Carlos Jorge de Sousa Gonçalves

Mestre em Engenharia Informática

Parallel and Distributed Statistical-based Extraction of Relevant Multiwords from Large *Corpora*

Dissertação para obtenção do Grau de Doutor em

Informática

Orientador:	José Alberto Cardoso e Cunha, Professor Catedrático (Aposentado), Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
Co-orientador:	Joaquim Francisco Ferreira da Silva, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente:	José A. Legatheaux Martins, Prof. Catedrático, FCT/Univ. Nova de Lisboa
Arguentes:	Salvador P. de Abreu, Prof. Catedrático, Univ. de Évora
	Pedro M. Pinto Ribeiro, Prof. Auxiliar, Fac. de Ciências da Univ. do Porto
Vogais:	José A. Cardoso e Cunha, Prof. Catedrático, FCT/Univ. Nova de Lisboa
	Manuel M. Barata, Prof. Coordenador, ISEL/Instituto Politécnico de Lisboa
	Vítor J. Ramos Rocio, Prof. Associado, Dep. Ciências e Tec. da Univ. Aberta
	Nuno M. Carvalheiro Marques, Prof. Auxiliar, FCT/Univ. Nova de Lisboa
	Vítor M. Alves Duarte, Prof. Auxiliar, FCT/Univ. Nova de Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2017

Parallel and Distributed Statistical-based Extraction of Relevant Multiwords from Large *Corpora*

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Para todos os que, directa ou indirectamente, tornaram este trabalho possível.

Acknowledgements

Ao terminar este projeto de doutoramento encerro um importante marco no meu percurso de vida, onde só a força de vontade e perseverança tornaram possível trilhar este árduo caminho repleto de constantes desafios pautados por avanços e recuos, deceções e vitórias. Embora se trate de um caminho por vezes longo e solitário, felizmente o meu não foi percorrido na solidão. Tive o privilégio de contar com o apoio de uma rede de pessoas e instituições a quem quero expressar os meus mais sinceros agradecimentos.

Em primeiro lugar quero agradecer ao meu orientador e co-orientador, os Professores José Cardoso e Cunha e Joaquim Ferreira da Silva, não só pelo sentido de rigor científico e exigência desafiadora; mas sobretudo pelo apoio, constante disponibilidade na troca de ideias e discussão de alternativas e estratégias a seguir.

Aos membros da Comissão de Acompanhamento da Tese, os Professores Omer Rana, Gaël Dias, Vítor Rocio e Vítor Duarte, o meu agradecimento pelos importantes contributos no âmbito da realização das provas intermédias de avaliação deste trabalho.

No plano institucional, quero agradecer os apoios disponibilizados pela Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (FCT/UNL), pelo Instituto Politécnico de Lisboa (IPL) e pelo Instituto Superior de Engenharia de Lisboa (ISEL). Nomeadamente, o apoio logístico e financeiro prestado pelo Departamento de Informática da FCT/UNL e pelo centro de investigação NOVALINCS que possibilitou a minha participação e consequente divulgação do presente trabalho, em vários painéis de conferências de âmbito nacional e internacional, assim como, a comparticipação dos custos decorrentes do trabalho de investigação e experimentação prática, nas pessoas do Presidente do Departamento de Informática, Professor Luís Caires; do coordenador do programa doutoral de informática, Professor Nuno Correia e aos membros do secretariado do Departamento de Informática: Anabela Duarte, Sandra Rainha, Susana Pereira e Joana Dâmaso.

Sendo Professor a tempo integral no ISEL, na Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores (ADEETC), a realização deste trabalho só foi possível com o apoio disponibilizado pelo IPL/ISEL, através do "Programa de apoio à formação avançada de docentes do Ensino Superior Politécnico" (bolsa PROTEC), pelas condições de trabalho proporcionadas pela ADEETC e pelo Grupo de Investigação Aplicada em Tecnologias de Informação (GIATSI) da ADEETC.

Sendo este um projeto de investigação suportado por uma forte componente de experimentação prática realizada em ambientes de *cloud* públicas com custos de operacionalidade associados, não posso deixar de agradecer à LunaCloud as condições facilitadoras de acesso, fundamentais para a conclusão deste estudo.

Um agradecimento a todos os amigos, por serem tão bons ouvintes, e aos colegas de trabalho, em especial, ao Luís Assunção, que mais do que um companheiro de viagem, foi acima de tudo um grande amigo. Obrigada Luís, não só pelas contribuições, mas sobretudo, pelas palavras de encorajamento nos momentos mais difíceis.

O meu agradecimento sentido a todos os familiares que me ajudaram e contribuíram positivamente para este trabalho. Nomeadamente, aos meus sogros, Delfina e Fausto e aos meus pais Elisabete e José, por todo o apoio e incentivo que me deram ao longo deste percurso.

Finalmente, o meu agradecimento de coração, aos meus filhos, Guilherme e Francisco e à minha esposa Estela, por estarem sempre ao meu lado, por toda a paciência e compreensão ao longo destes anos, pois só o vosso amor incondicional me alimentou nesta caminhada.

And, when you want something, all the universe conspires in helping you to achieve it

---

Paulo Coelho

Abstract

The amount of information available through the Internet has been showing a significant growth in the last decade. The information can result from various sources such as scientific experiments resulting from particle acceleration, recording the flight data of a commercial aircraft, or sets of documents from a given domain such as medical articles, news headlines from a newspaper, or social networks contents.

Due to the volume of data that must be analyzed, it is necessary to endow the search engines with new tools that allow the user to obtain the desired information in a timely and accurate manner. One approach is the annotation of documents with their relevant expressions. The extraction of relevant expressions from natural language text documents can be accomplished by the use of semantic, syntactic, or statistical techniques. Although the latter tend to be not so accurate, they have the advantage of being independent of the language. This investigation was performed in the context of LocalMaxs, which is a statistical method, thus language-independent, capable of extracting relevant expressions from natural language *corpora*.

However, due to the large volume of data involved, the sequential implementations of the above techniques have severe limitations both in terms of execution time and memory space. In this thesis we propose a distributed architecture and strategies for parallel implementations of statistical-based extraction of relevant expressions from large *corpora*.

A methodology was developed for modeling and evaluating those strategies based on empirical and theoretical approaches to estimate the statistical distribution of n-grams in natural language *corpora*. These approaches were applied to guide the design and evaluation of the behavior of LocalMaxs parallel and distributed implementations on cluster and cloud computing platforms. The implementation alternatives were compared regarding their precision and recall, and their performance metrics, namely, execution time, parallel speedup and sizeup. The performance results indicate almost linear speedup and sizeup for the range of large *corpora* sizes.

Keywords: Parallel and Distributed Computing; Extraction of Relevant Expressions; Statistical n-gram Methods; Caching Strategies.

Resumo

A quantidade de informação existente na Internet tem vindo a crescer a um ritmo significativo na última década. A informação pode ter origem em diversas fontes tais como experiências científicas que efetuam aceleração de partículas, o registo de dados do voo de um avião comercial ou o conjunto de documentos de um dado domínio tais como artigos médicos, notícias de um jornal ou conteúdos de redes sociais.

Tendo em conta o volume de dados a analisar, é necessário dotar os motores de busca de novas ferramentas que permitam ao utilizador comum obter a informação desejada em tempo útil. Uma abordagem possível é a anotação dos documentos com as suas expressões relevantes. A extração de expressões relevantes de documentos de texto de língua natural pode ser efetuada com recurso a técnicas semânticas, análise sintática ou técnicas estatísticas. Apesar de as últimas não serem tão precisas, têm a vantagem de serem independentes da língua. O trabalho desenvolvido nesta tese é baseado no método LocalMaxs que, por ser estatístico, pode ser utilizado para extrair expressões relevantes de textos de língua natural sem necessitar de conhecimento prévio da língua utilizada.

No entanto, devido ao grande volume dos dados envolvidos, as implementações sequenciais das técnicas acima referidas têm geralmente fortes limitações em termos de tempo de execução e espaço de memória. Nesta tese propõe-se uma arquitetura distribuída e estratégias de implementação paralela de métodos estatísticos de extração de expressões relevantes para *corpora* de grande tamanho.

Foi desenvolvida uma metodologia, baseada em abordagens empírica e teórica, para estimar a distribuição estatística dos *n*-grams em *corpus* de língua natural. As abordagens foram utilizadas na conceção e avaliação das implementações paralelas e distribuídas do método LocalMaxs em ambientes de computação em *cluster* e *cloud*. As implementações foram comparadas face à sua precisão, cobertura, tempo de execução e capacidade de expansão da escala para lidar com *corpora* de grande tamanho. Os resultados indicam um comportamento quase linear quanto à aceleração de desempenho e à capacidade de lidar com a escala, para a gama de *corpora* de grande tamanho.

Palavras-chave: Processamento Paralelo e Distribuído; Extração de Expressões Relevantes; Métodos Estatísticos de Extração de *n*-grams, Estratégias de *cache*.

Contents

List of Figures

List of Tables

Listings

Glossary

C	*Corpus*.
C_i	Individual cache for n-grams of size i.
$D_{All(1\cdots n)_{in}LocalPartition}$	Set of distinct n-grams from sizes 1 up to n in a local partition.
$D_{All(1\ldots n)_{in}C}$	Set of distinct n-grams from sizes 1 to n in the *corpus*.
D_i	Set of distinct n-grams of size i.
$D_{i_{in}D_n}$	Set of distinct n-grams of size i in table D_n.
$D_{i_{in}all_{glue_{g_n}Ref}}$	Set of distinct n-grams of size i within the input reference stream $all_{glue_{g_n}Ref}$.
E_0	Absolute efficiency relative to an ideal sequential machine.
$E_{K_1 \to K_2}$	Relative efficiency when going from K_1 to K_2 machines.
G	Granularity ratio of the computation time over the total overheads time.
K	Number of machines.
$NS_{All(1\ldots n)_{in}C}$	Set of nonsingleton n-grams from sizes 1 to n in the *corpus*.
NS_i	Set of nonsingleton n-grams of size i.
N_n	Problem size.
R_i	Average repetition factor of n-grams of size i.
$S_{All(1\ldots n)_{in}C}$	Set of singleton n-grams from sizes 1 to n in the *corpus*.
S_i	Set of singleton n-grams of size i.
Set_{All_i}	Set of all occurrences of n-grams of size i.
Sp_0	Absolute speedup relative to an ideal sequential machine.
$Sp_{K_1 \to K_2}$	Relative speedup when going from K_1 to K_2 machines.
T_{comm}	Time for intermediate communications during the execution of the LocalMaxs method.
T_{comp}	Computation time of the LocalMaxs functions with no overheads.
$all_{\Omega_i Ref}$	Set of remote references for applying the LocalMaxs relevance criterion to n-grams of size i.
$all_{glue_{g_n}Ref}$	Set of remote references needed for glue calculation g_n.
g_i	Glue calculation function (also used to denote the set of glue values of n-grams of size i).

$g_{n_1 \ldots n_2}$	Glue calculation function (also used to denote the set of glue values for the n-grams of size n_1 up to and including n_2).
h_{RO}	Hit ratio of static cache.
h_{RO+RW}	Hit ratio of a composite static plus dynamic cache.
h_{RW}	Hit ratio of a dynamic cache.
mr	Miss ratio of the cache.
mr_{Cold}	Cold-start miss ratio.
mr_{RO}	Miss ratio of a static cache.
mr_{RO+RW}	Miss ratio of a composite static plus dynamic cache.
mr_{RW}	Miss ratio of a dynamic cache.
mr_{Warm}	Warm-start miss ratio.
$re_{2 \ldots n}$	Relevance evaluation function (also used to denote the set of relevant expressions of bigrams up to n-grams of size n).
$t_{n\text{–}gram}$	Average time for sending or fetching an n-gram to/from the KVS servers.
t_{op}	Average time for locally executing a basic algorithm operation.

Acronyms

Amazon S3	Amazon Simple Storage Service.
Amazon AWS	Amazon Web Services.
API	Application Program Interface.
AWARD	Autonomic Workflow Activities Reconfigurable and Dynamic.
BATON	BAlanced Tree Overlay Network.
CAN	Content Addressable Network.
CN	Computing Node.
CPU	Central Processing Unit.
DASS	Data Access Storage Service.
DBMS	Database Management System.
DHT	Distributed Hash Table.
Ew	Exaword (10^{18}).
FAset	Fixed Frequency Accumulation Set.
G*n–gram*	Giga *n*-gram (10^9).
GB	Gigabyte (10^9).
GFS	Google File System.

Gw	Gigaword (10^9).
HDFS	Hadoop Distributed File System.
HTML	HyperText Markup Language.
IBS	Input Buffer Size.
JVM	Java Virtual Machine.
KVS	Key-Value Store.
MPI	Message Passing Interface.
Mw	Megaword (10^6).
NFS	Network File System.
OpenMP	Open Multi-Processing.
P2P	Peer-to-peer.
PB	Petabyte (10^{15}).
Pw	Petaword (10^{15}).
RMI	Remote Method Invocation.
SCP	Secure Copy Program.
SSH	Secure Shell.
T*n–gram*	Tera *n*-gram (10^{12}).
TB	Terabyte (10^{12}).
TCP/IP	Transmission Control Protocol/Internet Protocol.

Tw	Teraword (10^{12}).
vCPU	virtual CPU.
VM	Virtual Machine.
XML	eXtensible Markup Language.

Part I

Introduction and Background

Chapter 1

Introduction

The building of scalable solutions for enabling data-intensive applications, as found in science and engineering or in business data analytics, for example in Web applications and social networking sites, requires capabilities to provide relevant information to users and applications. Due to such huge amount of generated information, there is a need for methods, algorithms and tools that enable an automatic identification and extraction of relevant expressions from documents, so that their analysis and processing are made feasible.

1.1 Objectives

Importance of Extraction Methods. Document descriptors can be built automatically from their most relevant n-grams, contributing to guide a more focused search. In the context of this work an n-gram is a sequence of n consecutive words, where n is the n-gram size. An expression or term is an n-gram with a defined morphological class. As the number of Internet users and applications grows, the volume of generated and stored data is increasing very rapidly. Due to such huge amount of generated information, there is a need for methods, algorithms and tools that enable an automatic identification and extraction of relevant expressions from documents, so that their analysis and processing are made feasible. This is only possible by relying on parallel and distributed computing, with access to large numbers of processors and massive storage capabilities.

The automatic extraction of relevant expressions from raw text is an area of great interest. Relevant expressions can be extracted by different approaches: symbolic/linguistic, statistical or hybrid, as discussed with more detail in chapter 2.

Challenges Regarding the Extraction of Relevant Expressions. The extraction of relevant expressions involves several challenges and difficulties encompassing, on one hand, general issues concerning the extraction approaches and their practical validation. For example, how to deal with the complexity of interpreting and detecting the significant terms or expressions (based on syntactic, semantic, statistical or hybrid methods); how to meet the increasing need of dealing with a diversity of different languages, and conciliating this with favoring neutral approaches as happens with statistical-based ones, but also enabling specific domain and semantic interpretations; and how to integrate automatic extraction and user interaction tools to validate the obtained results. On the other hand, in the last decade, due to the rapid growth of Internet information systems and applications, there have been increasing concerns with large-scale data extraction issues, and the need to deal with the diversity of Internet access media and the growing amount of generated information, for example in Web and social networking applications. Those concerns led to an increased interest in the investigation of solutions to address the large scale of generated information and its dynamics, the need of filtering the raw information using knowledge based on the application domain and the user profiling in order to extract relevant information, and devising efficient solutions to process such large scale of information in acceptable time and memory space.

Problem Description. In this dissertation we propose solutions to the extraction of relevant expressions, up to hexagrams, from large *corpora*, using parallelism to reduce the execution time and distribution to handle large data sets. The investigation is centered on the LocalMaxs method but can also be applied to other extraction methods and applications based on n-gram models. The proposed solutions (to the extraction of relevant expressions) must accomplish an adequate trade-off between execution time and solutions quality, defined in terms of the precision and recall metrics, and computing resources/cost.

LocalMaxs [SL99; Sil+99] is a statistical method, thus language-independent, capable of extracting relevant expressions of a given size from a given *corpus*, based on a cohesion (also named "glue" in this document) metric and a relevance criterion. It relies on an exhaustive approach based on counting all occurrences of the n-grams in a *corpus*. In the context of this work we use the expressions "number of occurrences of an n-gram in the *corpus*" or "frequency of an n-gram in the *corpus*" as equivalent. The typical precision and recall achieved by this method are around 75 % [SL99] when considering n-grams up to hexagrams, but to achieve these values it requires a document *corpus* of at least 1 Megaword (10^6) (Mw) as reported in [SL99]. However, its sequential implementations pose limitations in terms of execution time and memory consumption, being unable to process large *corpora*, typically beyond a few hundred million words [SL99].

Considered Approaches. We assume that the *corpora* being analyzed are located in an accessible storage repository, and that the *corpus* data sets are static and unchanged during

the entire extraction process. Issues concerning geographical data distribution as well as dynamic and continuous data generation in data streaming systems, although important, are left out of the scope of this dissertation.

We developed an approach to support extraction methods, which are based on a first phase collecting *n*-gram *corpus* statistics, followed by other phases to execute different methods for calculating the relevance of *n*-grams by applying specific filtering criteria. An entire application can process a given input *corpus* as a single shot batch job, or any of its individual phases can be executed on its own. For example, after the first phase, multiple independent analyses can be performed based on the *n*-gram statistics describing the *corpus* characteristics.

The functionality to support the sequencing and execution of the application phases and the storage of the intermediate *n*-gram data is provided by a generic distributed architecture for *n*-gram based extraction. It relies on a collection of distributed machines that supports the parallelization of the extraction functions and the distribution of the *n*-gram data. This architecture was instantiated to support the execution of the LocalMaxs functions with their associated *n*-gram data structures.

Issues regarding the memory space efficiency of the *n*-gram data representation raise an important concern that can influence the overall performance of an algorithm implementation. This has been intensively studied in related works as mentioned in chapter 2, but it was left out of the scope of this dissertation, because our main goal was to study the influence of alternative parallel and distributed processing approaches to implement relevant *n*-gram extraction based on LocalMaxs.

When considering each specific computing infrastructure there is a limit to the number of available machines and to the maximum per machine available memory. This poses a constraint upon the maximum size of the *corpora* and the maximum *n*-gram size that can be processed by the proposed architecture, so that the total memory required by each phase of the algorithm may fit into the total aggregated memory of the collection of available machines. If a more efficient *n*-gram representation is used instead, the above size constraint will occur at slightly higher *corpus* sizes than with the currently developed implementation, but eventually it will also occur beyond a certain *corpus* scale.

However, from the conducted experimentation for the range of analyzed *corpora*, the above memory/machine constraint did not preclude the demonstration of the usefulness of the proposed approaches as far as parallelism and distribution are concerned. Furthermore, per machine local memory optimizations can be applied orthogonally to the proposed parallel and distributed architecture.

Two methods were considered for the parallel and distributed implementation of LocalMaxs. Both methods use data-parallel approaches to achieve acceptable execution time and use multiple machines to implement a scalable distributed memory model: i) The Global method ensures the same precision and recall as the LocalMaxs definition, but incurs a significant communication overhead for the cohesion (glue) and relevant expression calculations; ii) The Local method gives approximate results when compared

to the LocalMaxs definition, and also allows a data-parallel approach but has a reduced communication volume (compared to the Global method) that improves its performance and scalability.

The two methods are evaluated regarding a trade-off between the solutions quality (precision and recall) compared to the LocalMaxs definition and the system performance (execution time, memory space, parallel efficiency, scale), in a broad spectrum of *corpus* sizes and n-gram sizes.

1.2 Contributions and Main Results

The main contributions of this thesis are discussed in detail in chapters 3 to 9. They are briefly identified in the following:

Statistical *n*-gram Distribution in Natural Language *corpora*. We conducted an analysis of the statistical distribution of n-grams in natural language *corpora* and applied it to understand the behavior of the alternative parallel LocalMaxs implementations, including the design and analysis of an n-gram cache. Three dimensions were considered: i) **An empirical analysis** of the n-grams distribution, from unigrams to hexagrams, in *corpora* up to 1 Gigaword (10^9) (Gw), that was used as input to an analytical model of the behavior of the parallel LocalMaxs implementations; ii) **A theoretical model** for efficiently estimating the number of distinct unigrams, bigrams, ..., hexagrams, for any *corpus* size, based on Zipf-Mandelbrot Law and Poisson distribution. It was used to estimate the asymptotic behavior of the parallel LocalMaxs implementations for very large *corpora* when the numbers of distinct n-grams tend to constant *plateaux* as the *corpus* size increases beyond well-identified thresholds; iii) **The concept of Fixed Frequency Accumulation Set (FAset)**, to model the cumulative sum of frequencies of a set of distinct n-grams that represents a percentage of the total n-gram occurrences in a *corpus*. This is useful to determine the minimal n-gram subset necessary to ensure a desired *corpus* coverage, identifying the culprit n-grams which should be kept in an n-gram cache, and also guiding possible cache prefetching and replacement strategies;

A Distributed Architecture for *n*-gram Extraction Applications with Multiple Phases. This distributed architecture is capable of executing extraction algorithms based on statistical n-gram models on cluster- or cloud-based infrastructures. It is based on a distributed collection of virtual machines: i) Each machine contains one or multiple **controller** processes to execute the algorithm functions, and one or multiple servers to contain tables with n-gram information; ii) The servers collectively implement a **distributed in-memory key-value store** supporting the intermediate n-gram data shared between multiple application phases; iii) Each machine contains an n-gram cache system, exploiting the locality and the statistical knowledge on the n-gram references to reduce the volume of remote accesses to the key-value store; iv) **A workflow framework** is used to coordinate the

execution of extraction applications with multiple phases and the mapping of the application components into the virtual machines supported by the underlying computing infrastructures.

Parallel and Distributed LocalMaxs Implementations. Two main approaches were considered. One is based on a global method (denoted as Global method) which ensures the same precision and recall as the LocalMaxs method definition. Its implementation, based on the above distributed architecture, exhibits almost linear relative speedup and sizeup, when executed in a public cloud environment with up to 54 virtual machines, for the extraction of relevant bigrams and trigrams from English *corpora* up to 1 Gw. A theoretical analysis of the Global method behavior predicts that the above performance metrics exhibit the same linear evolution even beyond *corpus* sizes of 1 Gw.

The other approach is based on a local method (denoted as Local method) whose precision and recall tend to be lower than LocalMaxs, but they tend to increase with the *corpus* partitions size. Its performance (execution time and scalability) and solutions quality (precision and recall) are evaluated and compared to the Global method, for the considered *corpus* ranges up to 1 Gw and also estimated for larger *corpus* sizes.

Study of the *n*-gram Cache Behavior. To reduce the remote data communication, a novel *n*-gram cache system, composed of a set of individual caches for different *n*-gram sizes, was designed and implemented. The *n*-gram cache system can be configured as an on-demand dynamic cache with a cooperative-based warm-up strategy leading to reduce the miss ratio and miss time penalty; or it can also be configured as a static prefetching cache based on the FAset concept, leading to further reductions in the miss ratio. Both cache configurations can be complemented with an integrated Bloom filter to select the singleton *n*-grams existing in the *corpus*.

A cache analytical model was proposed to estimate the performance of the Global method, considering cache cold-start and cache warm-up scenarios, for the glue calculation of *n*-gram expressions, based on the *corpus* empirical data. The model estimates agree with the real execution results. The model was also used to predict the *n*-gram cache behavior for the very large *corpora* asymptotic *plateaux*. The implementation can be configured to support a finite or infinite *n*-gram cache, depending on each algorithm memory requirements for each *corpus* size and according to the available memory per machine.

1.3 Methodology

Figure 1.1 illustrates the main dimensions of the methodology that guided this work.

The results of the analysis of the statistical distribution of *n-grams* in natural language *corpora* were provided as input to an analytical performance model to estimate the expected behavior of each approach regarding the execution time, when varying the

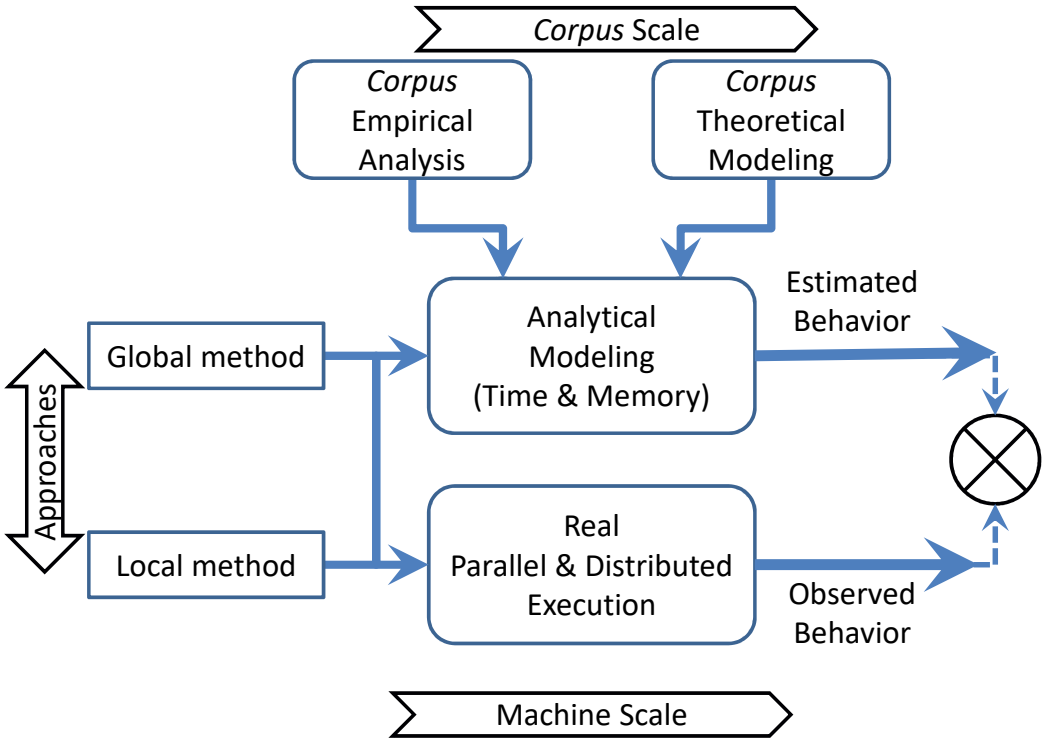


Figure 1.1: Methodology followed.

corpus size, the *n-gram* size, and the number of machines. This analysis was based on the empirical data for the *corpora* sizes up to 1 Gw, and was based on the theoretical model to identify the asymptotic behavior of the *n-gram* distribution for very large *corpora*. The analytical performance model estimates were then compared to the real execution results, obtained from a parallel and distributed implementation of each approach.

1.4 Publications

The following publications resulted from the work done in the context of this dissertation.

1. J. F. Silva, C. Gonçalves, and J. C. Cunha, *"A Theoretical Model for n-gram Distribution in Big Data Corpora"*. In 2016 IEEE International Conference on Big Data (IEEE Big Data 2016), December 2016, pp. 134—141. doi: 10.1109/BigData.2016.7840598

2. C. Gonçalves, J. F. Silva and J. C. Cunha, *"An n-gram Cache for Large-scale Parallel Extraction of Multiword Relevant Expressions with LocalMaxs"*. In 12^{th} IEEE International Conference on eScience (eScience 2016), October 2016, pp. 120—129. doi: 10.1109/eScience.2016.7870892

3. C. Gonçalves, J. F. Silva and J. C. Cunha, *"A Parallel Algorithm for Statistical Multiword Term Extraction from Very Large Corpora"*. In 17^{th} IEEE International Conference on High Performance Computing and Communications (HPCC 2015), August 2015, pp. 219—224. doi: 10.1109/HPCC-CSS-ICESS.2015.72

4. C. Gonçalves, L. Assunção, and J. C. Cunha. *"Flexible MapReduce Workflows for Cloud Data Analytics"*. In International Journal of Grid and High-Performance Computing (IJGHPC), Special Issue on Cloud Computing - Algorithmic, Experimental, Prototyping and Implementation, vol. 5, no. 4, pp. 48—64, October-December 2013. ISSN 1938-0259. doi: 10.4018/IJGHPC

5. C. Gonçalves, L. Assunção, and J. C. Cunha. *"Data Analytics in the Cloud with Flexible MapReduce Workflows"*. In 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012), pages 427—434. IEEE Computer Society, December 2012. doi: 10.1109/CloudCom.2012.6427527

6. L. Assunção, C. Gonçalves, and J. C. Cunha. *"Autonomic Workflow Activities: The AWARD Framework"*. In International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), vol. 5, no. 2, pp. 57—82, April-June 2014. ISSN 1947-9220. doi: 10.4018/IJARAS

7. L. Assunção, C. Gonçalves, and J. C. Cunha. *"Autonomic Activities in the Execution of Scientific Workflows: Evaluation of the AWARD Framework"*. In 9^{th} International Conference on Autonomic Trusted Computing (ATC 2012), pages 423 –430. IEEE Computer Society, September 2012. doi: 10.1109/UIC-ATC.2012.14

8. L. Assunção, C. Gonçalves, and J. C. Cunha. *"On the Difficulties of Using Workflow Tools to Express Parallelism and Distribution - A Case Study in Geological Sciences"* (GPC 2009), pages 104—110. In International Workshop on Workflow Management of the International Conference on Grid and Pervasive Computing, IEEE Computer Society, 2009. doi: 10.1109/GPC.2009.30

1.5 Document Organization

Figure 1.2 shows the organization of this document in 5 parts. Part I, **Introduction and Background**, is composed of this chapter and chapter 2 that briefly surveys the background and related work. Part II, ***n*-gram Statistical Distribution**, is composed of chapter 3, where we analyze the statistical distribution of *n*-grams in natural language *corpora*. Part III, **Parallel and Distributed *n*-gram Extraction**, is divided in two chapters: 4 and 5 describing a distributed architecture for *n*-gram extraction and the implementation of the LocalMaxs Global method. Part IV, ***n*-gram Cache Models**, is presented in chapters 6 to 8, namely: chapter 6 describes a dynamic *n*-gram cache system; chapter 7 presents an *n*-gram cache system composed of a static cache followed by a dynamic one; chapter 8 describes the influence of filtering the singleton *n*-grams out of the cache system (dynamic or static plus dynamic) using Bloom filters. In Part V, **Global Evaluation**, we present an evaluation of the parallel and distributed LocalMaxs implementations, chapter 9, and the conclusions and future work in chapter 10. The document ends with the Bibliography and three appendixes: A and B, where we present an example of the configuration files

used in the developed architecture; and C, containing complementary data for the static plus dynamic cache system.

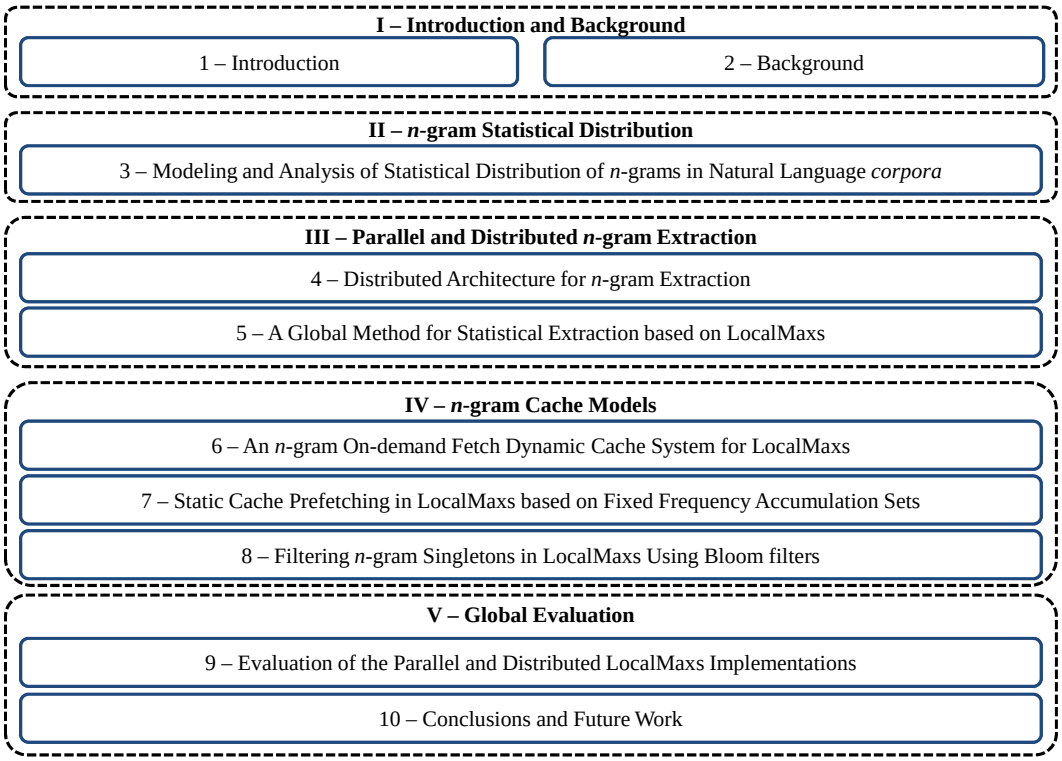


Figure 1.2: Document organization.

Chapter 2

Background

This chapter discusses background and related work in the context of this thesis.

The main goal of this thesis was to develop **parallel and distributed** solutions in order to improve the execution performance of computational methods for **statistical-based extraction of relevant expressions** in **natural language *corpora***. The focus of this work lies on the intersection of the areas represented in Figure 2.1.

The extraction of relevant expression in natural language *corpora* is related to the data mining area in the sense that it involves the process of finding meaningful patterns in data and present those patterns to the end users. Because of the complexity of those

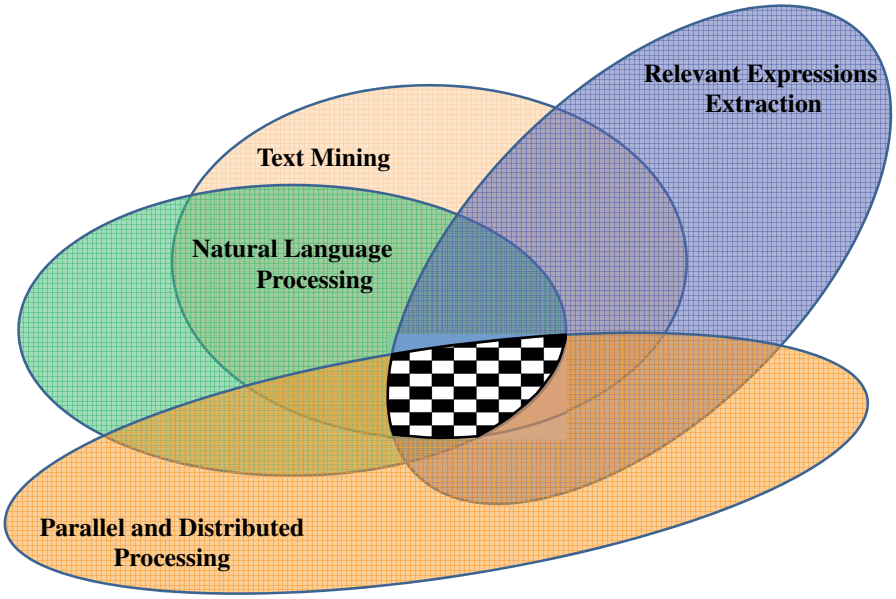


Figure 2.1: Areas in the research context.

problems it is common that data mining uses methods and tools from different domains such as computer science, statistics and mathematics (section 2.1).

Furthermore, due to the large amount of data typically involved in the actual Internet information systems and applications related to the field commonly referred to as Big Data, one of the important issues to be addressed is the capability to analyze large scale data in a timely manner, requiring the usage of extensive computing resources made available through parallel and distributed infrastructures (section 2.2).

The thesis encompasses the development of parallel implementations of the statistical-based extraction method LocalMaxs in distributed cluster and cloud computing infrastructures, and the study of their execution performance behavior concerning the parallel speedup and scale.

2.1 Extraction in *Corpus* Mining

In this section a succinct discussion of the approaches for extraction of relevant expression in natural language *corpora* is first presented (section 2.1.1) including the description of an *n*-gram based statistical extraction method, LocalMaxs (section 2.1.2).

2.1.1 Extraction of Relevant Expressions

Terminology mining, term extraction, term recognition, glossary extraction or relevant expression extraction are terms denoting a subtask of information extraction. The goal of extractors is to automatically identify relevant expressions from a given *corpus*.

One of the difficulties of extracting relevant expressions using statistical approaches is the quantity of data that must be processed to obtain good precision and recall, i.e., correctly identify an *n-gram* as a valid relevant expression (for good precision), and extract the greatest possible amount of existing relevant expression (for good recall). Precision, also called positive predictive value, can be defined as:

$$precison = \frac{\text{number of Relevant Expressions correctly identified by the Algorithm}}{\text{number of Relevant Expressions identified by the Algorithm}} \quad (2.1)$$

and recall, also known as sensitivity, can be defined as:

$$recal = \frac{\text{number of Relevant Expressions correctly identified by the Algorithm}}{\text{number of Relevant Expressions in the corpus}} \quad (2.2)$$

Due to the semantic richness of the relevant expressions, their automatic extraction from raw text has been an area of great interest. Documents can be summarily described by a set of relevant expressions such as "United Nations", "human rights", "International Court of Justice", etc.. Relevant expressions may also be used to index documents. Besides, they can be used as discriminant attributes to cluster documents. Relevant expressions

can be extracted by different approaches: symbolic/linguistic, statistical or hybrid. Basically, linguistic and hybrid approaches such as the ones mentioned in [Cop+02; Dai96; Dia03; KF11; MV10; Sin11; WSN10] or [BJ99; FAM00; PBB02; VND08; WH05; WLB07; Won07] need specific language information, usually syntactic filters such as Noun-Noun, Adjective-Noun, Verb-Noun, etc. to help on the extraction or on the identification of the relevant expressions type. However, as the texts have to be morphsyntactically tagged, this imposes a linguistic dependency — not all languages have high quality taggers and parsers available, especially when languages are unknown.

Besides, relevancy is not completely determined by morphsyntactic patterns. For instance, "triangle angle" and "greenhouse effect" share the Noun-Noun pattern, however only the second one can be considered relevant. Furthermore, most Noun phrases are not really relevant. Other approaches such as the ones in [Att+10] or [NV04] depend on other tools, such as WordNet and Wikipedia, usually available just for a small set of languages. Several statistical metrics used to extract relevant expressions, such as Mutual Information [CH90], Likelihood Ratio [Dun93], $\varphi 2$ [GC91], etc., have been used. Some of these metrics and others were evaluated in [Pea02]. The main problem with these metrics is that they only assess bigrams, i.e., sequences consisting of only two words. To circumvent this problem and to extract longer relevant expressions, other metrics and extractor algorithms such as LocalMaxs [Sil+99] have been proposed, with the advantage of being a pure language-independent extractor, but their sequential implementations perform poorly in terms of execution time and memory space.

The importance of extracting relevant expressions from text *corpus* is confirmed by a set of existing tools available nowadays. Some of them can be used directly on raw text but others are specific to working with well-defined formats. For example, this is the case of Anchovy [Anc13] that is a free multilingual cross-platform glossary editor and term extraction tool based on the open Glossary Markup Language [Glo13] format. However, many tools allow the extraction of relevant expressions from text documents. For example Yahoo offers some applications like Yahoo Quest [Yah13b] or Web services API, e.g., Content Analysis Yahoo [Yah13a], previously known as Yahoo Term Extraction [Yah13c] allowing the user to perform term extraction from documents. Tools like Termine [Ter13], Terminology Extraction [Ter13b] and Ngram Statistics Package [Ngr13] (formerly known as the Bigram Statistics Package) are other examples of tools to extract terms from text *corpora* that rely on statistical algorithms, such as C-value algorithm [FAM00], Poisson statistics, the Maximum Likelihood Estimation, Inverse Document Frequency, Fisher's exact test, the log Likelihood Ratio, Pearson's chi-squared test, the Dice Coefficient. Nevertheless they are tuned to work only with some languages, for example, English, Italian and French in the case of Terminology Extraction. There are also some open source initiatives. For example Term Extraction [Ter13a] aims to be a simple and free alternative to Yahoo's Term Extraction service. It is written in the Python language relying on Topia's Term Extraction [Top13], a package that determines important terms within a given piece of text [JSO13]. Sematext Key Phrase Extractor [Sem13] is a toolkit capable of extracting

relevant expressions ordered by rank that offers both a Java and HTTP API allowing its integration with other tools. Finally the AlchemyAPI [Alc13] is a text mining platform, being executed as a service in the cloud that, besides the extraction of relevant expressions from text, also allows other functionalities such as sentiment analysis, concept tagging, author extraction, relations extraction, etc.

2.1.2 *n*-gram Based Data Extraction

A diversity of extraction approaches are based on *n*-gram models as discussed in section 2.1.2.1. The LocalMaxs method is presented in section 2.1.2.2.

2.1.2.1 *n*-gram Models and Data Structures

Multiple language models have been investigated based on *n*-gram models [Gao+; HC04], and their relationships with probabilistic models of the *n*-gram distribution in natural language *corpora* have been studied [Egg07; LW05]. The latter studies have also been applied to the design and evaluation of search engines [Bre+99]. Namely, statistical studies on the history of word occurrences in natural language *corpora* have been made in different contexts: in particular cache-based *n*-gram models for linguistic applications were proposed by [Kuh88]. The relationships between the Zipf distribution [Zip35] and caching for Web search engines have also been extensively discussed [BY+07; Bre+99].

Issues related to data structures for large-scale applications concern the use of efficient indexing schemes [HZ03; HMC11; Kim+05; Kra+11; LB97], namely in the context of parallel and distributed processing [GD03a; GD03b; Mel+01].

Time and space optimizations have also been sought concerning data structures for representing *n*-grams, namely, regarding the use of suffix trees and arrays [Arr+14; MM90; MN03; McC76]. This thesis is centered on the extraction of relevant expressions in large *corpora* and memory limitations were overcome by using more machines. The use of local optimized data structures, such as suffix trees and arrays, is an orthogonal dimension to the work presented in this thesis, and their integration with the parallel LocalMaxs implementation developed will allow to increase the size of the *n-gram* data tables.

In this thesis we have also studied the influence of the statistical *n*-gram distribution in natural language *corpora* upon the design of parallel and distributed solutions to an *n*-gram based extraction method. Although the conducted developments were centered on LocalMaxs, the results of the statistical analysis can be applied to other *n*-gram based methods. A related dimension of this thesis concerns the proposal of a theoretical model for estimating the number of distinct *n*-grams, as a function of the *corpus* size and *n*-gram sizes, and its use to predict the asymptotic behavior of *n*-gram based applications, for very large *corpora* sizes. Comparing to other theoretical models, as discussed in chapter 3, we extended previous studies by considering *n*-grams beyond unigrams, and we also achieved better fittings of the model estimates *versus* the actual *corpora* data, validated with different languages [Smi12; TZ92; VV13].

2.1.2.2 The LocalMaxs Method

The LocalMaxs [SL99; Sil+99] method finds n-grams that can be classified as relevant expressions based on the cohesion/glue sticking the words together within an n-gram. Different n-grams usually have different cohesion values. One can intuitively accept that there is a strong cohesion between the words within the bigram "Giscard d'Estaing" or within the pentagram "European Court of Human Rights". However, one cannot say that there is a strong cohesion within the bigram "or uninterrupted" or within the bigram "of two". $SCP_f(.)$, SCP (Symmetric Conditional Probability), was defined [SL99; Sil+99] as a metric that can be used to calculate the glue of each n-gram that exists in a given *corpus*.

Definition 2.1: Cohesion SCP

$$SCP_f(w_1 \cdots w_n) = \frac{f(w_1 \cdots w_n)^2}{\frac{1}{n-1}\sum_{i=1}^{n-1} f(w_1 \cdots w_i) \times f(w_{i+1} \cdots w_n)}$$

where $f(w_1 \cdots w_n)$ is the frequency (the number of occurrences) of the n-gram $(w_1 \cdots w_n)$ in the *corpus*.

The denominator has the frequencies of all the leftmost and rightmost sub n-grams of sizes 1 to $n-1$ contained in the n-gram $(w_1 \cdots w_n)$. For example, f("European Court of Human Rights") is the number of occurrences of this pentagram in a *corpus*. Table 2.1 shows the leftmost and rightmost sub n-grams, that is two unigrams (*1*-gram), two bigrams (*2*-gram), two trigrams (*3*-gram), and two tetragrams (*4*-gram), needed for calculating the glue of that pentagram.

Table 2.1: Sub n-grams for *"European Court of Human Rights"*.

sub *n*-grams	**Leftmost and rightmost sub *n*-grams**
sub *1*-grams	{European}, {Rights}
sub *2*-grams	{European Court}, {Human Rights}
sub *3*-grams	{European Court of}, {of Human Rights}
sub *4*-grams	{European Court of Human}, {Court of Human Rights}

Definition 2.2: LocalMaxs Criterion

Let $W = (w_1 \ldots w_n)$ be an n-gram and $g(.)$ a generic cohesion metric. Let $\Omega_{n-1}(W)$ be the set of $g(.)$ values for all contiguous $(n-1)$-grams within the n-gram W; Let $\Omega_{n+1}(W)$ be the set of $g(.)$ values for all contiguous $(n+1)$-grams containing the n-gram W. The LocalMaxs method says that W is a relevant expression if and only if:

$$\forall_x \in \Omega_{n-1}(W), \forall_y \in \Omega_{n+1}(W)$$

$$length(W) = 2 \wedge g(W) > y \quad \vee \quad length(W) > 2 \wedge g(W) > \frac{x+y}{2}$$

Using the same example, for the n-gram $W = (European\,Court\,of\,Human\,Rights)$, the sets Ω_{n-1} and Ω_{n+1} are obtained as follows:

- $\Omega_{n-1}(W) = \{g(European\,Court\,of\,Human), g(Court\,of\,Human\,Rights)\}$
- $\Omega_{n+1}(W) = \{g(Y)\}$, such that $Y = \left(w_{Left}\,European\,Court\,of\,Human\,Rights\right)$ or $Y = \left(European\,Court\,of\,Human\,Rights\,w_{Right}\right)$ where w_{Left} and w_{Right} are unigrams appearing in the *corpus*.

In this work the cohesion metric $g(.)$ is defined by $SCP_f(.)$ (Definition 2.1). The criterion (Definition 2.2) is used to decide if a given n-gram can or can not be considered a relevant expression. Using this criterion the method outputs a list of relevant n-grams for each value of n.

2.2 Use of Parallel and Distributed Computing

Sequential implementations of the LocalMaxs method have performance limitations concerning the memory space and time needed to process large *corpora* beyond a few hundred million words, making it impractical to use a single commodity computer to process all this information sequentially.

In this section parallel approaches capable of supporting the data extraction are briefly surveyed (section 2.2.1), as well as distributed approaches to handle the data distribution required by the processing of very large *corpora* (section 2.2.2).

2.2.1 Parallelism

Due to the large amounts of data to be processed it is necessary to rely on parallel and distributed computing. The decomposition of an application towards parallelism can be specified by using generic or specific programming languages, or parallel programming libraries. There are many parallel programming models, ranging from the ones based on message passing (such as Message Passing Interface (MPI) [Gab+04]) or shared-memory (such as Open Multi-Processing (OpenMP) [Ope16]) to the implicitly parallel models such as MapReduce [DG04]. Different forms of parallelism can be exploited. In the data-parallel approach the same set of functions is applied to different data partitions. In the task-parallel approach multiple tasks are executed in parallel, each one performing a different function. Hybrid approaches exploit both data and task parallelism. According to the nature of the applications, their execution can be implemented as a single phase or in multiple phases.

The MapReduce programming model [DG04] has been widely used to support parallel approaches, for example, in data intensive text processing [LD10], computing n-grams statistics [BB13] and text mining applications [Bra+07; Dye+08; ELO08]. MapReduce provides a transparent and easy-to-use parallel programming abstraction. However, it requires the application to be modeled as a single phase (with two steps: Map and Reduce)

and provides a fixed workflow template [Lee+12] with a single input and a single output. As most of the data analytics applications require multiple phases, the use of MapReduce implies overheads due to the lack of reutilization of the mapper and reducer instances across the application phases. Also the MapReduce model does not provide support for intermediate communications during the execution of the mappers and the reducers. The above limitations led to other proposals extending the MapReduce model such as Twister [Eka+10] and Spark [Zah+10].

In preliminary studies [GAC12; GAC13] we used MapReduce to count the frequencies of n-grams in a *corpus*, corresponding to the first phase of the LocalMaxs implementation. However, in the work supporting this thesis we used a generic workflow framework providing all the required functionalities to express the application decomposition in multiple phases (e.g., count frequencies, calculate glues and evaluate relevance) and their logical dependencies. In our approach a distributed in-memory key-value store was proposed to support the storage of intermediate n-gram data between phases and also allowing the access to intermediate data during the execution of each phase.

2.2.2 Data Organization and Distribution

Processing large text *corpora* requires mechanisms to organize, access and store large quantities of data. Although we can use relational databases to organize and access large quantities of data, the NoSQL approach is gaining more and more importance, aiming to support a large amount of unstructured data that suggests a form of access based on key/value pairs. This has motivated the emergence of newer access mechanisms, more efficient and scalable. For example this has influenced the design of Application Program Interface (API) for NoSQL database systems like Amazon SimpleDB [Ama13b] and Amazon DynamoBD [Ama13a].

Distributed Hash Tables. The above concern has also motivated developments around the concept of distributed hash tables. Conceptually a Distributed Hash Table (DHT) is a hash table where the key/values pairs are distributed among a set of nodes. Implementations like Content Addressable Network (CAN) [Rat+01], Chord [Sto+01] and its derivatives like [KK03], Pastry [RD01] and Tapestry [Zha+04; ZKJ01] are examples of DHT on top of peer-to-peer overlay networks. All these systems support the concept of DHT on an Internet-like scale. Kademlia [MM02] also follows DHT approaches like Chord or Pastry and was designed with the aim of including all the desirable features of previous systems in a single system while being more efficient. BAlanced Tree Overlay Network (BATON) [JOV05] is a distributed tree structure for Peer-to-peer (P2P) systems. Different from other overlays that use a DHT, such as in the Chord system, BATON builds a P2P overlay network based on a balanced tree structure. In consequence, both exact match and range queries are efficiently supported. Voldemort [Vol13a] is another example of a DHT that combines in-memory caching with the disk storage system so that a

separate caching tier is not required. Data are automatically replicated and partitioned over multiple servers so each server contains only a subset of the total dataset.

Big Table Abstractions. All the above efforts motivated the appearance of models and API centered on high-level structures, namely tables and their corresponding abstractions like column-oriented operators. BigTable [Cha+06], proposed by Google, is one well-known example of such systems. BigTable is a distributed storage system for managing structured data, designed to scale to a very large size, i.e., Petabyte (10^{15}) (PB) of data across thousands of commodity servers. Following BigTable other similar approaches were proposed such as Hypertable [Hyp13], HBase [The13b], Apache Accumulo [The13a] and Windows Azure Table Storage [Mic13b]. H-store [H-S13] (and its commercial version VoltDB [Vol13b] or Apache Cassandra [The16a]) are examples of simplified Database Management System (DBMS) that try to get the best of the two worlds: the SQL and the NoSQL approaches.

Distributed File and Storage Systems. Access to the raw data usually relies on some form of distributed file service consisting of a naming service (manages the names, their logical organization in directories as a meta level service and their mappings to the physical blocks) and storage service (manages the blocks of raw data). For example, for a conventional cluster in a local area network the above functionality is usually supported for example by the Network File System (NFS) system. Distributed file systems for high performance computing clusters, such as the Google File System (GFS) [GGL03] or Apache Hadoop Distributed File System (HDFS) [The12], have optimizations regarding performance and reliability. They provide high throughput access to large datasets. Similar approaches were followed for cloud environments, for example as in Sector/Sphere [GG; Sec13].

As examples of basic storage abstraction there are Amazon S3 [Ama16c] and Windows Azure Blob Storage [Mic13a]. Amazon S3 is a service made available by Amazon that can be used to store objects. Each object is stored and retrieved using a unique developer-assigned key. The objects are stored in buckets that are the fundamental container in Amazon S3. Within a bucket a user can store practically an unlimited number of objects.

Distributed-memory and Caching Approaches. Distributed-memory solutions enable scalable behaviors. In applications with multiple phases the intermediate data produced by each phase can be placed in a disk-based storage system or can be kept in an in-memory store [Ama13a; Cha+06; Kal+08; The13b; The16a]. The latter approach is preferable in terms of performance but requires the use of multiple machines to support a distributed in-memory store. Solutions based on this approach are currently feasible due to the evolution of the memory and communication support infrastructures [Mem16; Ous+10], and are widely used to support large-scale data processing.

Using a distributed-memory model with parallelism favors obtaining scalable solutions. However, the most critical issue towards scalability is related to the algorithm design, which ideally should ensure that the parallel efficiency remains constant when the problem size and the number of machines are scaled-up by a given factor. This depends on reducing the algorithm and the system overheads. Concerning the algorithm the computation-to-overheads ratio must be large enough to ensure an acceptable efficiency. Among the observed overheads in a distributed solution, the ones due to remote data communication are usually the most critical, due to the network latencies. Thus, it is most important to guarantee the use of local data. This can be achieved by an adequate partitioning of the processes and the data, and by taking advantage of the temporal and spatial data localities of the algorithm. To ensure the latter objective, existing systems rely on the use of caching techniques, ranging from the hardware and operating system levels [GAV14], to the server and client levels [WR91], and also including caches that exploit specific knowledge about the program or application behavior [Vei+99]. For example, caches have been integrated with distributed in-memory key-value stores [DSL10], in web search engines [IC97], and as domain caches in the context of natural language processing and text mining statistics [Kuh88; KM90; KM92; Ros00].

Among the solutions proposed for distributed execution of n-gram based applications, we point out the work by Balkir *et al.* [BFR11] that also proposes a distributed architecture supporting efficient access to large n-gram tables for text mining applications. They describe a layered architecture for n-grams access based on a Bloom filter, local caching and a distributed in-memory key-value store (Memcached [Mem16]). They present a detailed performance study on the remote communication latencies and scalability of several alternative solutions based on combining Bloom filters, caching and distributed storage (disk-based or in-memory). In their solution they take advantage of the large number of singleton n-grams in a typical natural language data set in order to optimize the time and space of the n-gram structures. Other solutions aiming at optimizing access to distributed n-gram structures are described in [Bra+07; LKM09].

In our approach we also propose a distributed memory architecture used to address the large memory requirements, jointly with a distributed in-memory key-value store to large-scale intermediate data. This is complemented with an n-gram cache system in each machine to exploit the locality behavior of natural language n-gram based extraction methods.

2.2.3 Parallel Performance Metrics

The following metrics can be applied to each individual phase of the algorithm or to the entire algorithm execution. Traditionally, the speedup of a parallel algorithm implementation is compared to the best know sequential algorithm implementation. However, for large scale *corpora*, sequential implementations are in general unavailable, thus we need to consider relative performance metrics, i.e., that use as a basis, the implementations

achieved when using a minimum number of machines capable of processing such large *corpus* size.

2.2.3.1 Absolute Performance Metrics

Regarding absolute metrics in the context of this work we considered the speedup and efficiency definitions as follows:

Definition 2.3: Absolute Speedup

We define the absolute speedup (Sp_0) of a parallel implementation with K machines relative to an ideal hypothetical sequential implementation with one machine with infinite memory capacity and null input/output access times, as follows:

$$Sp_0(K) = \frac{T_0}{T_{parallel}(K)}$$

where T_0 is the execution time of a sequential implementation with one ideal machine, i.e., it only includes the computation time of the algorithm for a fixed problem size, without any overheads, and $T_{parallel}(K)$ is the total execution time with K machines. When $Sp_0(K) = K$, the absolute speedup is said to be linear.

Definition 2.4: Absolute Efficiency

We define the absolute efficiency (E_0) in the same conditions as above, by the following expression:

$$E_0(K) = \frac{1}{K} \times Sp_0(K) = \frac{1}{K} \times \frac{T_0}{T_{parallel}(K)}$$

2.2.3.2 Relative Performance Metrics

Due to the limitations of using a sequential single machine implementation for large *corpora*, the following metrics compare the parallel implementations when using different numbers of machines above a certain minimum.

Definition 2.5: Relative Speedup

We define the relative speedup of a parallel implementation, when going from K_1 to K_2 machines ($Sp_{K_1 \to K_2}$) for a fixed problem size, as follows:

$$Sp_{K_1 \to K_2} = \frac{T_{parallel}(K_1)}{T_{parallel}(K_2)}$$

where $T_{parallel}(K)$ is the total execution time with K machines. When $Sp_{K_1 \to K_2} = K_2/K_1$ then the relative speedup is said to be linear,

Definition 2.6: Relative Efficiency

We define the relative efficiency of a parallel implementation, when going from K_1 to K_2 machines ($E_{K_1 \to K_2}$) for a fixed problem size, as:

$$E_{K_1 \to K_2} = \frac{K_1}{K_2} \times \frac{T_{parallel}(K_1)}{T_{parallel}(K_2)}$$

Definition 2.7: Relative Sizeup

We define the relative sizeup of a parallel implementation, when scaling the input data size, i.e., the *corpus* size, from a *corpus* C_1 (with size $|C_1|$) with K_1 machines to a *corpus* C_2 (with size $|C_2|$) with K_2 machines, as:

$$Szp_{C_1K_1 \to C_2K_2} = \frac{CorpusSize\,C_2(K_2, T)}{CorpusSize\,C_1(K_1, T)}$$

where $CorpusSize\,C(K, T)$ is the *corpus* size which can be processed in time T using K machines. When $\frac{|C_2|}{|C_1|} = \frac{K_2}{K_1}$, then the relative sizeup is said to be linear.

The relative sizeup can also be defined in relation to the problem size (N_n) instead of the *corpus* size.

2.3 Chapter Summary

Among the known extraction approaches the ones based on statistical methods are particularly promising due to their language neutrality and potential for application to enable the automatic processing of large scale data, whose benefits are widely recognized due to the large amount of information generated nowadays.

However, there is still a need for developing methods and tools supporting an efficient processing of statistical-based extraction of relevant expressions. In particular, in order to ensure acceptable processing time and scalability, it is necessary to enable an efficient exploitation of parallelism and distribution that takes advantage of the available cluster and cloud computing infrastructures.

In this thesis the computational characteristics of a statistical extraction method, LocalMaxs, were studied in its two main dimensions of execution time and memory, in a broad scale of *corpora* sizes. The thesis proposes solutions to allow the exploitation of parallel processing to achieve reductions in the execution time while supporting distributed data to deal the large scale *corpora* and problem sizes.

Part II

n-gram Statistical Distribution

Chapter 3

Modeling and Analysis of Statistical Distribution of *n*-grams in Natural Language *corpora*

The characterization of intrinsic properties of n-grams in natural language corpora can be useful to the implementation of statistical methods such as LocalMaxs.

This chapter presents a study on the intrinsic properties of *n*-grams in natural language *corpora*. The properties considered include, among others, the distribution of the number of distinct *n*-grams and are very useful in the context of statistical extraction methods, such as LocalMaxs. The study was conducted for two languages, English and French, but the conclusions taken are valid for other languages. This chapter is organized in 8 sections. Section 3.1 presents a brief introduction to the underlying concepts and their relevance to the current thesis. A theoretical model to estimate the number of distinct *n*-grams in natural language *corpora* is presented in section 3.2 and in section 3.3 the model is used to analyze the number of distinct *n*-grams and their frequencies of occurrences in a *corpus*. A theoretical model to estimate the number of *n*-grams with a given frequency is presented in section 3.4. In sections 3.5 and 3.6 the model is used to analyze the evolution of the singleton and nonsingleton *n*-grams with increasing *corpus* sizes. The FAset concept is presented in section 3.7. Section3.8 presents the chapter conclusions.

3.1 Concepts

The analysis of the statistical distribution of the *n*-grams in natural language *corpora* is very useful in the characterization of the intrinsic properties of the *corpora*. The number of distinct *n*-grams, the number of singletons and the *n*-grams repetition factors are some examples of intrinsic quantities whose study can be useful for statistical methods such

as LocalMaxs. Figure 3.1 illustrates the approaches taken in the context of this thesis to characterize the intrinsic properties of n-grams in natural language *corpora*.

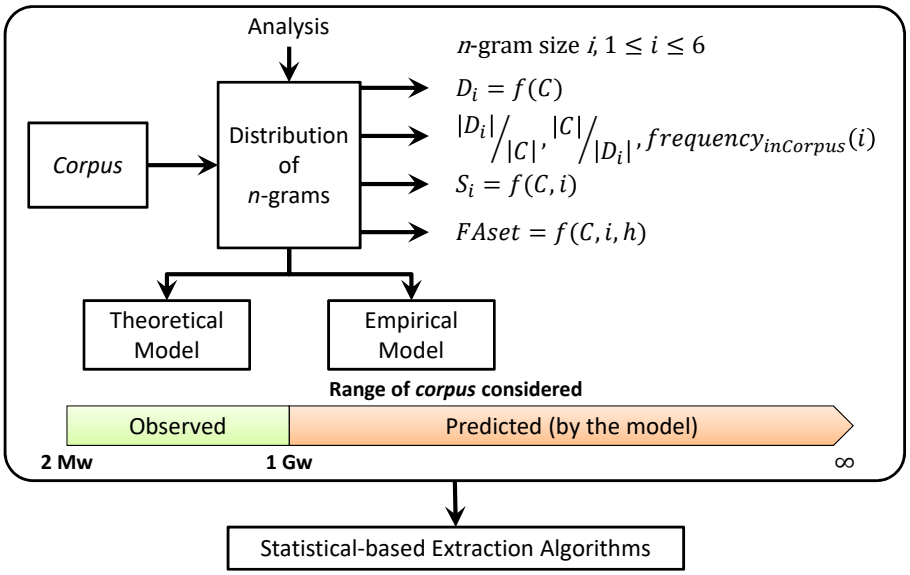


Figure 3.1: Dimensions of the study on the n-grams statistical properties of the natural language *corpora*.

Two approaches were followed in this study: i) An empirical approach, based on the actual counting of the number of distinct n-grams in the real *corpora*; ii) A theoretical model to capture the distribution of distinct n-grams and their frequencies of occurrences for *corpora* of a given language.

The distribution of the n-grams allowed us to characterize, for a given *corpus* C and n-grams of size i, the behavior of the following quantities: the total number of occurrences of n-grams of each size i in the *corpus* ($|Set_{All_i}|$); the total number of distinct n-grams of each size i ($|D_i|$); the repetition factor of n-grams of size i given by the ratio $|Set_{All_i}|/|D_i| \approx |C|/|D_i|$; and the evolution of the number ($|S_i|$) of singleton n-grams when the *corpus* size increases. This was then used to analyze the performance of the LocalMaxs method implementations for the entire *corpus* size range from 2 Mw up to infinity.

The empirical approach was applied to a range of *corpora* sizes from 2 Mw to 1 Gw, from Wikipedia [Wik16] with several topics, in the English and French languages. The theoretical model was validated for the English and French languages for *corpora* sizes ranging from 2 Mw to 1 Gw and was used to predict the number of distinct and singleton n-grams for the entire *corpus* size range.

For obtaining each one of the *corpus* used, random paragraphs were extracted from a *corpus* with 1 Gw words from Wikipedia articles in the English and French languages until the required sizes were approximately reached. Table 3.1 lists the main characteristics of the different *corpora* sets used in the context of this thesis.

Table 3.1: *Corpora* sets used.

***Corpora* set**	**Language**	**Sizes [Mw]**	***n*-gram size**
En_2-982/6	English	2, 4, 9, 18, 36, 73, 140, 245, 491, 982	6
Fr_2-970/6	French	2, 4, 8, 17, 35, 69, 139, 242, 484, 970	6
En_18-1023/4	English	18, 37, 75, 150, 300, 604, 1023	4
En_25-682/4	English	25, 227, 466, 682	4
En_1-689/2	English	1, 3, 6, 13, 26, 65, 129, 259, 689	2
En_1-259/3	English	1, 3, 6, 13, 26, 65, 129, 259	3
En_13-511/4	English	13, 25, 64, 128, 256, 511	4
En_50/7	English	50	7
En_256/6	English	252	6

The first column (*Corpora* set) identifies each *corpora* set using the following notation:

$$Lang\_C_1\text{–}C_2/n$$

where *Lang* represents the *corpus* language, e.g., *En* for English or *Fr* for French; C_1 and C_2 represent, respectively, the lowest and the highest *corpus* sizes in the set (in million words Mw); and n represents the maximum n-gram size considered ("n-gram size" column).

3.2 A Theoretical Model to Estimate the Number of Distinct *n*-grams

Many studies follow an empirical approach for determining the statistical distribution of the n-grams but are usually constrained by the *corpora* sizes, which for practical reasons stay far away from Big Data.

We propose a theoretical approach (first presented in [SGC16]) for estimating the number of distinct n-grams in each *corpus*. It is based on the Zipf-Mandelbrot Law and the Poisson distribution, and it allows an efficient estimation of the number of distinct unigrams, bigrams, ..., hexagrams, for any *corpus* size. The proposed model was validated for English and French *corpora*.

Words do not occur with similar probabilities in text. Everyday experience shows us that words such as "the", "and", or "in" are much more frequent than "agriculture" or "stomatology", whatever the topic of the text. This means words are more or less repeated throughout the text, so the number of distinct words in a *corpus* is less than the total number of words occurring in that *corpus*. This applies to single words or multiwords (sequences of n consecutive words where $n \geq 2$). Most of the empirical studies on the n-gram distribution only cover *corpora* of relatively small sizes. This precludes their usage towards understanding the behavior of an increasingly large number of Big Data applications that depend on the n-gram distributions. This requires an accurate estimation of the repetition patterns of the n-grams and their evolution for increasing *corpora* sizes.

The model proposed here also provides an efficient approach to estimate the number of distinct n-grams, including words and multiwords for each given *corpus* size.

3.2.1 Background

The frequency distribution of words in text has been studied in statistical linguistics [Car67; Man53; Zip35]. From Zipf's law [Zip35; Zip49] it can be stated that the frequency of any distinct single word (unigram) in a work of literature is inversely proportional to its rank in the frequency table. The frequency table lists all the existing distinct words sorted by their decreasing frequencies. The rank of the words is defined by their order number, where rank 1 corresponds to the most frequent word. Mandelbrot [Man53] proposed a generalization of this law for a better fitting of the frequency of some ranks, as it happens in some *corpora*. However, it is not sufficient to estimate the number of distinct unigrams.

Other improvements to Zipf's law were proposed in [Man09]. A critical review of the Zipf's word frequency law in natural language is made in [Pia14] claiming that semantics strongly influences word frequency. We think that is true, but it will not be the case for very Big Data *corpora*, where the existence of numerous topics tends not to favor any particular topic. Herdan-Heaps' law [Egg07] is an empirical law describing the number of distinct single words in documents as a function of the document length. It states that $V_R(n) = K \times n^{\beta}$ where V_R is the number of distinct words in the text of size n, and K and β are parameters determined empirically. According to [BYN00; Kor99; LW05], under mild assumptions, this law is asymptotically equivalent to Zipf's law concerning the frequencies of individual words.

In [LK06; LK10], although their aim is not to propose an approach to estimate the number of distinct n-grams, some estimate of cardinality of their sets is presented for hashing design. However, these estimates incur some computational weight, depending on the data volume. A clustering model for words distribution is proposed in [TZ92] based on estimating a probability (p) for each word occurring in a document of a given length. Then, the number of distinct words can be estimated but the model is not very accurate for large p values, and it is not always a close fit to observed data. There is no evidence that this approach could be extended to larger n-gram sizes: bigrams, trigrams, and so on.

To the best of our knowledge, we are not aware of other approach focused on the estimation of the number of distinct multiword n-grams. In this thesis a new approach, described in the following, is proposed to estimate the cardinality of the set of distinct n-grams (single or multiwords) in languages whose phrases are composed of alphanumeric character tokens separated by punctuation symbols and spaces, and which have a phonetic reproduction, such as English, French and other known Occidental languages.

3.2.2 The Number of Distinct *n*-grams in *Corpora*

In section 3.2.2.1 we discuss the Zipfian models. The estimation of the number of distinct single words is presented in section 3.2.2.2 and the estimation of the number of distinct multiwords is presented in section 3.2.2.3. An efficient implementation of the method that allows the estimation of the number of distinct n-grams is presented in section 3.2.2.4.

3.2.2.1 The Zipfian Models

The Zipf's law [Zip35] states that the frequency of the r^{th} most frequent word in natural language *corpora*, $f(r)$, scales according to:

$$f(r,\alpha) \propto \frac{1}{r^{\alpha}} \tag{3.1}$$

where r is the frequency rank of a word, and $\alpha \approx 1$. The most frequent word corresponds to $r = 1$; for the i^{th} most frequent word ($r = i$), its frequency $f(i)$ is proportional to $1/i^{\alpha}$. Though Zipf's law works as a good model for some *corpora*, it presents some deviations with respect to actual real *corpora* mainly for low and high ranks. To minimize these deviations, Mandelbrot [Man53] proposed a generalization of this law by *shifting* rank r by a value β:

$$f(r,\alpha,\beta) \propto \frac{1}{(r+\beta)^{\alpha}} \tag{3.2}$$

Then we state a corresponding equation for the frequency of rank r, denoted by $f(r,(\alpha,\beta))$ for $r = 1, 2, \ldots$, keeping the proportionality as in the Mandelbrot model (3.2):

$$f(r,(\alpha,\beta)) = \left(\frac{1+\beta}{r+\beta}\right)^{\alpha} \times f(1) \tag{3.3}$$

where $f(1)$ means the frequency of the most frequently occurring word in a *corpus*. The particular case of $\beta = 0$ in expression (3.3) corresponds to the Zipf's law model.

By using expression (3.3), (α,β) combinations can be searched for each *corpus* so that the estimate of the frequency of each rank is as close as possible to the actual frequency found in the real *corpus*. As shown in section 3.2.3 and resulting from our experiments, we consider that for each language and each n-gram size, there is a *best* (α,β) *combination*, which produces the most possible accurate results when estimating the number of distinct n-grams, with similar accuracies for different *corpus* sizes.

3.2.2.2 Estimating the Number of Distinct Single Words in *Corpora*

Having analyzed the relative frequency of the most common word in English, that is "the", corresponding to $r = 1$, — it can be taken as the probability of rank 1 in the *corpus*, denoted by p_1 — we noticed that it tends to be constant when *corpora* sizes grow. In fact, this probability presents very slight variations for small and median size *corpora*, but

converges to a value keeping more and more decimal digits unchanged for large *corpora*; for example, in the case of English *corpora* over 500 million words, the first 7 decimal digits showed to be the same for p_1, which was 0.0503705.

In order to obtain a more correct counting for the actual number of occurrences of each word, words should be separated from some punctuation marks by using spaces. This does not affect the semantic of texts. Then, the following set of punctuation mark characters were considered as separation targets for all *corpora* in this work: {'<', '>', '"', '!', '?', ':', ';', ' . ', ', ', '(', ')', '[', ']'}. This criterion was used in all *corpora* analyzed in this thesis, both for the empirical study as well as for validating the theoretical model. If other criteria are used for the content of this set, p_1 can tend to a different constant value.

For other ranks of very frequent English words such as "of", "and", "in", among others, it was easily verified that their individual probabilities also tend to constant values. We noticed this tendency still exists for other not so frequent words such as "late" and "again", although larger *corpora* were needed to reach their corresponding constant probabilities. Thus, there is no reason not to believe that this tendency is applicable to all words, even for rare ones, which will certainly be verified in huge *corpora*. For the other language considered in this work (French), the same behavior was verified. This leads us to the belief that as *corpora* sizes grow for the same language, words tend to have fixed ranks, which is consistent with the existence of what we called the *best* (α, β) *combination* for each language.

Thus, assuming that the probability of rank 1 in a *corpus* C (where, in this chapter, its size in words is denoted by c, which is equivalent to $|C|$) tends to remain constant for large *corpora*, then its expected frequency is $f(1) = p_1 \times c$. Also, the frequency of rank r can be estimated by expression (3.3), for a given (α, β) combination, such that $f(r,(\alpha,\beta)) = ((1+\beta)^{\alpha}/(r+\beta)^{\alpha}) \times f(1)$. So, the expected frequency of rank r in a *corpus* C, having c words, for a (α, β) *combination*, is:

$$f(r,(\alpha,\beta),c) = \left(\frac{1+\beta}{r+\beta}\right)^{\alpha} \times (p_1 \times c) \tag{3.4}$$

The values of the actual observed frequencies of the n-grams in a *corpus* are non negative integers but for two consecutive low r values there may be two non consecutive values of frequencies. Expression (3.4) models an approximation to this behavior and, by its structure, it can return non integer values.

Once there is a *best* (α, β) *combination* for each language l and that *best combination* must be used to obtain the frequency of rank r in a *corpus* C, in a language l, then α and β depend on l. Similarly, the probability of rank 1 depends on the language l too:

$$f(r,l,c) = \left(\frac{1+\beta(l)}{r+\beta(l)}\right)^{\alpha(l)} \times (p_1(l) \times c) \tag{3.5}$$

Considering Poisson Distribution. A random variable X follows a Poisson distribution [Hai67] with parameter $\lambda > 0$, if, for $k = 0, 1, 2, \ldots$, the probability mass function is, according to the classical notation:

$$f(k;\lambda) = Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{3.6}$$

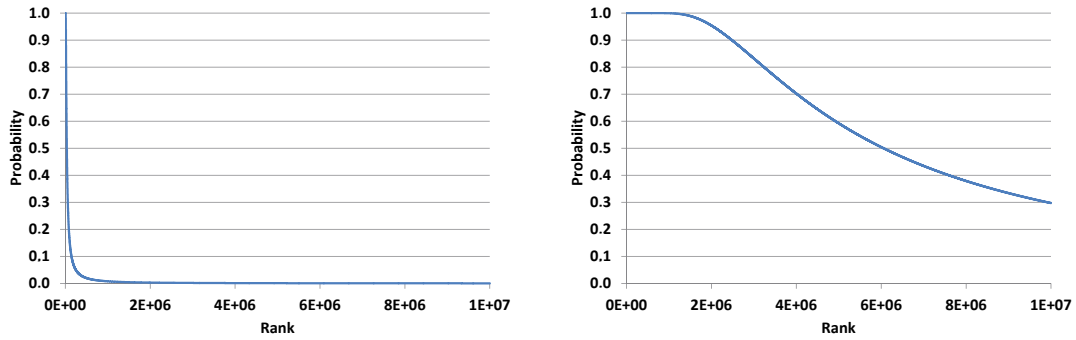
where e is the Euler's constant ($e = 2.71828\ldots$) and $k!$ is the factorial of k. λ is a positive real number equal to the expected value of X. This distribution provides a realistic model for many random phenomena for which a count of some sort is of interest, such as the number of traffic accidents per week, given its rate λ. So, let X be the number of times word w in rank r occurs in a *corpus* C, written in language l. Then, the probability of non-occurrence of w is:

$$Pr(X = 0) = \frac{\lambda^0 e^{-\lambda}}{0!} = e^{-\lambda} = e^{-f(r,l,c)} \tag{3.7}$$

where $f(r,l,c)$ stands for the expected frequency of rank r, given by expression (3.5). Thus, the probability of w occurring in the same *corpus* is:

$$Pr(X \geq 1) = 1 - e^{-f(r,l,c)} \tag{3.8}$$

Considering for example, the size of the smallest English *corpus* used in the derivation of this model (c = 2,226,162 words; l = "English") and using the *best* (α, β) *combination* searched for English unigrams ($\alpha = 1.3466$; $\beta = 7.7950$) — explained in section 3.2.3 — by application of expression (3.8), the probabilities of occurrence of the words corresponding, for example, to ranks 1; 100,000; and 3,000,000 are, respectively, $Pr(X \geq 1) = 1.0$; $Pr(X \geq 1) = 0.32120$; and $Pr(X \geq 1) = 0.00396$. For a *corpus* 10 times larger, those probabilities are, respectively, $Pr(X \geq 1) = 1.0$; $Pr(X \geq 1) = 0.97923$; and $Pr(X \geq 1) = 0.03895$. This matches our intuition, as we expect that the probability of occurrence of frequent words is high, even in small *corpora*; and the probability of rare words (higher ranks, such as 3,000,000), though low, grows with the *corpus* size. So, by expression (3.8) it is possible to calculate the probability of any word (its rank) in the vocabulary of a language ($v(l)$). This is depicted in Figure 3.2 showing the probabilities, as obtained by expression (3.8), of the first interval of unigram ranks, namely from rank 1 up to rank 1×10^7, for two *corpus* sizes: one with 1 Mw (Figure 3.2-a)) and another with 1 Gw (Figure 3.2-b)).



a) *Corpus size 1 Mw* ***b)*** *Corpus size 1 Gw*

Figure 3.2: Probabilities of the ranks from 1 up to 1×10^7 rank for unigrams. The Y axis represents the probability and the X axis represents the rank.

Now, in order to calculate $Dist(l,c)$ the number of distinct words in a *corpus* C, with c words, in a language l, we propose to sum the probabilities of occurrence of all words in the language vocabulary for that *corpus*, considering the entire vocabulary of l:

$$\begin{gathered} Dist(l,c) = \sum_{r=1}^{r=|\nu(l)|} \left(1 - e^{-f(r,l,c)}\right) \\ Dist(l,c) = |\nu(l)| - \sum_{r=1}^{r=|\nu(l)|} e^{-\left(\left(\frac{1+\beta(l)}{r+\beta(l)}\right)^{\alpha(l)} \times (p_1(l) \times c)\right)} \end{gathered} \tag{3.9}$$

where $|\nu(l)|$ is the size of $\nu(l)$, being the number of distinct words in the vocabulary of language l, which can reach some million words for languages such as English or French, as we noticed for large *corpora*. This sum of probabilities must not be taken as a probability. Indeed, the sum of probabilities can be used to estimate some population sizes. For example, the number of heads from tossing a fair coin 1000 times can be correctly estimated by summing the probability of a head in a single toss (0.5), 1000 times, i.e., $\sum_{t=1}^{t=1000} 0.5 = 500$.

3.2.2.3 Estimating the Number of Distinct Multiwords in *Corpora*

We noticed that the frequency of bigrams in *corpora* also follows a Zipfian distribution. For English *corpora*, the most common (i.e., $r = 1$) bigram, "of the", also tends to have a relative frequency that converges to a fixed value when *corpora* grow. The same happens to ranks $2, 3, \ldots$. So, the estimate of frequency given by expression (3.4) can also be applied to bigrams, however, there is a different *best* (α, β) *combination* for bigrams for each of the considered languages. Likewise for larger size n-grams: trigrams, tetragrams,.... Thus, due to their dependence on the language l and on the n-gram size n, α and β are denoted by $\alpha(l,n)$ and $\beta(l,n)$. Similarly, the probability of rank 1 also depends on the n-gram size and on each specific language, being denoted by $p_1(l,n)$. On the other hand, for the same language, the number of distinct single words is different from the number of distinct bigrams, trigrams,.... This means that the size of the vocabulary depends not only on the language, but also on the n-gram size; we use $|\nu(l,n)|$ to denote this number.

Therefore, expression (3.9) can be generalized also to any n-gram size n:

$$Dist(l,n,c) = |v(l,n)| - \sum_{r=1}^{r=|v(l,n)|} e^{-\left(\left(\frac{1+\beta(l,n)}{r+\beta(l,n)}\right)^{\alpha(l,n)} \times (p_1(l,n)\times c)\right)} \tag{3.10}$$

3.2.2.4 An Efficient Implementation

Using expression (3.10) we can estimate the number of distinct n-grams of each size n in a language for each *corpus* size. However, this may become computationally heavy; e.g., for the case of hexagrams ($n = 6$) for any English *corpus*, due to the large vocabulary size, the sum in expression (3.10) may lead to an order of magnitude of 10^{11} iterations.

So, we propose a more efficient implementation of expression (3.10), where the heavily iterated sums are replaced with an integral, leading to expression (3.18), whose detailed derivation is given in the following paragraphs. The practical efficiency of this alternative implementation was assessed in a single machine, where Python 2.5.1 based implementations of the two methods (3.10) and (3.18) were compared, using the GNU Scientific Library 2.2 implementation of the Incomplete Gamma Function, in a laptop with Mac OS X 10.5.8, 1 Central Processing Unit (CPU) @ 2.4 GHz Intel, and 4 Gigabyte (10^9) (GB) 667 MHz DDR2 SDRAM. When expression (3.10) was used to estimate the number of distinct unigrams in a 100 Mw English *corpus*, it took 178.73 minutes to complete. The same estimate took 0.0078408 seconds to complete when using expression (3.18). Similar gains were obtained for larger n-gram sizes ($n \geq 2$). The vocabulary size and the α and β values used in these tests result from a tuning phase as presented in section 3.2.3.

Derivation of the integral based formula (3.18). According to the Euler-Maclaurin formula [Abr72], the finite sum $\sum_{n=a}^{n=b} g(n)$ can be substituted by an integral as follows:

$$\sum_{n=a}^{n=b} g(n) \approx \int_a^b g(x)\,dx + B \tag{3.11}$$

where $B = \frac{g(b)+g(a)}{2} + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!}\left(g^{(2k-1)}(b) - g^{(2k-1)}(a)\right)$ and $g^{(2k-1)}(b)$ stands for the $(2k-1)^{th}$ derivative of $g(.)$ in b. And B_m, a Bernoulli number, is given by:

$$B_m \approx \sum_{v=0}^{v=m} \sum_{j=0}^{j=v} (-1)^j \binom{v}{j} \frac{j^m}{v+1} \tag{3.12}$$

So, by expression (3.12), $B_2 = 1/6$, $B_4 = -1/30$, etc..

On the other hand, variables l, n, c, $v(l,n)$, $\alpha(l,n)$ and $\beta(l,n)$ in expression (3.10), can be taken as constants in the context of each specific $Dist(l,n,c)$ calculation, that is, in the context of the estimation of the number of distinct n-grams of size n of a *corpus* C (with size c) in a specific language l. Thus, expression (3.10) can be simplified for better mathematical manipulation, as follows:

$$A = \alpha(l,n) \quad R = r + \beta(l,n)$$

$$Q = (1+\beta(l,n))^{\alpha(l,n)} \times p_1(l,n) \times c \tag{3.13}$$

$$r_1^\star = 1 + \beta(l,n) \quad r_v^\star = |\nu(l,n)| + \beta(l,n)$$

Thus

$$Dist(l,n,c) = |\nu(l,n)| - \sum_{R=r_1^\star}^{R=r_v^\star} e^{-Q\times R^{-A}} \tag{3.14}$$

Then, by applying the Euler-Maclaurin formula, given by expression (3.11), taking R as the integration variable,

$$Dist(l,n,c) = |\nu(l,n)| - \sum_{R=r_1^\star}^{R=r_v^\star} e^{-Q\times R^{-A}} = |\nu(l,n)| - \left(\int_{r_1^\star}^{r_v^\star} e^{-Q\times R^{-A}} dR + B \right) \tag{3.15}$$

where

$$B = \frac{g\left(r_1^\star\right) + g\left(r_v^\star\right)}{2} + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!} \left(g^{(2k-1)}\left(r_v^\star\right) - g^{(2k-1)}\left(r_1^\star\right) \right) \tag{3.16}$$

$g(R) = e^{-Q\times R^{-A}}$ and B_{2k} is given by expression (3.12). Our experiments showed us that it made a negligible difference to the result of $Dist(l,n,c)$ in expression (3.15), including or not the derivatives of $g(.)$ in the sum of expression (3.16). So, for simplicity, their respective equations are ignored in the following, where B is simplified to:

$$B \approx \frac{g\left(r_1^\star\right) + g\left(r_v^\star\right)}{2} \tag{3.17}$$

So considering the following expression:

$$Dist(l,n,c) - |\nu(l,n)| + B = -\int_{r_1^\star}^{r_v^\star} e^{-Q\times R^{-A}} dR$$

by using an integral calculator site[1] we obtain:

$$Dist(l,n,c) - |\nu(l,n)| + B = -\frac{Q^{\frac{1}{A}}}{A} \left[\gamma\left(-\frac{1}{A}, QR^{-A}\right) + Const \right]_{R=r_1^\star}^{R=r_v^\star}$$

where $Const$ denotes the integration constant. Then:

$$Dist(l,n,c) = E - \frac{Q^{\frac{1}{A}}}{A} \left[\gamma\left(-\frac{1}{A}, Q\times\left(r_v^\star\right)^{-A}\right) - \gamma\left(-\frac{1}{A}, Q\times\left(r_1^\star\right)^{-A}\right) \right] \tag{3.18}$$

[1] For example the one available in https://www.integral-calculator.com/

where $Dist(l,n,c)$ is the number of distinct n-grams; $E = |\nu(l,n)| - B$; $\gamma(.,.)$ is the Lower Incomplete Gamma function; the substitutions for symbols Q, A, $r_v^\star$ and $r_1^\star$ are in expression (3.13); $g(R) = e^{-Q \times R^{-A}}$; and B is given by expression (3.17).

3.2.3 Instantiation of the Model Parameters: $\alpha(l,n)$, $\beta(l,n)$, $|\nu(l,n)|$ and $p_1(l,n)$

To achieve a confidence degree in the determination of the model parameters suitable to provide a low level of relative errors when applying the model to existing available *corpus*, we assessed the accuracy of the model estimates in a wide range of *corpus* sizes. For each language, *corpora* were generated by doubling approximately the size of each *corpus*, from about 2×10^6 to 10^9 words: 2×10^6, 4×10^6, 8×10^6,... . For obtaining each of these specific *corpora* sizes, random paragraphs were extracted from the largest *corpus* (10^9 words) in each language until the required size is approximately reached. The proposed model was instantiated with English and French Wikipedia based *corpora* [Wik16]. Tables 3.2 and 3.3 show, for each *corpus*, its size and the number of actual observed distinct n-grams ($|D_i|$), from unigrams to hexagrams.

Table 3.2: Number of distinct n-grams of size $1 \leq i \leq 6$ (in millions of n-grams) *versus* *corpus* size for the English language.

***Corpus* size** [**Mw**]	$\|D_1\|$	$\|D_2\|$	$\|D_3\|$	$\|D_4\|$	$\|D_5\|$	$\|D_6\|$
2.226	0.171	0.918	1.682	2.032	2.146	2.189
4.450	0.275	1.604	3.169	3.971	4.254	4.358
8.955	0.447	2.797	5.961	7.776	8.466	8.722
18.007	0.729	4.834	11.104	15.129	16.790	17.422
35.772	1.187	8.208	20.258	28.896	32.773	34.304
72.678	1.966	14.086	37.404	56.035	65.160	68.935
140.276	3.155	23.084	65.483	102.858	122.712	131.339
245.492	4.718	34.961	104.707	171.430	209.427	226.713
490.847	7.784	57.968	185.347	319.964	403.573	444.168
981.996	12.814	94.705	323.192	589.842	770.074	863.071

In order to use expression (3.18) to obtain the best possible estimate of the number of distinct n-grams for a given *corpus*, language and n-gram size, the values $\alpha(l,n)$, $\beta(l,n)$, $|\nu(l,n)|$, and $p_1(l,n)$ must be determined, corresponding to: the *best* (α,β) *combination*, the vocabulary size ($|\nu(l,n)|$), and the probability of the n-gram with rank 1 ($p_1(l,n)$).

The above values were found by applying the following methodology. Concerning $p_1(l,n)$, for each language and n-gram size, the value of $p_1(l,n)$ was calculated as the relative frequency of the first ranked n-gram, for the above *corpus* size ranges. Concerning $|\nu(l,n)|$, although actual vocabularies are open, as new words and multiwords arise and others tend to stop being used, they are finite. However, the lack of consensus about the real vocabulary sizes prevents us from assessing how far the $|\nu(l,n)|$ values are from the actual sizes. According to the considered criterion, the vocabulary size parameter $|\nu(l,n)|$

Table 3.3: Number of distinct n-grams of size $1 \leq i \leq 6$ (in millions of n-grams) *versus* *corpus* size for the French language.

***Corpus* size** [**Mw**]	$\|D_1\|$	$\|D_2\|$	$\|D_3\|$	$\|D_4\|$	$\|D_5\|$	$\|D_6\|$
2.172	0.157	0.767	1.510	1.904	2.050	2.108
4.322	0.248	1.302	2.770	3.653	4.008	4.151
8.710	0.394	2.209	5.088	7.042	7.899	8.256
17.443	0.628	3.714	9.205	13.383	15.385	16.253
34.745	0.996	6.161	16.421	25.131	29.685	31.749
69.331	1.583	10.163	29.033	46.815	56.957	61.783
139.025	2.518	16.682	50.978	86.678	108.859	119.960
242.346	3.655	24.663	79.264	140.575	181.301	202.674
484.315	5.779	39.560	135.285	253.300	338.324	385.397
970.351	9.254	63.155	228.935	451.752	625.663	726.464

is the minimum value such as the best pair (α, β) ensures the lowest relative error in the model estimates.

The best pair (α, β) is found by a tuning process such that, for each $(|\nu(l,n)|, \alpha(l,n), \beta(l,n), p_1(l,n))$ combination, two estimates are obtained using expression (3.18). Considering an ordering of the *corpus* sizes from the smallest to the largest *corpus*, two points are selected close to both extremes in the *corpus* sizes range: one for the second smallest and the other for the second largest *corpus* size. Then, if these two estimates do not deviate more than 5% from the actual number of distinct n-grams of the real *corpus*, the search for the best (α, β) pair stops, as we consider that the actual size of the corresponding vocabulary can be approximated by $|\nu(l,n)|$ and the *best* (α, β) *combination* has been found. Thus, assuming that the minimum size of the vocabularies for the two considered languages is at least 2×10^7 n-grams, an exhaustive search is made for each pair (α, β) by varying $|\nu(l,n)|$ from 2×10^7 up to a maximum of 4×10^{11} n-grams, by steps of 1×10^6 n-grams or larger; then, for each $|\nu(l,n)|$ value, different (α, β) combinations were assessed by varying α from 0.5 to 1.8 and β from -0.5 to 80, by steps of 0.005 and 0.0001 respectively.

Tables 3.4 and 3.5 show the *best* (α, β) *combination* and vocabulary size for each triple (language, n-gram size, p_1 values), resulting from the above methodology.

Table 3.4: Best α, β, $|\nu|$ (in number of n-grams) and p_1 for the English *corpora*.

parameter	**unigrams**	**bigrams**	**trigrams**	**tetragrams**	**pentagrams**	**hexagrams**
$\boldsymbol{\alpha}$	1.3466	1.1873	0.9800	0.8252	0.8000	0.8000
$\boldsymbol{\beta}$	7.7950	48.1500	21.8550	0.4200	-0.4400	0.6150
$\boldsymbol{\|\nu\|}$	1.95×10^8	7.08×10^8	3.54×10^9	9.80×10^9	5.06×10^{10}	3.92×10^{11}
$\boldsymbol{p_1}$	0.05037	0.00827	0.00239	0.00238	0.00238	0.00067

Table 3.5: Best α, β, $|\nu|$ (in number of n-grams) and p_1 for the French *corpora*.

parameter	unigrams	bigrams	trigrams	tetragrams	pentagrams	hexagrams
α	1.4496	1.2653	1.0444	0.8843	0.7835	0.8602
β	16.4850	73.8550	71.3800	7.3500	−0.0700	2.4400
$\lvert\nu\rvert$	1.60×10^{8}	4.25×10^{8}	1.52×10^{9}	4.19×10^{9}	7.98×10^{9}	8.50×10^{10}
p_1	0.05338	0.00978	0.00165	0.00165	0.00165	0.00165

3.2.4 Validation of Model

Once the model parameters have been found, they were used in expression (3.18) to estimate the number of distinct n-grams for all *corpora* of Tables 3.2 and 3.2. The relative errors of these estimates are shown in Tables 3.6 and 3.7.

Table 3.6: Relative errors of the estimated number of distinct n-grams for each English *corpus*. Values are in percentage (%).

***Corpus* size** [Mw]	unigrams	bigrams	trigrams	tetragrams	pentagrams	hexagrams
2.226	-2.9	-0.5	0.0	-0.1	0.5	-0.5
4.450	-0.3	-0.9	-0.7	-1.1	0.6	4.7
8.955	2.2	-0.9	-0.8	-0.8	0.6	-3.0
18.007	3.7	-0.5	-0.5	-0.5	1.0	-4.0
35.772	4.2	0.0	0.0	0.0	0.9	0.9
72.678	4.3	0.5	0.3	0.3	0.6	0.7
140.276	3.4	0.8	0.6	0.6	0.5	1.0
245.492	2.2	0.8	0.6	0.7	0.3	-0.3
490.847	-0.1	0.3	0.4	0.3	-0.1	-0.5
981.996	2.9	-0.5	-0.3	-0.5	0.4	-0.5

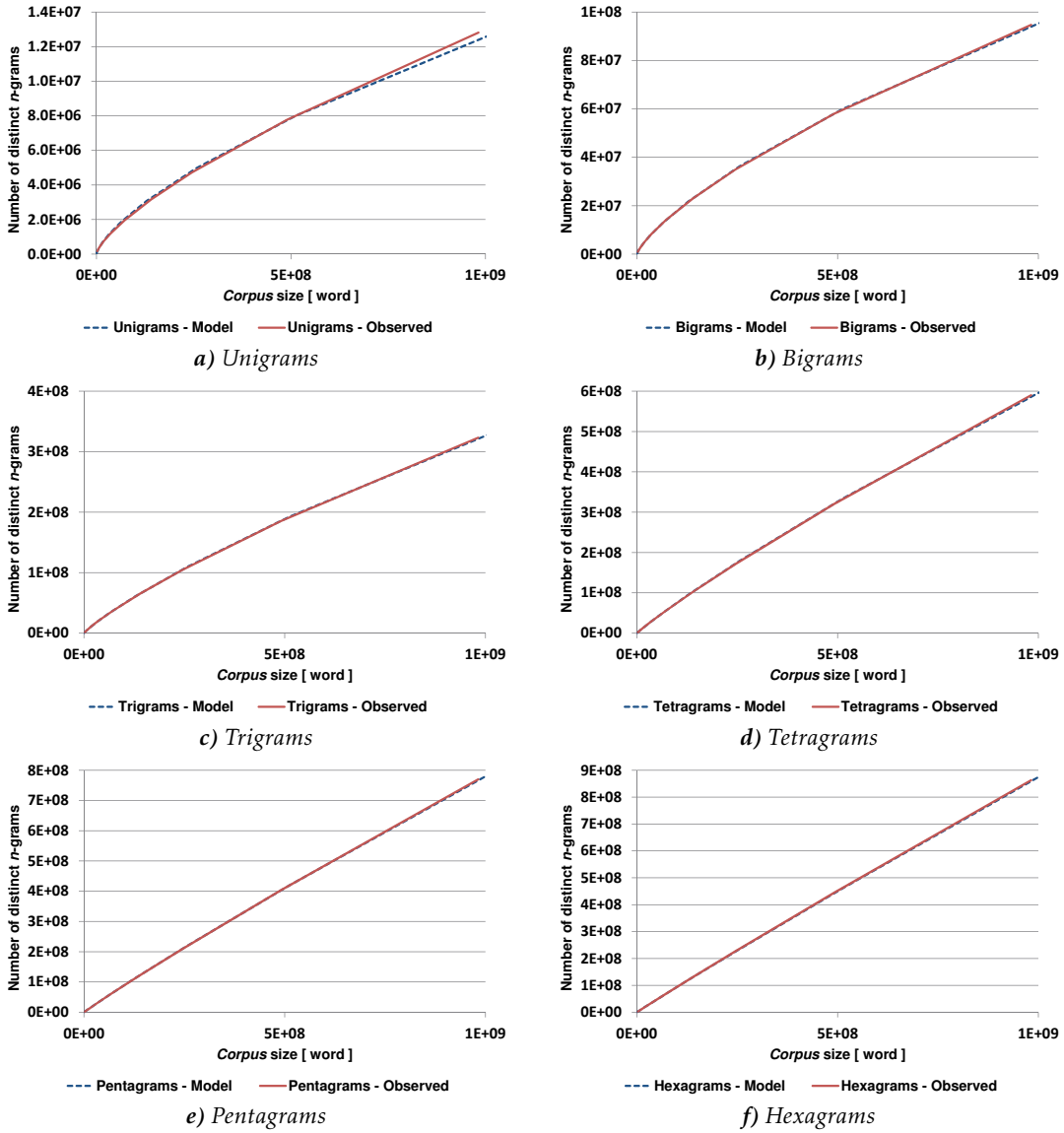
Table 3.7: Relative errors of the estimated number of distinct n-grams for each French *corpus*. Values are in percentage (%).

***Corpus* size** [Mw]	unigrams	bigrams	trigrams	tetragrams	pentagrams	hexagrams
2.172	-1.8	-0.6	0.2	-0.5	-0.2	0.0
4.322	-0.7	-1.0	-0.3	-0.5	-1.0	2.7
8.710	0.5	-0.9	-0.4	-0.6	-0.6	0.4
17.443	1.1	-0.8	-0.2	-0.2	-0.3	1.1
34.745	1.6	-0.3	0.1	0.1	0.1	0.1
69.331	1.7	0.2	0.5	0.5	0.5	-0.4
139.025	1.6	0.8	0.8	0.8	0.8	-0.6
242.346	1.1	1.0	0.9	0.9	0.8	-0.7
484.315	0.6	1.5	0.8	0.8	0.7	-0.4
970.351	-1.7	0.9	-0.5	-0.4	-0.3	0.2

This relative error, in Tables 3.6 and 3.7, is given by $(Es/Act-1)\times100\%$, where Es and Act stand for the estimate and the corresponding actual number in the *corpus*, respectively.

The results show that the relative error is generally less than 1% for estimates of bigrams up to pentagrams for the full range of *corpora* sizes in both languages. However, errors are slightly higher for some of the *corpora* sizes, reaching a maximum of 4.3% for unigrams, and 4.7% for hexagrams.

Figure 3.3 illustrates the evolution, along the *corpus* size range, of the actual numbers of distinct unigrams up to hexagrams, and their respective estimates obtained by this approach for the English *corpora*.



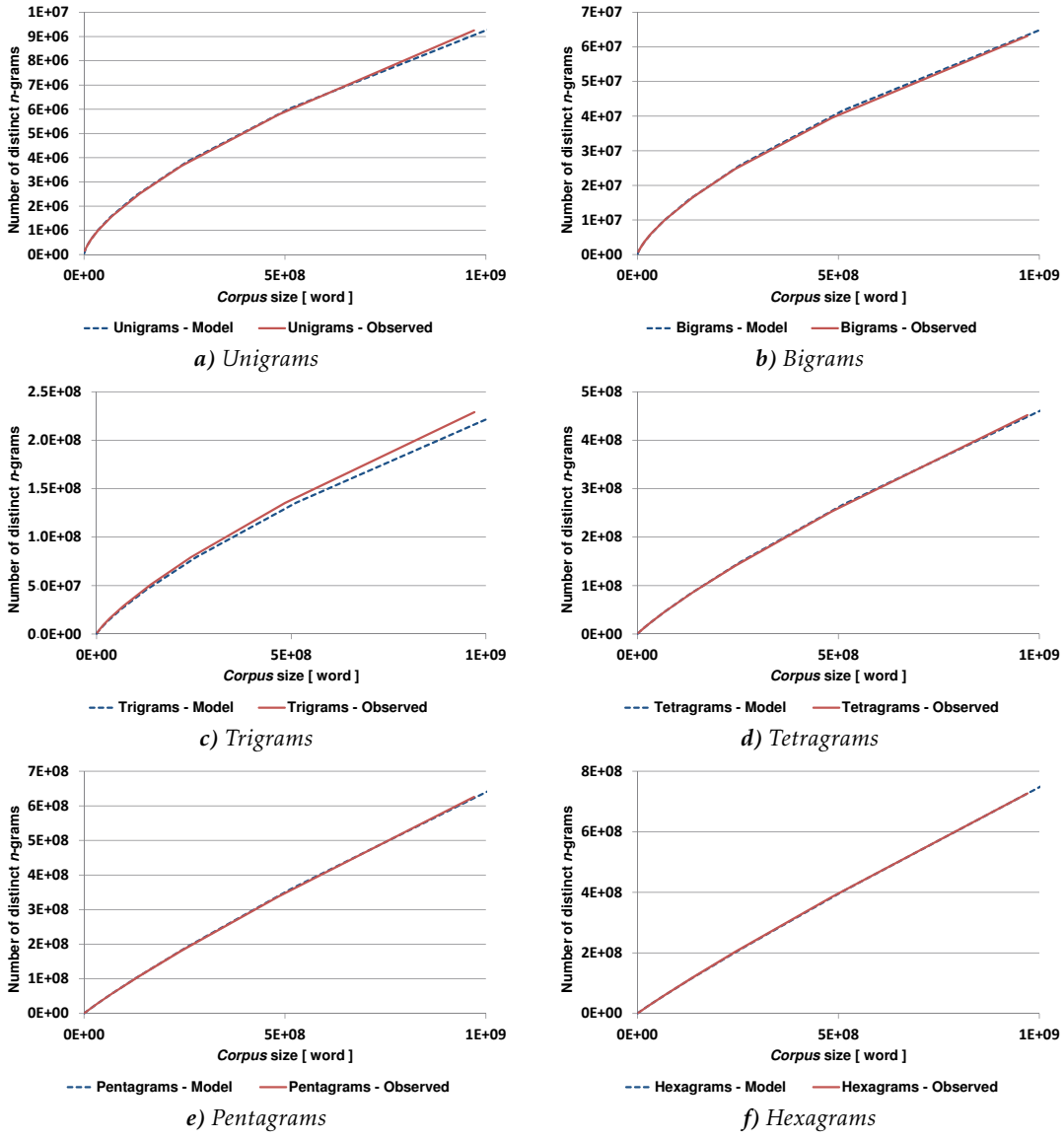
a) Unigrams *b) Bigrams* *c) Trigrams* *d) Tetragrams* *e) Pentagrams* *f) Hexagrams*

Figure 3.3: Number of distinct *n*-grams for English language — model estimation *versus* observed. The Y axis represents the number of distinct *n*-grams and the X axis represents the *corpus* size in words.

It shows a small deviation between the curves of unigrams, however not more than 4.3% as we know from Table 3.6. For bigrams, curves coincide apparently, since they deviate less than 1% for all *corpus* sizes. Table 3.6 allows us to preview similar coincidence

for trigrams, tetragrams, pentagrams and hexagrams as shown in Figure 3.3.

Similar curves were obtained for French according to Table 3.3 and Table 3.7. Figure 3.4 shows the evolution, with the *corpus* size, of the actual numbers of distinct unigrams up to hexagrams, and their respective estimates obtained by this approach for the French *corpora*.



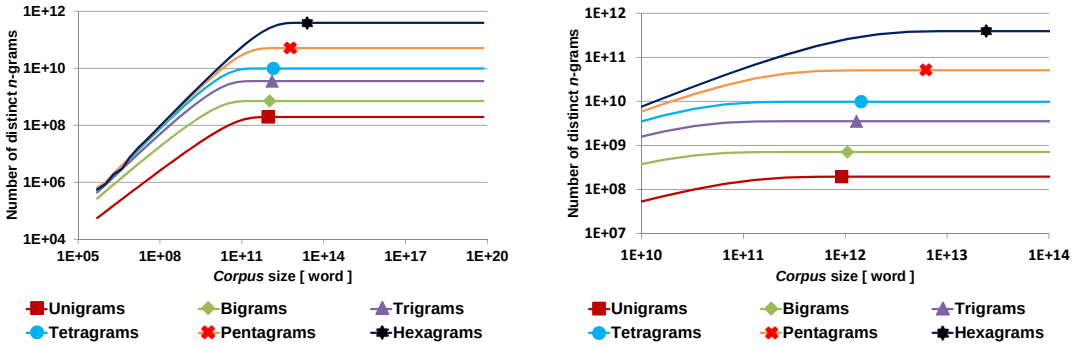
a) Unigrams *b) Bigrams* *c) Trigrams* *d) Tetragrams* *e) Pentagrams* *f) Hexagrams*

Figure 3.4: Number of distinct n-grams for French language — model estimation *versus* observed. The Y axis represents the number of distinct n-grams and the X axis represents the *corpus* size in words.

3.2.5 Applying the Model to Large *Corpus* Sizes

Once the model was validated, it can be applied to larger *corpus* sizes beyond the above considered ranges. A closer analysis of expression (3.10) suggests that, for each n-gram size, as the *corpus* size increases towards infinity, the sum in the second parcel tends to

zero, meaning that there are no more distinct n-grams in the vocabulary remaining to appear, so the number of distinct n-grams appearing in the *corpus* tends to the size of the vocabulary. Due to this, the evolution of the number of distinct n-grams for each size, as a function of the *corpus* size, exhibits a *plateau* which corresponds to the respective vocabulary size. This is illustrated in Figure 3.5 from unigrams up to hexagrams.



a) *Global view* ***b)*** *Knee detail*

Figure 3.5: Number of distinct n-grams estimated by the model *versus corpus* size, for the English language. The Y axis represents the number of distinct n-grams and the X axis represents the *corpus* size in words.

In Figure 3.5-a) the *corpus* size ranges from 1×10^6 to 1×10^{20} words. Figure 3.5-b) is a detail of image Figure 3.5-a) (knee zone) in the *corpus* range from 1×10^{10} to 1×10^{14} words.

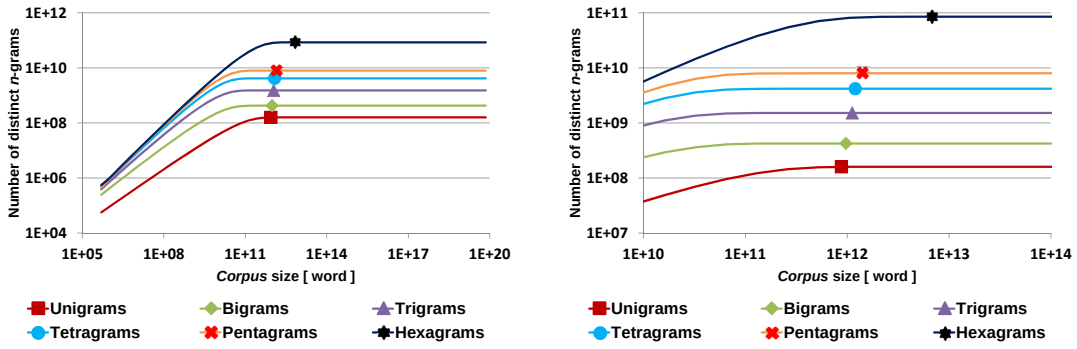
Table 3.8 shows all obtained *plateau* values estimated by the model for the different n-gram sizes and the corresponding *corpus* size thresholds from which these *plateau* values are reached in the case of the English language. The quantities are also expressed in terms of Giga n-grams (Giga n-gram (10^9) (Gn–*gram*)) and Tera n-grams (Tera n-gram (10^{12}) (Tn–*gram*)), or Tera words (Teraword (10^{12}) (Tw)).

Table 3.8: *Plateau* values for distinct English n-grams ($|D_i|$) and corresponding *corpus* size thresholds ($|C|$).

	unigrams	**bigrams**	**trigrams**	**tetragrams**	**pentagrams**	**hexagrams**
$\|\mathbf{D}_i\|$	1.95×10^8	7.08×10^8	3.54×10^9	9.80×10^9	5.06×10^{10}	3.92×10^{11}
	(0.2 *Gn-gram*)	(0.7 *Gn-gram*)	(3.5 *Gn-gram*)	(9.8 *Gn-gram*)	(50.6 *Gn-gram*)	(0.4 *Tn-gram*)
$\|C\|$	9.22×10^{11}	1.05×10^{12}	1.29×10^{12}	1.43×10^{12}	6.18×10^{12}	2.39×10^{13}
	(0.9 *Tw*)	(1.1 *Tw*)	(1.3 *Tw*)	(1.4 *Tw*)	(6.2 *Tw*)	(23.9 *Tw*)

For example, the curve in Figure 3.5-a) shows that for English *corpus* sizes larger than a *threshold* of 9.22×10^{11} words, the estimated number of distinct unigrams will not grow further, having reached the unigrams vocabulary size, that is 1.95×10^8 unigrams.

The values for the French language are shown in Figure 3.6 and Table 3.9.



a) *Global view* ***b)*** *Knee detail*

Figure 3.6: Number of distinct n-grams *versus corpus* estimated by the model for the French language. The Y axis represents the number of distinct n-grams and the X axis represents the *corpus* size in words.

Table 3.9: *Plateau* values for distinct French n-grams ($|D_i|$) and corresponding *corpus* size thresholds ($|C|$).

	unigrams	**bigrams**	**trigrams**	**tetragrams**	**pentagrams**	**hexagrams**
$\|D_i\|$	1.60×10^{8} (0.2 *Gn-gram*)	4.25×10^{8} (0.4 *Gn-gram*)	1.52×10^{9} (1.5 *Gn-gram*)	4.19×10^{9} (4.2 *Gn-gram*)	7.98×10^{9} (8.0 *Gn-gram*)	8.50×10^{10} (85.0 *Gn-gram*)
$\|C\|$	8.78×10^{11} (0.9 *Tw*)	9.69×10^{11} (1.0 *Tw*)	1.12×10^{12} (1.1 *Tw*)	1.20×10^{12} (1.2 *Tw*)	1.43×10^{12} (1.4 *Tw*)	6.80×10^{12} (6.8 *Tw*)

3.2.6 Summary

The estimation of the number of distinct n-grams of different sizes in different *corpora* sizes is critical to support Big Data algorithm design and implementation. The identification of such *plateau* levels allows to determine the maximum required capacity of memory and the number of machines (for distributed implementations) in applications whose problem size is proportional to the number of distinct n-grams, e.g., in the LocalMaxs method [SL99], which counts n-gram frequencies, calculates n-gram glues, and extracts relevant n-grams.

3.3 Analysis of Distinct *n*-grams and their Frequencies of Occurrences in the *Corpus*

In section 3.3.1 we present an analysis of the evolution of the percentages of the distinct n-grams with the *corpus* size. The evolution of the n-grams repetition factors with the *corpus* size is discussed in section 3.3.2.

3.3.1 Percentages of Distinct *n*-grams

For each n-gram size i the ratio $|D_i|/|C|$ decreases with $|C|$ and is higher for the larger n-gram sizes. In fact this ratio tends to zero as the *corpus* size grows towards infinity. This

is due to the fact that the number of distinct n-grams in a *corpus* is constrained by the size of the actual available language vocabulary. The larger n-gram sizes tend to dominate the entire population of n-grams. Table 3.10 and Figure 3.7 show the ratio $|D_i|/|C|$ for the ranges of *corpus* sizes analyzed, with i ranging from 1 to 6, for the English language.

Table 3.10: Ratio $|D_i|/|C|$ *versus corpus* size for the English language.

***Corpus* size [word]**	$\|D_1\|/\|C\|$	$\|D_2\|/\|C\|$	$\|D_3\|/\|C\|$	$\|D_4\|/\|C\|$	$\|D_5\|/\|C\|$	$\|D_6\|/\|C\|$
2.00×10^6	0.077	0.419	0.758	0.902	1.040*	1.350*
8.00×10^6	0.053	0.317	0.668	0.863	0.959	0.972
1.60×10^7	0.043	0.274	0.621	0.841	0.937	1.012*
6.40×10^7	0.029	0.201	0.525	0.780	0.909	0.947
1.28×10^8	0.024	0.170	0.476	0.743	0.884	0.936
5.12×10^8	0.016	0.117	0.376	0.651	0.819	0.900
1.02×10^9	0.012	0.095	0.325	0.594	0.779	0.874
4.10×10^9	0.008	0.057	0.223	0.457	0.674	0.811
8.19×10^9	0.006	0.042	0.173	0.377	0.607	0.770
1.64×10^{10}	0.004	0.029	0.126	0.291	0.531	0.722
1.31×10^{11}	0.001	0.005	0.027	0.073	0.253	0.523
1.05×10^{12}	0.000	0.001	0.003	0.009	0.048	0.247
1.68×10^{13}	0.000	0.000	0.000	0.001	0.003	0.023
1.34×10^{14}	0.000	0.000	0.000	0.000	0.000	0.003
1.07×10^{15}	0.000	0.000	0.000	0.000	0.000	0.000

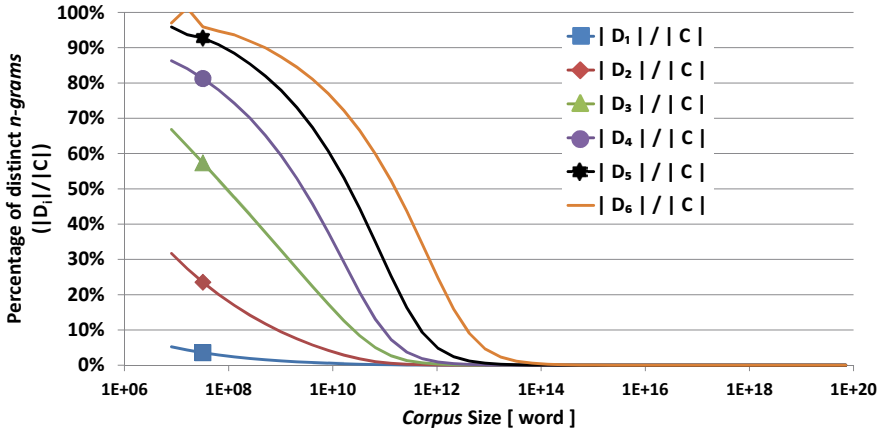


Figure 3.7: Ratio $|D_i|/|C|$ *versus corpus* size for English. The Y axis represents the ratio $|D_i|/|C|$ and the X axis represents the *corpus* size in words.

Note that the anomalies in the three * marked values (in Table 3.10) are due to the inaccuracies of the theoretical model discussed in section 3.2.4 in the cases of smaller *corpus* sizes. From our observations, when applying the model for *corpus* sizes larger than 20 Mw in the case of the English language, the model estimates exhibit non significant deviations from the actual *corpora* values, which is related to the statistical robustness of the large numbers. Thus, the model estimates can be safely used for large *corpora*.

3.3.2 Average Repetition of *n*-grams in *corpora* up to Infinity

From the ratio $|D_i|/|C|$ presented above we can define an average repetition factor (R_i) for each n-gram size, defined as $R_i = \left|Set_{All_i}\right|/|D_i|$, where $\left|Set_{All_i}\right|$ is the total number of occurrences of n-grams of size i, and $|D_i|$ is the number of corresponding distinct n-grams of size i in the *corpus*. For each n-gram size i, this factor increases with the *corpus* size.

Figures 3.8 shows the average repetition factor R_i for the ranges of *corpus* analyzed, with i from 1 to 6, for English up to the *plateau* regions.

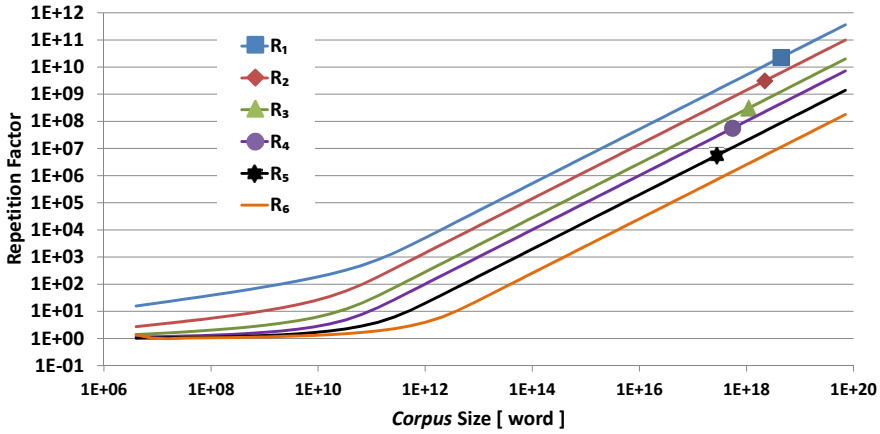


Figure 3.8: Average repetition factor $R_i = \left|Set_{All_i}\right|/|D_i|$ *versus corpus* size for English up to the *plateau* regions. The Y axis represents the repetition factor, and the X axis represents the *corpus* size ranging from 8 Mw up to the *plateau* regions.

Algorithms can benefit from the knowledge about this repetition behavior to exploit the temporal locality of the n-gram references, which can be captured by an n-gram cache. The above benefit is greater for the smaller n-gram sizes because they have higher repetition factors. A similar analysis can be made to other languages regarding these quantities $\left|Set_{All_i}\right|$, $|D_i|$, $|R_i|$, or $|D_i|/|C|$.

3.4 A Theoretical Model to Estimate the Number of *n*-grams with a Given Frequency

If we can estimate how frequently the distinct n-grams occur in a very large *corpus* we can calculate, for each specific application, the number of access requests to those n-grams and then a more efficient usage of the storage space can be achieved.

In this section we present a generic model to estimate the number of n-grams with a given frequency $k \geq 0$. In particular, for $k = 1$, the model provides an estimate of the number of singletons in a *corpus* with a given size that can be very useful, since, by occurring less frequently, these n-grams must be the first to be discarded from memory when this is a critical resource.

This model shares the same theoretical foundation as the model presented in section 3.2.2, namely it is also based on the properties of the Poisson distribution and the Zipf-Mandelbrot law, and also applies to single words or multiwords.

3.4.1 Background

The authors of the study presented in [Boo67] analyze the distribution of low frequency words and claim that the ratio of the number of singletons over the number of distinct words is still independent of the text length. We will show that this is far from true in real *corpora* singletons distribution, specially for Big Data *corpora*.

In [LA09], authors claim the median and the mode of the frequency of words tend to have the same value in all *corpora*. We show that this is not correct, since for Big Data *corpora*, the most common number of occurrences of each word, i.e., the mode referred to by those authors, tends to stop being 1, as opposed to what happens for non-Big Data *corpora*.

Other works [Sch10; Tay15] analyze the distribution of words in text as well as the number of words with the same frequency, but not as a function of the *corpus* size, in particular concerning the evolution of the number of singletons. In [Baa01], it is said that $V(m,N) \propto \frac{1}{m^a}$ where $V(m,N)$ is the number of words with frequency m in a *corpus* having N tokens, and a is a free parameter of the model determining the slope of the regression (see [Baa01] for details). We show that this does not match for singletons ($m = 1$) in Big Data *corpora* since $V(1,N)$ tends to 0 in that context. In [CO02], the authors study the stability of the words distribution in text, and show a graphic with the probability ratio of the singletons when the *corpus* size varies but no law or formula is shown to estimate it as a function of the *corpus* size. We present an approach to estimate the number of n-grams with the same frequency as a function of the *corpus* size, with an application to the case of singletons in the context of Big Data.

3.4.2 The Number of *n*-grams with the Same Frequency in a *Corpus*

According to expression (3.6), we obtain the following expression for the probability that n-gram w in rank r occurs $X = k$ times:

$$Pr(X = k) = \frac{\lambda_r^k e^{-\lambda_r}}{k!} \tag{3.19}$$

where λ_r stands for the expected frequency of rank r, that is $f(r,l,n,c)$, which is given by expression (3.5) once generalized to the case of n-grams of size n. Now, in order to estimate the number of distinct n-grams ($W(k,l,n,c)$) of that size (n), that occur k times in a *corpus* of size c, written in a language l, we propose to sum the probabilities of all the distinct n-grams of size n occurring k times in that *corpus*, considering the entire vocabulary ($v(l,n)$) of language l for n-grams of size n:

$$W(k,l,n,c) = \sum_{r=1}^{r=|\nu(l,n)|} \frac{\lambda_r^k \times e^{-\lambda_r}}{k!} = \sum_{r=1}^{r=|\nu(l,n)|} \frac{f(r,l,n,c)^k \times e^{-f(r,l,n,c)}}{k!} \tag{3.20}$$

Similarly to the case of the distinct n-grams model, we propose an efficient implementation of (3.20), where the heavily iterated sums are replaced with an integral, leading to expression (3.23), which is derived as follows using the same symbols for A, Q, R, $\alpha(l,n)$, $\beta(l,n)$, $r_1^\star$ and $r_\nu^\star$ as defined in expression (3.13) on page 34.

$$\begin{aligned} W(k,l,n,c) &= \textstyle\sum_{R=r_1^\star}^{R=r_\nu^\star} \frac{\left(Q\times R^{-A}\right)^k \times e^{-Q\times R^{-A}}}{k!} \\ W(k,l,n,c) &= \tfrac{Q^k}{k!} \textstyle\sum_{R=r_1^\star}^{R=r_\nu^\star} R^{-A\times k} \times e^{-Q\times R^{-A}} \end{aligned} \tag{3.21}$$

Assuming the accuracy obtained with an integral for substitution of this sum, we obtain the following:

$$\begin{aligned} W(k,l,n,c) &\approx \tfrac{Q^k}{k!} \textstyle\int_{r_1^\star}^{r_\nu^\star} R^{-A\times k} \times e^{-Q\times R^{-A}} dR \\ W(k,l,n,c) &\approx \tfrac{Q^k \times Q^{\left(\frac{1}{A}-k\right)}}{k!\times A} \left[\gamma\left(k-\tfrac{1}{A}, \tfrac{Q}{R^A}\right) + Const\right]_{R=r_1^\star}^{R=r_\nu^\star} \end{aligned} \tag{3.22}$$

$$W(k,l,n,c) \approx \frac{Q^{\frac{1}{A}}}{k!\times A}\left[\gamma\left(k-\frac{1}{A}, \frac{Q}{\left(r_\nu^\star\right)^A}\right) - \gamma\left(k-\frac{1}{A}, \frac{Q}{\left(r_1^\star\right)^A}\right)\right] \tag{3.23}$$

Notice that the latter expression is structurally similar to the second parcel in expression (3.18) on page 34 when $k = 0$, which corresponds to the total number of distinct n-grams with frequency 0, i.e., not occurring in the *corpus*. In fact, the meaning of expression (3.9), on page 32, states that the number of distinct n-grams in a *corpus* with a given size is equal to the total number of vocabulary n-grams minus the number of all the n-grams which have not yet occurred in the *corpus*.

3.4.3 Validation of the Model

Since the distinct n-grams model (expression (3.10) or expression (3.18)) and the equal frequencies model (expression (3.20) or expression (3.23)) have the same theoretical foundation, their parameters (α, β, ν and p_1 as presented in Tables 3.4 and 3.5) are the same. Then, the accuracy of expression (3.23) was assessed with the English and French languages with three large *corpora*: about 250×10^6, 500×10^6 and 1000×10^6 words. Smaller *corpora* were not considered as the proposed approach focus mainly on large and Big Data *corpora* where the large number of occurrences guarantees the statistical robustness necessary to obtain accurate estimates.

In this assessment the model was used to estimate the number of distinct n-grams of sizes from 1 to 6, having frequencies k, ranging from 1 to 128 in power of 2 increments,

although for simplicity, Tables 3.11 and 3.12 only show the cases of unigrams, trigrams and pentagrams. Values for bigrams, tetragrams and hexagrams are similar.

Tables 3.11 and 3.12 show the calculated relative errors, obtained as in Tables 3.6 and 3.7, between the model estimates and the actual observed numbers.

Table 3.11: Relative errors of the estimates of the number of distinct n-grams with the same frequency in English *corpora*. Values in percentage (%). *Corpus* size in words.

	***Corpus* size**	**Frequency**							
	[word]	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
	245,492,006	1.1	12.0	3.0	0.0	-2.0	-5.6	-9.0	-9.0
unigrams	490,846,877	-4.6	13.1	9.2	2.6	0.2	-2.1	-5.4	-6.5
	981,996,022	-7.0	11.3	8.1	6.6	7.1	0.3	1.7	-9.1
	245,492,006	0.9	5.5	-5.0	-8.0	-8.9	-10.8	-5.0	1.8
trigrams	490,846,877	-0.6	11.3	-0.2	-6.0	-8.8	-10.7	-9.5	-7.8
	981,996,022	-2.7	12.1	-2.0	-6.8	-9.4	-8.0	-7.1	-6.9
	245,492,006	-0.1	11.6	8.2	5.6	5.2	5.8	9.8	9.9
pentagrams	490,846,877	-0.7	10.5	9.1	6.4	4.8	3.2	8.3	6.2
	981,996,022	-1.2	11.9	10.0	8.0	5.4	3.0	2.5	-2.7

Table 3.12: Relative errors of the estimates of the number of distinct n-grams with the same frequency in French *corpora*. Values in percentage (%). *Corpus* size in words.

	***Corpus* size**	**Frequency**							
	[word]	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
	242,346,014	1.8	11.2	2.9	1.4	-2.3	-4.1	-7.6	-8.2
unigrams	484,314,987	-0.3	12.1	7.3	1.6	0.9	-1.1	-3.9	-5.8
	970,351,308	-4.0	13.0	9.0	7.1	0.8	0.1	1.1	8.3
	242,346,014	1.0	4.2	-6.1	-7.8	-9.2	-10.5	-2.5	-1.3
trigrams	484,314,987	-4.9	9.6	-0.3	-4.5	-8.3	-7.0	-8.4	-6.9
	970,351,308	-3.0	13.1	7.1	-2.5	-7.1	-6.4	-7.1	-7.0
	242,346,014	-1.1	8.8	7.4	3.9	5.1	6.6	9.9	14.0
pentagrams	484,314,987	-0.4	11.5	8.1	8.6	3.6	4.4	7.8	7.2
	970,351,308	-1.4	9.9	9.0	7.7	6.0	3.9	1.8	2.1

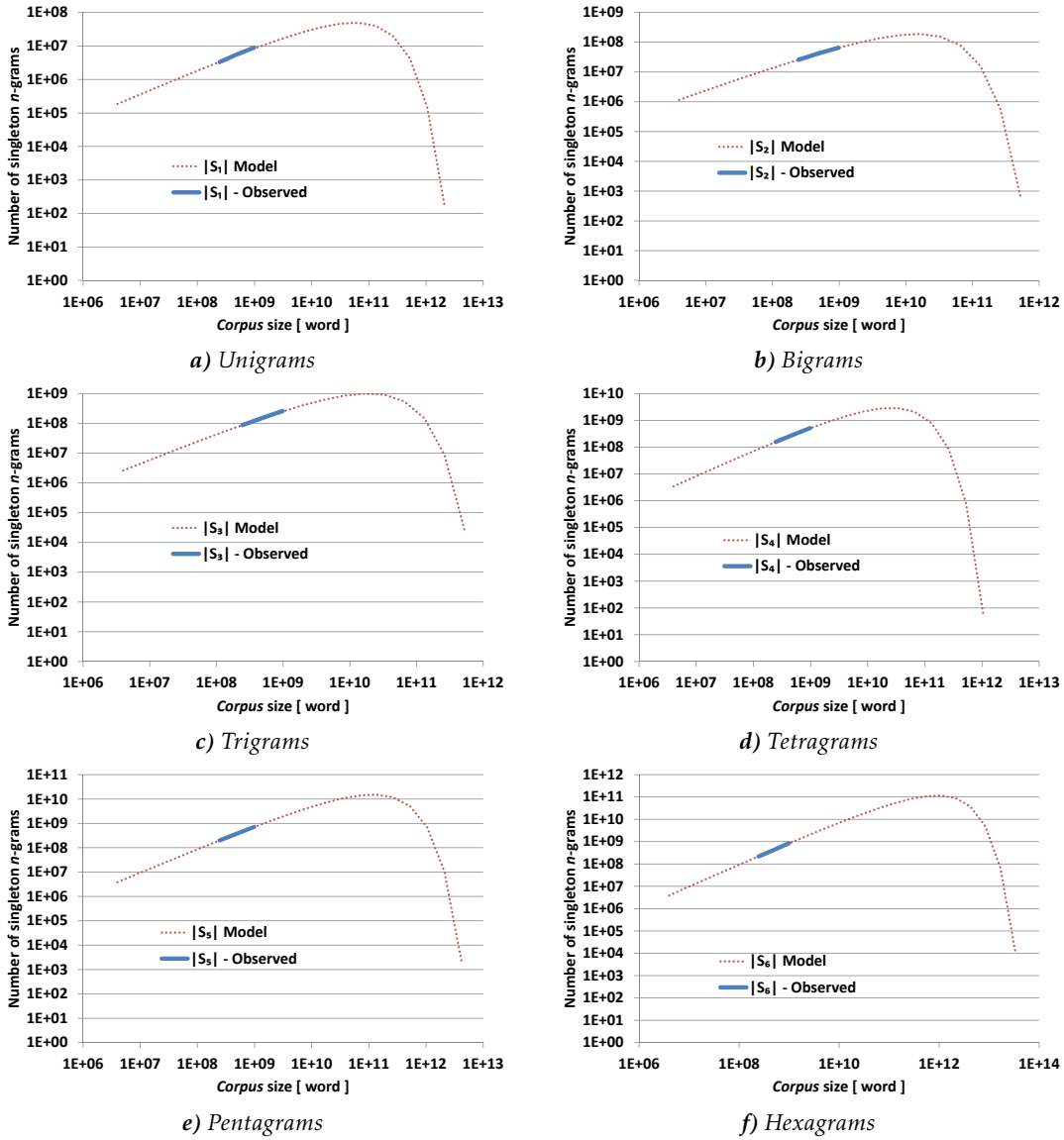
The relative errors can be considered relatively low, since the average of the absolute values over the entire range of considered frequencies is about 6% for unigrams, trigrams and pentagrams for both languages. However, relative errors tend to be higher than the average for the frequency of 2, which becomes a motivation for the improvement of this approach.

Due to the fact that the Lower Incomplete Gamma function (expression (3.23)) originates a calculation error for values of k greater than 171 and because it is impossible to directly represent the factorial of large values of k, the calculation of $W(k,l,n,c)$, for values higher than those shown in Tables 3.11 and 3.12, requires some additional work to overcome this limitation.

However, from our experiments, we found out that for values of frequencies below 200, the model estimates provide sufficiently low relative errors. Namely, this enables us to proceed with the application of the model to estimate the number of singleton n-grams occurring in a *corpus* of a given size.

3.5 Analysis of the Singletons in the *Corpus*

The low relative errors allow us to study the evolution of the number of singletons as a function of the *corpus* size. Figure 3.9 depicts that evolution, which was also obtained from expression (3.23).



a) *Unigrams* ***b)*** *Bigrams*

c) *Trigrams* ***d)*** *Tetragrams*

e) *Pentagrams* ***f)*** *Hexagrams*

Figure 3.9: Number of singletons *versus corpus* size for the English language. The Y axis represents the number of singleton n-grams and the X axis represents the *corpus* size in words.

Thus, by Figure 3.9-a) we can see that the maximum number of singleton unigrams in English *corpora* is reached for a *corpus* size of about $10^{10.7}$ words. More precisely, $10^{10.7033}$ = 50,501,002,567 words *corpus* and the corresponding estimate, that is the expected number of singletons is 49,272,372.2 unigrams. The same figure suggests that the expected number of singleton unigrams is about 0 when the *corpus* size is greater that 10^{12} words. In fact, for a size of $10^{12.28} \approx 1.9$ Tw, the expected number of singletons is 617 unigrams — a very small fraction of the *corpus* size or of its number of distinct words. For a size of $10^{12.816}$ = 6,546,361,740,673 words the singletons estimate is 5.837×10^{-11} unigrams. Figure 3.9-a) to Figure 3.9-f) show that the curves of the estimates deviate to the right and present higher maximum numbers of singletons as the size of the *n*-gram grows.

This happens both for English and French. Although for simplicity we do not present the values for French, we also observed, for example, that in a French *corpus* of $10^{13.344}$ = 22,080,047,330,189 words, the expected number of singleton hexagrams is 6.883×10^{-8}. It means that in practice no hexagram is expected to occur with frequency 1 for a French *corpus* of this size; and for obvious reasons, no singleton unigrams, bigrams, up to pentagrams are expected for the same *corpus*, that is, every *n*-gram will certainly occur with higher frequency than 1.

So, we conclude that 1 is not the most common number for the frequency of *n*-grams in Big Data, contrary to what happens for small and medium size *corpora*.

3.6 Analysis of the Nonsingletons in the *Corpus*

By using expression (3.18), on page 34, which for a *corpus* C gives the total number of distinct *n*-grams ($|D_i|$) of size i and by using expression (3.23), on page 45, which for the same *corpus* C and $k = 1$, gives the total number of singleton *n*-grams of size i ($|S_i|$), we can derive the total number of distinct nonsingleton *n*-grams ($|NS_i|$) of size i in the *corpus* C as:

$$|NS_i| = |D_i| - |S_i| \tag{3.24}$$

In the remaining of this thesis document when the term "nonsingleton" is mentioned, the intended meaning is "distinct nonsingleton".

Figures 3.10 to 3.13 show the evolution of singletons, nonsingletons and distinct *n*-grams of different sizes, with $1 \leq n \leq 6$, when the *corpus* size grows up to the *plateaux* for the English language.

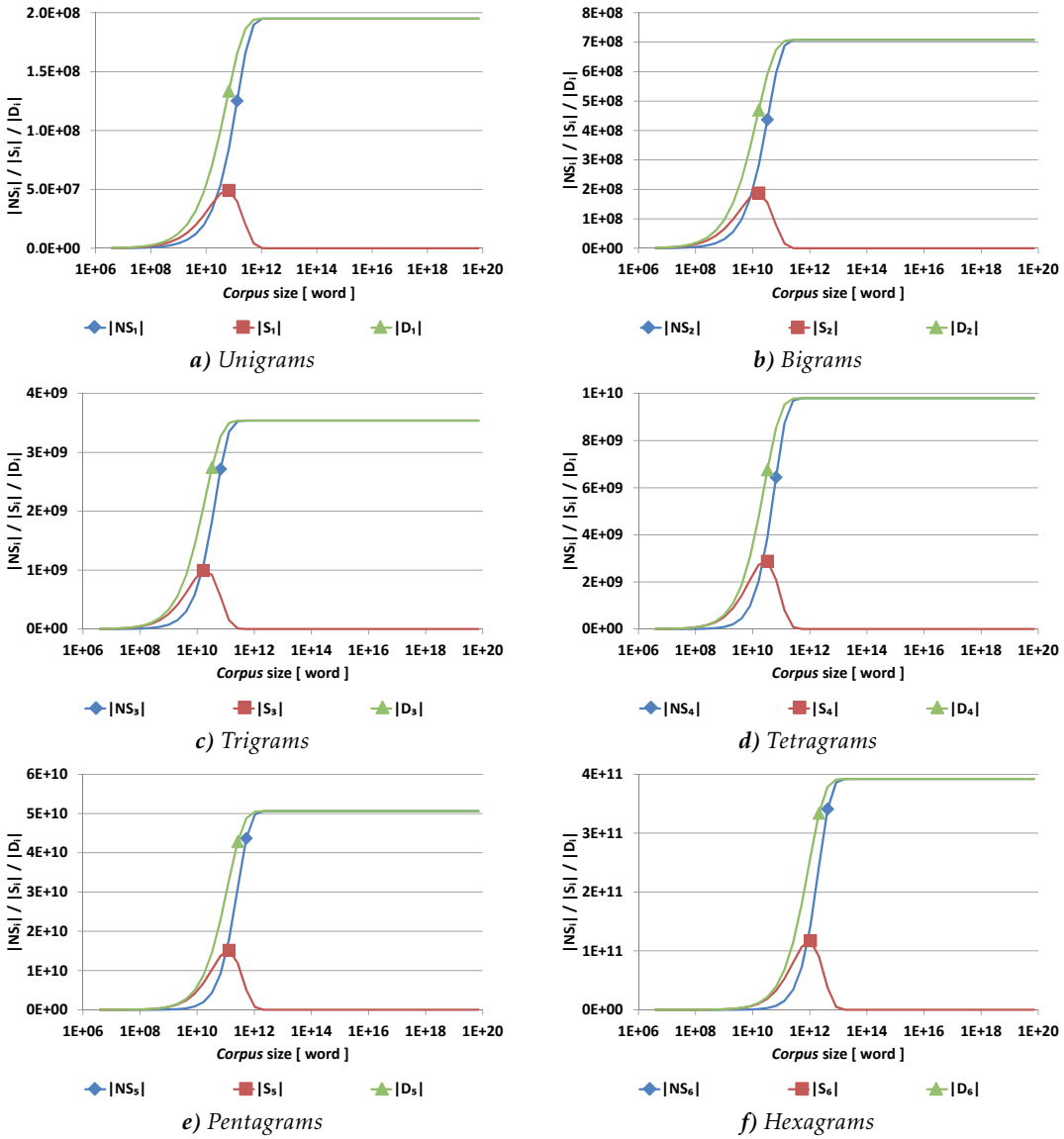


Figure 3.10: Number singletons ($|S_i|$), nonsingletons ($|NS_i|$) and distinct ($|D_i|$) n-grams *versus corpus* size. The Y axis represents the number of nonsingleton, singleton and distinct n-grams; and the X axis represents the *corpus* size in words.

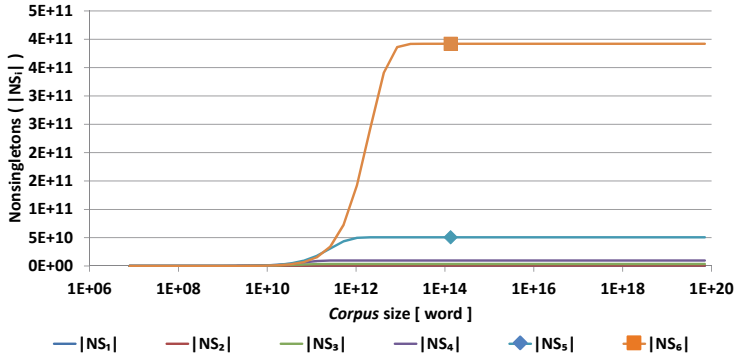


Figure 3.11: Number of nonsingletons ($|NS_i|$) for n-grams $1 \leq i \leq 6$ *versus corpus* size. The Y axis represents the number of nonsingleton n-grams and the X axis indicates the *corpus* size in words.

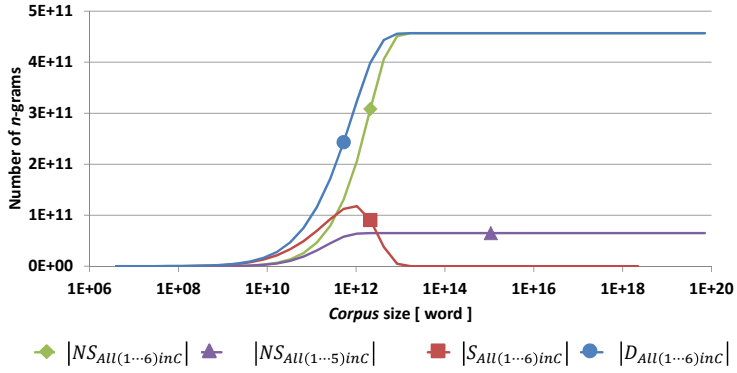


Figure 3.12: Total number of distinct ($\left|D_{All(1\cdots n)_{in}C}\right|$), singletons ($|S_{All(1\ldots n)_{in}C}|$) and nonsingletons ($|NS_{All(1\ldots n)_{in}C}|$) n-grams ($n = 6$) from unigrams up to hexagrams *versus corpus* size. Nonsingletons ($\left|NS_{All(1\cdots 5)_{in}C}\right|$) is also shown. The Y axis represents the number of nonsingleton, singleton and distinct n-grams and the X axis represents the *corpus* size in words.

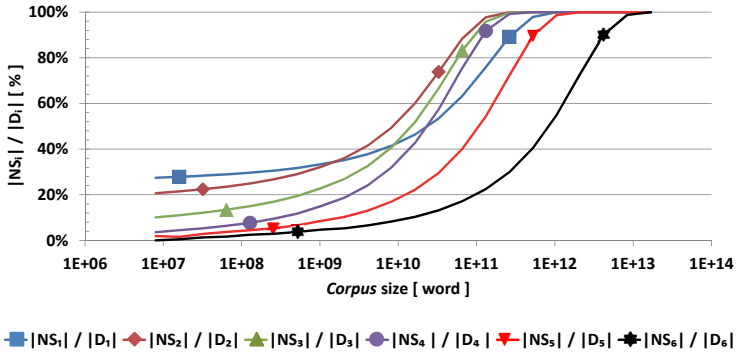


Figure 3.13: Ratio of nonsingletons ($|NS_i|$) over distinct ($|D_i|$) n-grams *versus corpus* size. The Y axis represents the ratio $|NS_i|/|D_i|$ $1 \leq i \leq 6$ and the X axis represents the *corpus* size in words.

3.7 Fixed Frequency Accumulation Set — FAset

In section 3.7.1 we present the concept of Fixed Frequency Accumulation Set (FAset) and in section 3.7.2 we present its formal definition.

3.7.1 Concept

Our experiments have shown that, for *corpora* sizes larger than a given value, there is a number of distinct *n*-grams whose occurrences are responsible for a large percentage of the total occurrences of *n*-grams. For example, in the case of the English language, for *corpora* larger than 29 Mw, 50% of the total occurrences of unigrams are only due to the most frequent 509 distinct unigrams. We name this set, FAset-50%, whose membership becomes practically constant for increasing *corpora* sizes. For *corpora* larger than 406 Mw, 75% of the total occurrences of unigrams lie only on the most frequent 8,190 unigrams. This behavior also occurs for greater *n*-gram sizes, but it takes larger *corpora* sizes to reach the thresholds for the corresponding FAset, which have much higher cardinalities. For example, for bigrams the FAset-50% contains 180,000 distinct bigrams beyond *corpora* of size 812 Mw. Other human languages may exhibit slightly different FAset size values. However, the cardinalities of the FAset are very small with respect to the *corpora* total size, mostly for unigrams and bigrams.

3.7.2 Definition

Among all the *n*-grams of each size there are individual *n*-grams that occur much more frequently than others. Their distribution exhibits a Zipf-like [Zip35] behavior where for each *n*-gram size, a small percentage of *n*-grams is responsible for most of the *n*-gram occurrences.

This behavior can be explored by an algorithm implementation that can take advantage of the higher frequencies of certain *n*-gram subsets within each *n*-gram size. Those *n*-grams have a higher contribution to improving a cache efficiency (hit ratio).

The concept of Fixed Frequency Accumulation Set (FAset) is used to model the cumulative sum of frequencies of a set of distinct *n*-grams, and to select the minimal subset of *n*-grams which represent a target percentage of the total *n*-gram occurrences in a *corpus*.

Definition 3.1: Hit ratio of a FAset

For a given n-gram size n, a FAset FA for ensuring a target hit ratio $h(n,FA)$ is the minimal subset of distinct n-grams of size n, whose cumulative sum of frequencies within the *corpus* is responsible for ensuring a percentage of the total number of occurrences of the n-grams of size n in the entire *corpus* (Set_{All_n}), such that:

$$h(n,FA) = \frac{\sum_{ng \in FA} freq_{inCorpus}(ng)}{\left|Set_{All_n}\right|}$$

where ng denotes a distinct n-gram included in the FAset and $freq_{inCorpus}(ng)$ is the frequency of the n-gram ng in the *corpus*.

Note that that $\left|Set_{All_n}\right| \approx |C|$ for any n-gram size n.

As illustrated in Figure 3.14 the distinct unigrams with ranks from 1 to r_h ensure an intended relative coverage, determined by the shaded area.

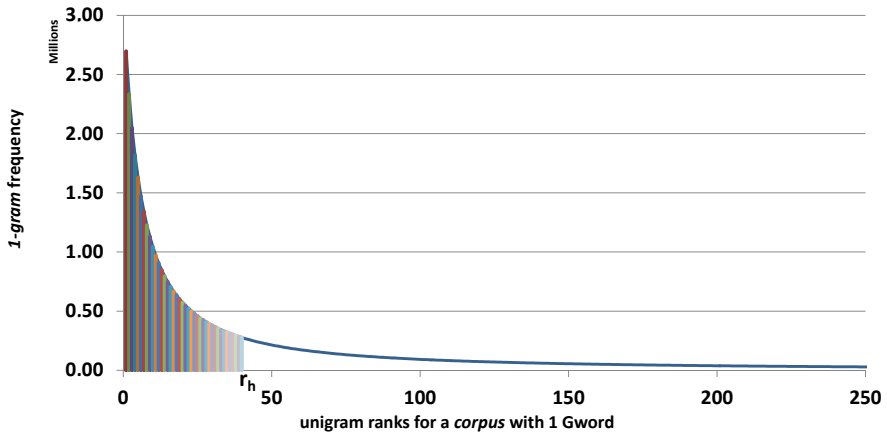


Figure 3.14: Distribution of unigram ranks for a *corpus* of 1 Gw. The Y axis represents the frequency of distinct unigrams, in millions; and X axis represents their ranks.

This definition can be used to determine the minimal FAset size necessary to ensure a target relative coverage, i.e., hit ratio, and to identify the culprit n-grams which should be kept in an n-gram cache to provide such desired hit ratio. This identification can be made by a static analysis of the *corpus*, and can be useful to guide cache prefetching and replacement strategies.

3.8 Chapter Summary

We proposed an approach to estimate the number of distinct n-grams of a given size n, for any *corpora* size. The approach was evaluated for $1 \leq n \leq 6$, for the English and French languages. It can be used for English or other languages, just requiring a small and a large *corpus* to be used to tune the model parameters for that language and for

each n-gram size. The approach is based on Zipf-Mandelbrot Law and on the Poisson distribution. Computationally heavy sums are replaced with an integral in order to provide more efficient calculations. This can be useful for Big Data applications in the context of data mining, database systems or for cache size and hashing size calculation, where the memory space to accommodate n-gram set cardinalities needs to be quickly estimated. In the context of this development, we noticed that the probability of each distinct n-gram in a *corpus* tends to remain constant when *corpora* of the same language grow in size, which became evident for frequent n-grams, in the experiments we made. And there is no reason to think that, for Big Data *corpora*, the same will not happen for less frequent n-grams. This property led us to develop this approach. Although language vocabularies for each n-gram size are open, as new words may arise and others tend to stop being used in each language, they are finite. This sets a *plateau* for the maximum number of distinct n-grams of each size n, which exists in any *corpus* of a given language, as our approach shows when estimates are calculated for Big Data *corpora*. Tests showed promising results for the calculations of those estimates, as the highest relative error regarding the prediction of the total number of distinct n-grams was lower than 5%.

In this thesis, the results of the study on the n-grams distribution presented in this chapter, were applied to support the analysis and the design of the parallel and distributed implementation of a statistical extraction method, LocalMaxs. The model estimates led to conclusions regarding the LocalMaxs implementation behavior which were confirmed by the experimental results of real execution, for *corpus* up to 1 Gw and n-gram sizes up to hexagrams.

Also the model estimates allowed to predict the evolution of the LocalMaxs implementation for *corpus* sizes up to the *plateau* regions, thus allowing to estimate their behavior for Big Data.

Part III

Parallel and Distributed *n*-gram Extraction

Chapter 4

A Distributed Architecture for *n*-gram Extraction

A Distributed Architecture for n-gram Extraction.

As mentioned in chapter 1, the main goal of this dissertation is to propose solutions to the statistical-based extraction of relevant expressions from large *corpora*, using parallelism to achieve acceptable execution time and using distribution to enable the handling of large data sets. In order to achieve this goal, a distributed architecture for *n*-gram extraction algorithms was designed and implemented. Although all the experimentation and the validation of the architecture were conducted centered on the LocalMaxs method and its parallel and distributed implementations, the architecture design can also be useful when adapted to other *n*-gram based extraction methods that have a similar set of the characteristics, as outlined in the following.

This chapter is organized in 4 sections. The rationale of the architecture is presented in section 4.1. Section 4.2 presents an outline of the architecture. Section 4.3 describes the implementation of the logical architecture on top of the runtime environments. This chapter ends with a summary in section 4.4.

4.1 Rationale of a Distributed Architecture for *n*-gram Extraction

In the following sections we present the main requirements that should be addressed by the proposed architecture: i) To be used in the contexts of multiphase *n*-gram statistical-based extraction methods (section 4.1.1); ii) To support intermediate and final data sets as distributed *n*-gram tables (section 4.1.2); iii) To be based on a three layered approach (section 4.1.3).

4.1.1 Multiphase *n*-gram Statistical-based Extraction

Multiple phases. Typically, many *n*-gram statistical-based extraction methods rely on a first phase for collecting *n*-gram statistics from an input *corpus*, or use existing available *n*-gram statistical data sets, for example, *Google N-gram Viewer* [Goo16] or *Microsoft Web N-gram Corpus* [Wan+10], possibly with some preprocessing, and then proceed with several other phases where specific metrics of the corresponding extraction method are evaluated and/or further operations are applied.

For example, in the LocalMaxs method three successive phases are identified, for *n*-gram counting, glue calculation, and relevance evaluation. The dependencies between phases can be described as a simple pipeline whose stages correspond to the individual algorithm phases. More generally, they can be described as a workflow, i.e., a graph of activities with arcs denoting their dependencies, which allows more freedom in the exploitation of a more fine algorithm decomposition concerning its execution and data flows.

Workflow-based approach. Due to the above, a workflow-based description seems more adequate to express the alternatives for parallel and distribution execution. In fact, the mentioned multiphase characteristic is common to typical data analytics (and other...) applications and a diversity of implementations relies on workflow-based approaches. For example, a widely used model for this purpose is MapReduce, which corresponds to an implicitly defined two-step workflow, being adequate to exploit the application data parallelism by splitting the input data, processing its parts independently and in parallel and finally merging the partial results. As an alternative, in this work we rely on a more generic and flexible workflow tool in order to remove the two-step limitation of the original MapReduce model (although in the meanwhile the functionality of this model has been subject to several extensions, e.g., to support multiple iterations, etc...(as mentioned in chapter 2)). In our proposal, it is possible to express an extraction algorithm with any number of phases, and the designer can express alternative forms of phase sequencing and coordination, e.g., ranging from a strict sequential execution (as in a simple sequential pipeline) to several degrees of overlapped execution involving multiple algorithm phases. This provides more flexibility, e.g., by allowing a complex phase to be decomposed into multiple internal tasks and by coordinating the task dependencies (both inter-task and among phases) according to the workflow graph. This is illustrated later on this chapter in section 4.1.3.1. Although this approach may incur some more complexity to the algorithm specification by the designer when compared to the implicit MapReduce approach, we believe this is overcome by the benefits from its increased flexibility.

4.1.2 Distributed *n*-gram Table Storage

The architecture should support an independent global *n*-gram table store that can be accessed as input, intermediate, or output data by the algorithm tasks in its multiple phases. In order to allow scalable processing of large *corpora* this *n*-gram table was designed as a distributed in-memory key-value store. Its supported functionalities are similar to a diversity of other proposals [Mem16], including some related to *n*-gram data storage (e.g., as in [BFR11]). In our proposed design and implementation, the distributed *n*-gram table store is decoupled into a set of separate and independent *n*-gram tables, one dedicated to each *n*-gram size and these tables are uniformly distributed among a set of servers. The *n*-gram tables can be first used by an initial algorithm phase collecting the *n*-gram statistics from an input *corpus*, and can later be consulted and updated by the remaining phases as the algorithm execution proceeds. Depending on the problem requirements, phase one can be completely decoupled from the remaining phases. This has the advantage that, once phase one has been completed, further phases are allowed to apply different evaluations, e.g., alternative metrics and extraction operations, using the global *n*-gram data for a given *corpus* data set. Alternatively, phase one can run concurrently with the other algorithm phases but this requires a suitable coordination of the accesses to the distributed *n*-gram tables by the tasks running in the overlapped phases. In fact this only amounts to ensuring that the algorithm task dependencies, as expressed in the logical algorithm workflow, are preserved. For example, concerning the LocalMaxs method it is possible to perform the evaluation of counting, glue and relevance of *n*-grams of different sizes concurrently, as illustrated by the logical algorithm workflow in Figure 4.2 on page 61.

Although during the algorithm execution it is beneficial to rely on the in-memory implementation of the distributed *n*-gram tables, these data can also be saved and kept in a persistent storage system, for further future use without requiring rerunning phase one for the same *corpus*. This is similar to using existing available *n*-gram statistics data sets (for example *Google N-gram Viewer*) as a basis for statistical-based evaluations.

4.1.3 A Three-layered Architecture

The developed architecture was designed with three layers of abstraction, as illustrated in Figure 4.1: i) Algorithm Layer, described in section 4.1.3.1; ii) Logical Virtual Architecture, described in section 4.1.3.2; iii) Runtime Environment and Infrastructure, described in section 4.1.3.3.

The rationale of this design is well-known to promote a top-down development for each algorithm design and implementation, from a high-level workflow-based description, to an intermediate logical virtual machine implementation, down to the physical execution on each specific hardware infrastructure.

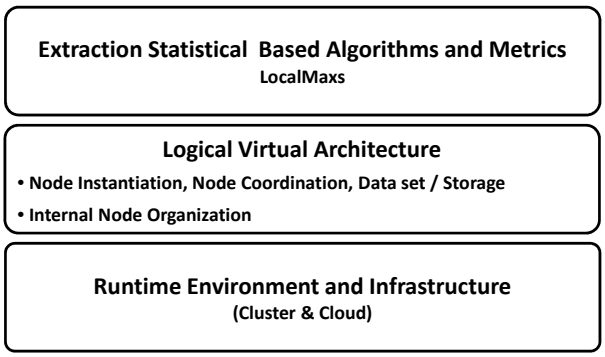


Figure 4.1: A layered architecture for n-gram extraction.

4.1.3.1 Algorithm Layer

In order to keep a high-level view focused on the logics of the algorithm specification, as far as the end user is concerned, the top (first) layer allows a specification of the algorithm functions and their multiphase decomposition as a logical workflow, and completely hides the lower level concerns of algorithm execution. This upper layer is responsible for the operations concerning the extraction algorithms, their corresponding metrics and the specification of the algorithm phases and their logical dependencies.

Statistical algorithms typically are composed of a set of operations, applied in sequence to the n-grams of different sizes, and they can be described as a graph, whose nodes represent the algorithm operations and the links represent the dependencies among the operations. In this work we propose that the sequencing and the interdependencies among the different operations are specified as a workflow. For phase one the input data set consists of the *corpus* documents. The intermediate data between phases and the algorithm output data are kept in tables with entries of the form $\langle key, values \rangle$, where key is an n-gram and $values$ are the results of evaluating metrics associated with the n-gram (for example frequency, glue, relevance criterion, etc.).

As an example Figure 4.2 shows the logical workflow that describes the dependencies among the operations needed to identify relevant expressions using a statistical method such as LocalMaxs for the case of finding relevant expressions for bigrams (2-grams) up to pentagrams (5-grams).

This logical workflow assumes that each workflow node is in charge of processing the entire data set that composes the *corpus*. The workflow nodes shown in Figure 4.2 correspond to the three algorithm phases and to the different sizes of the n-grams, in this case unigrams up to hexagrams (in the case of phases one and two) and pentagrams (for phase three). Each workflow node executes the algorithm operation to calculate the associated function for each corresponding phase, according to the control flow dependencies shown.

Processing of large *corpora* in acceptable time is only feasible using parallel and distributed computing. Thus, the workflow nodes are mapped into virtual computing nodes

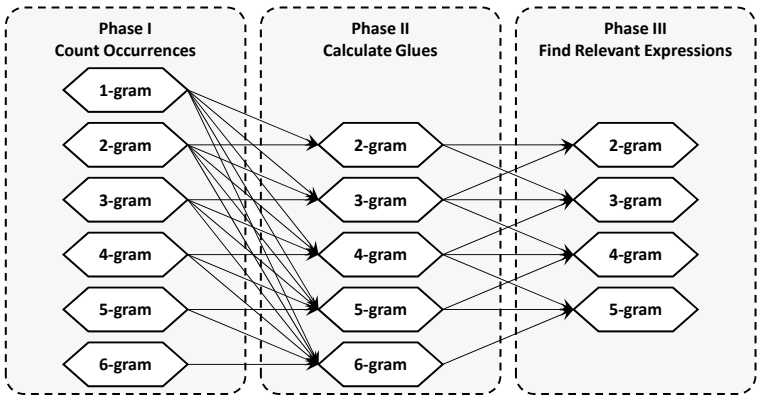


Figure 4.2: Logical workflow to express dependencies among extraction operations.

that are supported by the system architecture layer described in the following. During execution, the workflow nodes have access to the input *corpus* data set and the intermediate *n*-gram tables by using the data set / storage operations provided by the system architecture layer.

4.1.3.2 Logical Architecture Layer

An intermediate (second) layer, named the logical virtual architecture, provides abstractions to express dimensions related to the parallel and distributed execution of the logical workflow tasks, such as their agglomeration and granularity, the sequencing and overlapping of their execution, the degree of distribution of the *n*-gram tables, and the degree of locality of accesses to the distributed *n*-gram table store. However, they are still transparent to (and independent of) the execution mechanisms provided by each specific underlying computing infrastructure.

The logical virtual architecture layer is responsible for two main groups of operations: i) Operations for implementing the algorithm functions applied to the *n*-grams; ii) Data set and storage operations for accessing input data sets and the intermediate *n*-gram tables. These operations are supported by a distributed architecture consisting of a set of interconnected virtual computing nodes — Computing Node (CN) — as illustrated in Figure 4.3. In the context of this dissertation we assume a distributed memory architecture.

Within each virtual computing node (represented by the white boxes), there are a number of controller processes (represented by the green hexagonal forms) for executing the algorithm functions, and a number of server processes (represented by the blue ellipses) for implementing a distributed in-memory Key-Value Store (KVS). The key-value servers collectively implement a set of distributed in-memory *n*-gram tables, one for each *n*-gram size, used to store the *n*-gram data. A local cache (represented by the white rectangles) in each virtual computing node allows to store the reused remote sub *n*-grams accessed during the execution of the algorithm operations. Integrated with the architecture there is a workflow tool for specifying the algorithm graph.

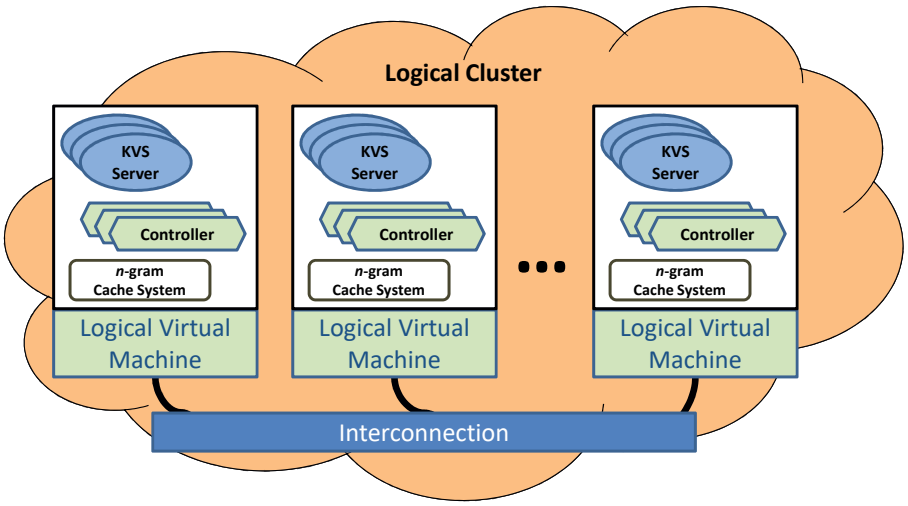


Figure 4.3: Logical architecture mapped to logical cluster.

The above workflow specification enables its parallel and distributed execution on the underlying runtime environment and infrastructure.

4.1.3.3 Runtime Environment and Infrastructure

The lowest (third) layer corresponds to the computing system infrastructure and its runtime environment, which provides the execution mechanisms that are specifically supported on top of the physical hardware infrastructure.

Virtual computing nodes can be mapped directly to system virtual machine instances, or they can be mapped to native operating system processes, depending on the virtualization support provided by each runtime environment. In each case the numbers of controllers and KVS servers are configured during the initialization of the corresponding virtual computing node. The virtual machines are logically interconnected among each other and implement a logical cluster as the one presented in Figure 4.3.

At the workflow level, associated tools support the mappings from the top to the intermediate levels. At the logical virtual architecture level, associated tools support the mappings to the actual computing system runtime environment. In this work, the strategies and mechanisms supporting the execution of the algorithm tasks at the lowest level are out of the control of the designer and developer: they are the sole responsibility of the existing computing system and runtime environment. This is due to the focus on cloud platforms as target infrastructure where there is a high level virtualization of the runtime environment.

4.2 Description of the Distributed Architecture for *n*-gram Extraction

In this section we describe the developed distributed architecture for *n*-gram extraction capable of supporting the execution of statistical methods. In the context of this work the architecture was used to support the execution of parallel and distributed implementations of the LocalMaxs method.

The distributed logical virtual architecture was designed as a collection of logical virtual machines. These are named as *"logical virtual machines"* to distinguish them from the system-level virtual machines directly supported by the machine virtualization functionalities of the operating system environment. Figure 4.3 shows the developed architecture, which supports two main functionalities:

- Algorithm functions — Each logical virtual machine contains one or multiple controller processes to execute the main algorithm functions (e.g, *n*-gram counting, glue calculation, and relevance evaluation). A logical virtual machine that only contains controllers is named a compute-only machine;

- Distributed in-memory *n*-gram store — Each logical virtual machine can also contain one or more key-value server processes (KVS) that collectively implement the distributed in-memory *n*-gram table store, as a set of distributed in-memory *n*-gram tables, one set for each *n*-gram size. The *n*-grams are distributed across the KVS servers using an hash function applied to the *n*-gram or part of the *n*-gram. A logical virtual machine that only contains key-value servers is named a serve-only machine. The distributed KVS store ensures a scalable solution to handle very large *n*-gram data tables. Besides, being based on in-memory tables it provides lower latencies than distributed disk-based databases. However, due to the physical distribution of the tables, it incurs remote communication overheads, influenced by the network characteristics. These overheads can be significantly reduced by an *n*-gram cache, which, depending on each application, exploits the algorithm temporal or spatial localities.

The controller and the server processes can be combined within the same logical virtual machine, which is then named a compute-and-serve machine. In order to allow the algorithm tasks to be executed by the controllers within a virtual machine to access the *n*-gram table store, the machine controllers can send requests to the distributed *n*-gram table servers, and get the corresponding replies. The set of KVS servers is exposed to the controllers using an uniform interface defined by the Data Access Storage Service (DASS)[1] object. This distributed memory model contributes to a scalable behavior for large *corpora* sizes and numbers of involved machines. At the distributed logical virtual machine layer,

[1] A detailed description is not provided in this dissertation although it is available in http://cjsg.ddns.net/~cajo/phd/, as well as the Java code of the architecture implementation.

several design decisions have impact upon the performance and the scalability behavior of an algorithm implementation:

- The allocation of logical workflow nodes (Figure 4.5) to logical virtual machines results from a decision taken by the designer, when configuring the logical virtual machines. For example, it is possible to have a one-to-one correspondence between the algorithm nodes and the logical virtual machines, or to have multiple algorithm nodes mapped to the same logical virtual machine. It is also possible to allocate logical virtual machines in a per phase basis, or to (re)use the same machine for algorithm tasks as the algorithm phases proceed with their execution;

- As described ahead, internally each controller relies on multiple threads to support the overlapped execution of different controller activities, such as reading from the input data set, executing its assigned algorithm tasks, handling the requests/replies to fetch data from the *n*-gram store, and writing the output results into the *n*-gram table store;

- Concerning the distributed in-memory *n*-gram store, the allocation of key-value servers to logical virtual machines is related to the strategy used for the partitioning of the *n*-gram tables. The effectiveness of a partitioning strategy is in general dependent on the *n*-gram distribution in the input *corpora* (as discussed in chapter 3) and also depends on each specific algorithm. This is discussed in chapter 5 on section 5.4.2 on page 84 in the case of LocalMaxs. In general a good data partitioning should ensure a well-balanced load among the key-value servers, during the algorithm execution;

- Being based on a distributed collection of logical virtual machines, this design leads to communication overheads due to the requests made by the controller processes to access the *n*-gram data in the distributed in-memory store. These overheads will of course depend on the mapping of the logical virtual machines to the actual hardware machines of the underlying computing system. At the logical virtual machine architecture level, by configuring compute-serve logical virtual machines, it is possible to colocate some controllers and servers within the same machine, thus reducing some of the accesses to the *n*-gram tables to local accesses. However, this will lead only to a minor reduction of those overheads because the processing of large *corpora* will also require a large number of machines with servers, so most of the accesses will be to remote servers. Using replication of all the *n*-gram tables within each machine would in general not be feasible due to lack of memory to keep the *n*-gram tables, when the *corpora* sizes grow;

- In order to reduce the above mentioned communication overheads, in this work we developed an *n*-gram cache system local to each controller within a logical virtual machine, that exploits the temporal locality of the *n*-gram occurrences in natural

language *corpora*, as discussed in chapter 3. The *n*-gram cache system is composed of a set of individual caches, one for each *n*-gram size. Depending on the characteristics of the algorithms, each individual cache can be separately configured and tuned, according to the following dimensions: i) Infinite *versus* finite capacity; ii) Cache warming up strategy; iii) Cache prefetch strategy; iv) Filtering *n*-gram singletons with Bloom filters. Algorithms that process natural language *corpora*, based on *n*-gram statistical information, are influenced by the *n*-gram distribution and repetition patterns, thus can benefit from specific configurations of the *n*-gram cache system.

Controller Organization. Figure 4.4 shows the internal organization of the controller used to implement the functions required by LocalMaxs (or other statistical extraction methods).

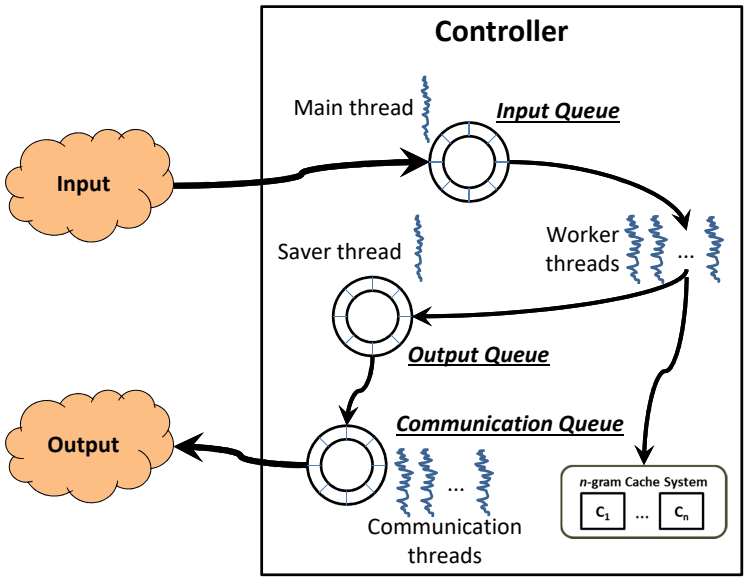


Figure 4.4: Controller internal organization.

Within each controller there is a main thread responsible for reading the input data set into an input queue. A collection of worker threads applies the algorithm functions to the input data set. In the case of the algorithm functions that need to fetch remote *n*-gram data from the remote KVS store, the local *n*-gram cache system is used and a set of communication threads handles the *n*-gram fetch requests made by the controller. A saver thread is responsible for sending the output data to the local key-value server through the communication queue. Both the input, the output and communication operations are overlapped with the worker threads processing. Internally to each controller, an *n*-gram and its associated data is represented by a `Record` (in Java) which contains the *n*-gram identification and a number, that can be the frequency, glue or relevance or other metric related to the *n*-gram.

4.3 Implementing the Logical Architecture on top of Runtime Environments

The current implementation of this architecture (controllers, KVS servers and *n*-gram cache system) was developed in Java, relies on an existing workflow framework [AGC12] and can be executed in standalone computers, cluster or cloud infrastructures. The only requirement is that on each node (CN) there must be a Secure Shell (SSH) server and a Java Virtual Machine (JVM) with version at least 1.7.X.

A set of associated support tools allows the configuration of the mappings of the controllers and KVS servers into the logical virtual machine images and their instantiation in the underlying execution environment. Namely, it is possible to allocate multiple controllers and/or KVS servers within the same virtual machine, thus allowing to benefit from the local interactions between such controllers and the colocated KVS servers. In all the experimentation work reported in this thesis the architecture was configured with a single controller and a single KVS server within each logical virtual machine. This is also the assumption considered in the performance analysis of the parallel and distributed LocalMaxs implementations.

Figure 4.5 shows how the mapping of the workflow nodes to a concrete Logical Machine Image and the mappings of each logical virtual machine to the physical infrastructure are performed using the developed tools.

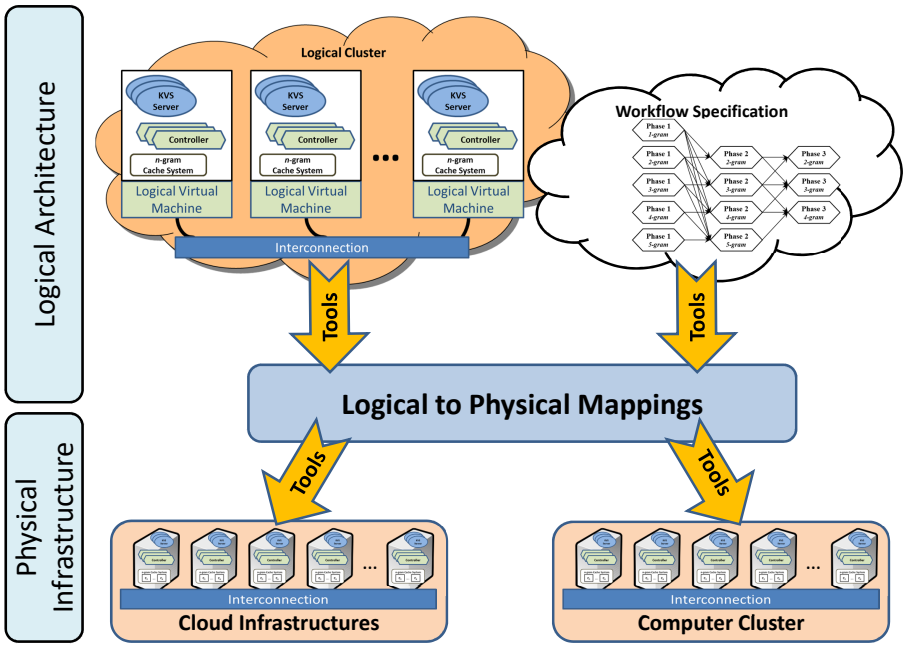


Figure 4.5: Logical to physical mappings.

4.3.1 Characterization of the Target Runtime Environments

Figure 4.6 shows the interaction of a developer with the distributed architecture using the developed tools.

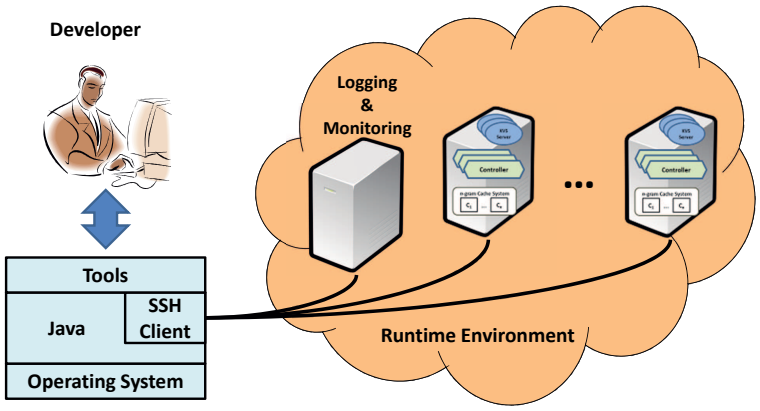


Figure 4.6: Developer interacting with the distributed architecture.

The monitoring and logging of events generated by the architecture components is made using the Log4Java [The16d] package. All the components of the architecture (controllers, KVS servers and n-gram cache system) are configured with a Log4Java appender capable of writing the logging events to a logging server. The developer can interact with the remote virtual computing nodes (CN), to execute the necessary scripts, using the standard SSH protocol. The developer is able to monitoring the state of the components by checking the logging server.

4.3.2 Development and Support Tools

A set of tools was developed to enable the specification of the logical workflow specific to each algorithm, the configuration of the logical virtual machine images and their instantiation with the desired number of controller and server processes, followed by the activation of the corresponding workflow instance. During the execution there are tools integrated with the controllers and servers to perform the logging of useful algorithm and virtual machine performance metrics. The logged information can be inspected during execution or used for a *post mortem* analysis.

The correspondence between the above functionalities and the three layers of the developed architecture is illustrated in Figure 4.7.

The logical workflow specification using the workflow tools supported by the *Autonomic Workflow Activities Reconfigurable and Dynamic (AWARD)* environment generates an eXtensible Markup Language (XML) description. The Java classes for the algorithm functions, the controller, the cache and the KVS server implementations, plus a set of activation scripts, are aggregated in a logical virtual machine image (stored in a file). The instantiation of the image is performed through the available tools provided by each

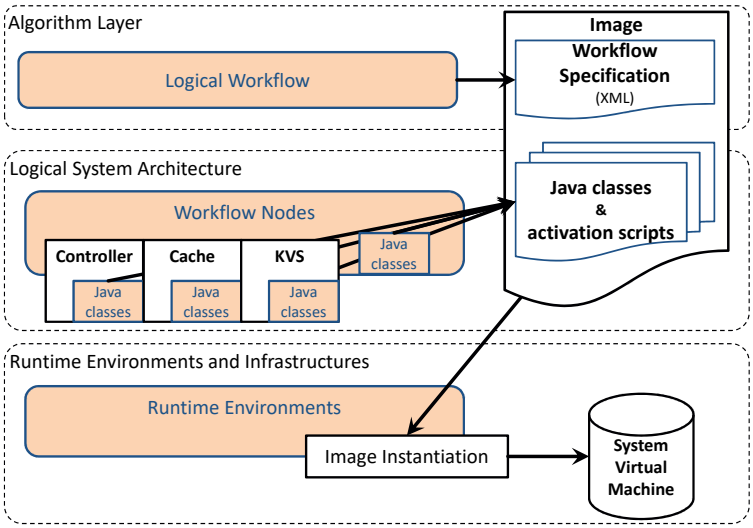


Figure 4.7: Development functionalities and architecture layers.

specific runtime environment (cluster or cloud). During execution the state of the system virtual machines can be inspected by using the available runtime system tools.

Next, a succinct high level description of the above tools is presented in the same sequence as they are used along the corresponding development steps (from step 1 to step 6).

Step 1: Activating the "Tools Launcher"

This is the entry point to the developed tools (Figure 4.8).

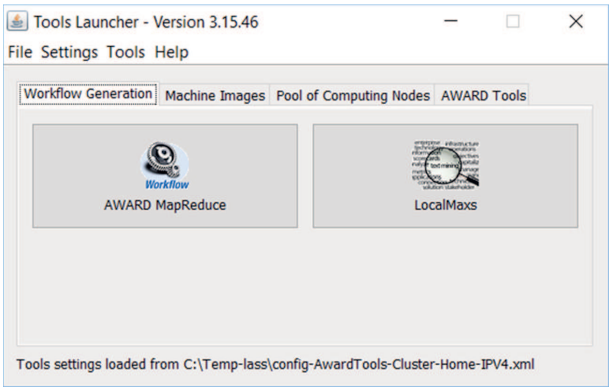


Figure 4.8: Tools launcher for "Workflow Generation".

The menu bar is used to manage the default settings of the tools developed, generate the AWARD workflow configuration files and manage the logging. The default settings for all the tools are defined in a XML configuration file (presented in Appendix B).

The tabbed panel allows the access to the tools developed and is divided in four main areas: i) "Workflow Generation" for each algorithm (e.g., LocalMaxs) settings and workflow generation; ii) "Machine Images" for specification and instantiation of virtual

machines; iii) "Pool of Computing Nodes" for management of the system virtual machines; iv) "AWARD Tools" to access the available AWARD workflow tools. Except for the AWARD tools, which were previously available from the work by Luís Assunção [CA16], all the above tools were developed anew, by the author, for supporting this thesis work.

Step 2: Using the Tool "Workflow Generation"

Using this graphical interface it is possible to activate specific tools for each algorithm, e.g., in order to specify the algorithm parameters and the input data set, and generate the corresponding workflow template. For example, in Figure 4.8 two cases are illustrated: the "AWARD MapReduce" tool supports a flexible implementation of the MapReduce model on top of the AWARD tool, developed in joint work reported in [GAC12; GAC13]; and the "LocalMaxs" tool allows to manage the developed parallel and distributed implementation of this method as proposed in this thesis (Figure 4.9).

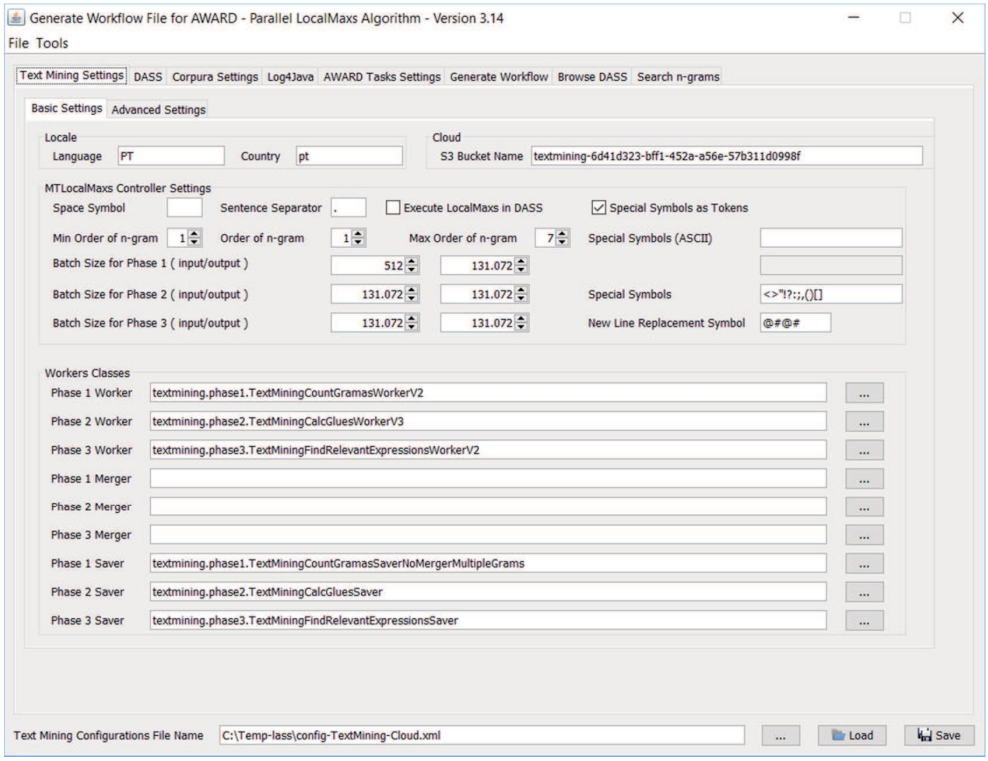


Figure 4.9: LocalMaxs settings.

The "LocalMaxs" tool allows the user to specify the LocalMaxs specific parameters, such as the maximum n-gram size for extracting relevant expressions and the input data set containing the *corpus* to be processed, namely, it allows to specify the list of separator symbols that are used to delimit the *corpus* words. It also allows to specify the Java classes that implement the algorithm functions and to generate the workflow template (Figure 4.10). Configuration parameters related to the architecture implementation are also specified using this tool, namely, the controller parameters such as its internal buffer sizes, the cache configuration, and the configuration of the controller interfacing with

specific input/output devices including the distributed n-gram store servers, through the selection of Java classes from the developed DASS interface. Also it allows to specify KVS server configuration parameters, such as the n-gram tables entry types (unigrams, bigrams, etc.), and the servers system network access location. The tool also allows to obtain statistics regarding the number of distinct n-grams and total number of n-grams.

The LocalMaxs specific parameters are stored in a configuration file specified in XML (an example is presented in Appendix A).

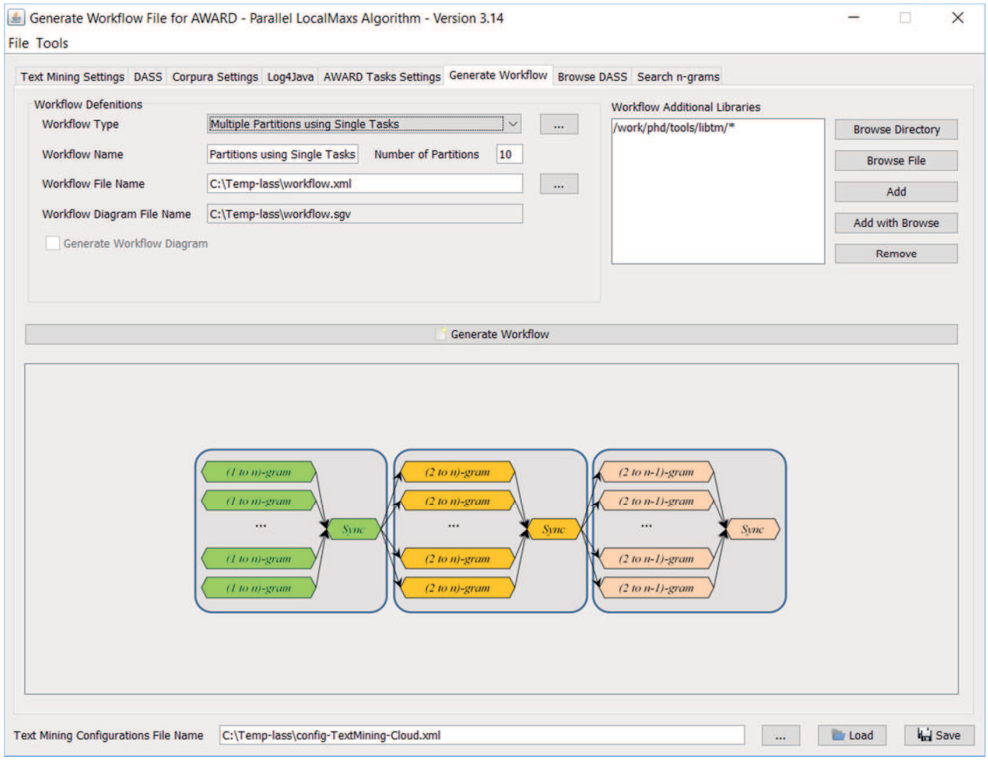


Figure 4.10: Workflow generation.

Step 3: Instantiation of the Virtual Machines

Having generated a workflow template and developed the Java classes for the logical virtual machine image, one must rely on the available runtime environment tools, specific to each cluster or cloud infrastructure, in order to instantiate the necessary virtual machines. This is made using the "Machine Images" tool providing the following functions:

- Management of virtual machine images, which includes the operations for listing, creating and deleting images. Namely, it is possible to configure the properties of each machine instance in terms of number of CPU, disk size and amount of memory;
- Management of virtual machine instances, which includes operations for creating, starting, stopping and deleting machine instances;
- For debugging purposes this tool also allows the establishment of remote connections (using SSH) to the created machine instances.

To access each different computing infrastructure it is necessary to rely on a custom tool. Figure 4.11 shows several tools developed for this purpose.

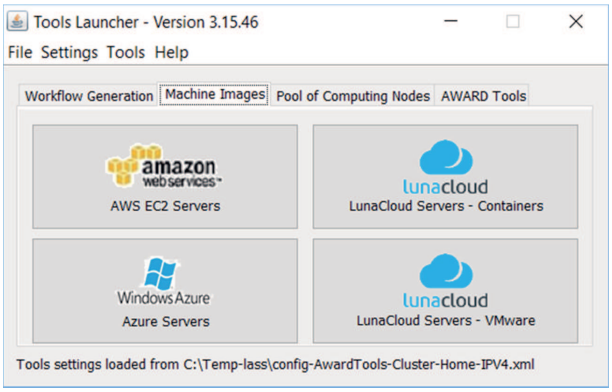
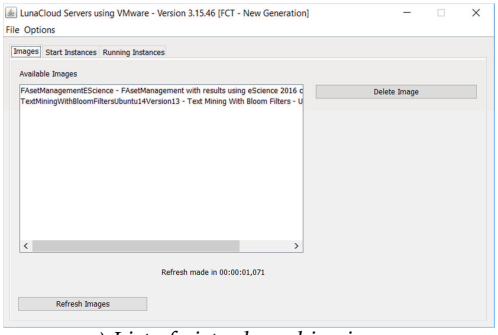
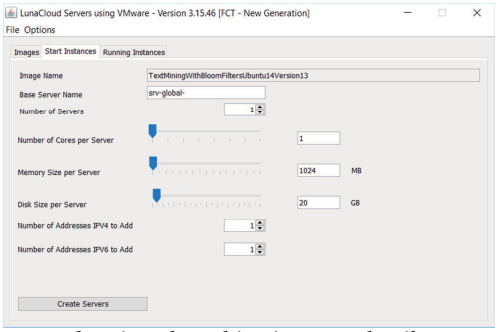


Figure 4.11: Tools launcher for "Machine Images".

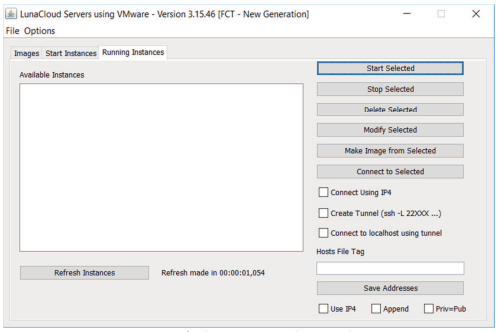
Figure 4.12 shows the tool developed to access the Lunacloud [Lun15] public cloud environment. With this tool the user is able to: i) List and delete virtual machine images — Figure 4.12-a); ii) Create virtual machine images and modify their properties — Figure 4.12-b); Control the execution of virtual machine images instances, namely, start or stop; and Create a new image from an existing one — Figure 4.12-c). A similar tool allows to access the Amazon EC2 environment [Ama16b].



a) *List of virtual machine images*



b) *Virtual machine instance details*



c) *Management of the virtual machine instances*

Figure 4.12: Management of server instances and images using Lunacloud.

The tool also allows to manage the assignment of IP addresses to the system virtual machines created.

Step 4: Mapping the Controllers and KVS Servers into the Created System Virtual Machines

Through the panel "Pool of Computing Nodes" (Figure 4.13) the following tools can be invoked: i) "File Copy" (Figure 4.14), to copy the workflow definition files to remote computing nodes using the Secure Copy Program (SCP); ii) "Remote Commands on Computing Nodes" (Figure 4.15), to launch the remote execution of commands in the created system virtual machines, e.g., to launch the execution of the KVS servers; iii) "Start Workflow Activities" (Figure 4.16), to allocate the controllers into the system virtual machines and to start workflow execution by using the tools provided by the AWARD framework.

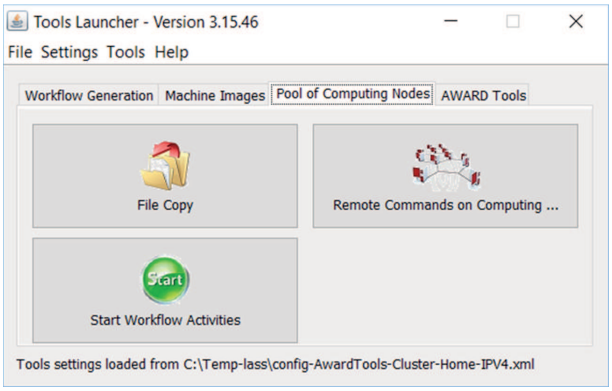


Figure 4.13: Tools launcher for "Pool of Computing Nodes".

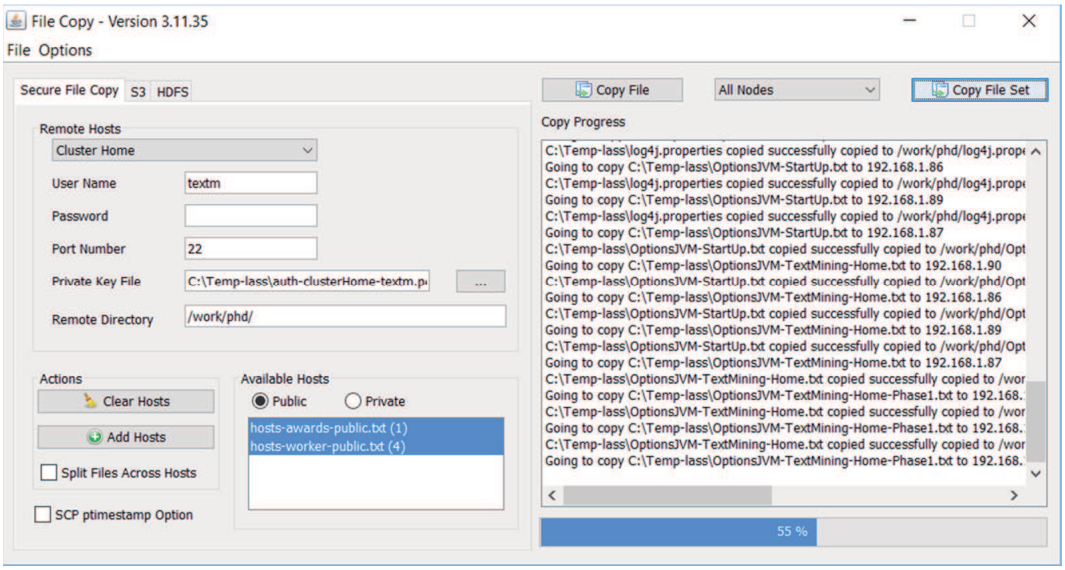


Figure 4.14: File copy.

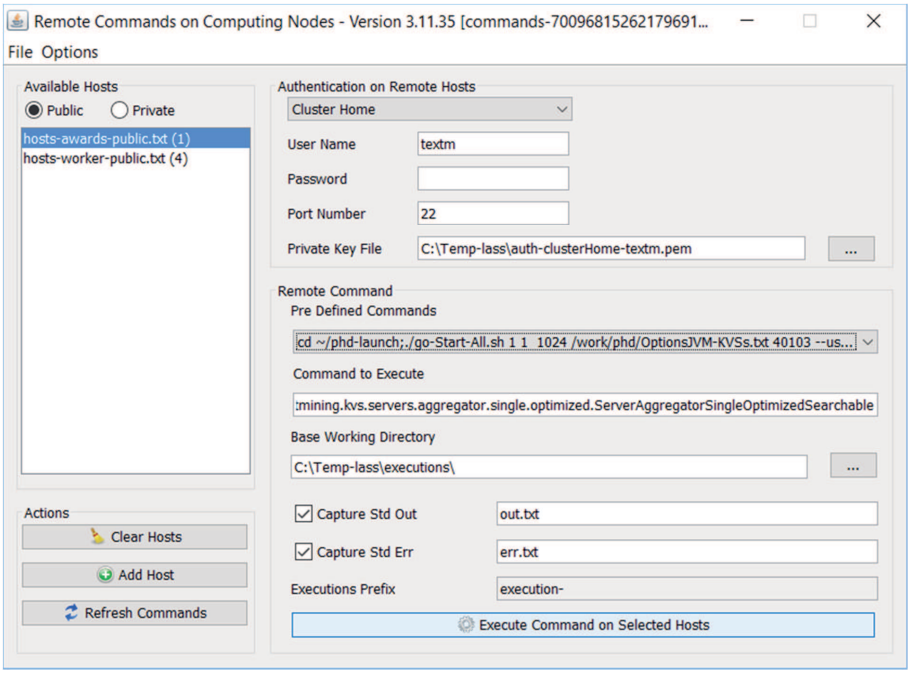


Figure 4.15: Execution of remote commands.

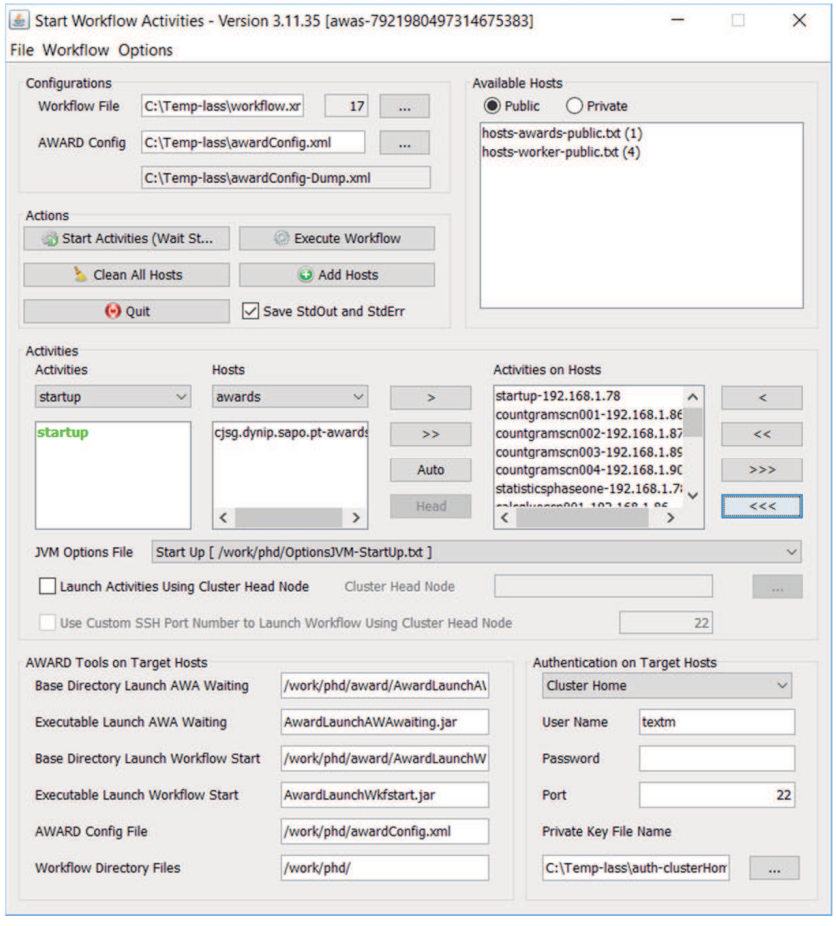


Figure 4.16: Workflow management.

Step 5: Monitoring the Algorithm Execution

The logging events generated during the algorithm execution are stored in a set of servers (supported by the workflow environment), that make the execution trace available for inspection through a set of HyperText Markup Language (HTML) pages. During the execution this logging information can be accessed using the menu bar of the tools launcher.

Step 6: Terminating the Execution

The results of the execution can be saved in external storage (using the DASS interface) according to the configuration parameters. Furthermore, the *n*-gram KVS tables can be saved in a persistent disk repository to be later accessed/reused. Once the execution is terminated, the virtual machine instances can be terminated using the tool "Machine Image".

Tools Implementation and Dependencies. Due to the need of supporting the experimentation in different environments, namely, different cluster and cloud infrastructures, and due to the lack of existing unified and integrated development environments allowing access to such a diversity of heterogeneous systems, in the beginning of this thesis work, it was necessary to take a significant implementation effort towards the development of the above tools. All the developed tools are fully operational and were tested in two different public cloud environments, Amazon and Lunacloud, and enabled to perform all the experimentation reported in this thesis.

The tools can be accessed separately and individually by the developer or their interaction can be encapsulated into high-level scripts made available to higher level users, e.g., by requiring only a specification of the algorithm parameters and input/output data.

All the tools were developed using the Java language. The connection to the remote computing nodes relies on SSH connections. The tools use some external libraries, also developed in Java (open source available in Jar files), namely:

- AWARD framework tools [AGC12];
- Amazon Web Services (Amazon AWS) libraries to manage virtual machines created in the Amazon Cloud [Ama16a];
- Apache Commons CLI used to parse arguments passed to the controller [The16b];
- Apache Logging Services to support the Log4Java [The16d];
- Hadoop libraries to support the access to the HDFS file system [The16c];
- Java Secure Channel to support the SSH and SCP protocols in Java [JSc16].

The architecture and the tools developed run under different operating systems, such as Windows or Linux (with support for a SSH server and a JVM with version 1.7.X), in environments ranging from standalone computers, private clusters (supported in Slax/Linux

operating system) and public cloud environments, namely Amazon EC2 ([Ama16b]) and Lunacloud ([Lun15]).

4.4 Chapter Summary

In this chapter a description of the developed architecture that supports the parallel and distributed execution of statistical-based extraction methods was presented. The architecture was implemented in Java and includes the mechanisms for supporting the basic operations of the extraction methods, the storage of the n-gram data, and the caching of n-grams, using multiple machines in a cluster or cloud environment. A set of associated tools was also implemented to support the development and execution of multiphase extraction algorithms.

The developed prototype runs in cluster and public cloud environments (Amazon and Lunacloud). Namely, it was used to support the parallel and distributed implementation of LocalMaxs, as discussed in the following chapters.

Chapter 5

A Global Method for Statistical Extraction based on LocalMaxs

This chapter presents a Global method for the parallel and distributed implementation of LocalMaxs.

In this chapter the Global method is presented. The name Global means that the global frequency of occurrences of the *n*-grams in the entire *corpus* is taken into account in the relevance evaluation. The rationale of the Global method is presented in section 5.1. In section 5.2 we model the LocalMaxs Global method as a workflow. The implementation of the Global method on top of the distributed architecture is presented in section 5.3. The influence of the distribution of *n*-grams across the distributed in-memory server tables is discussed in section 5.4. The time complexity of the Global method is discussed in section 5.5. This chapter ends with a summary (section 5.6).

5.1 Rationale of the Global Method

For a given *corpus* C and n-gram size n the method calculates the set of relevant expressions according to the definition of LocalMaxs. Depending on the number of available machines and the maximum memory per machine, the implementation takes advantage of the subdivision of work and of the distribution of n-gram data to enable the method evaluation for large *corpora*.

Figure 5.1 shows the main tasks and the global data structures used by the Global method. In the first phase of the Global method the *corpus* is decomposed into equal-size partitions. Each machine counts the number of occurrences of n-grams in its local partition. The n-gram frequency counts are stored in the in-memory distributed store, as an n-gram table, with an entry for each distinct n-gram. In the second phase the

glue (cohesion) evaluation of all the distinct n-grams is performed in parallel by all the machines, each working upon its own table partition. Then in the third phase the machines perform the parallel evaluation of the relevance of the n-grams in their local table partitions.

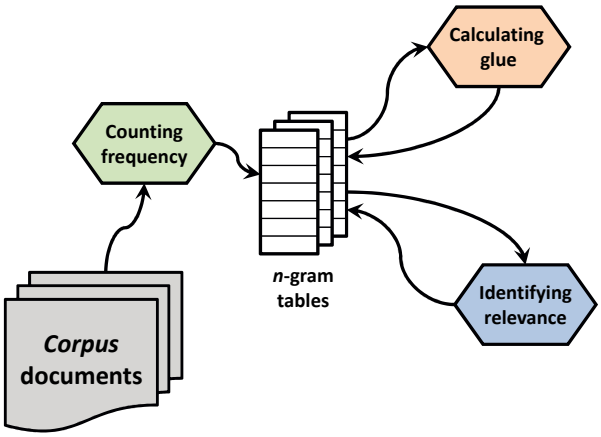


Figure 5.1: Global method tasks and structure.

In order to be able to handle large size *corpora* the Global method follows a data partitioning approach. The advantages of the Global method are:

- **Precision and recall.** The Global method is faithful to the LocalMaxs definition, i.e., it achieves the same precision and recall;
- **Phase decoupling.** Because each phase is decoupled from the others it is possible to implement other n-gram statistical methods besides LocalMaxs and to use different strategies in each phase. For example it is possible to implement phase one as a MapReduce job, use different numbers of controllers in each phase, or different optimizations such as a cache system to reduce the communications overheads;
- ***n*-gram data supported in tables.** The n-gram tables filled during the execution of the Global method work as a repository that can be used by other algorithms. Because the n-gram tables act as an external repository it is possible to execute initially phase one of the Global method to obtain the n-gram frequency counts, followed by multiple executions of phase two and three in order to evaluate the precision and recall of different glue/relevance metrics.

A consequence of the in-memory distributed store is that accessing the n-gram distributed data will introduce communication overheads, which are further increased due to the exhaustive nature of the method. This led to the proposal of an n-gram cache whose design was guided by the analysis of the properties of the n-gram distribution in natural language *corpora* presented in chapter 3.

As the Global method is based on the generation of a set of distributed n-gram tables, which are then processed by phases two and three, this method originates a distribution

of the n-gram data across the tables, which breaks the n-gram spatial locality in the input *corpus* text. Thus, in this implementation the cache system is only able to benefit from the temporal locality of the n-gram occurrences.

5.2 Logical Workflow

The template workflow presented in Figure 5.2 represents the logical dependencies imposed by the LocalMaxs method when used to extract relevant expressions of sizes from 2 up to 5. In general for a given maximum n-gram size n, as input parameter, the set of relevant expressions from 2 to n (denoted $re_{2...n}$) are identified in the *corpus*: this requires counting all n-gram occurrences from size 1 up to $(n+1)$, and calculating the glue for all n-grams from size 2 up to $(n+1)$ (denoted $g_{2...(n+1)}$).

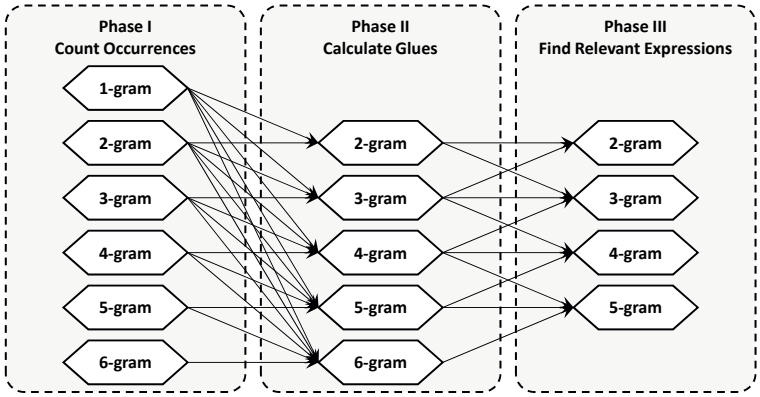


Figure 5.2: Global method logical workflow for $re_{2...5}$.

Each workflow node executes the algorithm task to calculate the associated function for phase i (i from 1 up to 3) and n-gram of size n, according to the dependencies shown.

The workflow in Figure 5.2 allows execution overlap between nodes in different phases, thus possibly leading to the completion of the algorithm execution earlier for the smaller n-gram sizes, but this requires a dynamic workflow scheduling strategy to keep the load balanced.

The above logical workflow allows several alternatives concerning the mapping of the individual nodes to concrete implementations, and also concerning different ways of combining the algorithm tasks within each workflow node. For example, this combination can be useful to achieve an adequate granularity of the computations assigned to each workflow node.

In this logical description, each workflow node is in charge of processing the entire data set that composes the *corpus*. However, it is possible to explore a data parallel approach such that each node is in charge of a separate partition of the input data set. For example, Figure 5.3 shows an alternative workflow decomposition of the algorithm tasks that enables the parallel handling of multiple partitions of the input data set, such that each node is responsible for a separate data partition.

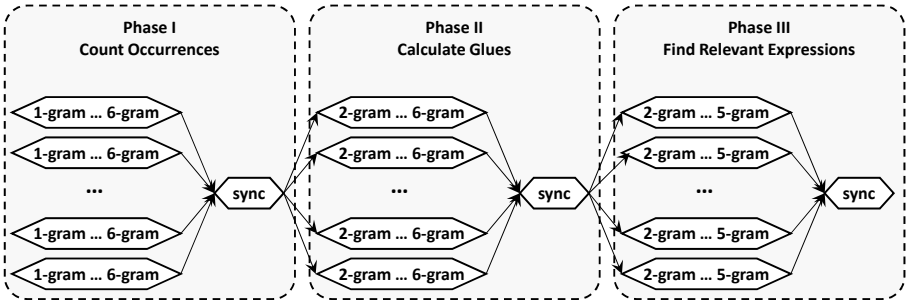


Figure 5.3: An alternative workflow decomposition for the Global method.

Each node with a label "$n_1-gram\cdots n_2-gram$" represents a task to sequentially calculate the associated algorithm function for each n-gram size in the range of values from n_1 to n_2. For instance, in phase one each node is responsible for consecutively counting the n-gram occurrences, starting with unigrams and ending with hexagrams.

The workflow in Figure 5.3 enforces a synchronization barrier at the end of each phase (denoted by the nodes labeled "sync"). This solution represents a decomposition of the algorithm in tasks that leads to a well-balanced application load among the workflow nodes, thus when mapped to an homogeneous cluster of machines (physical or virtual), all the nodes within each phase tend to exhibit similar completion times. This was confirmed by the conducted experiments (Figure 6.34 on page 178).

Once a workflow decomposition is defined it is necessary to map it to a concrete implementation by assigning the workflow nodes to the components of an underlying support architecture, as presented in the following.

5.3 Implementation of the Global Method

In this section we describe the implementation of the Global method. Sections 5.3.1 to 5.3.3 describe the implementation of each of the three phases.

5.3.1 Implementing Phase One

In the Global method, phase one corresponds to the n-gram counting problem that is commonly used by n-gram models and for which there is a wide range of implementations [Lin+10]. The *corpus* is divided into equal-size input data partitions, each separately processed by different controllers. Phase one generates the distinct n-gram tables, one for each n-gram size, containing the total counts of all the n-gram occurrences in the *corpus*. These tables are stored in a distributed collection of KVS servers, being partitioned equally by hashing. Each server performs a global aggregation of the local counts received for its corresponding n-grams.

For phase one, the input data set consists of the *corpus* to be analyzed. The *corpus*, composed of a single document or a set of documents, can be located in different media such as: a local file system, a distributed file system or a cloud storage system such as the

Amazon Simple Storage Service (Amazon S3) [Ama16c]. In the context of phase one the following definitions are considered:

Definition 5.1: Sentence

A sentence is a sequence of one or more unigrams separated by white spaces and ending with a dot symbol (.).

Definition 5.2: Chunk for Phase One

A chunk is a sequence of one or more sentences.

The number of sentences (chunk size) is specified in a configuration file. The controller uses a chunk extractor to read sentences of text from the input *corpus*. Each chunk is available as a Java `Record`, that is processed by one worker thread. The worker thread counts the occurrences of the n-grams contained in chunks and returns a list of objects of type `Record`, corresponding to all the distinct n-grams contained in the chunks and the corresponding frequencies. The lists produced by the worker are collected by a saver thread that sends the n-gram data (encapsulated in objects of type `Record`) to the KVS servers. The distribution of the n-gram data records to the KVS servers is made using an hash function. In order to minimize the latency due to the underlying network infrastructure, the n-gram records are sent to the KVS servers in buffers whose size (B) is specified in a configuration file.

5.3.2 Implementing Phase Two

In phase two, the input data set to each controller in a machine consists of the n-gram table partitions kept by the local KVS server in the same machine that were produced during phase one. In phase two the following definition is considered:

Definition 5.3: Chunk for Phase Two

A chunk is a list of `Record`, in the form $\langle key, frequency \rangle$, where key is the n-gram identifier and $frequency$ is the frequency of the n-gram in the *corpus*.

The chunks have equal sizes, expressed in number of records, denoted as Input Buffer Size (IBS) which is configured according to the desired computation-to-communication granularity (as discussed in section 5.5.5.1, on page 117). For each n-gram size n the glue calculation consists of the following steps:

1. Reading the n-gram chunks from the local n-gram tables for n-grams of size n;
2. Calculating an n-gram glue requires knowing the frequency of all the included leftmost and rightmost sub n-grams of sizes from 1 to $(n-1)$ according to LocalMaxs Definition 2.1, on page 15. Once those sub n-grams are identified, if they are missing from the local n-gram cache system, then they are fetched from the KVS server tables;

3. Performing the glue calculation of the n-grams in a chunk;

4. Putting the values of the n-gram glue in an output queue so that they are sent to the local KVS server. Phase two output is the updated n-gram table with the calculated glue for each distinct n-gram, with an entry in the form $\langle key, \langle frequency, glue \rangle \rangle$, where key is the n-gram identifier and $frequency$ and $glue$ are respectively the frequency of the n-gram in the *corpus* and its glue value.

Both the input and the output operations (in steps 1 and 4) are overlapped with the glue calculation (in steps 2 and 3).

5.3.3 Implementing Phase Three

Phase three input consists of the local n-gram tables that result from the n-gram glue calculation. The sequence of steps taken in phase three is similar to phase two:

1. Reading the n-gram chunks from the local n-gram tables for n-grams of each size $n \geq 2$ considered for relevance evaluation. The main thread only fetches the n-gram records whose frequency is greater than 1, because singleton n-grams are not considered as candidates for relevant expressions;

2. Identifying the sets Ω_{n-1} and Ω_{n+1} and the corresponding maximum values according to LocalMaxs Definition 2.1, on page 15. Identifying the relevance of an n-gram requires the glue of the n-gram of size n as well as the glues of the two leftmost and rightmost sub $(n-1)$-grams enclosed in the n-gram and the maximum glue of the $(n+1)$-grams that include the n-gram;

3. Applying the relevant expression criterion (Definition 2.2, on page 15);

4. Updating the local n-gram table with a relevance boolean flag for the n-grams that were considered relevant expressions.

At the end of phase three the local KVS server n-gram tables used to store the n-gram frequencies and glues are updated with the n-gram relevances. Each table entry is in the form $\langle key, \langle frequency, glue, relevance \rangle \rangle$, where key is the n-gram identifier and $frequency$, $glue$ and $relevance$ are respectively the values of the frequency of the n-gram in the *corpus*, its glue and relevance.

5.4 Work Partitioning and Data Partitioning

In this section we discuss the work partitioning approach followed (section 5.4.1), the influence of the hash methods used for the data partitioning (section 5.4.2), and an analysis of the distribution of the distinct n-grams among the KVS server tables used in phase two and three (section 87).

5.4.1 Work Partitioning

The architecture is configured as a collection of homogeneous logical virtual machines, each one supported by an identical system configuration in terms of the number of virtual CPU, RAM size, local storage and network bandwidth specification. The logical virtual machines also have identical internal component configurations, with one controller, one KVS server, and equal size caches (Figure 5.4).

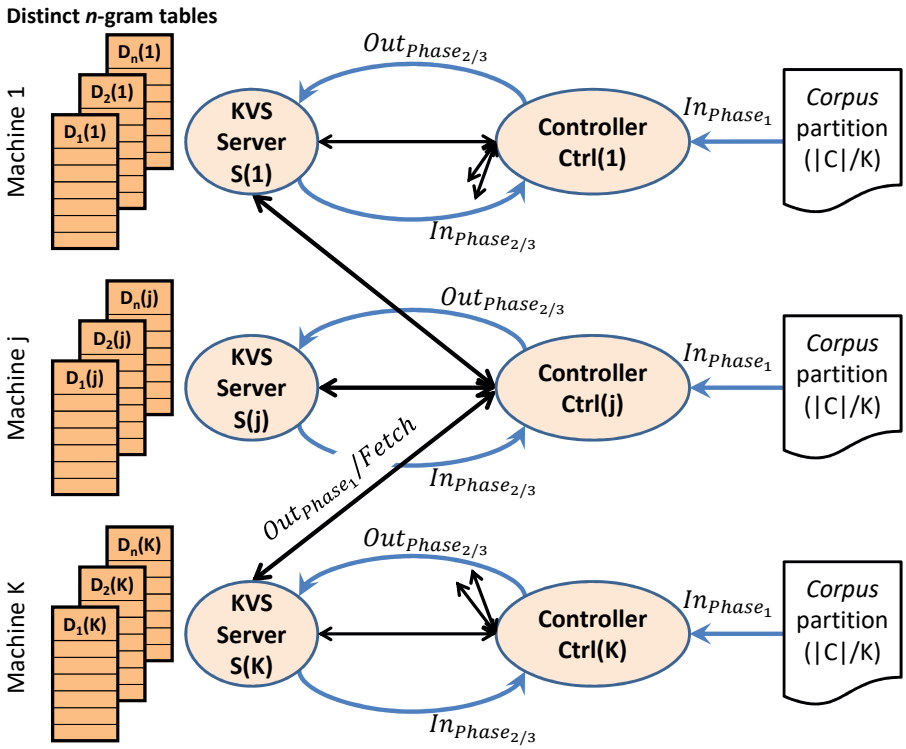


Figure 5.4: Work table and data partitioning within the architecture.

The Figure 5.4 illustrates how each controller ($Ctrl(j)$ in machine $1 \leq j \leq K$) performs the input in phase one (In_{Phase_1}) from its assigned *corpus* partition $C(j)$ (of size $|C|/K$); how it outputs the n-gram output data (Out_{Phase_1}) to the KVS servers; and how it performs the input and output in phases two (In_{Phase_2}, Out_{Phase_2}) and three (In_{Phase_3}, Out_{Phase_3}) from/to the local KVS server ($S(j)$) n-gram tables.

Each local table of distinct n-grams of size i in machine j contains the distinct n-grams of that size ($D_i(j)$) that were assigned to server $S(j)$. For simplicity we denote this table by the same label $D_i(j)$. The set of n-gram tables within the server $S(j)$ in machine j is composed of: $\{D_1(j), D_2(j), \cdots, D_i(j), \cdots, D_n(j)\}$. The set of distinct n-grams of size i in the *corpus* (D_i) is contained in the union of the tables $D_i(j)$ in all servers.

The figure also illustrates how in phase two and three each controller performs the fetching ($Fetch$) of the needed sub n-gram data from the corresponding distributed server tables ($S(1)$ to $S(K)$).

As the *corpus* data set is static and unchanged during the algorithm execution, a static work distribution leads to a good load balancing in all three phases because, on one hand,

the local n-gram tables for each n-gram size assigned to the servers are of approximately equal sizes, i.e., $|D_i(j)| \approx (|D_i|/K)$, for $1 \leq i \leq n$ and $1 \leq j \leq K$, and on the other hand, in any of the three phases the input partitions assigned to the controllers are of equal sizes.

5.4.2 Hash-based *n*-gram Table Partitioning

In order to be able to process large *corpora*, the n-gram data are partitioned among the servers according to a set of disjoint tables. These tables are generated during phase one as the controller in each machine obtains the frequency counts in its *corpus* partition and sends them to the KVS servers.

A decision was taken regarding not to use replication of the n-gram tables. There is a set of distributed tables for each n-gram size, thus each KVS server contains a subset of the n-gram tables, encompassing the complete range of n-gram sizes handled, from unigrams up to the highest n-gram size required by the problem parameters (*corpus* size and n-gram sizes). The total number of required servers will depend on the scale of the problem (*corpus* size and n-gram sizes) and the available machine configurations.

The distribution of n-grams among different servers is made using hash functions, applied to the entire n-gram or part of it. The distribution of n-grams across the different KVS servers should ensure a well balanced load among all the servers that depends on the patterns of the access requests made by the controllers determined by the statistical distribution of the frequencies of occurrences of the n-grams in the *corpus*. Due to the skew of that statistical distribution, certain n-grams will be more requested than others.

The distribution of distinct n-grams using hash functions applied to individual n-grams will break the n-gram spatial locality of the n-grams in the original texts. For example let us consider the *pangram*[1] "The quick brown fox jumps over a lazy dog" and four different ways of applying the hash[2] function to an n-gram $(w_1 \cdots w_n)$:

(h_0) to the entire n-gram: $(w_1 \cdots w_n)$

(h_1) to the leftmost *(n-1)*-gram: $\left(w_1 \cdots w_{(n-1)}\right)$

(h_2) to the leftmost and rightmost unigrams: w_1 and w_n

(h_3) to the leftmost unigram: w_1

If for example we consider four KVS servers, identified by integers 0, 1, 2 and 3, then the distribution of distinct of n-grams, with sizes from unigrams up to tetragrams, is as presented in Tables 5.1 to 5.4.

[1] *Pangram* a term from the Greek for all letters (***pan*** = all + ***grámma*** = letter), is a series of words which contains all the letters of the alphabet

[2] The hash function considered is the one provided by the class `String` available within Java

Table 5.1: Distribution of unigrams across four KVS servers.

unigrams	h_0	h_1	h_2	h_3
The	1	1	1	1
quick	1	1	1	1
brown	2	2	2	2
fox	3	3	3	3
jumps	1	1	1	1
over	0	0	0	0
a	1	1	1	1
lazy	0	0	0	0
dog	0	0	0	0

Table 5.2: Distribution of bigrams across four KVS servers.

bigrams	h_0	h_1	h_2	h_3
The quick	2	1	2	1
quick brown	1	1	1	1
brown fox	3	2	3	2
fox jumps	0	3	0	3
jumps over	3	1	3	1
over a	3	0	3	0
a lazy	1	1	1	1
lazy dog	0	0	0	0

Table 5.3: Distribution of trigrams across four KVS servers.

trigrams	h_0	h_1	h_2	h_3
The quick brown	0	2	1	1
quick brown fox	3	1	1	1
brown fox jumps	2	3	3	2
fox jumps over	0	0	1	3
jumps over a	0	3	2	1
over a lazy	3	3	0	0
a lazy dog	2	1	0	1

Table 5.4: Distribution of tetragrams across four KVS servers.

tetragrams	h_0	h_1	h_2	h_3
The quick brown fox	1	0	0	1
quick brown fox jumps	2	2	2	2
brown fox jumps over	3	0	0	3
fox jumps over a	0	0	3	1
jumps over a lazy	1	3	0	0
over a lazy dog	3	2	2	1

As an example assume that we want to calculate the glues of the n-grams ***The quick***, ***The quick brown*** and ***The quick brown fox***. Besides accessing the n-grams mentioned it

is also necessary to access the leftmost and rightmost sub n-grams contained within them. Using Tables 5.2 to 5.4 we can determine the servers containing the n-grams, when using the four different hash methods. Tables 5.5 and 5.6 show the needed sub n-grams and their locations (assuming the four KVS servers) when using the hash method h_1 (Table 5.5) and hash method h_3 (Table 5.6). In both tables the notation (n-gram $\rightarrow$ server identifier) denotes the assignment of the given n-gram to the given server.

From the analysis of tables Tables 5.5 and 5.6 this example suggests that hash method h_3 would seem a good choice since it is the one that places most of the n-grams and the sub n-grams needed during the glue calculation in the same KVS server.

Table 5.5: Sub n-grams and corresponding KVS server identifier using hash method h_1.

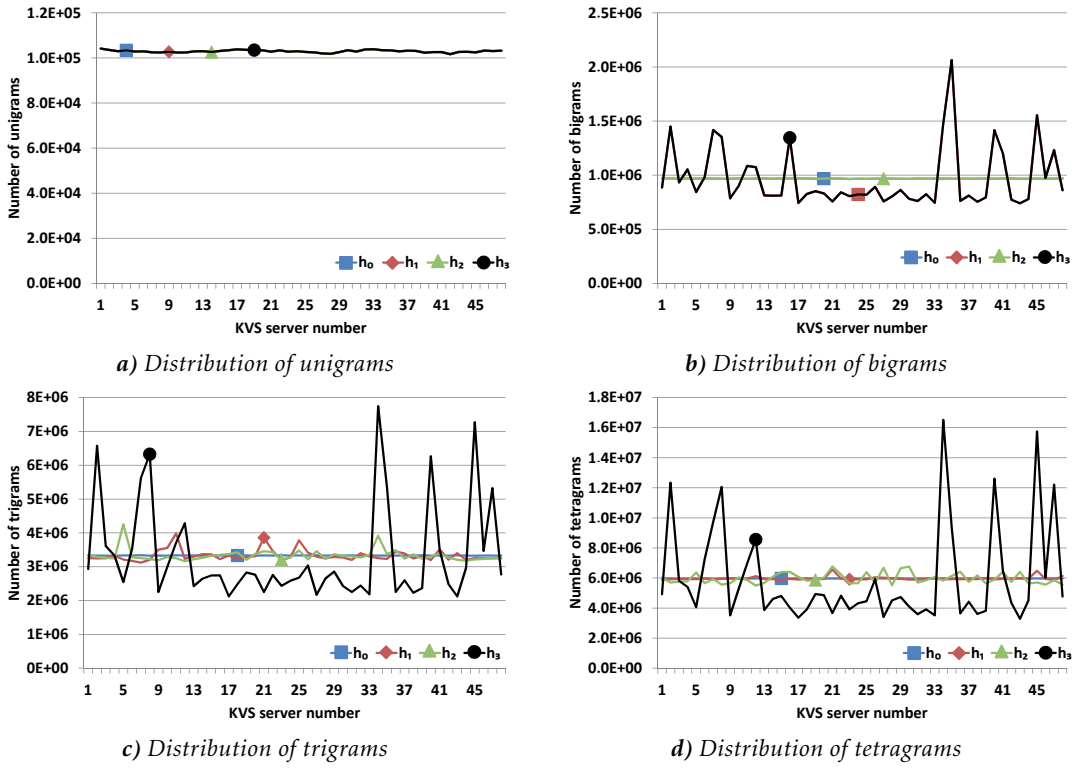
***n*-gram**	**unigrams**	**bigrams**	**trigrams**
The quick → 1	The → 1 quick→ 1		
The quick brown→ 2	The → 1 brown→ 2	The quick → 1 quick brown → 1	
The quick brown fox→ 0	The → 1 fox → 3	The quick → 1 brown fox → 2	The quick brown → 2 quick brown fox → 1

Table 5.6: Sub n-grams and corresponding KVS server identifier using hash method h_3.

***n*-gram**	**unigrams**	**bigrams**	**trigrams**
The quick→ 1	The → 1 quick→ 1		
The quick brown→ 1	The → 1 brown→ 2	The quick → 1 quick brown → 1	
The quick brown fox→ 1	The → 1 fox → 3	The quick → 1 brown fox → 2	The quick brown → 1 quick brown fox → 1

However, the n-gram distribution across all the servers should also ensure a uniform partitioning of the distinct n-grams among all the servers. Figure 5.5 shows the distribution of unigrams, bigrams, trigrams and tetragrams using the above mentioned hash methods for a *corpus* of 466 Mw when the number of KVS servers is 48.

In the case of unigrams all the above hash methods are always applied to the same input n-gram, thus the distribution among the KVS servers is the same. For higher n-gram sizes, such as bigrams, trigrams and tetragrams, the methods h_0, h_1 and h_2 tend to present a more balanced n-gram distribution among all the KVS servers than the method h_3. Indeed, when using method h_3 the most frequently occurring unigrams, e.g., "the" or "and" in the case of the English language or "de" for the Portuguese language, due to their popularity, are cited as leftmost unigrams within a larger number of higher size n-grams ($n \geq 2$), which according to the h_3 method are assigned to the same KVS server as the corresponding leftmost unigram. This originates that a small number of servers will contain a much higher number of distinct n-grams than other servers.



a) Distribution of unigrams

b) Distribution of bigrams

c) Distribution of trigrams

d) Distribution of tetragrams

Figure 5.5: Distribution of distinct n-grams across 48 KVS servers using the hash methods $h0$, $h1$, $h2$ and $h3$ for a *corpus* of 466 Mw. The Y axis represents the number of distinct n-grams and the X axis represents the KVS server number.

In all the experimentation conducted with the execution of the parallel and distributed implementation of the LocalMaxs Global method, the hash method h_1 was used.

5.4.3 Influence of the Distribution of the Distinct n-gram Tables upon Phase Two

Handling large *corpus* sizes is the major requirement that must be met by the implementation of the Global method. Thus, its behavior as a function of the number of machines becomes a decisive aspect of the design and analysis of a parallel and distributed implementation of this method.

As discussed above, the local n-gram server tables for keeping the data of the distinct n-grams of each size i ($D_i(j)$) are disjoint and represent equal size partitions of the total number of distinct n-grams of the corresponding size i in the *corpus*, thus $|D_i(j)|$ is proportional to $1/K$ being approximately equal to $|D_i|/K$.

However, in phase two of LocalMaxs Global method, the calculation of the glue of each distinct n-gram of size n requires getting the values of the frequencies of its sub n-grams of sizes from 1 up to $(n-1)$ (Definition 2.1, on page 15) schematically illustrated in Figure 5.6.

The sets of sub n-grams that are needed by the glue calculations of all the distinct

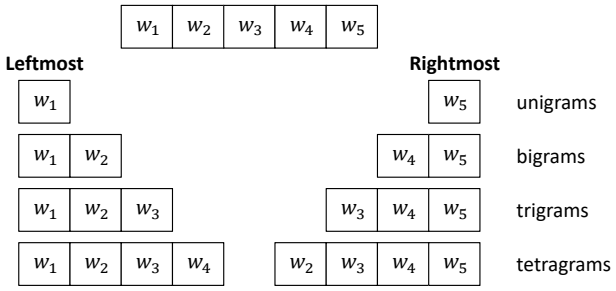


Figure 5.6: Sub n-grams needed for glue calculation.

n-grams in the local tables, $D_2(j)$ to $D_n(j)$, in each machine are not disjoint, i.e., there are repeated occurrences in those sub n-grams. As the access to the frequencies of these sub n-grams, which are kept in the distributed server tables, implies communication overheads, a local cache in each controller machine allows to reduce the number of accesses down to the number of distinct sub n-grams that are needed.

The set of all the occurrences of the sub n-grams needed by the glue calculation g_n of all the distinct n-grams of size n in the table $D_n(j)$ in machine j, where $2 \leq n \leq n_{max}$ and n_{max} is the maximum n-gram size considered, is denoted as $all_{glue_{g_n}Ref}(j)$.

Among those occurrences there is a set of distinct sub n-grams from sizes 1 up to $(n-1)$, which corresponds to the set $D_{1_{in}D_n} \cup D_{2_{in}D_n} \cup \cdots \cup D_{(n-1)_{in}D_n}$. Each set $D_{i_{in}D_n}$, with $1 \leq i \leq (n-1)$, represents the set of distinct leftmost and rightmost sub n-grams of size i, occurring within the n-grams of size n in the D_n table.

For simplicity, in the following we denote "the leftmost and rightmost sub n-grams of size i" just as "the sub n-grams of size i".

When the glue calculations $g_{2\cdots n}$ of the n-grams in tables $D_2(j)$ to $D_n(j)$ are considered, the set of all the accesses to sub n-grams is denoted as $all_{glue_{g_{2\cdots n}}Ref}(j)$ and the set of distinct n-grams within $all_{glue_{g_{2\cdots n}}Ref}(j)$ is obtained as follows: $D_{1_{in}D_2\cdots D_n}(j) \cup D_{2_{in}D_3\cdots D_n}(j) \cup \cdots \cup D_{(n-1)_{in}D_n}(j)$, where each set $D_{i_{in}D_{(i+1)}\cdots D_n}(j)$ represents the set of distinct sub n-grams of size i $(1 \leq i \leq (n-1))$ within the n-grams in the tables $D_{(i+1)}(j)$ up to $D_n(j)$ in each machine. When using a single machine, the set $D_{i_{in}D_n}$ for each fixed i from 1 up to $(n-1)$, is equal to the set of distinct n-grams of size i (denoted as D_i) in the *corpus*.

When considering multiple machines, each machine handles the distinct sub n-gram references in its local n-gram tables ($D_2(j)$, $D_3(j)$, etc.). Unlike the local n-gram tables, which are equal size partitions, the same behavior was not observed concerning the set of distinct sub n-grams of size i ($D_{i_{in}D_n}(j)$) in each local partition table $D_n(j)$, with $i+1 \leq n \leq n_{max}$, where n_{max} is the maximum n-gram size considered for glue calculation.

In fact, for each pair of values of i and n, when K>1, we observe that $\left|D_{i_{in}D_n}(j)\right| > \left|\frac{D_i}{K}\right|$, where $|D_i|$ is the total number of distinct sub n-grams of size i in the *corpus*. For example, when K=2 each machine gets a local table of distinct bigrams ($D_2(j)$), whose size is half of the total distinct number of bigrams ($|D_2|$), while each set $D_{1_{in}D_2}(j)$ in each machine j: 1, 2, is larger than half of the total number of distinct unigrams in the *corpus* ($|D_1|$).

As a result of this behavior, the overheads in each machine due to the communication to fetch the needed sub n-grams do not decrease proportionally to $1/K$.

In order to quantify the impact of this behavior when using K machines, for example in the case of distinct subunigrams, for the glue calculation of bigrams, first we would need to estimate in how many distinct bigrams in the *corpus* (D_2 table) each distinct unigram occurs. Then, we would need to estimate how the partitioning of the distinct bigram table D_2 among the multiple machines affects the distribution of the included subunigrams. Namely, the most frequently occurring unigrams in the *corpus*, which tend to appear in many distinct bigrams, have a greater opportunity of spreading (get repeated) in multiple machines.

In this work an empirical approach was followed to investigate this issue. In the following we illustrate the observed behavior for selected cases aiming at identifying the global trends of the number of distinct sub n-grams per machine as a function of the number of machines (K), for all the glue calculations.

The distribution of the number of distinct sub n-grams per machine ($\left|D_{i_{in}D_n}(j)\right|$) as a function of K, for different values of i and n is shown in Figure 5.7 for the case $i = 1$ when n goes from 2 up to 4: this corresponds to the number of distinct subunigrams ($\left|D_{1_{in}D_2}(j)\right|$) observed per machine in its local bigram table (needed by glue calculation g_2), and to the distinct subunigrams ($\left|D_{1_{in}D_3}(j)\right|$) in its local trigrams table (needed by glue calculation g_3), and to the distinct subunigrams ($\left|D_{1_{in}D_4}(j)\right|$) in its local tetragrams table (needed by glue calculation g_4), when K varies from 1 to 48, for a fixed *corpus* size of 466 Mw.

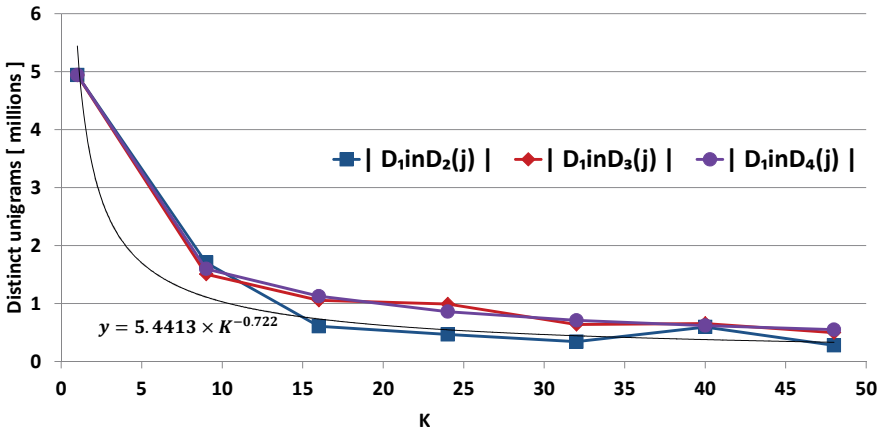


Figure 5.7: Distinct subunigrams in $D_2(j)$, $D_3(j)$ and $D_4(j)$ per machine for a *corpus* size of 466 Mw. The Y axis represents the number of subunigrams in millions and the X axis represents the number of machines.

The data presented were obtained by executing the phase one of the Global method in order to generate the distinct bigram, trigram and tetragram tables, using the hash method h_1, and then instrumenting each KVS server to obtain the number of distinct sub n-grams of the sets $D_{1_{in}D_2}(j)$, $D_{1_{in}D_3}(j)$ and $D_{1_{in}D_4}(j)$ in each machine. The above procedure was repeated for each value in the above mentioned range of K values. For

each value of K, the average of the cardinalities of the above sets over the K machines was calculated and is shown in Figure 5.7. The trend of the average cardinality of $D_{1_{in}D_2}(j)$ was derived through fitting of the corresponding curve, as shown in the figure.

For the *corpus* size of 466 Mw presented in Figure 5.7 the trend of the distinct subunigrams in the bigram table as a function of the number of machines (K) is approximated by the following expression:

$$\left|D_{1_{in}D_2(j)}\right| = 5.4413 \times K^{-0.722} \tag{5.1}$$

Similar behaviors were observed for the distributions of distinct subbigrams, subtrigrams, etc, that is $D_{i_{in}D_n}(j)$ for values of i going from 2 up to 5 for the considered range of *corpus* sizes (2 up to 998 Mw). For example Figure 5.8 shows the number of distinct sub n-grams per machine ($\left|D_{i_{in}D_n}(j)\right|$) for a *corpus* of 1 Mw when we vary the number of machines from 1 to 48, with $1 \leq i \leq 5$ and $i < n \leq 6$. For example, in the Figure 5.8 the curve Average $D_{1_{in}D_n}(j)$ was obtained by the average of the values of $\left|D_{1_{in}D_n}(j)\right|$ when n ranges from 2 up to 6. The curve Average $D_{2_{in}D_n}(j)$ was obtained by the average of the values of $\left|D_{2_{in}D_n}(j)\right|$ when n ranges from 3 up to 6, and so on for the other curves.

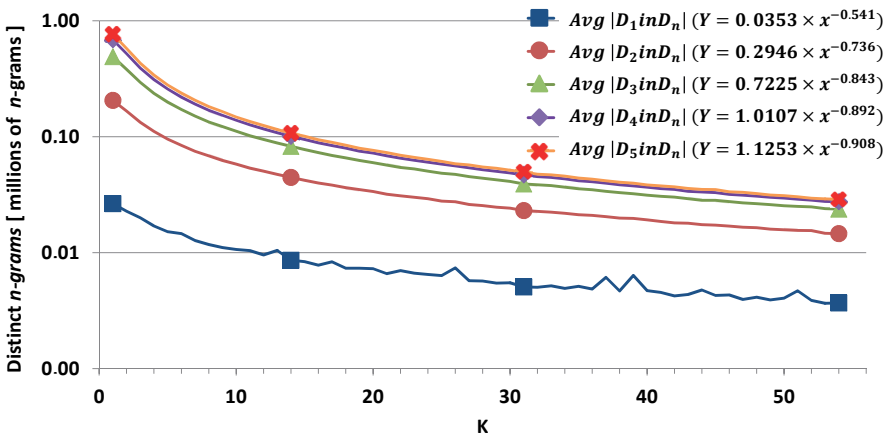


Figure 5.8: Average distinct numbers of sub n-grams in $D_n(j)$ per machine for a *corpus* size of 1 Mw. The Y axis represents the total number of distinct sub n-grams in millions of n-grams and the X axis represents the number of machines.

As shown in Figure 5.8 the variation of the distinct sub n-grams $\left|D_{i_{in}D_n}(j)\right|$ follows a power law of the form K^{-b}, with $0 < b < 1$, while the size of the table D_n is proportional to $1/K$.

Figures 5.9 to 5.11 show the average percentage ($\left|D_{i_{in}D_n}\right|/|D_i|$) of distinct sub n-grams of size i ($D_{i_{in}D_n}$) in D_n per machine relative to the total number of distinct n-grams of the same size in the *corpus* ($|D_i|$), for different *corpus* sizes, ranging from 13 to 511 Mw, when the number of machines ranges from 1 to 54.

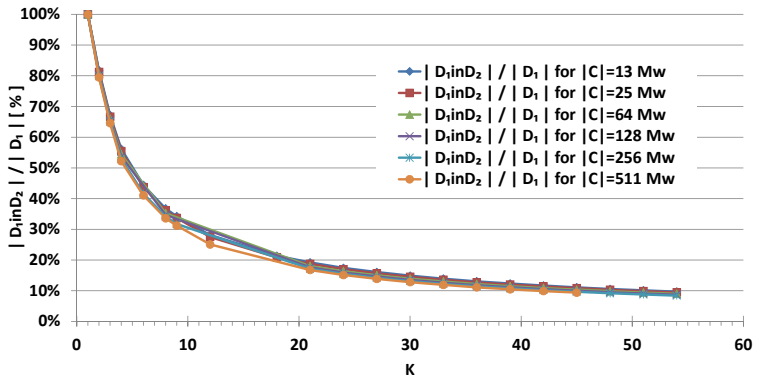


Figure 5.9: Average percentage of distinct subunigrams in D_2 per machine for different *corpus* sizes (from 13 Mw to 511 Mw). The Y axis represents the ratio $\left|D_{1_{in}D_2}(j)\right|/|D_1|$ and the X axis represents the number of machines.

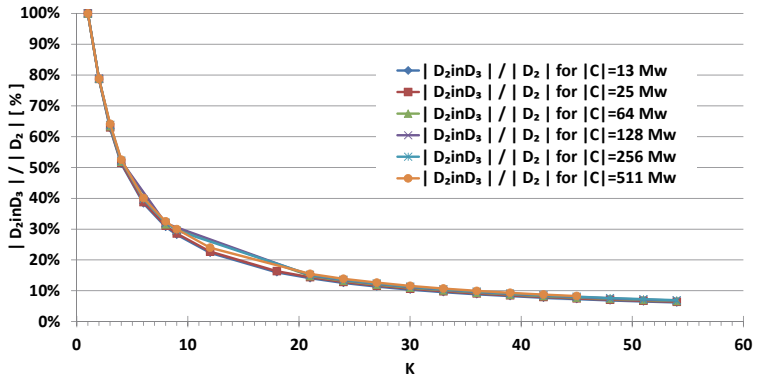


Figure 5.10: Average percentage of subbigrams in D_3 per machine for different *corpus* sizes (from 13 Mw to 511 Mw). The Y axis represents the ratio $\left|D_{2_{in}D_3}(j)\right|/|D_2|$ and the X axis represents the number of machines.

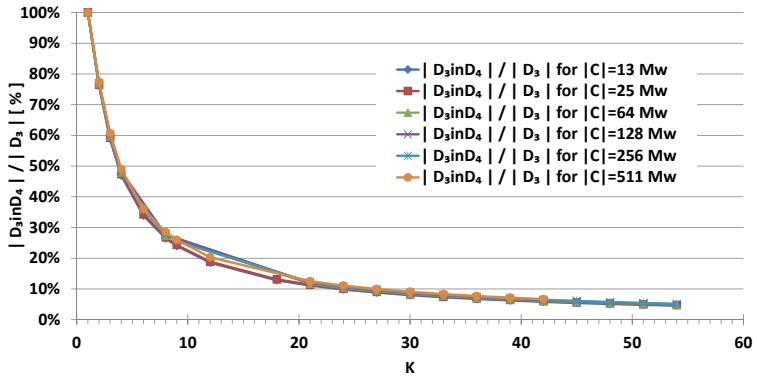


Figure 5.11: Average percentage of distinct subtrigrams in D_4 per machine for different *corpus* sizes (from 13 Mw to 511 Mw). The Y axis represents the ratio $\left|D_{3_{in}D_4}(j)\right|/|D_3|$ and the X axis represents the number of machines.

For all observed cases, the number of distinct sub n-grams per machine decreases following a power law K^{-b}, with $0 < b < 1$. This represents a reduction in the number of

distinct references to sub n-grams by each machine for an increasing number of machines. The impact of this behavior upon the phase two complexity is analyzed in the following.

5.5 Time Complexity of the Global Method

This section presents a preliminary analysis of the time complexity of the Global method in order to identify the influence of its two main components: i) The time complexity of the computation due to the execution of the intrinsic functions of the LocalMaxs method (Definition 2.1, on page 15) for the problem size determined by each *corpus* size and the n-gram sizes considered; ii) The communication overheads.

The basic concepts are introduced in section 5.5.1. The time complexities of the three phases of the Global method are analyzed in sections 5.5.2 through 5.5.4. The experimental measures of the time parameters of the basic computation and communication operations, which influence the performance of the Global method, are presented in section 5.5.5. A discussion about the complexity of the Global method is presented in section 5.5.6.

5.5.1 Basic Concepts

We analyze each phase of the LocalMaxs Global method implementation separately and assume that the three phases are executed in sequence. The total execution time (T_{exec}) is given by expression (5.2):

$$T_{exec} = T_{comp} + T_{overheads} = T_{comp}\left(1 + \frac{1}{G}\right)$$

$$G = \frac{Tcomp}{Toverheads} \tag{5.2}$$

$$T_{overheads} = T_{input} + T_{comm} + T_{output} + T_{others}$$

where T_{comp} is the computation time of the LocalMaxs functions with no overheads, and $T_{overheads}$ is the time of the overheads components. T_{input} and T_{output} are the input and output times, T_{comm} is the time for intermediate communication, and T_{others} represents the remaining overheads. The latter component includes the overheads due to the management of the algorithm workflow, the Java virtual machine, and the system overheads due to management of the parallel and distributed execution of the virtual machines environment. The G factor is the granularity ratio of the computation time over the total overheads time.

The expressions (5.2) can also be applied to model the execution time of each individual phase. Also, the expressions apply to a single machine case, or to a multiple machines case ($K > 1$). In the latter case each term in the expressions is annotated with an index j ($1 \leq j \leq K$) to describe the behavior in each machine j, e.g., $T_{exec}(j)$. Then, the

total execution time with K machines is determined by the execution time of the slowest machine, i.e., $T_{exec} = maximum\{T_{exec}(j) : 1 \leq j \leq K\}$.

5.5.1.1 The Computation Time

For each phase this component (T_{comp}) is modeled as the product of the problem size $N_n(phase)$ times the average time $t_{op}(phase)$ for executing a basic algorithm operation locally. In phase one, $N_n(phase_1)$ is equal to the sum of all the n-gram occurrences in the *corpus* for each of the considered n-gram sizes, and $t_{op}(phase_1)$ refers to a basic count and aggregate operation for an individual n-gram. For phase two, $N_n(phase_2)$ is the total number of distinct n-grams in the *corpus* for each of the considered n-gram sizes, excluding the unigrams, and $t_{op}(phase_2)$ refers to a basic glue calculation operation for an individual n-gram. Finally, in phase three, $N_n(phase_3)$ is the total number of distinct n-grams in the *corpus* for each of the considered n-gram sizes, excluding the unigrams and the singletons, and $t_{op}(phase_3)$ refers to a basic relevance evaluation operation for an individual n-gram.

For example, when evaluating the relevant expressions from bigrams to pentagrams (as illustrated in Figure 5.2), $N_n(phase_1)$ considers the total counts from unigrams to hexagrams, $N_n(phase_2)$ considers the distinct n-grams from bigrams to hexagrams, and $N_n(phase_3)$ considers the distinct nonsingleton n-grams from bigrams to pentagrams.

For an ideal sequential machine with infinite memory capacity and without any overheads, the total execution time for each individual phase would just be equal to the problem computation time:

$$T_0(phase) = N_n(phase) \times t_{op}(phase) \tag{5.3}$$

This time is used as an ideal reference to compare with the performance of the real implementation. In the case of a real machine with enough memory to hold the entire problem size there are some local implementation overheads, which are not considered in our analysis. And, when K machines are used, additional overheads appear due to the communication and data distribution.

For an homogeneous collection of K machines, the computation time in machine j ($T_{comp}(phase, j)$) of the algorithm for each phase would be given by $T_0(phase)/K$. However, in the analytical model presented in this thesis we also consider the additional communication overheads ($T_{comm}(phase, j)$) in each machine, thus the model assumes $T_{exec}(phase, j) \approx T_{comp}(phase, j) + T_{comm}(phase, j)$.

If the workload is also equally distributed among the K machines, then $T_{exec}(j)$ will be equal for all the machines.

5.5.1.2 The Input and Output Times

In each phase the $T_{input}(j)$ component in each machine only involves local reads from the local disk storage in phase one, or from the local partition table in phases two and

three. In phase one the $T_{output}(j)$ component corresponds to the sending of the n-gram frequency counts to the KVS servers (so it is analyzed as a remote communication time), while in phases two and three it corresponds to a local communication with the local KVS server in machine j to update the local n-gram table with the glue or relevance information.

5.5.1.3 The Communication Time

The communication component (T_{comm}) plays a significant role in the algorithm execution. This can be analyzed in terms of two equivalent dimensions: i) The number of messages exchanged and the message size; ii) The average per n-gram communication time and the volume of the n-gram data exchanged.

Number of Messages

Figure 5.12 shows an outline of the message interactions between the controllers and the KVS servers. Note that the controllers are independent of each other.

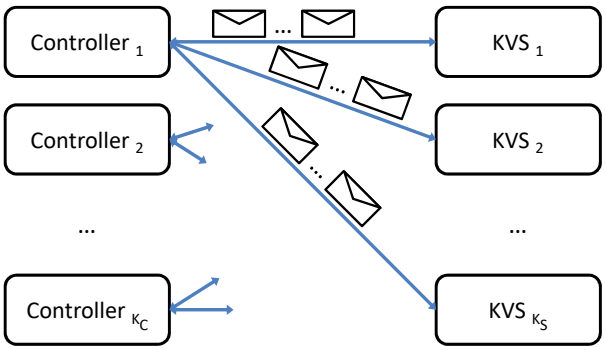


Figure 5.12: Messages exchanged between controllers and KVS servers.

This analysis applies both to the produced output (as in phase one) or to the volume of data (as in phases two and three) requested from each controller.

We use the following notation, although for simplicity here we omit the phase index: i) K_C is the number of controllers and K_S is the number of servers, and we assume $K_C = K_S = K$, with exactly the same number of controllers and servers per machine, namely 1; ii) M is the total number of messages exchanged by all controllers with all servers during each phase of the algorithm; iii) $M_C = K_S \times M_{C \to S}$ is the total number of messages exchanged by each controller with all the servers, where $M_{C \to S}$ is the number of messages exchanged by a controller with a single server; iv) The size of each message is denoted by M_{msg} measured in terms of the number of n-gram records included (for example an n-gram identification and its frequency count).

We assume a uniform distribution of the communicated data by all the controllers, as well as a uniform distribution of the data exchanged by each controller and the K_S servers. This assumption is consistent with the distribution of n-grams by hashing (section 5.4) and with the even work partition distribution.

The total number of exchanged messages is $M = K_C \times M_C$, and the total volume of exchanged data is:

$$V = M \times M_{msg} \tag{5.4}$$

Average per *n*-gram Communication Time

The above message-based analysis allows to quantify the intrinsic communication complexity, i.e., only depends on the patterns on the message interactions between the controllers and servers, and is independent of the time-related parameters of the underlying system communication infrastructure.

The relationship between the above aspects can be expressed in terms of the per-message latency and transmission time, giving the average time (T_{msg}) for a message interaction:

$$T_{msg} = latency + t_{transmission} \times M_{msg} \tag{5.5}$$

where $latency$, equivalent to the sending time of a null message, reflects the setup and delay overheads related to establishing a connection, depending on the underlying software/hardware communications layers, and $t_{transmission}$ is the average transmission time of the elementary items included in the message of M_{msg} size, and is the reciprocal of the communication bandwidth measured in terms of number of items, i.e., n-gram records, per second.

In order to model the time behavior of the communication interactions in a meaningful way regarding the n-gram based exchange between the controllers and the KVS servers we define the parameter $t_{n\text{–}gram}$ which is the average time for sending or fetching an n-gram to/from the KVS servers. This allows to abstract away the network specific effects of the message latency and transmission times, which in our architecture are the outcome of the combined effects of multiple software and physical layers from the Java Remote Method Invocation (RMI) communication, to the Transmission Control Protocol/Internet Protocol (TCP/IP) system protocols down to the network infrastructure. The average time for a message interaction is given by:

$$T_{msg} = M_{msg} \times t_{n\text{–}gram} = latency + t_{transmission} \times M_{msg} \tag{5.6}$$

Thus:

$$t_{n\text{–}gram} = \frac{latency}{M_{msg}} + t_{transmission} \tag{5.7}$$

This expression emphasizes the advantage of using large messages to hide the latency overhead.

In order to support this approach, the $t_{n\text{–}gram}$ parameter was experimentally measured in the used real execution environments as presented in section 5.5.5, on page 117. In this

way its value reflects an average indicator of the per n-gram communication time, which already takes into consideration the degree of message overlapping occurring in the real infrastructure. Thus, we can use this value as a rough approximation to the n-gram communication time for performance evaluation purposes. The average communication time that each controller takes to interact with all the servers is given by:

$$T_{comm}(j) = V_{K_C} \times t_{n\text{–}gram} = M_C \times M_{msg} \times t_{n\text{–}gram} \tag{5.8}$$

where V_{K_C} is the number of n-grams records exchanged by each controller with all the servers, i.e., $V_{K_C} = M_C \times M_{msg}$.

5.5.1.4 Efficiency and Granularity

The above defined concept of the T_0 computation time, in expression (5.3), allows us to compare the parallel implementation against the ideal case where only the computation component of the algorithm is considered, namely, in terms of the fundamental operations performed for a given problem size. However, in a real implementations other components representing the overheads must be taken into consideration.

In the ideal case of K homogeneous machines and equal distribution of work among the K machines, the following expression gives the ideal cost product representing the accumulated execution time due to the total number of operations performed by all the K machines:

$$K \times T_{exec} = T_0 \tag{5.9}$$

where T_{exec} is the total execution time (that corresponds to $T_{parallel}(K)$ in Definition 2.3) and T_0 is the total computation time with an ideal sequential machine. This corresponds to the ideal situation without any overheads, leading to a linear speedup .

In a real case with K homogeneous machines, the overheads must be considered leading to:

$$K \times T_{exec} = T_0 + T_{overheads} = T_0 \times \left(1 + \frac{T_{overheads}}{T_0}\right) \rightarrow T_{exec} = \frac{T_0}{K} + \frac{T_{overheads}}{K} \tag{5.10}$$

where $T_{overheads}$ are the total accumulated overheads for all the K machines. In this ideal case all the machines have equal computation and overhead times. Due to the non homogeneity of the real cases, the total execution time is given by the execution time in the slowest machine.

Thus, for a fixed problem size, the speedup of a configuration with K machines relative to the ideal sequential implementation can be expressed as:

$$Sp_0(K) = \frac{T_0}{T_{exec}(K)} = K \times \frac{T_0}{T_0 + T_{overheads}} \tag{5.11}$$

We define the ratio $T_0/T_{overheads}$ as the computation-to-overheads granularity $G(K)$. As in this model we only consider the communication overheads, then $T_{overheads} = T_{comm}$, so $G(K) = T_0/T_{comm}$. The speedup relative to the ideal sequential implementation can be expressed as:

$$Sp_0(K) = \frac{K}{1 + \frac{1}{G(K)}} \tag{5.12}$$

Ideally $Sp_0(K) = K$. The efficiency relative to the ideal sequential implementation can be expressed as:

$$E_0(K) = \frac{Sp_0(K)}{K} = \frac{1}{1 + \frac{1}{G(K)}} = 1 - \frac{T_{overheads}}{K \times T_{exec}} \tag{5.13}$$

Ideally $E_0(K) = 1$ (or 100%) corresponding to a complete absence of overheads. Note that the efficiency $E_0(K) = T_0/(K \times T_{exec}(K))$ becomes:

$$E_0(K = 1) = \frac{T_0}{T_{exec}(1)} = \frac{T_0}{T_0 + T_{overheads}(1)} = \frac{1}{1 + \frac{1}{G(1)}} \tag{5.14}$$

where, when $K = 1$, $T_{overheads}(K = 1)$ represents the overheads of a sequential real machine with limited memory when compared to an ideal machine with unlimited memory. For example, in a single real machine with limited memory there are communication overheads due to the access to an external KVS.

When a single machine is not able to execute the algorithm for large problem sizes we need to rely on multiple machines. Then we use the relative speedup and efficiency metrics (Definition 2.5 and 2.6, respectively, on pages 20 and 21). Thus,

$$Sp_{K_1 \to K_2} = \frac{K_2}{K_1} \times \frac{1 + \frac{1}{G(K_1)}}{1 + \frac{1}{G(K_2)}} = \frac{K_2}{K_1} \times E_{K_1 \to K_2} \tag{5.15}$$

where $G(K_1) = T_0/T_{overheads}(K_1)$ and $G(K_2) = T_0/T_{overheads}(K_2)$, respectively, for configurations with K_1 and K_2 machines.

These latter definitions are particularly important when processing large *corpus* sizes, which preclude the execution of the algorithm using less than a minimum number of machines. This number must be big enough to ensure that the problem size can fit into the total aggregated memory of the machines.

For a fixed problem size, when going from K_1 to K_2 machines, if $G(K_1) = G(K_2)$, then the relative speedup is linear and the relative efficiency is 100%. This may happen even if the absolute efficiency, i.e., comparing to the ideal sequential machine, is much lower than 100%.

In the following sections we discuss the time complexity of each phase of the LocalMaxs Global method implementation by presenting a simplified model that is used to estimate its performance and scalability and allows to estimate how far we are from the ideal case. The model only considers the pure local computation (T_{comp}) and the

communication (T_{comm}) components. In this modeling we rely on the knowledge about the statistical distribution of n-grams presented in chapter 3. The remaining overhead components, which are less significant than the communications, are evaluated macroscopically through the experimental measurements conducted during the real execution of the algorithm in different environments (chapters 6 to 9).

5.5.2 Time Complexity of Phase One

In this section we discuss the time complexity of phase one. Section 5.5.2.1 presents the execution model and section 5.5.2.2 presents an analysis of the granularity and efficiency.

5.5.2.1 Execution Model for Phase One

In phase one, $T_{input}(j)$ is the time for inputting the *n-grams* from the local partition, that is, the n-gram tables in the local KVS servers in machine j (section 5.3); $T_{comp}(j)$ is the time for counting the number of occurrences of all those *n-grams*; and $T_{output}(j)$ is the time for sending the *n-gram* frequency counts to the corresponding KVS servers. $T_{input}(j)$ and $T_{comp}(j)$ are proportional to the partition size. As the partition size is given by $|C|/K$, thus both $T_{input}(j)$ and $T_{comp}(j)$ are proportional to $1/K$. During phase one the output from each machine involves the communication with the remote KVS servers and its corresponding time is denoted as $T_{send}(j)$ in the following. This communication is performed asynchronously regarding the n-gram counting in each machine, and the output from the K machines to the KVS servers are overlapped.

If phase one was executed in a strictly sequential mode within each machine j, $1 \leq j \leq K$, the execution time in each machine would be given by the following expression.

$$T_1(j) = T_{input}(j) + T_{count}(j) + T_{aggregate}(j) + T_{send}(j) \tag{5.16}$$

Neglecting the T_{input} local component, we can identify the local computation component $T_{Count}(j) = T_{count}(j) + T_{aggregate}(j)$ and the communication component $T_{comm}(j)$. Instead of a pure sequential execution in each machine we assume an execution strategy where a thread performs the input, overlapped with another thread that performs the full counting and aggregation, before sending any messages to the KVS servers.

The parameter $M_{C \to S}$, that is the number of messages that each controller sends to each KVS server, is determined by the controller strategy to locally aggregate the output data produced.

Full Local Aggregation

Namely, for phase one in the case of each controller performing a local aggregation of the frequency counts of all the n-grams found in its local partition, before sending them to the KVS servers, then in this case $M_{C \to S} = 1$, that is a single message is sent by each controller to each server with a size given by $M_{msg} = V_{K_C}/K_S$, where V_{K_C} is the number of distinct n-grams of each size (1 up to n) in the local partition.

If each machine performs a full local aggregation of the frequency counts of the n-grams found in its partition, before sending them to the KVS servers, then the estimate of the volume of the output communication for machine j sent to each server is given by:

$$V(j) = \frac{\sum_{i=1}^{n} \left| D_{i_{in}LocalPartition}(j) \right|}{K} = \frac{\left| D_{All(1\cdots n)_{in}LocalPartition}(j) \right|}{K} \tag{5.17}$$

where $\left| D_{All(1\cdots n)_{in}LocalPartition}(j) \right|$ is the total number of distinct n-grams of sizes i from 1 up to n in the local partition of the machine j, $\left| D_{i_{in}LocalPartition}(j) \right|$ is the number of distinct n-grams of size i in local partition, n is the maximum n-gram size considered in this phase (Figure 5.2), and we assume $K = K_C = K_S$. The average value over the K machines can be considered as a good approximation of the per machine volume of data.

The full local aggregation strategy requires enough local memory to keep the entire table of $D_{All(1\cdots n)_{in}LocalPartition}(j)$ entries in each machine. It can also imply some network communication overheads due to the likelihood of an almost simultaneous sending of the n-grams by all the machines and the corresponding peak overload of the KVS servers.

No Local Aggregation

On the other hand, in the worst case corresponding to no local aggregation of the n-gram counts, we estimate the per machine volume of the output communication sent to each server as given by:

$$V(j) = \frac{\sum_{i=1}^{n} \left| Set_{All_i} \right|}{K} \approx n \times \frac{|C|}{K} \tag{5.18}$$

where $\left| Set_{All_i} \right| \approx |C|$ is the total number of n-grams of size i ($1 \leq i \leq n$, where n is the maximum n-gram size). However, this alternative would lead to a volume of output data much higher than in the full local aggregation case.

Partial Local Aggregation

As an intermediate approach the local execution of counting and aggregation can be overlapped with the communication. Although it generates a higher volume of output data compared to a full local aggregation case, because distinct n-grams get repeated in multiple segments, it allows the sending of n-gram data to the servers to be performed incrementally and time overlapped among the multiple machines during the execution of phase one.

In this alternative each controller sends parts of its produced data during the execution of phase one: this could be made without any local aggregation of the counts by the controller, but preferably it should be accompanied by a segment based partial aggregation. The data segments should be big enough to ensure that the generated M_{msg} sizes are above the threshold beyond which the message sending latency is minimized (expression (5.7)). Although this latter alternative generates more messages than the full local aggregation (that only sends one message in the end of phase one) it allows to spread the

sending of the data by the controllers to the KVS servers, all along the execution of phase one. This has a potential contribution to reduce the network or KVS server contention due to an almost simultaneous sending of messages by all the controllers which tends to occur in the case of full local aggregation due to the balanced work distribution. This latter effect can be more likely to appear as the scale grows for large numbers of machines. It also requires less local memory to hold the output n-gram data, compared to the full local aggregation.

Thus, in the followed approach, the execution within each machine is performed by multiple concurrent threads as illustrated in Figure 5.13:

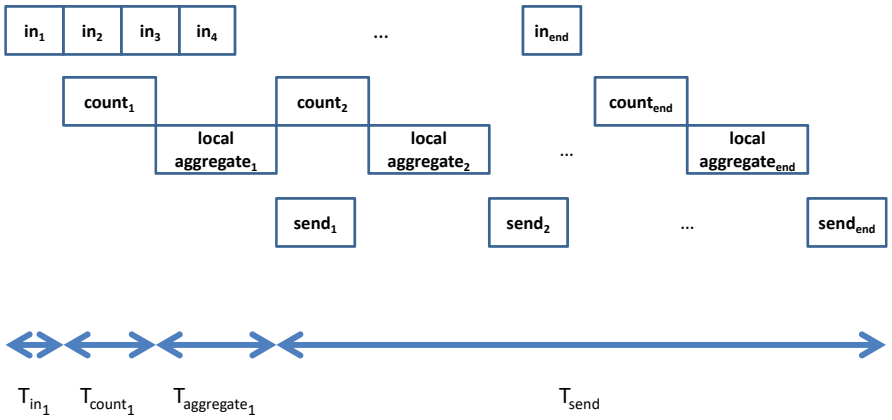


Figure 5.13: Phase one multithreaded operations sequence.

where in_i represent the input of the chunk i ($1 \leq i \leq end$) of sentences from the local *corpus* partition; $count_i$ represents the counting of the distinct n-grams within chunk i; the aggregation of frequencies of distinct n-grams in chunk i is represented by $localAggregate_i$; and $send_i$ is the parcel corresponding to the sending of the frequencies of the n-grams to the KVS servers.

Thus, the corresponding execution time in each machine j, $1 \leq j \leq K$, is given by the following expression:

$$T_1(j) = T_{in_1}(j) + T_{count_1}(j) + T_{localAggregate_1}(j) + Time\left(T_{send_{end}}(j)\right) - Time\left(T_{send_1}(j)\right) \quad (5.19)$$

where $Time(\cdots)$ is the wall clock time of the indicated output events, namely, the beginning of the sending operations for the first chunk ($T_{send_1}(j)$) and for the last chunk ($T_{send_{end}}(j)$). The parcels $T_{in_1}(j)$, $T_{count_1}(j)$ and $T_{localAggregate_1}(j)$ can be neglected compared to the sending time of all the chunk data, because they are both local operations and regard a single chunk.

With reference to the analysis of the messages as presented in section 5.5.1.3, in this case the message size sent by each controller is given by:

$$M_{msg} = \frac{V'_{K_C}/K_s}{M_{C \to S}} \quad (5.20)$$

In this case $V_{K_C}^{'}$ is the total number of the distinct n-grams in all the data segments mentioned above, thus it is higher than the V_{K_C} value in the full local aggregation case.

$$T_{msg}^{'} = \frac{V_{K_C}^{'}/K_s}{M_{C\rightarrow S}} \times t_{n\text{–}gram}^{'} \rightarrow T_{msg}^{'} \times M_{C\rightarrow S} = \frac{V_{K_C}^{'}}{K_S} \times t_{n\text{–}gram}^{'} \tag{5.21}$$

In expression (5.21) the value $t_{n\text{–}gram}^{'}$ can become greater than the value of $t_{n\text{–}gram}$ in the case of full local aggregation with a single message sent by the controller to each KVS server, in case the message size becomes smaller. Indeed, as shown in expression (5.7), a message size small enough can make the latency parcel significant. This can be avoided by ensuring that each message size (expression (5.20)) is always greater than the output buffer threshold beyond which the message latency is minimized (as discussed in section 5.5.5 on page 117).

5.5.2.2 Granularity Analysis for Phase One

The critical issue regarding the comparison between the local aggregation alternatives is the computation-to-communication granularity obtained in each case.

Full Local Aggregation

The corresponding granularity factor G in phase one for each machine j is:

$$G(j) = \frac{T_{Count}(j)}{T_{comm}(j)} = \frac{N_n(phase1, j) \times t_{op}}{\left|D_{All(1\cdots n)_{in}LocalPartition}(j)\right| \times t_{n\text{–}gram}} \tag{5.22}$$

In expression (5.22) $T_{Count}(j)$ is equal to the local problem size in machine j, times the average time for a local count and aggregate operation (t_{op}) and $T_{comm}(j)$ is the total number of distinct n-grams in the local partition of machine j, times the average time for sending an n-gram to a KVS server ($t_{n\text{–}gram}$).

When considering a single machine, $K = 1$, expression (5.22) gives the phase one intrinsic problem computation-to-communication granularity:

$$G_{Full}(K = 1) = \frac{|C| \times n \times t_{op}}{\left|D_{All(1\cdots n)_{in}C}\right| \times t_{n\text{–}gram}} \tag{5.23}$$

where n is the maximum n-gram size considered, for example 6 in the Figure 5.2, and $D_{All(1\ldots n)_{in}C}$ is the total number of distinct n-grams from sizes 1 to n, in the *corpus* C.

In order to estimate the above granularity for $K = 1$ we need to calculate the number of distinct n-grams in the *corpus* ($\left|D_{All(1\cdots n)_{in}C}\right|$). This was done by applying the theoretical model, expression (3.18) presented in chapter 3, leading to Table 5.7 in the case of the English *corpora* (section 3.2 on page 27), when considering the number of distinct n-grams from unigrams up to hexagrams ($n = 6$).

Table 5.7: Phase one problem size and total number of distinct n-grams, in a single machine scenario ($K = 1$), when counting unigrams up to hexagrams. *Corpus* size in words.

$\|C\|$	$N_n(phase_1) = \|C\| \times 6$	$\left\|D_{All(1\cdots6)_{in}C}\right\|$	$N_n(phase_1) / \left\|D_{All(1\cdots6)_{in}C}\right\|$
$\mathbf{1.6 \times 10^7}$	9.60×10^7	5.97×10^7	1.61
$\mathbf{1.3 \times 10^8}$	7.68×10^8	4.14×10^8	1.86
$\mathbf{1.0 \times 10^9}$	6.14×10^9	2.74×10^9	2.24
$\mathbf{1.6 \times 10^{10}}$	9.83×10^{10}	2.79×10^{10}	3.52
$\mathbf{1.3 \times 10^{11}}$	7.86×10^{11}	1.16×10^{11}	6.80
$\mathbf{1.0 \times 10^{12}}$	6.29×10^{12}	3.24×10^{11}	19.41
$\mathbf{1.7 \times 10^{13}}$	1.01×10^{14}	4.57×10^{11}	220.35
$\mathbf{1.3 \times 10^{14}}$	8.05×10^{14}	4.57×10^{11}	1762.76

Figure 5.14 shows the problem size (denoted by $N_n(phase_1)$) and the total number of distinct n-grams of sizes from 1 up to 6 (denoted by $\left|D_{All(1\cdots6)_{in}C}\right|$), both measured in terms of number of n-grams, and the ratio $N_n(phase_1) / \left|D_{All(1\cdots6)_{in}C}\right|$ for a single machine when $n = 6$.

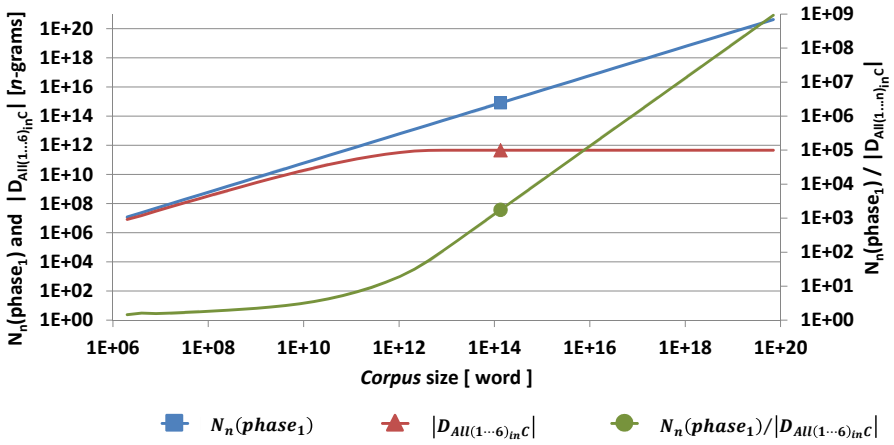


Figure 5.14: Phase one problem size and total number of distinct n-grams, in a single machine scenario ($K = 1$), when counting unigrams up to hexagrams and varying the *corpus* size. The left Y axis represents the phase one problem size and the total number of distinct n-grams; the right Y axis represents the ratio $N_n(phase_1) / \left|D_{All(1\cdots6)_{in}C}\right|$; and the X axis represents the *corpus* size in words.

The ratio $N_n(phase_1) / \left|D_{All(1\cdots6)_{in}C}\right|$ shows that for $K = 1$ and $n = 6$, and assuming a ratio $t_{op}/t_{n\text{–}gram}$ of 1, the granularity only reaches values of 10 or more, beyond *corpus* sizes of around 0.5 Tw. For still larger *corpus* sizes the granularity increases proportionally to the *corpus* size.

In order to expose analytically the behavior of the granularity we derived the fitting expressions for the $\left|D_{All(1\cdots6)_{in}C}\right|$ curve in Figure 5.14 and obtained the following formulas, which is an approximation to the ones derived from the theoretical model presented in chapter 3 (expression (3.18) on page 34):

A) *Corpus* range from 2×10^6 up to about 1.3×10^{11} words: $\left|D_{All(1\cdots6)_{in}C}\right| \approx 31.06 \times |C|^{0.8743}$ n-grams

B) *Corpus* range from 1.3×10^{11} up to about 4.2×10^{12} words: $\left|D_{All(1\cdots6)_{in}C}\right| \approx 1 \times 10^{11} \times \lg(|C|) - 2 \times 10^{12}$ n-grams

C) *Corpus* range beyond 4.2×10^{12} words: almost constant and equal to $\left|D_{All_{(1\cdots6)in}C}\right| \approx 4.57 \times 10^{11}$ n-grams

When considering multiple machines, $K > 1$, each one with full local aggregation, expression (5.22) becomes:

$$G_{Full}(j) = \frac{|C(j)| \times n \times t_{op}}{\left|D_{All(1\cdots n)_{in}LocalPartition}(j)\right| \times t_{n\text{–}gram}} \tag{5.24}$$

which gives the phase one computation-to-communication granularity of a machine j, $1 \leq j \leq K$, where $|C(j)| = |C|/K$ represents the size of the local partition on machine j, and $|D_{All(1\cdots n)_{in}LocalPartition}(j)|$ is the total number of distinct n-grams from sizes 1 to n, in the local partition.

The above analysis that led to Figure 5.14 remains valid as long as we replace the *corpus* size $|C|$ in the X axis with the partition size $|C(j)| = |C|/K$. Thus, we can obtain an expression for the $G(j)$ for the above fitting ranges:

A) $G_{Full}(j) = \frac{|C(j)| \times n \times t_{op}}{31.06 \times \left(\frac{|C|}{K}\right)^{0.8743} \times t_{n\text{–}gram}} = \frac{|C| \times n \times t_{op}}{31.06 \times |C|^{0.8743} \times t_{n\text{–}gram}} \times K^{-0.1257} = G_{Full}(K=1) \times K^{-0.1257}$

B) $G_{Full}(j) = \frac{|C(j)| \times n \times t_{op}}{\left(1 \times 10^{11} \times \lg\left(\frac{|C|}{K}\right) - 2 \times 10^{12}\right) \times t_{n\text{–}gram}}$

C) $G_{Full}(j) = \frac{|C(j)| \times n \times t_{op}}{4.57 \times 10^{11} \times t_{n\text{–}gram}} = \frac{|C| \times n \times t_{op}}{4.57 \times 10^{11} \times t_{n\text{–}gram}} \times \frac{1}{K}$

For the first range (***A)***) of *corpus* partition sizes $|C(j)|$ until about 130 Gw the granularity with K machines is only slightly lower than the granularity with a single machine, and decreases slowly with K, for each fixed size *corpus*. This is due to the fact that the distinct n-grams in this range closely follow the evolution of the partition size.

On the contrary, for the third range (***C)***) of *corpus* partition sizes $|C(j)|$ beyond 4.2 Tw the granularity with K machines grows proportionally to the partition size, leading to efficiency values, relatively to the ideal sequential machine, which become close to 100%. So, if we keep the partition size large enough to stay within this range, and keep the *corpus* size fixed and vary the number of machines, but keeping the granularity values of at least above 10, then the efficiency is at least above 90% and the speedup relative to the ideal sequential machine is at least $0.9 \times K$.

Furthermore, also for the third range (***C)***), we can keep the value of the partition size $|C(j)|$ fixed and be able to process progressively large *corpus* sizes by increasing the number of machines (K) in the same proportion as the *corpus* size ($|C|$) while keeping

the same granularity, thus the same parallel efficiency relatively to the ideal sequential machine.

The above analysis assumed that the ratio $t_{op}/t_{n\text{–}gram}$ does not change significantly with K.

Partial Local Aggregation

When considering partial local aggregation, expression (5.22) becomes:

$$G_{Partial}(j) = \frac{|C(j)| \times n \times t_{op}}{\left|D_{All(1\cdots n)_{in}SubPartition}(j)\right| \times NumberOfSubPartitions(j) \times t_{n\text{–}gram}} \tag{5.25}$$

where $NumberOfSubPartitions(j)$ gives the number of equal-sized subpartitions in which the local partition of size $|C(j)|$ is subdivided, and $\left|D_{All(1\cdots n)_{in}SubPartition}(j)\right|$ is the total number of distinct n-grams from sizes 1 to n, in a subpartition on machine j. Although this approach has the advantage of requiring less local memory for the n-gram output data, because it sends the subpartition data one at a time, and also having the advantage of not concentrating all the sending at the end of phase one, it leads to a slightly lower granularity than the full local aggregation alternative. This is due to generating a larger volume of output data for the same partition size when compared with the full local aggregation case, due to the repetition of some distinct n-grams in multiple subpartitions.

Given that the number of subpartitions within each partition is defined as:

$$NumberOfSubPartitions = \frac{|C(j)|}{SubPartitionSize} = \frac{\frac{|C|}{K}}{SubPartitionSize} \tag{5.26}$$

expression (5.25) becomes:

$$G_{Partial}(j) = \frac{SubPartitionSize \times n \times t_{op}}{\left|D_{All(1\cdots n)_{in}SubPartition}(j)\right| \times t_{n\text{–}gram}} \tag{5.27}$$

For range ***A)*** of *corpus* sizes expression (5.27) simplifies to:

$$\begin{gathered} G_{Partial}(j) \simeq \frac{SubPartitionSize \times n \times t_{op}}{31.06 \times SubPartitionSize^{0.8743} \times t_{n\text{–}gram}} = \\ \frac{n \times t_{op}}{31.06 \times t_{n\text{–}gram}} \times SubPartitionSize^{0.1257} = \\ \frac{n \times t_{op}}{31.06 \times t_{n\text{–}gram}} \times \left(\frac{\frac{|C|}{K}}{NumberOfSubPartitions}\right)^{0.1257} \end{gathered} \tag{5.28}$$

leading to:

$$G_{Partial}(j) \simeq G_{Full}(j) \times NumberOfSubPartitions^{-0.1257} \tag{5.29}$$

Thus, for this range, although the granularity decreases with the number of subpartitions within each partition, compared to the full local aggregation, it is only a slight

reduction. On the other hand, by increasing the number of subpartitions, it is possible to reduce the required local memory for the output n-gram data, corresponding to a reduction in the number of distinct n-grams in the corresponding subpartition.

Figure 5.15 shows, for the case of partial local aggregation, the evolution of the granularity factor for phase one *versus* the *corpus* partition size when considering multiple subpartitions (in the of cases of 1, 4, 16 and 64 subpartitions), and assuming a ratio $t_{op}/t_{n\text{–}gram}$ of 1. It also shows the evolution of the total number of distinct n-grams per partition ($\left|D_{All(1\cdots6)_{in}C(j)}\right|$) represented by the dotted curve.

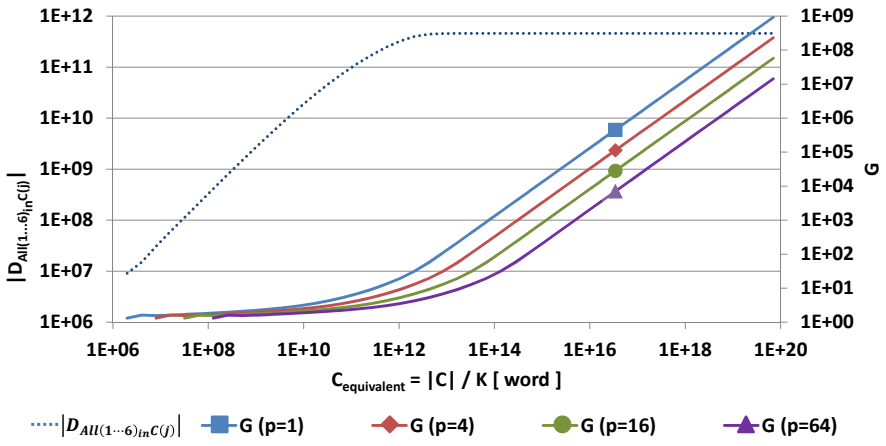


Figure 5.15: Granularity for phase one when using partial local aggregation for different numbers of subpartitions when varying the partition size. The left Y axis represents the total number of distinct n-grams per partition; the right Y axis represents the granularity factor; and the X axis represents the partition size in words.

Table 5.8 shows the values of the granularity factor, for the curves presented in Figure 5.15, when the *corpus* partition size ranges from 16 Mw up to 130 Tw.

Table 5.8: Granularity for phase one when using partial local aggregation for different numbers of subpartitions (p). Partitions size in words.

$\|C\|/K$	$G\,(p=1)$	$G\,(p=4)$	$G\,(p=16)$	$G\,(p=64)$
$\mathbf{1.6\times10^{7}}$	1.61	1.61		
$\mathbf{1.3\times10^{8}}$	1.86	1.69	1.57	1.45
$\mathbf{1.0\times10^{9}}$	2.24	1.96	1.77	1.61
$\mathbf{1.6\times10^{10}}$	3.52	2.69	2.24	1.96
$\mathbf{1.3\times10^{11}}$	6.80	4.22	3.04	2.44
$\mathbf{1.0\times10^{12}}$	19.41	9.16	5.26	3.52
$\mathbf{1.7\times10^{13}}$	220.35	56.76	19.41	9.16
$\mathbf{1.3\times10^{14}}$	1762.76	440.69	110.40	31.56

The increments in the granularity factor when going from subpartition numbers p_1 to p_2 are denoted by $\Delta\left(G_{p_1}\rightarrow G_{p_2}\right)$, and the increments in the total number of distinct

n-grams per subpartition, also when going from subpartition numbers p_1 to p_2 are denoted by $\Delta D_A\left(G_{p_1} \rightarrow G_{p_2}\right)$. The factors $\Delta\left(G_{p_1} \rightarrow G_{p_2}\right)$ and $\Delta D_A\left(G_{p_1} \rightarrow G_{p_2}\right)$ are defined as follows:

$$\begin{aligned} \Delta\left(G_{p_1} \rightarrow G_{p_2}\right) &= \frac{G_{p_2} - G_{p_1}}{G_{p_1}} \\ \Delta D_A\left(G_{p_1} \rightarrow G_{p_2}\right) &= \frac{\left|D_{All(1\cdots n)_{in}SubPartition}\left(G_{p_2}\right)\right| - \left|D_{All(1\cdots n)_{in}SubPartition}\left(G_{p_1}\right)\right|}{\left|D_{All(1\cdots n)_{in}SubPartition}\left(G_{p_1}\right)\right|} \end{aligned} \tag{5.30}$$

where G_{p_i} is the granularity factor when considering p_i number of subpartitions and $\left|D_{All(1\cdots n)_{in}SubPartition}\left(G_{p_i}\right)\right|$ is the total number of distinct n-grams of sizes 1 up to n per subpartition, when considering p_i number of subpartitions.

Figure 5.16 shows, for the case of partial local aggregation, the increment in the granularity when the number of subpartitions varies, represented by the filled curves $\Delta\left(G_{p_1} \rightarrow G_{p_2}\right)$, and the increment in the total number of distinct n-grams per subpartition when the number of subpartitions change (dotted curves $\Delta D_a\left(G_{p_1} \rightarrow G_{p_2}\right)$).

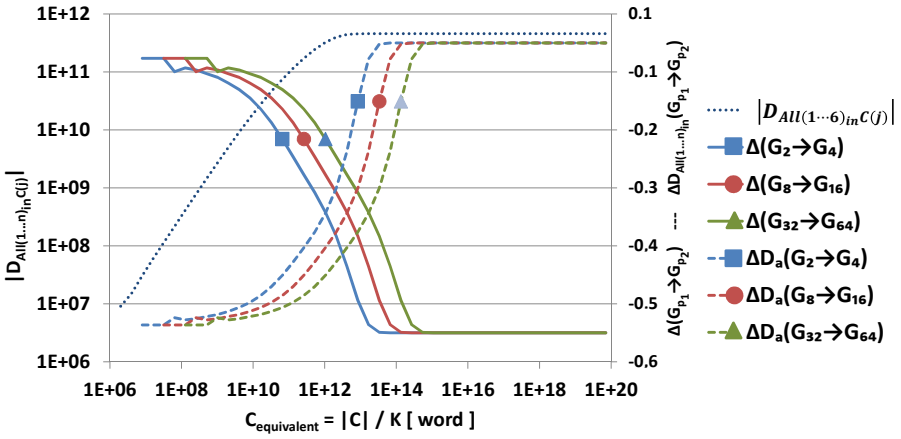


Figure 5.16: Granularity increment for phase one when using partial local aggregation for different numbers of subpartitions. The left Y axis represents the total number of distinct n-grams per partition; the right Y axis represents the increments in the granularity and in the number of distinct per subpartition when going from partition numbers p_1 to p_2; and the X axis represents the partition size per machine in words.

Table 5.9 shows the values for the increments in the granularity factor and the corresponding increments in the total number of distinct n-grams per subpartition, for the curves presented in Figure 5.16, when the *corpus* partition size ranges from 16 Mw up to 130 Tw.

Table 5.9: Granularity increment for phase one when using partial local aggregation for different numbers of subpartitions. *Corpus* partition size in words.

$\|C\|/K$	$\Delta(G_2 \to G_4)$	$\Delta(G_8 \to G_{16})$	$\Delta(G_{32} \to G_{64})$	$\Delta D_A(G_2 \to G_4)$	$\Delta D_A(G_8 \to G_{16})$	$\Delta D_A(G_{32} \to G_{64})$
$\mathbf{1.6 \times 10^7}$	-0.026			-0.486		
$\mathbf{1.3 \times 10^8}$	-0.043	-0.026	-0.026	-0.478	-0.486	-0.486
$\mathbf{1.0 \times 10^9}$	-0.059	-0.047	-0.050	-0.468	-0.476	-0.474
$\mathbf{1.6 \times 10^{10}}$	-0.114	-0.080	-0.059	-0.436	-0.456	-0.468
$\mathbf{1.3 \times 10^{11}}$	-0.197	-0.138	-0.095	-0.378	-0.420	-0.447
$\mathbf{1.0 \times 10^{12}}$	-0.292	-0.227	-0.166	-0.294	-0.353	-0.401
$\mathbf{1.7 \times 10^{13}}$	-0.486	-0.385	-0.292	-0.027	-0.187	-0.294
$\mathbf{1.3 \times 10^{14}}$	-0.500	-0.499	-0.444	0.000	-0.002	-0.101

From Figure 5.16 and Table 5.9 we conclude that in first range of partition sizes considered (***A***)), doubling the number of subpartitions per partition leads to a granularity decrease of about 10% while it leads to a reduction in the number of distinct n-grams per subpartition of about 50%. Thus there is room for a trade off between reducing the subpartition size to fit the local memory constraint and not reducing the granularity significantly. This is opposite to what happens in the third range (***C***)), after reaching the *plateaux*.

5.5.3 Time Complexity of Phase Two

In this section we discuss the time complexity of phase two. Section 5.5.3.1 presents the execution model and section 5.5.3.2 presents an analysis of the granularity of phase two.

5.5.3.1 Execution Model for Phase Two

Phase two problem size ($N_n(phase_2)$) is the total number of glue operations for finding relevant expressions for a given range of n-gram sizes and this directly determines the computation time ($T_{comp}(phase_2)$). We consider two cases: i) Isolated glue calculation, consisting only of the glue g_i for n-grams of size i, with $i \geq 2$; ii) Combined glue calculation, consisting of all the glues $g_{n_1 \ldots n_2}$ for the n-grams from n_1 up to and including n_2, with $2 \leq n_1 < n_2$. For example, calculating the combined glue g_{234} (needed for the relevance re_{23} of bigrams and trigrams) involves the glues for all bigrams, trigrams and tetragrams, $N_n(phase_2) = |D_2| + |D_3| + |D_4|$. For the isolated glue calculation of bigrams (g_2) $N_n(phase_2) = |D_2|$; for trigrams (g_3) $N_n(phase_2) = |D_3|$; for tetragrams (g_4) $N_n(phase_2) = |D_4|$.

Generated References

The number of references to all sub n-grams generated ($|all_{glue_{g_n} Ref}|$) by a glue calculation g_n determines the communication time ($T_{comm}(phase_2)$) because the corresponding n-gram data is stored in the KVS servers. To analyze the references generated we consider as example the glue calculation g_2 where the table $D_2(j)$ represents the local partition of

the bigrams in machine j, with $1 \leq j \leq K$. The glue for each individual bigram depends on the frequencies of its two subunigrams. Thus, the total number of references to subunigrams (denoted as $\left|all_{glue_{g_2}Ref}(j)\right|$) in machine j is $2 \times |D_2(j)|$. Among these references there are repeated unigram occurrences, which can be filtered out by a cache mechanism, leading to $D_{1_{in}D_2}(j)$, the set of all the distinct subunigrams within the bigrams found in table $D_2(j)$. Table 5.10 shows the number of references to all sub n-grams (with repetitions in the left column) and the number of distinct sub n-grams (in the right column) for the glue calculations g_2, g_3, g_4.

Table 5.10: Glue references.

glue	**Sub n-grams** $\left\|all_{glue_{g_i}Ref}(j)\right\|$	**Distinct sub n-grams for glue g_i**
g_2	$2 \times \|D_2(j)\|$	$\left\|D_{1_{in}D_2(j)}\right\|$
g_3	$2 \times \|D_3(j)\| + 2 \times \|D_3(j)\|$	$\left\|D_{1_{in}D_3(j)}\right\| + \left\|D_{2_{in}D_3(j)}\right\|$
g_4	$2 \times \|D_4(j)\| + 2 \times \|D_4(j)\| + 2 \times \|D_4(j)\|$	$\left\|D_{1_{in}D_4(j)}\right\| + \left\|D_{2_{in}D_4(j)}\right\| + \left\|D_{3_{in}D_4(j)}\right\|$

For glue g_i, with $2 \leq i \leq 4$, the sub n-grams in the left column include all the references to the sub n-grams from sizes 1 up to $(i-1)$ (as in Definition 2.1, on page 15). The right column gives the total number of distinct sub n-grams cited by those references. Although not shown in the above table, for the combined glue g_{234} the total number of references to sub n-grams (denoted as $\left|all_{glue_{g234}Ref}(j)\right|$) is the sum of the individual references for the considered glues, that is $2 \times |D_2(j)| + 4 \times |D_3(j)| + 6 \times |D_4(j)|$.

Execution Time

Assuming a strict sequential execution in each machine (j), the phase two execution time in each machine would be:

$$T_2(j) = T_{input}(j) + T_{Glue}(j) + T_{comm}(j) + T_{output}(j) \tag{5.31}$$

where: $T_{input}(j)$ is the input time; $T_{Glue}(j)$ is the local glue calculation time in each machine, corresponding to the T_{comp} parcel in expression (5.2), which includes the parsing of the sub n-grams, and the glue calculation assuming all arguments are already local, i.e., were already fetched if necessary; $T_{comm}(j)$ is the remote sub n-gram fetch time; and $T_{output}(j)$ is the local output time of the glue value. The glue time is: $T_{Glue}(j) = N_n(phase_2, j) \times t_{g_n}$, where $t_{g_n} = t_{op}(phase_2)$ (section 5.5.1.1) is the average time for the parsing and the glue calculation of any n-gram with size from 2 to n, assuming that all the glue arguments are already local. The communication time is: $T_{comm}(j) = \left|all_{glue_{g_i}Ref}(j)\right| \times f(j) \times t_{fetch_n}$, where $\left|all_{glue_{g_i}Ref}(j)\right|$ is the number of references to the sub n-grams needed by the calculations in each machine, $f(j)$ is the fraction of those references which are remote (i.e., the miss ratio), and t_{fetch_n} is the average time to remotely fetch any n-gram of size from 1 to $n-1$.

Both t_{g_n} and t_{fetch_n} were experimentally measured (section 5.5.5). For a single machine ($K = 1$) T_2 gives the total phase two execution time. For K machines working in parallel, each one independently calculates the glue of the n-grams in its local n-gram partition, thus $T_2(j)$ gives the phase two execution time for each machine, and the total phase two execution time is the maximum $T_2(j)$ among the K machines. As $N_n(phase_2)$ is equally divided by K machines, in each machine $T_{input}(j)$, $T_{output}(j)$ and $T_{Glue}(j)$ are proportional to the local partition size, that is $N_n(phase_2)/K$. Thus, they are proportional to $1/K$. The glue time is $T_{Glue}(j)$=$N_n(j) \times t_{g_n}$=$(N_n/K) \times t_{g_n}$. The communication time is $T_{comm}(j) = \left|all_{glue_{g_i}Ref}(j)\right| \times f(j) \times t_{fetch_n}$. The parcel $\left|all_{glue_{g_i}Ref}(j)\right|$ is proportional to $1/K$, as $\left|all_{glue_{g_i}Ref}(j)\right| = \left|all_{glue_{g_i}Ref}\right|/K$, where $\left|all_{glue_{g_i}Ref}\right|$ is the total number of references to sub n-grams in the single machine case ($K = 1$), while the behavior of fraction $f(j)$ as a function of K depends on the type of the cache operation, as discussed in the following chapters, where we also discuss how the communication $T_{comm}(j)$, for a fixed *corpus* size, decreases with K, as long as there is no network congestion. By using multiple machines (K) in parallel, we can reduce the time for local glue calculation (T_{Glue}) by a factor of K, while T_{comm} also decreases.

However, instead of a strict sequential execution, the execution of phase two within each machine is performed by multiple concurrent threads as illustrated in Figure 5.17:

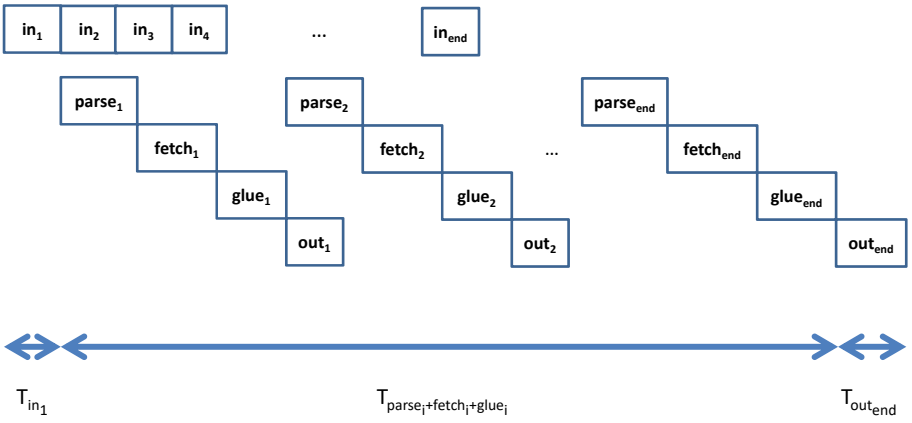


Figure 5.17: Phase two multithreaded operations sequence.

where in_i represents the input of a chunk i of n-grams from the local KVS server; $parse_i$ is the time spent in the parsing of the chunk i of data to identify the sub n-grams required for the glue calculation; $fetch_i$ represents the time used to check the local cache for the sub n-grams in the n-grams of chunk i, and fetch them from the KVS servers if they are not present in the local cache; $glue_i$ is the local calculation of the glue of the n-grams contained in the input chunk i after fetching all the needed sub n-grams; and out_i is the time used to output the glues of chunk i of n-grams to the local KVS server.

Thus, the corresponding execution time in each machine is given by the following expression:

$$T_2(j) = T_{in_1}(j) + \left(T_{parse_i}(j) + T_{fetch_i}(j) + T_{glue_i}(j)\right) \times NumberOfChunks + T_{out_{end}}(j) \quad (5.32)$$

The parcels $T_{in_1}(j)$ and $T_{out_{end}}(j)$ can be neglected regarding the other parcel, because they are both local operations to read or write a single chunk of n-grams. The middle parcel can be decomposed in two main components: i) $T_{parse}(j)$ and $T_{glue}(j)$ which correspond to the total accumulated sum of the parse and glue calculations time for all chunks processed in each machine, i.e., $T_{Glue}(j)$ as defined in expression (5.31); ii) $T_{comm}(j)$ corresponds to the total time for fetching the remote sub n-grams from the KVS servers, that count as cache misses. Thus, expression (5.32) can be simplified to:

$$T_2(j) \simeq T_{Glue}(j) + T_{comm}(j) \quad (5.33)$$

Even considering that in the real execution the phase two components of expression (5.31) overlap in time, being executed by four concurrent threads (one for input; one for parse and glue calculation; one for fetch; and one for output), the communication time $T_{comm}(j)$ dominates the phase two execution time.

5.5.3.2 Granularity Analysis for Phase Two

The computation-to-communication ratio in each machine $1 \leq j \leq K$, corresponding to a granularity factor of the algorithm implementation when only the communication overheads are considered, is defined by:

$$G(j) = \frac{T_{Glue}(j)}{T_{comm}(j)} = \frac{N_n(phase_2)}{\left|all_{glue_{g_i}Ref}\right|} \times \frac{1}{f(j)} \times \frac{t_{g_n}}{t_{fetch_n}} \quad (5.34)$$

The corresponding expression of the phase two execution time in each machine is:

$$T_2(j) = T_{input}(j) + T_{Glue}(j) \times \left(1 + \frac{1}{G(j)}\right) + T_{output}(j) \quad (5.35)$$

Even if we use K machines to speed up the local glue calculation by a factor of K, and also achieving some reduction in the communication time, the $N_n(phase_2)/\left|all_{glue_{g_i}Ref}\right|$ ratio for each individual machine would not change as it does not depends on K, and is less than 1. Furthermore, the ratio t_{g_n}/t_{fetch_n} is also less than 1. On the other hand, the fraction of remote sub n-gram references $f \leq 1$, thus it contributes to increase G. However, a very low value of f would be required in order to compensate the combined effect of the above two factors and lead to G greater than 1. In the following chapters we discuss approaches for significantly reducing the value of f, corresponding to lowering the penalty of fetching the remote sub *n-gram* references by transforming most of them into local accesses to an n-gram cache.

Table 5.11 shows the problem size for phase two ($N_n(phase_2)$), the total number of sub n-gram references needed for glue calculation $g_{2\cdots6}$ ($\left|all_{glue_{g2\cdots6}Ref}\right|$) and the ratio of

the problem size over the needed references, when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 5.11: Phase two problem size and total number of sub n-gram references needed, in a single machine scenario ($K = 1$), when calculating glue $g_{2...6}$. *Corpus* size in words.

$\|C\|$	$N_n(phase_2) = \sum_{i=2}^{6} \|D_i\|$	$\left\|all_{glue_{g_{2...6}}Ref}\right\|$	$N_n(phase_2) / \left\|all_{glue_{g_{2...6}}Ref}\right\|$
$\mathbf{1.6 \times 10^7}$	5.90×10^7	4.11×10^8	0.14
$\mathbf{1.3 \times 10^8}$	4.11×10^8	2.96×10^9	0.14
$\mathbf{1.0 \times 10^9}$	2.73×10^9	2.05×10^{10}	0.13
$\mathbf{1.6 \times 10^{10}}$	2.78×10^{10}	2.26×10^{11}	0.12
$\mathbf{1.3 \times 10^{11}}$	1.15×10^{11}	1.02×10^{12}	0.11
$\mathbf{1.0 \times 10^{12}}$	3.24×10^{11}	3.07×10^{12}	0.11
$\mathbf{1.7 \times 10^{13}}$	4.57×10^{11}	4.40×10^{12}	0.10
$\mathbf{1.3 \times 10^{14}}$	4.57×10^{11}	4.40×10^{12}	0.10

Figure 5.18 shows, for phase two, the problem size, the total number of sub n-gram references needed for glue calculation $g_{2...6}$ and the ratio of problem size over the needed references, when the *corpus* size ranges from 2 Mw up to the *plateaux*.

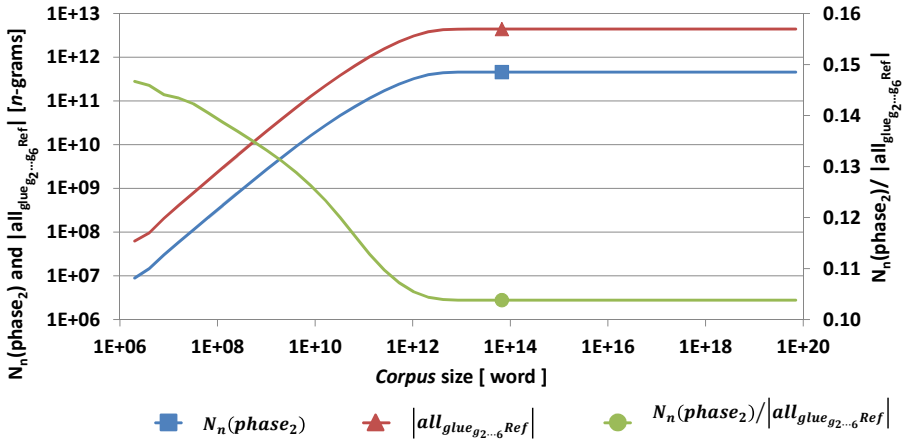
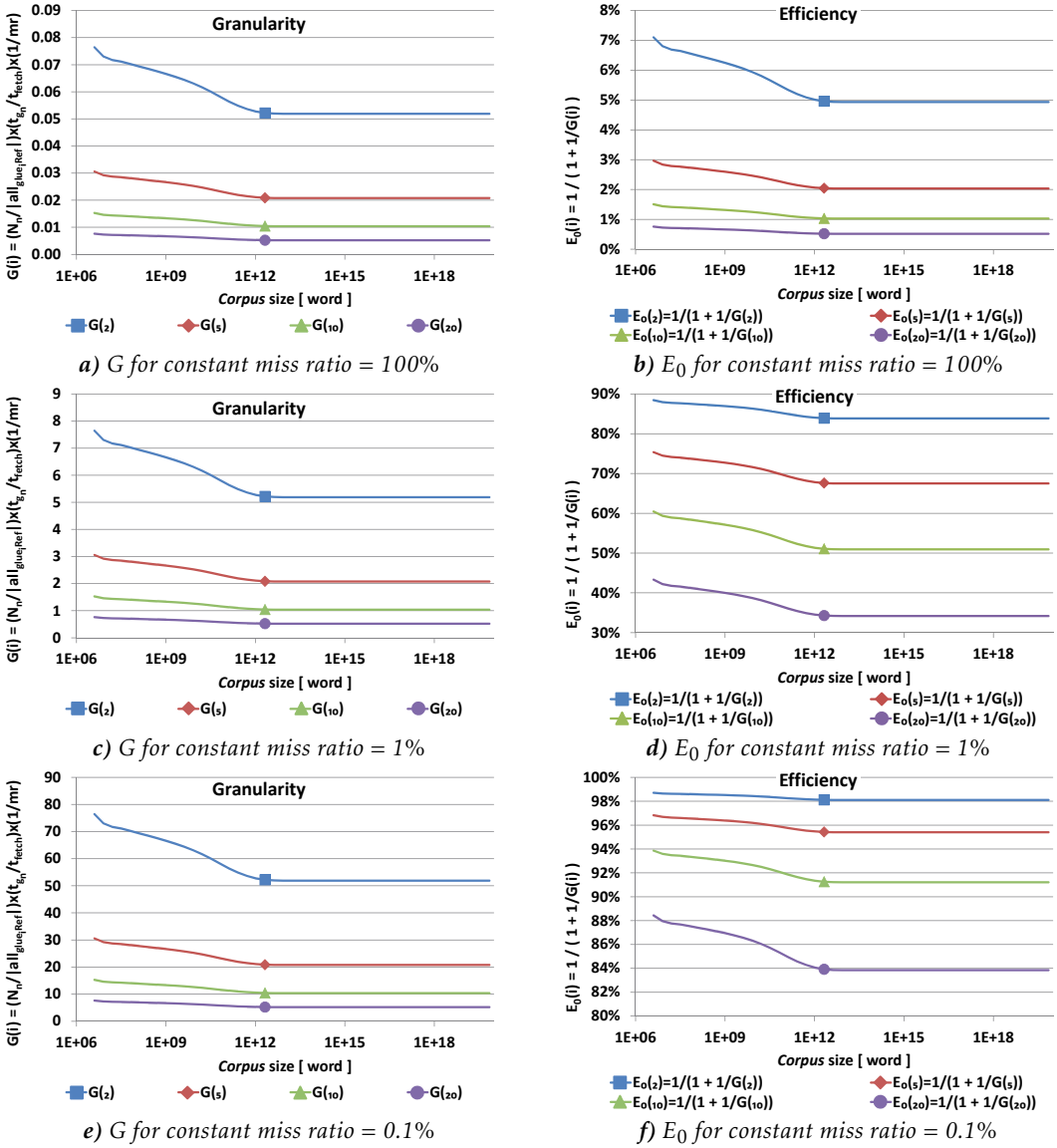


Figure 5.18: Phase two problem size and total number of sub n-gram references needed, in a single machine scenario ($K = 1$), when calculating glue $g_{2...6}$ and varying the *corpus* size. The left Y axis represents the phase two problem size and the total number of needed references in number of n-grams; the right Y axis represents the ratio $N_n(phase_2) / \left|all_{glue_{g_{2...6}}Ref}\right|$; and the X axis represents the *corpus* size in words.

Table 5.11 and Figure 5.18 illustrate that the ratio $N_n(phase_2) / \left|all_{glue_{g_{2...6}}Ref}\right|$ changes only slightly when the *corpus* size ranges from 2 Mw up to the *plateaux*, and stabilizes when the *plateaux* are reached.

Figure 5.19 shows, for a single machine case ($K = 1$), the granularity factor (G) and the corresponding values of efficiency (E_0) *versus* the *corpus* size when considering different values for the ratio t_{g_n}/t_{fetch} and for three cases of the n-gram cache miss ratio (mr): Figure 5.19-a) and 5.19-b) $mr = 100\%$; Figure 5.19-c) and 5.19-d) $mr = 1\%$; and Figure 5.19-d)

and 5.19-e) $mr = 0.1\%$. In all subfigures for each curve $G(i)$ and $E_0(i)$, the parameter i represents the ratio t_{fetch}/t_{g_n} with values: 2, 5, 10 and 20.



a) G *for constant miss ratio = 100%*

b) E_0 *for constant miss ratio = 100%*

c) G *for constant miss ratio = 1%*

d) E_0 *for constant miss ratio = 1%*

e) G *for constant miss ratio = 0.1%*

f) E_0 *for constant miss ratio = 0.1%*

Figure 5.19: Granularity and efficiency for phase two for a single machine case and glue $g_{2...6}$. The Y axis represents the G factor in the case of Figures a), c) and e); and the efficiency (E_0) in the case of Figures b), d) and f). The X axis represents the *corpus* size in words.

Figure 5.19 illustrates that the value of the miss ratio (mr) of an n-gram cache in phase two has a critical influence upon the granularity and efficiency in the full range of *corpus* sizes from about 2 Mw up to and including the *plateau* regions. Namely, without a cache, for $K = 1$, the phase two efficiency is always below 7% in all cases for the entire *corpus* range. In fact, for a given infrastructure the best that can be done to increase the $t_{g_n}/t_{n\text{–}gram}$ ratio is to ensure that the $t_{n\text{–}gram}$ average time corresponds to the value resulting from minimizing the per message latency, as discussed in section 5.5.5. Thus,

in the context of the Global method, our efforts to achieve an "acceptable" computation-to-communication granularity, at least greater than 1, which corresponds to an efficiency over 50% relative to an ideal sequential implementation, are focused on reducing the miss ratio of an n-gram cache as much as possible, as discussed in detail in chapters 6, 7, and 8.

5.5.4 Time Complexity of Phase Three

In this section we discuss the time complexity of phase three. Section 5.5.4.1 presents the execution model and section 5.5.4.2 presents an analysis of its computation-to-communica tion granularity.

5.5.4.1 Execution Model for Phase Three

The phase three problem size ($N_n(phase_3)$) for the evaluation of $re_{n_1 \cdots n_2}$ is the total number of relevant expression calculation operations for the range of n-gram sizes from $n_1 \geq 2$ to $n_2 > n_1$. This is equal to the sum of all the nonsingleton n-grams in the *corpus* from size n_1 up to n_2, which are included in the corresponding distinct n-gram tables. In the beginning of phase three these tables already include the calculated glue values for each n-gram.

Generated References

The phase three communication time component depends on the number of references generated ($|all_{\Omega_i Ref}|$) to the n-gram data stored in the distributed KVS server by each relevant expression calculation. According to the LocalMaxs Definition 2.2, on page 15, and Figure 5.20, each n-gram $W = (w_1\, w_2\, w_3\, w_4\, w_5)$ relevance calculation requires three glue values:

1. $g(W)$. In the beginning of phase three this value is already in the local n-gram input table in the entry containing the n-gram data for W;

2. The maximum of the set $\{g(X_L = (w_1\, w_2\, w_3\, w_4)), g(X_R = (w_2\, w_3\, w_4\, w_5))\}$, where $g(X)$ denotes the glue of the n-gram X. Obtaining these values requires two references to the distributed n-gram tables;

3. The maximum of the set $\Omega_{n+1}(W) = \{g(Y)\}$, such that Y satisfies $Y = Y_L$ or $Y = Y_R$ and $Y_L = \left(w_{Left}\, w_1\, w_2\, w_3\, w_4\, w_5\right)$ and $Y_R = \left(w_1\, w_2\, w_3\, w_4\, w_5\, w_{Right}\right)$ where w_{Left} and w_{Right} are unigrams appearing in the *corpus*. This maximum value is also already in the local table entry containing the n-gram data for W, because it was updated during the execution of phase two.

From the above analysis, excluding the two local references (items 1 and 3), we conclude that only two remote references (item 2) to the distributed n-gram table servers are required for each n-gram of size $i \geq 3$ whose relevance must be calculated in phase three,

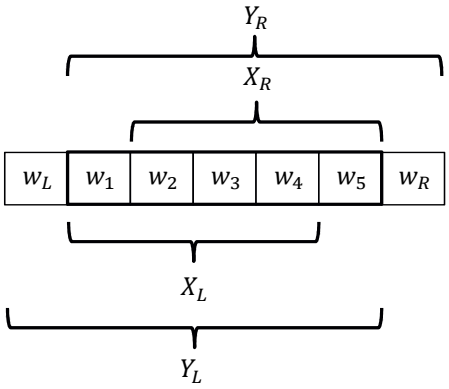


Figure 5.20: Glues needed for relevance evaluation.

thus $\left|all_{\Omega_i Ref}\right| = \left|NS_{All(3\cdots n)_{in}LocalPartition}\right| \times 2$, where $\left|NS_{All(3\cdots n)_{in}LocalPartition}\right|$ is the total number of nonsingleton n-grams with sizes from 3 up to n in the local table partition.

Execution Time

Assuming a strict sequential execution in each machine j, $1 \leq j \leq K$, the phase three execution time in each machine would be given by:

$$T_3(j) = T_{input}(j) + T_{Relevance}(j) + T_{comm}(j) + T_{output}(j) \tag{5.36}$$

where: $T_{input}(j)$ is the local input time; $T_{Relevance}(j)$ is the relevant expression calculation time, assuming that the required auxiliary n-grams are already present locally; $T_{comm}(j)$ is the remote auxiliary n-gram fetch time; and $T_{output}(j)$ is the local output time of the relevance value. The relevance calculation time is: $T_{Relevance}(j) = N_n(phase_3, j) \times t_{re_n}$, where t_{re_n} is the average time for the parsing and calculation of the relevance of any nonsingleton n-gram with size from 2 to n. The communication time is: $T_{comm}(j) = \left|all_{\Omega_i Ref}(j)\right| \times f(j) \times t_{fetch_n}$, where $\left|all_{\Omega_i Ref}(j)\right|$ is the total number of references to the needed auxiliary n-grams, as in Definition 15, on page 2.2, of LocalMaxs, $f(j)$ is the fraction of those references which are remote (i.e., the miss ratio), and t_{fetch_n} is the average time to remotely fetch any n-gram of size from 2 to $(n+1)$. Both t_{re_n} and t_{fetch_n} are experimentally measured. For bigrams $\left|all_{\Omega_i Ref}(j)\right| = 0$ because all the references are to the local KVS server (corresponds to the $(\Omega+1)$-set in Definition 2.2).

For longer n-grams, $i \geq 3$, $\left|all_{\Omega_i Ref}(j)\right| = 2 \times \left|NS_{All(3\cdots n)_{in}LocalPartition}\right|$ and $0 \leq f(j) \leq 1$ depending on the miss ratio of the n-gram cache system.

As in phase two, $T_{comm}(j)$ decreases with K, as long as there is no network congestion. By using multiple machines (K) in parallel, we can reduce the time for local relevance evaluation ($T_{Relevance}$) by a factor of K, while T_{comm} also decreases.

However, the execution within each machine is performed by multiple concurrent threads as illustrated in Figure 5.21:

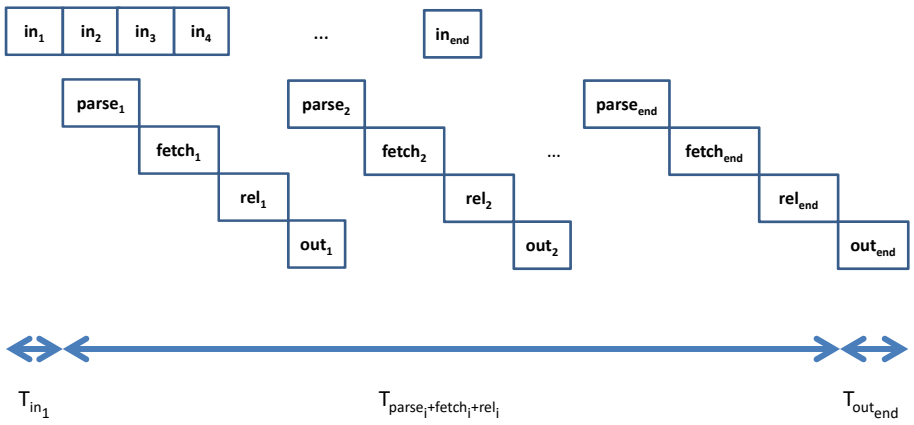


Figure 5.21: Phase three multithreaded operations sequence.

where in_i represents the input of a chunk i of n-grams from the local KVS server; $parse_i$ represents the parsing of a chunk i of data to identify the Ω sets; $fetch_i$ represents the fetching of the glues of the Ω sets from the KVS servers; rel_i verifies which are the n-grams that can be considered relevant expressions; and out_i is the output of the relevance results of a chunk i of n-grams to the local KVS server. Thus, the corresponding execution time in each machine is given by the following expression:

$$T_3(j) = T_{in_1}(j) + \left(T_{parse_i}(j) + T_{fetch_i}(j) + T_{rel_i}(j)\right) \times NumberOfChunks + T_{out_{end}}(j) \quad (5.37)$$

The parcels $T_{in_1}(j)$ and $T_{out_{end}}(j)$ can be neglected regarding the middle parcel, because they are both local operations to read or write a single chunk of n-grams. The middle parcel can be decomposed in two main components: i) $T_{parse}(j)$ and $T_{rel}(j)$ which correspond to the total accumulated sum of the parse and relevant expression identification calculation times for all chunks processed in each machine, i.e., is equal to $T_{Relevance}(j)$ in expression (5.36); ii) $T_{comm}(j)$ corresponds to the total time for fetching the Ω sets glues from the KVS servers. Thus, expression (5.37) can be simplified to:

$$T_3(j) \simeq T_{Relevance}(j) + T_{comm}(j) \quad (5.38)$$

5.5.4.2 Granularity Analysis for Phase Three

The computation-to-communication ratio is now given by:

$$G(j) = \frac{T_{Relevance}(j)}{T_{comm}(j)} = \frac{N_n(phase_3, j)}{\left|all_{\Omega_i Ref}(j)\right|} \times \frac{1}{f(j)} \times \frac{t_{re_n}}{t_{fetch_n}} \quad (5.39)$$

The ratio t_{re_n}/t_{fetch_n} is less than 1. In the case of bigrams there are no remote references required. For longer n-gram sizes than bigrams the use of an n-gram cache system with a miss ratio small enough contributes to increase G.

Table 5.12 shows the problem size for phase three ($N_n(phase_3)$) for $re_{2...5}$, the total number of n-gram references needed for evaluation of the relevance ($\left|all_{\Omega_{2...5}Ref}\right|$) and the

ratio of the problem size over the needed references, when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 5.12: Phase three problem size and total number of n-gram references needed, in a single machine scenario ($K = 1$), when evaluating relevance of bigrams up to pentagrams. *Corpus* size in words.

$\|C\|$	$N_n(phase_3) = \sum_{i=2}^{5} \|NS_i\|$	$\left\|all_{\Omega_{2...5}Ref}\right\|$	$N_n(phase_3) / \left\|all_{\Omega_{2...5}Ref}\right\|$
$\mathbf{1.6 \times 10^7}$	7.30×10^5	8.73×10^5	0.84
$\mathbf{1.3 \times 10^8}$	6.14×10^6	8.90×10^6	0.69
$\mathbf{1.0 \times 10^9}$	5.69×10^7	9.43×10^7	0.60
$\mathbf{1.6 \times 10^{10}}$	1.21×10^9	2.22×10^9	0.54
$\mathbf{1.3 \times 10^{11}}$	1.05×10^{10}	2.00×10^{10}	0.52
$\mathbf{1.0 \times 10^{12}}$	4.49×10^{10}	8.83×10^{10}	0.51
$\mathbf{1.7 \times 10^{13}}$	6.46×10^{10}	1.28×10^{11}	0.51
$\mathbf{1.3 \times 10^{14}}$	6.46×10^{10}	1.28×10^{11}	0.51

Figure 5.22 shows, for phase three, the problem size for $re_{2...5}$, the total number of remote n-gram references needed for evaluation of the relevance and the ratio of the problem size over the needed references, when the *corpus* size ranges from 2 Mw up to the *plateaux*.

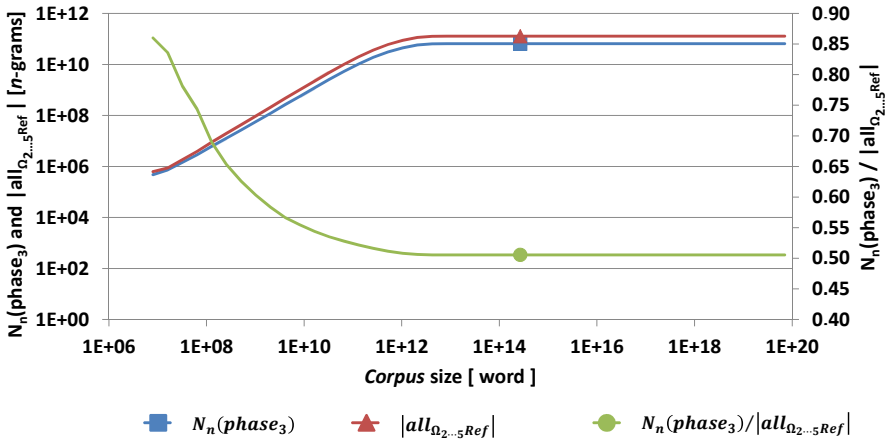


Figure 5.22: Phase three problem size and total number of remote n-gram references needed, in a single machine scenario ($K = 1$), when evaluating relevance of bigrams up to pentagrams and varying the *corpus* size. The left Y axis represents the phase three problem size and the total number of needed remote references in n-grams; the right Y axis represents the ratio $N_n(phase_3) / \left|all_{\Omega_{2...5}Ref}\right|$; and the X axis represents the *corpus* size in words.

5.5.4.3 Use of Cache on Phase Three

Although the use of a cache in phase three would contribute to increase the efficiency, the current implementation only uses an n-gram cache for phase two because the most significant communication overheads are due to glue calculation, and phase three represents

a small component in the total execution time.

5.5.5 Experimental Measurement of the $t_{n\text{–}gram}$ and t_{op} Times

The granularity, i.e., the G factor, depends on the ratio $T_0/T_{overheads}$. Due to the fact that the input and output operations in any phase — expression (5.16) in the case of phase one, expression (5.31) for phase two, and expression (5.36) for phase three — are local, the $T_{overheads}$ can be reasonably approximated by the communication component of each phase. The generic parameter $t_{n\text{–}gram}$ has the following interpretation according to the basic communication operations in each phase: i) Sending of n-gram counts in phase one (t_{send}); ii) Fetching n-gram frequencies or glues, respectively in phase two or three (t_{fetch}). Thus, for phase one $T_{overheads}(phase_1) \approx NumberOfItemsSent \times t_{send}$, where $NumberOfItemsSent$ is the total number of elements, each containing a frequency count of an n-gram, that are sent to the KVS servers, and t_{send}, corresponding to the $t_{n\text{–}gram}$ parameter, is the average time to send an n-gram in a range of sizes, e.g., 1, 2, 3 and 4. Likewise for phase two $T_{overheads}(phase_2) \approx NumberOfMissedNGramsPhase2 \times t_{fetch}$, where $NumberOfMissedNGramsPhase2$ is the total number of n-grams which must be fetched from the KVS servers during the glue calculations. For phase three $T_{overheads}(phase_3) \approx NumberOfMissedNGramsPhase3 \times t_{fetch}$, where the total number of n-grams which must be fetched from the KVS servers during the evaluation of relevance is represented by $NumberOfMissedNGramsPhase3$. The parameter t_{fetch}, corresponding to the $t_{n\text{–}gram}$ parameter, is the average time to fetch an n-gram in a range of sizes, e.g., 2, 3 and 4.

Both the t_{send} and t_{fetch} parameters depend on the network infrastructure and on the sending or fetching strategy, namely, the use of as large as possible output buffers (with size B) of multiple n-gram elements, to hide the latency of the interaction with the KVS servers. Besides, the above parameters are also affected by the allocation of virtual to physical machines that is automatically performed by the underlying cloud or cluster management layer.

In the following section we determine the order of magnitude of $t_{n\text{–}gram}$ based on experimental measurements conducted on the Amazon EC2 public cloud infrastructure using `m2.xlarge` instances. An instance `m2.xlarge` is a 64 bit machine, with 4 virtual CPU (vCPU), 34.2 GB of memory and moderate network performance (according to Amazon specifications [Ama17]).

5.5.5.1 Measuring $t_{n\text{–}gram}$

As illustrated in Figure 5.13, on page 100, the sending of n-gram counts by each machine during phase one is performed by a separate thread, thus it takes place asynchronously regarding the input and counting operations. However, for each output buffer of n-gram counts, the sender thread performs a synchronous request to the KVS servers, i.e., it waits for a positive acknowledge. In phase two and three, the glue calculation or relevance evaluation can not proceed until the n-gram data regarding the requested n-gram blocks

are delivered. Thus, for phase two and three (Figure 5.17 and Figure 5.21, respectively, on pages 109 and 115) a fetcher communication thread synchronously waits for the reply from the KVS servers with the requested n-gram data.

Due to the above approach, t_{send} and t_{fetch} have the same order of magnitude. However, due to the fact that phase two is the most critical phase regarding the execution time (in the experimental range of *corpus* size up to 1 Gw) we only investigated the behavior of the t_{fetch} parameter for phase two, as a function of the n-gram output buffer size (B).

Measuring t_{fetch} in Phase Two

We define t_{fetch} as the average time to receive an n-gram from a KVS server. Different experiments, conducted in different infrastructures, showed that sending n-grams in batch can affect the average value of t_{fetch}. To better understand what would be the best size of the batch buffer, denoted by B_0, that could lead to a minimum value of t_{fetch} we made the following experiment in different infrastructures (a private cluster, Amazon EC2 public cloud and Lunacloud public cloud):

1. Phase one of the LocalMaxs Global method was executed for a *corpus* size of 64 Mw and 10 KVS servers in order to build the n-gram tables from unigrams up to tetragrams;

2. For each n-gram size from 1 to 4, a test client was executed that fetches a set of random n-grams from each server by varying the number of servers (K) from 1 to 10. For each value of K the size of the output buffer (B) varied from 1 to 32×1024 n-grams, using in each case the same buffer size for all server requests. This simulates the behavior of the LocalMaxs Global method in phase two because, due to the hash-based distribution among the servers, the n-gram tables in each server keep an uniform number of n-grams. For each of these individual experiments and for each value of B, we measured the turnaround time $T_{buffer}(B, j)$ for getting a reply from each server j, $1 \leq j \leq K$.

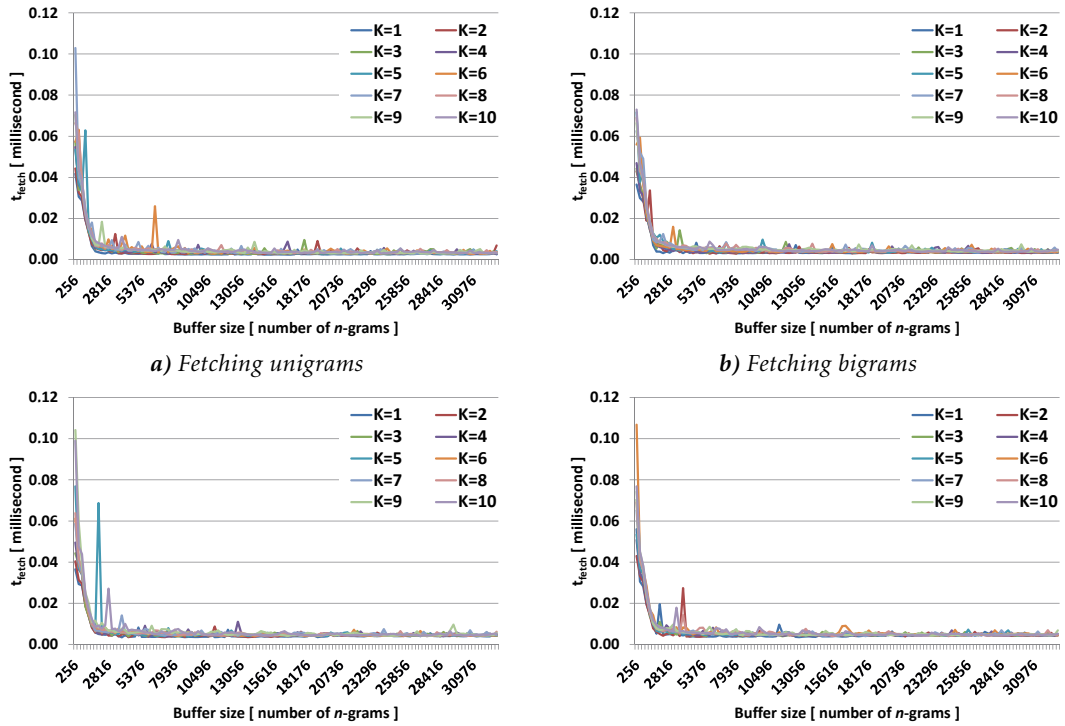
The average time ($\bar{T}_{buffer}(B, K)$) for receiving a buffer of size B, from each one of the K servers was calculated as:

$$\bar{T}_{buffer}(B, K) = \frac{\sum_{j=1}^{K} T_{buffer}(B, j)}{K} \tag{5.40}$$

The average time ($\bar{t}_{fetch}(B, K)$) for receiving an n-gram, for each of the above cases was calculated as:

$$\bar{t}_{fetch}(B, K) = \frac{\bar{T}_{buffer}(B, K)}{B} \tag{5.41}$$

As illustrated in Figure 5.23, the obtained values $\bar{t}_{fetch}(B, K)$ vary with B and with K. These are the results of the conducted experiments using the Amazon EC2 cloud



a) Fetching unigrams

b) Fetching bigrams

c) Fetching trigrams

d) Fetching tetragrams

Figure 5.23: Time for fetching ($\bar{t}_{fetch}(B,K)$) n-grams from different number of KVS servers *versus* the buffer size (B), using Amazon EC2 cloud `m2.xlarge` instances. The Y axis represents the average time to fetch an n-gram in millisecond; and the X axis represents the buffer size (B) in number of n-grams.

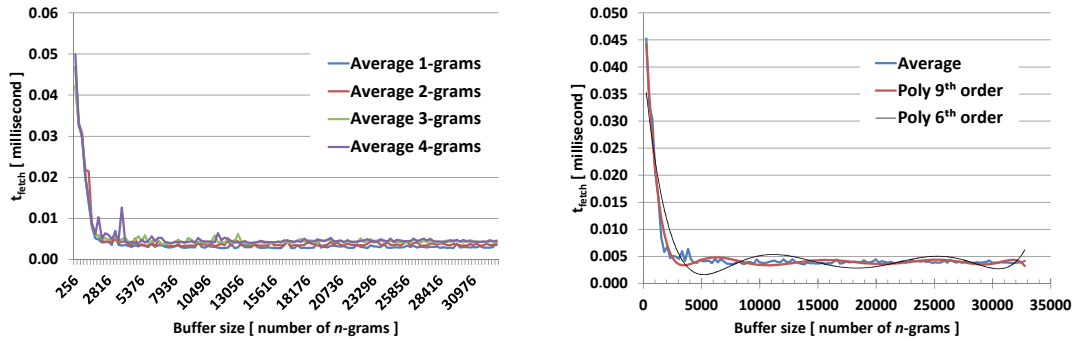
infrastructure, with `m2.xlarge` instances, to fetch n-grams (unigrams, bigrams, trigrams and tetragrams) from the KVS servers.

For the infrastructures used in our experiments, we observed that if the size of the output buffer request (B) is greater than a minimum of 2000 n-grams the communication time $\bar{t}_{fetch}(B,K)$ remains almost constant and is in the order of tens of microsecond. For other infrastructures the value of $\bar{t}_{fetch}(B,K)$ is different. Thus, the value of the output buffer B should not be lower than B_0, in order to ensure a minimum value of t_{fetch}. This is illustrated in Figure 5.24-a) that shows the average time ($\bar{t}_{fetch}(B,K)$) to fetch n-grams among the n-gram sizes 1 to 4 (unigrams, bigrams, trigrams and tetragrams) from the KVS servers. As we can see the average times to fetch an n-gram are similar irrespective of the n-gram size. Figure 5.24-b) shows the average time to fetch an n-gram, for n-gram sizes from 1 to 4, and two fitting curves. In this case the average time, for each output buffer size, was calculated over the values of t_{fetch} among unigrams, bigrams, trigrams and tetragrams.

The equations of the two fitting curves presented in Figure 5.24-b) are given by:

$$Poly^{6th} = m_6x^6 + m_5x^5 + m_4x^4 + m_3x^3 + +m_2x^2 + m_1x + m_0 \tag{5.42}$$

and



a) *Average time for fetching n-grams*

b) *Fitting curve for fetching n-grams*

Figure 5.24: Average time ($\bar{t}_{fetch}(B,K)$) for fetching n-grams from KVS servers and associated fitting curve in Amazon EC2 cloud using `m2.x1large` instances. The Y axis represents the average time to fetch an n-gram in millisecond; and the X axis represents the buffer size (B) in number of n-grams.

$$Poly^{9th} = m_9x^9 + m_8x^8 + m_7x^7 + m_6x^6 + m_5x^5 + m_4x^4 + m_3x^3 + m_2x^2 + m_1x^1 + m_0 \quad (5.43)$$

whose factors m_i are presented in Table 5.13.

Table 5.13: Polynomials factors for t_{fetch} fitting curves.

m_i factors	6^{th} order	9^{th} order
m_0	$4.0e^{-2}$	$5.6e^{-2}$
m_1	$2.2e^{-5}$	$-5.1e^{-5}$
m_2	$4.4e^{-9}$	$-1.9e^{-8}$
m_3	$-4.2e^{-13}$	$-3.8e^{-12}$
m_4	$2.0e^{-17}$	$4.4e^{-16}$
m_5	$-4.8e^{-22}$	$-3.1e^{-20}$
m_6	$4.4e^{-27}$	$1.3e^{-24}$
m_7		$-3.5e^{-29}$
m_8		$5.2e^{-34}$
m_9		$-3.2e^{-39}$

In all the cases the average time is minimized if $B > B_0$. Concerning the communication with the KVS servers during phase one, each controller must ensure that the output buffer used for sending the n-gram frequencies has a size above the buffer size threshold B_0 which, for each infrastructure, ensures that the average time is minimal.

In phase two, for glue calculation purposes, each table (D_n) of distinct n-grams of size n is analyzed to identify the included sub n-grams of sizes i from 1 up to $(n-1)$. For each size i there is a number of sub n-grams equal to $2 \times |D_n|$. As each table D_n is processed as a series of fixed size (IBS) chunks of n-grams, in each chunk the number of included sub n-grams of size i is equal to $2 \times IBS$ for i from 1 up to $(n-1)$.

Because the hash based distribution of the n-grams across the KVS servers is practically uniform, in this approximations the needed sub n-grams are also assumed distributed uniformly across the KVS servers. So in the presence of an n-gram cache C_i for n-grams of size i, with an individual miss ratio mr_i, for each chunk of size IBS, the average number of sub n-grams of size i missing in the cache C_i can be estimated by:

$$2 \times IBS \times mr_i \tag{5.44}$$

When calculating glue g_n (for n-grams of size n), it is necessary to estimate the above average number of missing sub n-grams in each of the chunks, corresponding to the sub n-grams of sizes from 1 to $(n-1)$ that is: $2 \times IBS \times mr_1$ for the subunigrams, $2 \times IBS \times mr_2$ for the subbigrams, ..., $2 \times IBS \times mr_{(n-1)}$ for the sub $(n-1)$-grams.

As discussed above, the minimal size of the buffer used to fetch the sub n-grams from each KVS in order to minimize the average time (t_{fetch}) is denoted as B_0. Thus, when having K servers, it is necessary to collect a number of sub n-grams, as a result of the above chunk processing, which must be big enough to ensure that each server gets a request for at least B_0 n-grams leading to:

$$2 \times IBS \times mr_i > K \times B_0 \tag{5.45}$$

As we are able to estimate the value of the cache miss ratio (mr_i) for each n-gram size, then for each configuration of K machines there is a minimal size IBS that must be imposed in order to satisfy the expression (5.45).

As discussed in the following chapters, the value of *mr* considered in the above expressions is obtained from the cache analytical modeling, and/or from the experimental measurements taken during real execution. Experimental results are presented in chapters 6, 7 and 8.

5.5.5.2 Measuring the Basic Operation Time in Phase Two (t_{g_n})

The average time spent in each basic glue operation is denoted by $t_{g_n}(g_i)$ for the glue of n-grams with size i, for example, $t_{g_n}(g_2)$ in the case of the bigrams glue g_2. It depends on the implementation of the basic operation on each specific execution environment. In the following we present the obtained values of t_{g_n} when calculating the glues g_2, g_3 and g_4 for four different *corpus* (64, 128, 256 and 511 Mw) and the number of machines varying from 3 to 18, in the Amazon EC2 public cloud (with the same type of machines that were used to determine the t_{fetch} time presented in section 5.5.5.1).

Table 5.14 shows the number of distinct n-grams for the *corpora* used in the determination of the values of t_{g_n} for bigrams, trigrams and tetragrams.

Table 5.14: Number of distinct n-grams for the *corpora* used in the determination of t_{g_n}.

$\|C\|$ [Mw]	$\|D_1\|$	$\|D_2\|$	$\|D_3\|$	$\|D_4\|$
64	1 591 515	11 855 860	31 413 238	47 477 297
128	2 620 322	19 915 076	56 537 305	89 860 495
256	4 316 177	33 195 356	100 792 182	168 630 935
511	7 129 215	54 834 546	177 770 692	313 683 177

Figure 5.25 shows the obtained values of t_{g_n} for bigrams, trigrams and tetragrams.

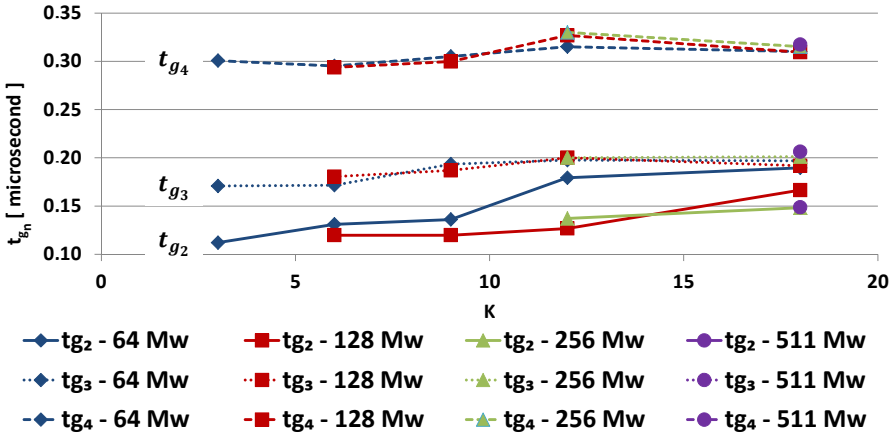


Figure 5.25: Obtained t_{g_n} times for glues g_2, g_3 and g_4 for *corpus* sizes from 64 to 511 Mw. The filled lines represent $t_{g_n}(g_2)$ shortened to t_{g_2}, the dotted lines represent $t_{g_n}(g_3)$ shortened to t_{g_3}, and the dashed lines represent $t_{g_n}(g_4)$ shortened to t_{g_4}. Each line is marked with several points corresponding to the different *corpus* sizes and numbers of machines. The Y axis represents the average time needed for a glue operation (g_i) in microsecond; and the X axis represents the number of machines.

For each *corpus* size and number of machines (K), the average time for a glue operation g_i was measured for each individual machine from 1 to K, and the average of these values over the K machines was calculated: each curve in the figure shows the calculated average time for each *corpus* size and value of K when the total number of machines was varied from 3 to 18. The obtained values for this average time are consistent with the fact that the basic glue operation time does not depend on the *corpus* size nor on the number of machines.

For each fixed *corpus* and value of K, the average time for a glue operation increases from the g_2 to the g_4 cases due to the increased complexity of the glue calculations when going from bigrams to tetragrams. Also for each fixed *corpus* the average glue operation time stays practically constant with K, as expected since only local operations are involved.

Figure 5.26 shows the average values of t_{g_n} for g_2 up to g_4, obtained by averaging of the values shown in Figure 5.25 over the four *corpus* sizes (64 to 511 Mw) and numbers of machines (3 to 18).

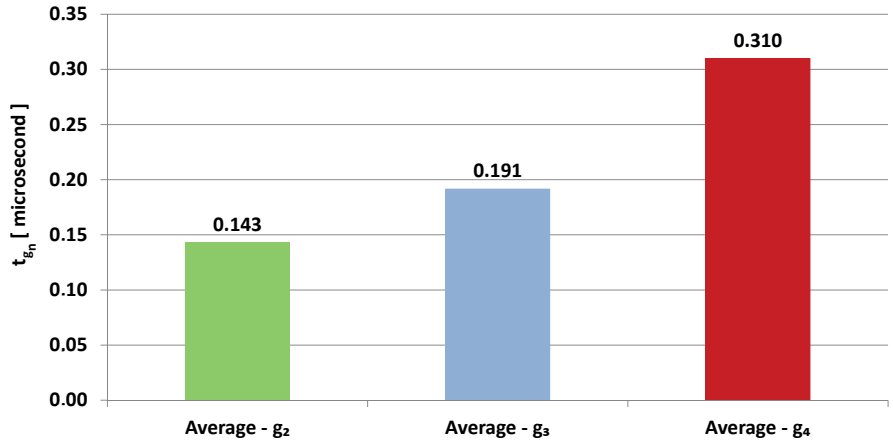


Figure 5.26: Average obtained t_{g_n} times for glues g_2, g_3 and g_4 with *corpus* sizes ranging from 64 to 511 Mw using from 3 to 18 `m2.xlarge` machine instances. The Y axis represents the average values of tg_n in microsecond and the X axis represents the glue calculation (g_3, g_3 or g_4).

From the above, we can calculate the average time for the basic operation ($t_{g_n}(g_{234})$) of the combined glue g_{234}, which represents the average time of a basic glue operation for bigrams, trigrams or tetragrams as follows:

$$t_{g_n}(g_{234}) = \frac{t_{g_n}(g_2) \times |D_2| + t_{g_n}(g_3) \times |D_3| + t_{g_n}(g_4) \times |D_4|}{|D_2| + |D_3| + |D_4|} \tag{5.46}$$

By using the number of distinct n-grams presented in Table 5.14 and the values of $t_{g_n}(g_2)$, $t_{g_n}(g_3)$ and $t_{g_n}(g_4)$ presented in Figure 5.26 we obtain the values of t_{g_n} (for glue g_{234}) presented in Figure 5.27.

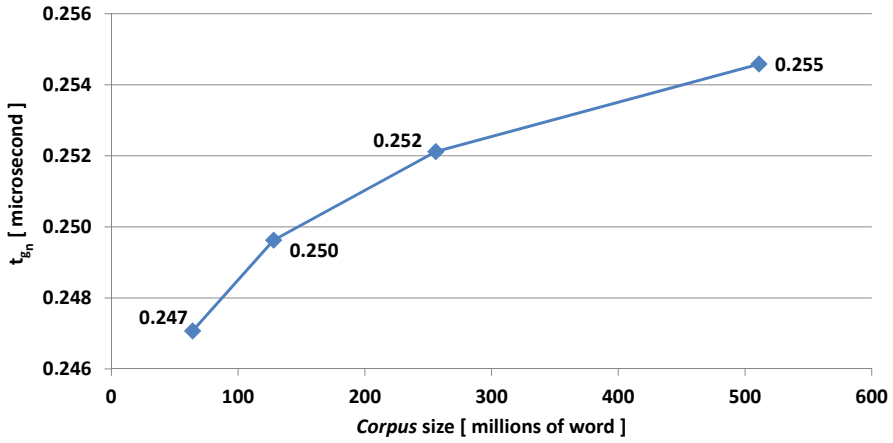


Figure 5.27: Calculated t_{g_n} for glue g_{234} *versus corpus* size for *corpus* sizes ranging from 64 to 511 Mw using from 3 to 18 `m2.xlarge` machine instances. The Y axis represents the value of $t_{g_n}(g_{234})$ in microsecond and the X axis represents the *corpus* size in millions of words.

Figure 5.27 shows that the value of $t_{g_n}(g_{234})$ slightly increases with the *corpus* size,

in the observed *corpus* range, due to the relative increase of the population of distinct tetragrams over the population of distinct trigrams as well as over the bigrams.

By using the theoretical model presented in chapter 3, Figure 5.28 shows the evolution of t_{g_n} as a function of the *corpus* size, from 2 Mw until the *plateau* regions. When the *corpus* size grows to the *plateaux* the value of $t_{g_n}(g_{234})$ tends to a constant value around 0.272 microsecond (for the Amazon EC2 public cloud using `m2.xlarge` instances). This was obtained by using the distinct numbers of n-grams presented in Table 3.8, on page 40.

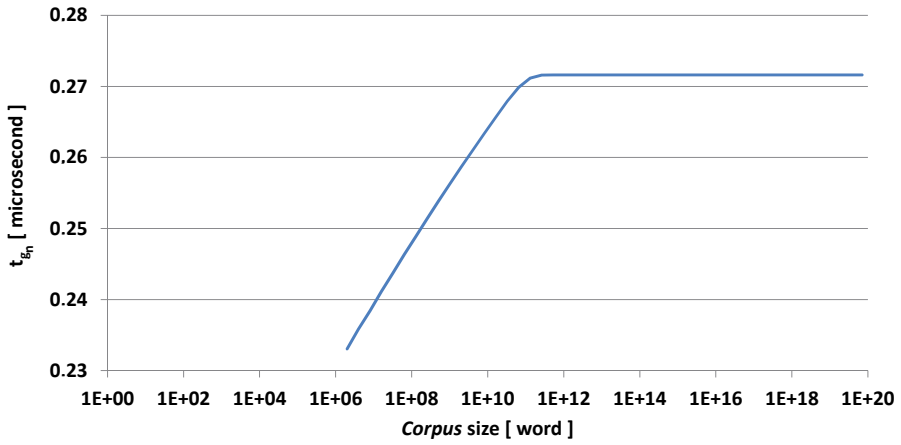


Figure 5.28: Calculated $t_{g_n}(g_{234})$ when *plateaux* are reached. The Y axis represents the value of t_{g_n} in microsecond and the X axis represents the *corpus* size in words.

Influence of the Infrastructure on the Value t_{g_n}

The value of t_{g_n} depends on the infrastructure used. Table 5.15 and Figure 5.29 show different values of t_{g_n}, for different values of glue and different infrastructures or different system implementations of system virtual machines.

Table 5.15: Values of t_{g_n}, expressed in microsecond, for different execution environments.

glue	*Amazon* (*m2.xlarge*)	*Amazon* (*r4.2xlarge*)	*LunaCloud* (*VMware*)	*LunaCloud* (*Containers*)
g_2	0.14	0.08	0.09	0.34
g_3	0.19	0.09	0.15	0.39
g_4	0.31	0.12	0.20	0.49
g_5		0.17	0.24	
g_6		0.20	0.33	
g_7			0.49	

For example in the case of Amazon EC2 cloud the instances types used were `m2xlarge`, for glue g_2 up to g_4, and `r4.2xlarge`, for glue g_2 up to g_6. An instance `r4.2xlarge` is a 64 bit machine, with 8 vCPU, 61 GB of memory and network bandwidth up to 10 *Gbit/s*. In the case of the LunaCloud infrastructure in all cases each virtual machine was configured with 4 vCPU, 64 GB of memory and a network bandwidth with 10 *Gbit/s*. The label "VMware" means that the system implementation of virtual machines was

supported using VMware technologies and the label "Containers" means that the system implementation of virtual machines was supported using Linux containers. The sizes of the *corpus* used ranged from 50 up to 682 Mw.

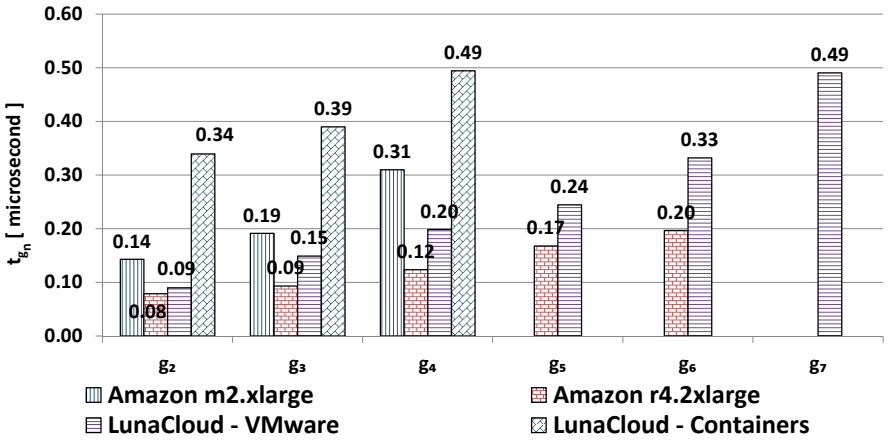


Figure 5.29: Range of obtained t_{g_n} times for glues g_2 up to g_7 with *corpus* sizes ranging from 50 to 682 Mw for different execution environments.

Figure 5.30 shows the variation of $t_{g_n}(g_{234})$, when using different infrastructures with *corpus* sizes ranging from 50 to 682 Mw.

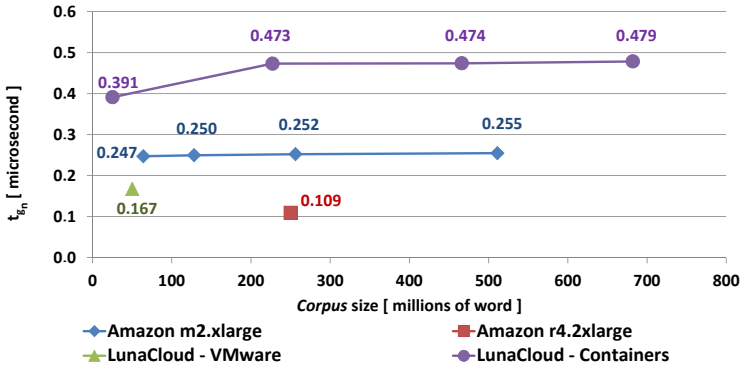


Figure 5.30: Obtained t_{g_n} for g_{234} *versus corpus* size in different execution environments with *corpus* sizes ranging from 50 to 682 Mw. The Y axis represents the value of t_{g_n} in microsecond and the X axis represents the *corpus* size in millions of words.

In all cases the value of t_{g_n}, as defined in expression (5.46), slightly increases with the *corpus* size. However, for a fixed sized *corpus* the absolute value of t_{g_n} depends on the infrastructures (types of Virtual Machine (VM)) used.

Figure 5.31 shows the variation of $t_{g_n}(g)$, for different glue calculations (g_2, g_{23}, g_{234}, $g_{2\cdots5}$ and $g_{2\cdots6}$), when using Amazon `r4.2x1arge` instances, when the *corpus* size ranges from 2 Mw up to the *plateau* regions.

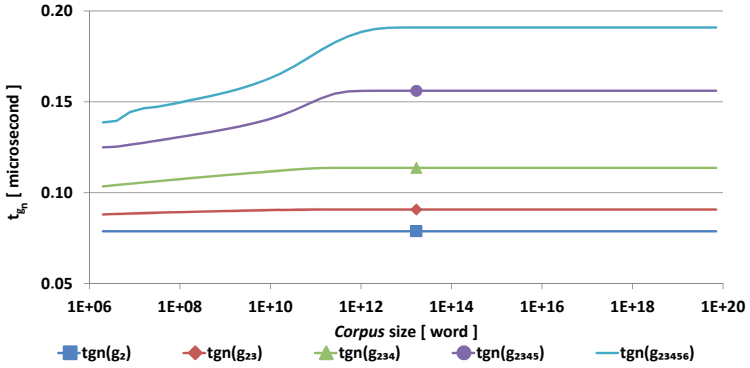


Figure 5.31: Obtained t_{g_n} for different combined glue calculations *versus corpus* size using Amazon `r4.2x1arge` instances *versus corpus* size. The Y axis represents the value of t_{g_n} in microsecond and the X axis represents the *corpus* size in words.

The value of t_{g_n} increases when the glue calculations become more complex, i.e., when going from g_2 to $g_{2\cdots6}$.

5.5.5.3 Revisiting Phase Two Granularity with a Real Time Ratio (t_{g_n}/t_{fetch})

From Figure 5.24-b) we can observe that the minimum t_{fetch} is around 5 microsecond and from Figure 5.27 the average value of t_{g_n} (g_{234}) is about 0.251 microsecond (for a range of *corpus* sizes from 64 Mw to 511 Mw). Thus, in the case of Amazon EC2 cloud when using `m2.x1arge` instances, for phase two the ratio t_{g_n}/t_{fetch}, for glue calculation g_{234}, is given approximately by:

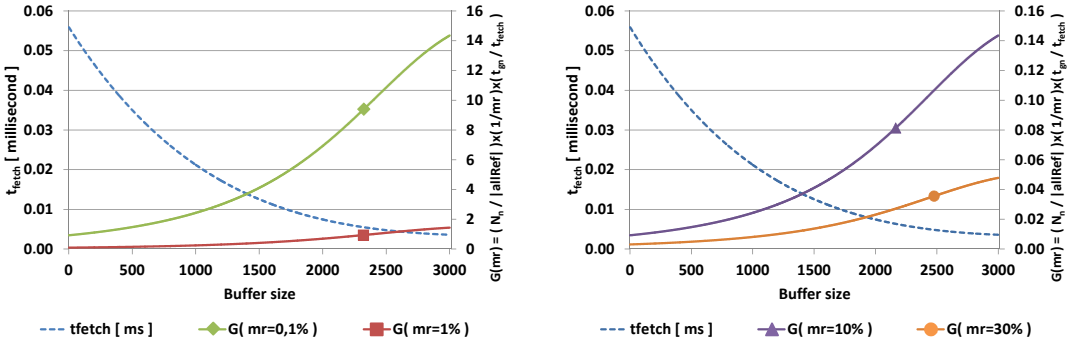
$$\frac{t_{g_n}}{t_{fetch}} = \frac{0.251\ \mu s}{5\ \mu s} \approx \frac{1}{20} \tag{5.47}$$

Using the above indicative value in expression (5.47) we can estimate the order of magnitude of the granularity and efficiency of phase two, compared to the ideal machine case. Table 5.16 shows the values of the granularity G and efficiency (E_0), for a range of *corpus* sizes from 64 to 511 Mw and glue calculation g_{234}, which exhibits the indicated average ratio $\bar{S_n} = N_n/\left|all_{g_{234}Refs}\right|$, and considering different values of the miss ratio from 0.1% up to 30%.

Table 5.16: Influence of the miss ratio on the granularity factor (G) and efficiency (E_0) in the case of glue calculation g_{234} and *corpus* from 64 to 511 Mw.

mr	$\bar{S_n}$	***G***	E_0
0.10%	0.205	10.309	91.16%
1.00%	0.205	1.031	50.76%
5.00%	0.205	0.206	17.09%
10.00%	0.205	0.103	9.35%
15.00%	0.205	0.069	6.43%
30.00%	0.205	0.034	3.32%

Figure 5.32 shows the variation of the granularity G using expression (5.34) for phase two as a function of the batch output buffer size (B) for different values of miss ratio. The variation of G is shown in Figure 5.32-a) for $mr = 0.1\%$ and $mr = 1\%$, and in Figure 5.32-b) for $mr = 10\%$ and $mr = 30\%$. The curve of t_{fetch} as a function of B is also shown in this figure.

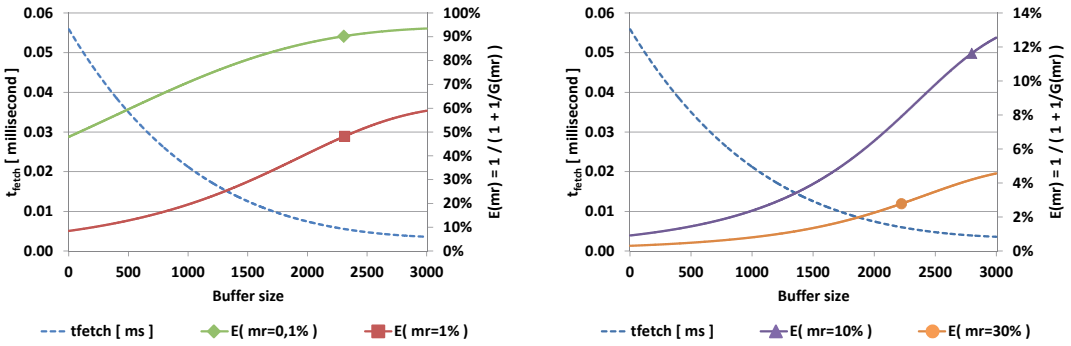


a) *G factor for mr = 0.1% and mr = 1%* ***b)*** *G factor for mr = 10% and mr = 30%*

Figure 5.32: Granularity *versus* buffer size for different miss ratio values. The left Y axis represents the average time to fetch an n-gram (t_{fetch}) shown in the dotted curve, in millisecond; the right Y axis represents the granularity factor shown in the filled curves; and the X axis represents the buffer size in number of n-grams.

The values of t_{fetch} as a function of the batch buffer size (B) were determined using the expression (5.43), on page 120, and the polynomial factors presented in column "9^{th}order" of Table 5.13, on page 120, when B varies from 1 to 3000 n-grams, for different values of miss ratio: 1%, 5%, 10%, 15% and 30%.

Figure 5.33 shows the variation of the efficiency E_0 and using the expression (5.13), for the corresponding values of granularity presented in Figure 5.32. The variation of the efficiency is shown in Figure 5.33-a) for $mr = 0.1\%$ and $mr = 1\%$, and in Figure 5.33-b) for $mr = 10\%$ and $mr = 30\%$.



a) *Efficiency for mr = 0.1% and mr = 1%* ***b)*** *Efficiency for mr = 10% and mr = 30%*

Figure 5.33: Efficiency *versus* buffer size for different miss ratio values. The left Y axis represents the average time to fetch an n-gram (t_{fetch}) shown in the dotted curve, in milliseconds; and the right Y axis represents the efficiency (E_0) shown in the filled curves; and the X axis represents the buffer size in number of n-grams.

5.5.6 Discussion of the Time Complexity of the Global Method

Execution of phase one is amenable to a parallel decomposition of the processing of the frequency counts among the available machines in such a way that the communication overheads are kept small enough to allow speedup (relative to an ideal sequential machine) close to linear, by choosing an appropriate task granularity for each problem size. On the contrary, in phase two the communication overhead is a significant component of the phase execution time. Thus, our major efforts were focused on improving the performance of phase two.

For the range of experimentally analyzed *corpora* up to 1 Gw we observed $T_2 > T_1 > T_3$, that is phase two time (T_2) is responsible for the major component of the total execution time, and phase three time (T_3) represents the minor component. This behavior was confirmed from our experimentation within that *corpora* range, where we found out that phase two dominates the execution time of the LocalMaxs Global method. Due to the monotonic increase of all the repetition factors for all the *n*-gram sizes with growing *corpora* sizes, the time complexity of phase one time (T_1), which grows linearly with the *corpus* size, becomes progressively more significant as discussed in chapters 9.

The significant communication overhead of phase two arises due to the need to fetch a large number of remote sub *n-grams* for glue calculation. In the following we discuss the design of an *n-gram* cache with the objective of reducing the above communication penalties, i.e., corresponding to lowering the values for the miss ratio (*mr*). We extend the above analytical model of the time complexity of phase two to incorporate the use of the *n-gram* cache, in order to predict the expected values of miss ratio and miss time penalties for different *corpora* sizes and numbers of machines. For different cache organizations we show that the experimental results obtained with the real implementation agree with the model estimates. We also show how the use of an *n-gram* cache contributes to a significant reduction in the execution time of phase two of the LocalMaxs Global method algorithm.

5.6 Chapter Summary

From the previous, albeit preliminary analysis, we conclude that the most critical issue in the implementation of the Global method for LocalMaxs is the low computation-to-communication granularity in phase two. As a result, we identify two main strategies to increase the above mentioned granularity: i) To increase the phase two problem size ($N_n(phase2)$) while keeping the same number of total sub *n*-gram references ($\left|all_{glue_g Ref_{n-gram}}\right|$), which can be achieved by boosting the reutilization of those subreferences through the cooperative evaluation of combined glues, and also by increasing the problem computation workload with other *n*-gram based algorithms that share the same statistical *n*-gram data; ii) To decrease the cache system miss ratio.

In order to evaluate the impact of alternative cache management strategies upon the reduction of the cache system miss ratio under the influence of the LocalMaxs Global

method algorithm implementation, we proceeded as follows:

1. We considered the design and implementation of a dynamic cache system, i.e., with an on-demand n-gram fetch strategy, and evaluated its cold-start and warm-start behaviors, for different *corpus* sizes, different combinations of glue calculations, and different numbers of machines;

2. In order to evaluate the possibilities of reducing the overheads due to the cold-start misses in the dynamic cache, we considered the design and implementation of a static cache, that is completely filled in before the execution of phase two starts. Furthermore, the cache prefetching relies on the concept of FAset presented in chapter 3, and its implementation takes advantage of the overlapped execution between the FAset loading and the glue calculations in phase two. Thus, it contributes to partially hiding the cold-start misses overhead;

3. To take advantage of the possibility of filtering the n-gram singletons appearing in the cache input reference stream, a Bloom filter was integrated as a stage within the cache system.

The three above issues are discussed, respectively, in chapters 6, 7, and 8.

Part IV

n-gram Cache Models

Chapter 6

An *n*-gram On-demand Fetch Dynamic Cache System for LocalMaxs

Dynamic n-gram cache system with on-demand fetch.

This chapter describes the execution of the parallel and distributed implementation of the LocalMaxs Global method with a dynamic *n*-gram cache system. In this study we only considered the use of the cache system in phase two. It is organized in 6 sections. In section 6.1 we present the generic structure of the *n*-gram cache system. The on-demand dynamic cache system is presented in section 6.2. The evaluation of the cache system using cold-start and warm-start strategies is presented in section 6.3 and 6.4. The experimental results are presented in section 6.5. The chapter summary is presented in section 6.6.

6.1 Generic Structure of the *n*-gram Cache System

In this thesis we only considered the usage of an *n*-gram cache system in phase two of the Global method because it is the phase that exhibits the most significant communication overheads due to the fetching of *n*-gram data. The repeated sub *n-gram* occurrences generated by glue calculations justify an *n-gram* cache system in each machine j, $1 \leq j \leq K$, with one individual cache C_i for each *n-gram* size (C_1, ..., C_{n-1}). Figure 6.1 shows the generic structure of the cache system, with the input stream of sub *n-gram* references generated ($all_{glue_{g_{n_1 \cdots n_2}} Ref}(j)$) during the glue calculation $g_{n_1 \cdots n_2}$ and how those references are distributed among the individual caches: C_1 responsible for handling the stream of unigram references ($all_{glue_{g_{n_1 \cdots n_2}} Ref_{1\text{–}gram}}(j)$); C_2 responsible for handling the bigram references ($all_{glue_{g_{n_1 \cdots n_2}} Ref_{2\text{–}gram}}(j)$), and so on.

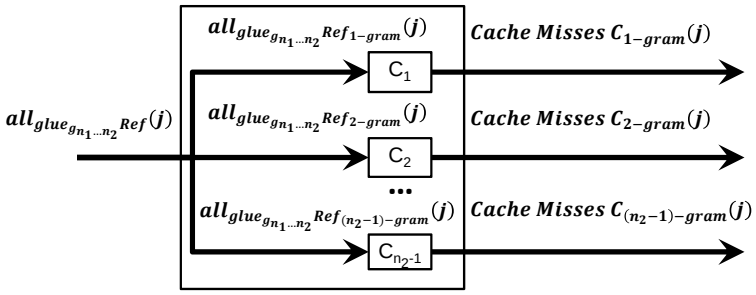


Figure 6.1: *n*-gram cache system in each machine j, $1 \leq j \leq K$.

The number of *n-gram* misses for the cache C_i ($\left|Cache\,Misses\,C_{i-gram}(j)\right|$), $1 \leq i \leq (n-1)$, is obtained as:

$$\left|Cache\,Misses\,C_{i-gram}(j)\right| = \left|all_{glue_{g_{n_1 \cdots n_2}} Ref_{\,i-gram}}(j)\right| \times mr_i(j) \tag{6.1}$$

where $mr_i(j)$ is the corresponding individual cache miss ratio. The entire system can be considered globally as a composite cache system in each machine j, with a global miss ratio $mr(j)$, whose input is given by the stream $all_{glue_{g_{n_1 \cdots n_2}} Ref}(j)$, and the output is the stream of missed references $Cache\,Misses(j)$ whose cardinality is given by the sum of the missed references from the individual caches. Thus, the total number of misses of the cache system in each machine j is given by:

$$\begin{aligned}\left|all_{glue_{g_{n_1 \cdots n_2}} Ref}(j)\right| \times mr(j) = {} & \left|all_{glue_{g_{n_1 \cdots n_2}} Ref_{\,1-gram}}(j)\right| \times mr_1(j) + \cdots + \\ & \left|all_{glue_{g_{n_1 \cdots n_2}} Ref_{\,i-gram}}(j)\right| \times mr_i(j) + \cdots + \\ & \left|all_{glue_{g_{n_1 \cdots n_2}} Ref_{\,(n_2-1)-gram}}(j)\right| \times mr_{(n_2-1)}(j)\end{aligned} \tag{6.2}$$

The *n*-gram cache system is characterized according to the following dimensions.

6.1.1 Cache Model — Finite *versus* Infinite Capacity

In this thesis, concerning the cache capacity we assume an infinite cache, in the sense that all the distinct sub *n-grams* of each size occurring in each table partition can be contained in the corresponding cache system in each machine. By applying the theoretical model that was presented in chapter 3 or as a result of executing phase one of the LocalMaxs Global method we can calculate the total numbers of distinct sub *n*-grams, thus for each given *corpus* size we can exactly determine the maximum cache capacity for each *n-gram* size required to ensure the infinite cache assumption.

Indeed, due to the logarithmic evolution of the number of distinct *n-grams* with respect to the *corpus* size and their asymptotic *plateaux*, there is a maximum finite required cache capacity, which remains constant when the distinct *n-gram plateaux* are reached. In our analysis we studied the behavior of the cold or first occurrence misses, which happen

when the cache is empty at the beginning of the execution, being the only type of misses occurring in an infinite cache. We show how we take advantage of the inherent repeatability of the *n-gram* occurrences in the *corpus*, and how this gets better for growing *corpora* sizes, mostly for the lower *n-gram* sizes.

This assumption can be enforced by the implementation by configuring the number of machines needed to execute phase two of LocalMaxs Global method in order to providing enough local memory to each machine. Otherwise, the infinite cache assumption does not hold. Nevertheless, the current implementation of the Global Method is able to handle also the case of a finite capacity cache, by processing the first occurrence (cold-start misses), as well as the capacity misses.

6.1.2 Cache Management — Dynamic, Static, and Static plus Dynamic

As a result of phase one of LocalMaxs Global method, the total number of distinct *n*-grams and their individual number of occurrences are calculated. These *n*-grams data can be loaded in anticipation in the cache systems to be used in phase two according to a static prefetch strategy. However, as the *n*-grams have different individual frequencies of occurrences, some having much higher numbers than others, their efficiency towards increasing the cache hit ratio are also different. As previously discussed, by using the FAset concept, we are able to identify the most adequate *n*-gram subsets to ensure a desired total number of occurrence counts which leads to cache hits in an infinite cache. Although this allows to implement a FAset-based static prefetch cache strategy, it is only useful if the time penalty for loading the FAset elements can be hidden through overlapping its execution with other useful computations (as discussed in detail in chapter 7). In the absence of such a strategy, the only mechanism that is provided by default is the on-demand fetch of the *n*-gram data as they are referenced during phase two execution. Additionally, the information on the numbers of *n*-gram occurrences can be used to support a dynamic cache replacement strategy in case of a finite cache system.

Due to the significant overhead of the cold-start misses, it is desirable to take advantage of the reutilization of the already fetched *n*-gram data. On one hand this is achieved by exploring the inherent repetition of the individual *n*-grams in the *corpus*. On the other hand this can be enhanced by a proposed cache warming-up strategy based on the combination of different glue calculations, similarly to the effect achieved by cooperative shared caches. Thus, for the cache evaluation in phase two, we consider isolated and combined glue calculations. Table 6.1 shows the number of glue calculation tasks using each cache, e.g., g_2 only uses C_1 while $g_{2\ldots6}$ has 5 tasks using C_1, 4 tasks using C_2, etc. A combined glue calculation is preferable as it contributes to an increased cache utilization, reducing the number of remote sub *n*-gram fetches.

This latter effect could be further boosted by supporting multiple concurrent users sharing the cache system, for example by executing multiple instances of the LocalMaxs or other *n*-gram based algorithm.

Table 6.1: Number of caches used by each glue calculation.

glue	C_1	C_2	C_3	C_4	C_5
g_2	1				
g_3	1	1			
g_4	1	1	1		
g_5	1	1	1	1	
g_6	1	1	1	1	1
g_{23}	2	1			
g_{234}	3	2	1		
$g_{2\cdots5}$	4	3	2	1	
$g_{2\cdots6}$	5	4	3	2	1

In any case of the characteristics of the LocalMaxs functions (glue or relevance calculation), all references to the cache correspond to read-only accesses. We evaluate the miss ratio of the cache system in order to understand its impact on the reduction of the percentage of remote n-gram references when compared to a system without cache. In order to understand the influence of the cache upon the absolute execution time, we also evaluate the cache miss time penalty. While the miss ratio only reflects the intrinsic properties of the LocalMaxs algorithm and the efficiency of the cache design, the miss time penalty, defined as $MissTimePenalty = MissPenalty \times t_{fetch}$, where $MissPenalty = NumberOfNGramsMisses$ and t_{fetch} is the average time to fetch an n-gram from the remote KVS server (chapter 5), provides additional information on the effective time overhead depending on the network infrastructure characteristics.

In this chapter we evaluate the impact of the cold-start misses upon the communication overheads of phase two and we propose and evaluate strategies for warming-up the n-gram caches by combined glue calculations.

We evaluate the behavior of the miss ratio of the cache system, concerning the cold-start and the warming-up operation modes, when varying the *corpus* size, the maximum n-gram glue calculation and the number of machines. The above evaluation was conducted in two complementary ways:

1. We extend the phase two time complexity model presented in chapter 5 with an analytical model for the infinite dynamic cache system. This model was first fed with statistical data on the n-gram distribution, obtained by an empirical analysis of selected *corpora* within the range from 2 Mw to 982 Mw (Table 3.2 on page 35); This study is extended in chapter 9 with n-gram data for the entire *corpus* range up to the *plateaux*, by applying the theoretical model of chapter 3;

2. The prototype of the LocalMaxs Global methods parallel and distributed implementation (presented in chapters 4 and 5) was extended with the implementation of an on-demand dynamic cache system. The developed prototype was evaluated in private cluster and public cloud environments, and performance measurements of the real execution times and the observed miss ratios and miss time penalties

were collected. The real execution results were compared to the analytical model predictions.

6.2 Dynamic Cache System

Figure 6.2 represents the per machine dynamic cache system as a black box, denoted by C_{RW}. This notation hints its basic operation mode, namely, that it performs the on-demand fetch of the needed sub n-grams during the glue calculation, and as such the cache is progressively filled in as the execution proceeds. The n-gram data in each entry of the cache is a pair in the form $\langle key, frequency \rangle$, where key uniquely identifies an n-gram and $frequency$ is its associated absolute frequency count in the entire *corpus* under analysis as obtained in the end of phase one. Although we used the Read Write (**RW**) notation, each cache entry, once loaded, does not change anymore during execution of phase two, because on one hand the n-gram frequency count is fixed for each given *corpus* (as obtained in phase one of the Global method), and on the other hand we assume an infinite capacity cache. An improvement on this could be to update an auxiliary counter with the number of remaining occurrences for each distinct n-gram on the fly as execution proceeds, and discard from the cache the entries having a current remaining frequency count equal to zero (0).

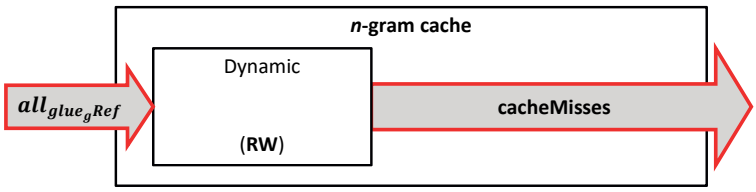


Figure 6.2: Diagram of a single dynamic cache system, used in phase two of the Global method.

In Figure 6.2 $all_{glue_g Ref}$ corresponds to the input reference stream consisting of a set of sequences of n-gram references, each one with n-grams of a given size. These sequences are generated as follows.

For a glue calculation $g_{n_1 \cdots n_2}$, the local partition tables of distinct n-grams with sizes from n_1 up to and including n_2 are parsed. For each glue calculation g_n the parsing of table D_n of n-grams of size n, $n_1 \leq n \leq n_2$, generates a sequence of references to the sub n-grams of sizes from 1 to $(n-1)$, which are contained in each of the n-gram entries of table D_n (Table 5.10 on page 108 and Figure 6.1 on page 134). Each one of these sequences for each sub n-gram size, from 1 to $(n-1)$, is forwarded to the corresponding individual cache, respectively C_1 for the unigrams, C_2 for the bigrams, up to C_{n-1} (as illustrated in Figure 6.3 for the case of glue g_{234}). This is performed for all the glue calculations from g_{n_1} to g_{n_2}.

The output misses of the cache system (denoted by *cacheMisses* in Figure 6.2) also form a stream of n-gram references which contains the missed sub n-gram references

generated by all the individual caches C_1 up to C_{n-1}. The global cache hit ratio (h_{RW}) and the corresponding global miss ratio (mr_{RW}) are defined as follows:

$$h_{RW} = \frac{\left|all_{glue_{g_n}Ref}\right| - |cacheMisses|}{\left|all_{glue_{g_n}Ref}\right|} \tag{6.3}$$

$$mr_{RW} = 1 - h_{RW} = \frac{|cacheMisses|}{\left|all_{glue_{g_n}Ref}\right|} \rightarrow |cacheMisses| = \left|all_{glue_{g_n}Ref}\right| \times (1 - h_{RW}) \tag{6.4}$$

In the assumption of an infinite cache with a cold-start, i.e., which starts empty, the total number of cache misses when evaluating the glue g_n is:

$$|cacheMisses(g_n)| = \sum_{i=1}^{n-1} \left|D_{i_{in}all_{glue_{g_n}Ref}}\right| \tag{6.5}$$

where $D_{i_{in}all_{glue_{g_n}Ref}}$ is the set of distinct n-grams of size i within the input reference stream $all_{glue_{g_n}Ref}$ and is equal to the set of leftmost and rightmost sub n-grams of size i found in the D_n table. The number ($\left|D_{i_{in}all_{glue_{g_n}Ref}}\right|$) corresponds to the total number of cold-start or first occurrence misses of cache C_i, $1 \leq i \leq (n-1)$.

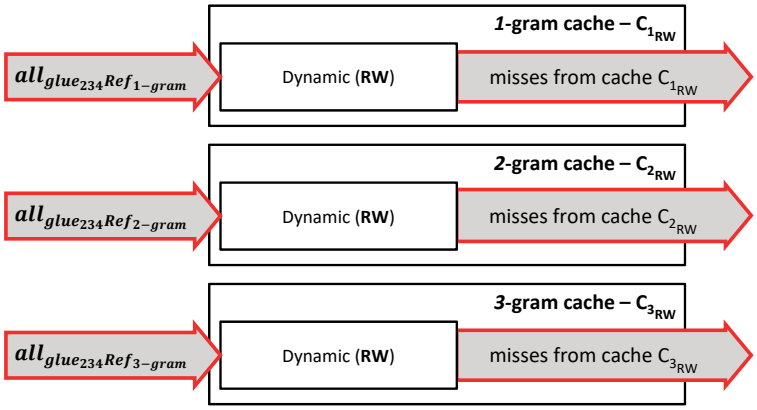


Figure 6.3: Dynamic cache system diagram for glue g_{234}.

In Figure 6.3 the unigrams cache is denoted by $C_{1_{RW}}$, the bigrams cache is denoted by $C_{2_{RW}}$, and the trigrams cache is denoted by $C_{3_{RW}}$. However, for a matter of simplicity, in the following analysis in this chapter we denote the unigrams, bigrams and trigrams caches, respectively, by C_1, C_2 and C_3.

Expression (6.2), which defines the global miss ratio of the local machine j cache system ($mr(j)$) in terms of its individual caches, also applies to this case of an on-demand dynamic cache (mr_{RW}).

6.3 n-gram Cache Evaluation with Cold-start

We estimate the number of missed references from each infinite cache C_i, which starts empty (cold-start), by considering a model where the cold-start misses are determined by the number of distinct sub *n-grams* of size i in the *n-gram* table partitions (D_n for $n \geq 2$). We consider separately the cases for a single machine and for multiple machines, because in the latter case there is a need to analyze the distribution of the number of distinct sub *n-grams* among the machines (as discussed in section 5.4.3 on page 87).

6.3.1 The Single Machine Case ($K = 1$)

Individual Cache Miss Ratio. When calculating the glue g_n the individual cache cold-start miss ratio ($mr_{Cold}(C_i, g_n)$) of an individual cache (C_i), as presented in Figure 6.3, is given by:

$$mr_{Cold}(C_i, g_n) = \frac{\left|D_{i_{in}D_n}\right|}{2 \times |D_n|} \tag{6.6}$$

for each i, $1 \leq i \leq (n-1)$, the numerator gives the total number of distinct sub n-grams of size i ($D_{i_{in}D_n}$) in the n-gram table D_n and the denominator gives the total number of references to the sub n-grams of size i in table D_n.

For a single machine, the total number of distinct subunigrams in the table D_n is equal to the total number of distinct unigrams in the *corpus*, i.e., $\left|D_{1_{in}D_n}\right| = |D_1|$, $2 \leq n \leq 6$, and for the case of bigrams $\left|D_{2_{in}D_n}\right| = |D_2|$, $3 \leq n \leq 6$, etc. The individual cache cold-start miss ratios for the isolated glues g_2 up to g_6 are shown in Table 6.2.

Table 6.2: Individual cache cold-start miss ratios for each isolated glue calculation in a single machine ($K = 1$). The *corpus* sizes (C) are expressed in millions of words and miss ratios in percentage (%). The last line indicates the caches C_i used in each glue calculation.

$\lvert C\rvert$	$mr_{Cold}(C_i,g_2) = \frac{\lvert D_i\rvert}{2\times\lvert D_2\rvert}$	$mr_{Cold}(C_i,g_3) = \frac{\lvert D_i\rvert}{2\times\lvert D_3\rvert}$		$mr_{Cold}(C_i,g_4) = \frac{\lvert D_i\rvert}{2\times\lvert D_4\rvert}$			$mr_{Cold}(C_i,g_5) = \frac{\lvert D_i\rvert}{2\times\lvert D_5\rvert}$				$mrCold(C_i,g_6) = \frac{\lvert D_i\rvert}{2\times\lvert D_6\rvert}$				
2	9.31	5.08	27.29	4.21	22.59	41.40	3.98	21.39	39.19	47.33	3.91	20.97	38.43	46.41	49.03
4	8.57	4.34	25.31	3.46	20.20	39.90	3.23	18.86	37.25	46.68	3.16	18.41	36.36	45.56	48.81
9	7.99	3.75	23.46	2.87	17.99	38.33	2.64	16.52	35.21	45.92	2.56	16.04	34.17	44.58	48.53
18	7.54	3.28	21.76	2.41	15.97	36.70	2.17	14.39	33.07	45.05	2.09	13.87	31.87	43.42	48.19
36	7.23	2.93	20.26	2.05	14.20	35.05	1.81	12.52	30.91	44.08	1.73	11.96	29.53	42.12	47.77
73	6.98	2.63	18.83	1.75	12.57	33.38	1.51	10.81	28.70	43.00	1.43	10.22	27.13	40.64	47.26
140	6.83	2.41	17.63	1.53	11.22	31.83	1.29	9.41	26.68	41.91	1.20	8.79	24.93	39.16	46.72
245	6.75	2.25	16.69	1.38	10.20	30.54	1.13	8.35	25.00	40.93	1.04	7.71	23.09	37.81	46.19
491	6.71	2.10	15.64	1.22	9.06	28.96	0.96	7.18	22.96	39.64	0.88	6.53	20.86	36.02	45.43
982	6.76	1.98	14.65	1.09	8.03	27.40	0.83	6.15	20.98	38.30	0.74	5.49	18.72	34.17	44.61
Cache	C_1	C_1	C_2	C_1	C_2	C_3	C_1	C_2	C_3	C_4	C_1	C_2	C_3	C_4	C_5

Each column in the table indicates the values of the individual cache miss ratio ($mr_{Cold}(C_i, g_n)$) for cache C_i ($1 \leq i \leq (n-1)$) for each indicated value of glue g_n ($2 \leq n \leq 6$). From Table 6.2 we observe the following behaviors:

- $mr_{Cold}(C_i, g_n)$ decreases when the *corpus* size increases because the ratio $|D_i|/|D_n|$ also decreases; This is due to the fact that $|D_i|$ grows slower than $|D_n|$ (Figure 3.5 on page 40 and Figure 3.6 on page 41);

- $mr_{Cold}(C_i, g_n)$ decreases as the glue calculation n-gram size n increases; For example, $mr_{Cold}(C_1, g_2) > mr_{Cold}(C_1, g_3) > mr_{Cold}(C_1, g_4)$ and so on; This is due to the increased reutilization of sub n-grams when increasing the n-gram sizes of the n-gram tables;

- For the largest *corpus* of about 1 Gw (982 Mw) and for glue calculation g_6 all caches are used but the individual cold-start miss ratio values in Table 6.2 are significantly different from each other: 1% for cache C_1, 5% for cache C_2, 19% for cache C_3, 34% for cache C_4, and 45% for cache C_5; This is due to the higher repetition factors for the smaller n-gram sizes, and to the higher proportions of singletons for the longer n-gram sizes, for this *corpus* size.

Global Cache Miss Ratio. The global cold-start miss ratio of the cache system formed by caches $C_{1+2+\cdots+5}$ for a given *corpus* size and for the glue $g_{2\ldots6}$ is:

$$mr_{Cold(g_{2\ldots6})} = \frac{\sum_{n=2}^{6} |cacheMisses(g_n)|}{\left|all_{glue_{g_{2\ldots6}}Ref}\right|} \tag{6.7}$$

which can be expanded to:

$$mr_{Cold(g_{2\ldots6})} = \frac{5\times|D_1| + 4\times|D_2| + 3\times|D_3| + 2\times|D_4| + |D_5|}{2\times|D_2| + 4\times|D_3| + 6\times|D_4| + 8\times|D_5| + 10\times|D_6|} \tag{6.8}$$

The numerator, both in expression (6.7) and expression (6.8), counts the sum of the distinct sub *n-gram* references of all the glue calculations g_n, $2 \leq n \leq 6$, when considered individually, because in the cold-start scenario each cache starts empty: the glues are calculated successively and each cache C_i is reset at the end of each glue calculation. The denominator counts the total number of sub n-gram references ($\left|all_{glue_{g_{2\ldots6}}Ref}\right|$).

Evolution of Miss Penalty and Miss Ratio with the *Corpus* Size. Figure 6.4 shows, in a scenario of a single machine and for the combined glue g_{234}, the global (*Global cs*) and individual cold-start miss penalties (C_i *cs*) as a function of the *corpus* size (in the range from 2 Mw to 982 Mw), where $Global\ cs = \sum_{n=2}^{4} |cacheMisses(g_n)|$ and C_1 *cs*, C_2 *cs*, and C_3 *cs* are the total numbers of misses for each individual cache. The global cold-start miss ratio $mr_{Cold}(g_{234})$ is also shown.

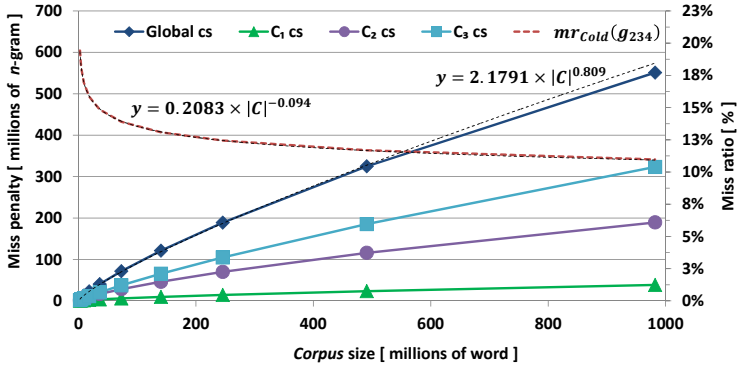


Figure 6.4: Cold-start miss penalties (global and individual) and global cold-start miss ratio when $K = 1$ for combined glue g_{234} *versus corpus* size. The left Y axis represents the cold-start miss penalty for each individual cache (C_1 *cs*, C_2 *cs*, and C_3 *cs*) and for the cache system (*Global cs*), in millions of *n*-grams; the right Y axis represents the global cold-start miss ratio ($mr_{Cold}(g_{234})$) in percentage; and the X axis represents the *corpus* size in millions of words.

The following conclusions can be taken:

- The global cold-start miss ratio ($mr_{Cold}(g_{234})$) decreases with the *corpus* size, but the decreasing rate is lower for larger *corpus* sizes;
- The global cold-start miss penalty (*Global cs*) increases with the *corpus* size;
- The caches with higher *n*-gram sizes have greater influence upon the global cold-start miss penalty, than the ones with lower sizes concerning the absolute number of misses and their growth rate when increasing the *corpus* size.

Figure 6.5 shows the global and individual cold-start miss penalties and the global miss ratio for the case of combined glue $g_{2...6}$, leading to similar conclusions as above.

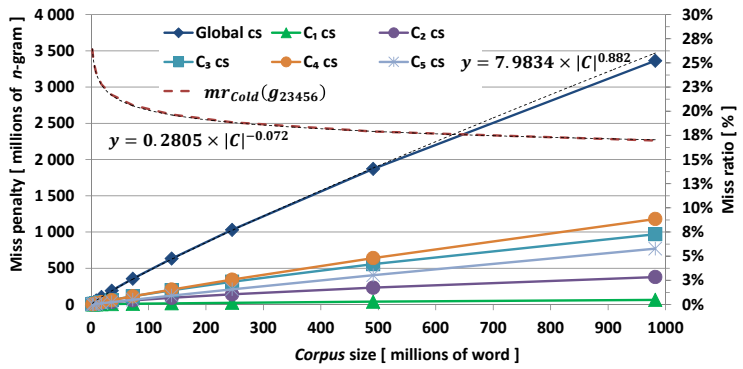
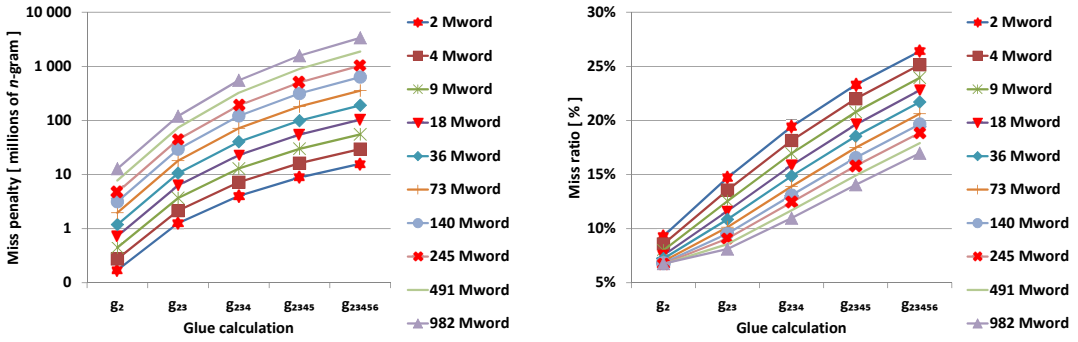


Figure 6.5: Cold-start miss penalties (global and individual) and global cold-start miss ratio when $K = 1$ for combined glue $g_{2...6}$ *versus corpus* size. The left Y axis represents the cold-start miss penalty in millions of *n*-grams; the right Y axis represents the global cold-start miss ratio in percentage; and the X axis represents the *corpus* size in millions of words.

The influence of the maximum n-gram size n of glue calculations when going from g_2 up to $g_{2\cdots6}$, for the same range of *corpus* sizes as above, is shown in Figure 6.6-a) regarding the global cold-start miss penalty (*Global cs*) and in Figure 6.6-b) regarding the global cold-start miss ratio ($mr_{Cold}(g_{2\cdots n})$), with the following observations:

- Although the global cold-start miss penalty increases with the complexity (n) of the combined glue calculation, its growth rate exhibits a slowdown as n increases (Figure 6.6-a)). This is due to the progressive increase of the cache reutilization of distinct *n-gram* references; For example, there are many more repetitions in the subunigram references to cache C_1 for the combined glue calculation of $g_{2\cdots6}$ than for the glue calculation of g_2 because cache C_1 is used by all the glue calculations from g_2 up to g_6 and the universe of distinct unigrams is finite;
- The behavior of the global cold-start miss ratio (Figure 6.6-b)) is determined by expression (6.8).



a) *Global miss penalty* ***b)*** *Global miss ratio*

Figure 6.6: Single machine global cold-start miss penalty and global cold-start miss ratio *versus* combined glue calculation for different *corpus* sizes. The Y axis in subfigure a) represents the miss penalty in millions of n-grams and the Y axis in subfigure b) represents the miss ratio in percentage. In both subfigures the X axis represents the glue calculation (g_2, g_{23}, ..., $g_{2\cdots6}$).

6.3.2 The Multiple Machines Case ($K > 1$)

As presented in section 5.4.3 the evolution of the distribution of distinct sub *n-grams* ($\left|D_{i_{in}D_n}(j)\right|$) in each local partition table $D_n(j)$, with n: 2, 3, 4, ..., can be approximately described by an expression in the form $|D_i| \times K^{-b}$, where $0 < b < 1$, K is the number of machines and $|D_i|$ is the number of distinct n-grams of size i in the *corpus*. Figure 6.7 summarizes this behavior when K varies from 1 to 48, for a fixed *corpus* size of 466 Mw, for the case of cache C_1 and the glue calculations g_2, g_3 and g_4. The corresponding cold-start miss ratio of the cache C_1 is also shown.

Let us consider the fitting expression $\left|D_{1_{in}D_2}(j)\right| = 5.4 \times K^{-0.7}$ approximately capturing the curve trend in Figure 6.7. As an approximation to the cold-start miss ratio of the cache C_1 for machine j for the glue g_2, we have:

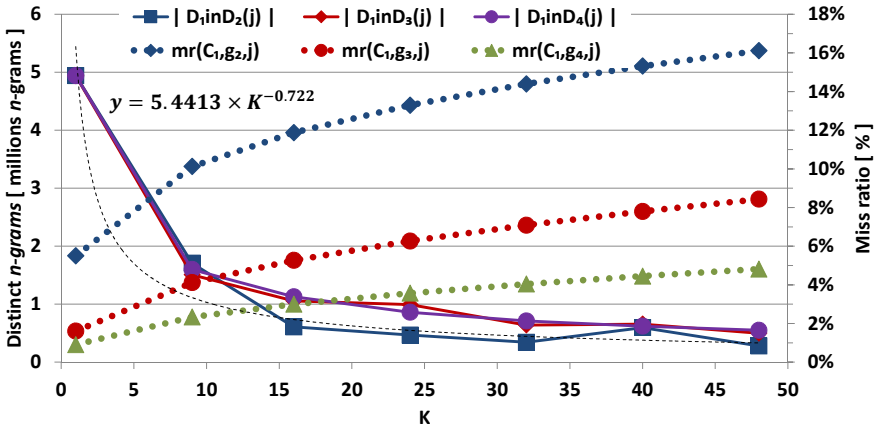


Figure 6.7: Distinct subunigrams in D_2, D_3 and D_4 and cold-start miss ratio for cache C_1 per machine for a *corpus* size of 466 Mw. The X axis represents the number of machines, the left Y axis represents the number of distinct sub n-grams, and the right Y axis represents the per machine cold-start miss ratio ($mr_{Cold}(C_1, g_n, j)$) for $n = 2, 3, 4$.

$$mr_{Cold}(C_1, g_2, j) = \frac{\left|D_{1_{in}D_2}(j)\right|}{2 \times |D_2(j)|} \tag{6.9}$$

where $|D_2(j)| = |D_2|/K$ is the total number of distinct bigrams in the $D_2(j)$ table. Thus, expression (6.9) becomes:

$$mr_{Cold}(C_1, g_2, j) = 5.4 \times K^{-0.7} \times \frac{K}{2 \times |D_2|} = \frac{2.7 \times K^{0.3}}{|D_2|} \tag{6.10}$$

Thus:

- $mr_{Cold}(C_1, g_n, j)$ increases with the number of machines (K) for all n, $2 \leq n \leq 4$;
- Also note that for the considered range of machines, although $mr_{Cold}(C_1, g_n, j)$ increases with K, the slope of the corresponding curve is comparatively lower when going from calculating the glue calculation g_2 to glue g_3 and glue g_4; Thus, when calculating the glues of larger n-gram sizes, the observed increase of the miss ratio with K is less pronounced;
- For each value of K, $mr_{Cold}(C_1, g_n, j)$ decreases from glue g_2 to glue g_3, and to glue g_4, due to the increase of the repetition of distinct subunigram occurrences when the total cache references go from $2 \times |D_2|$ to $2 \times |D_3|$ and to $2 \times |D_4|$;
- Although $mr_{Cold}(C_1, g_n, j)$ increases with the number of machines (K), the individual cache C_1 cold-start miss penalty ($C_1 cs = \left|D_{1_{in}D_n}(j)\right|$) decreases with K following a power law trend ($|D_1| \times K^{-b}$).
- Similar behaviors were observed for the distribution of distinct subbigrams, subtrigrams, etc, that is $D_{i_{in}D_n}(j)$ for values of i going from 2 up to 5, and the corresponding C_i caches: i) The per machine individual cache C_i cold-start miss penalty

($C_i\,cs = \left|D_{i_{in}D_n}(j)\right|$) decreases with K; ii) The per machine individual cache C_i cold-start miss ratio ($mr_{Cold}(C_i, g_n, j)$) increases with K and tends to stabilize for larger values of K.

The same behavior is observed when considering a combined glue calculation such as g_{234} as illustrated in Figure 6.8, concerning the evolution of the per machine cold-start misses and the global cold-start miss ratio of the cache system composed of caches C_1, C_2 and C_3 (denoted as C_{1+2+3}) as a function of the number of machines (K):

- The per machine global cold-start miss penalty (*Global cs*) of the cache system decreases with K as a power law function (whose fitting curve is shown), but the decreasing rate is lower for larger number of machines; The same behavior is observed for each of the individual cache cold-start miss penalties ($C_1\,cs$, $C_2\,cs$, $C_3\,cs$);

- The per machine global cold-start miss ratio ($mr_{Cold}(g_{234})$) of the cache system slowly increases with K, as illustrated by its fitting curve, and tends to stabilize for larger values of K.

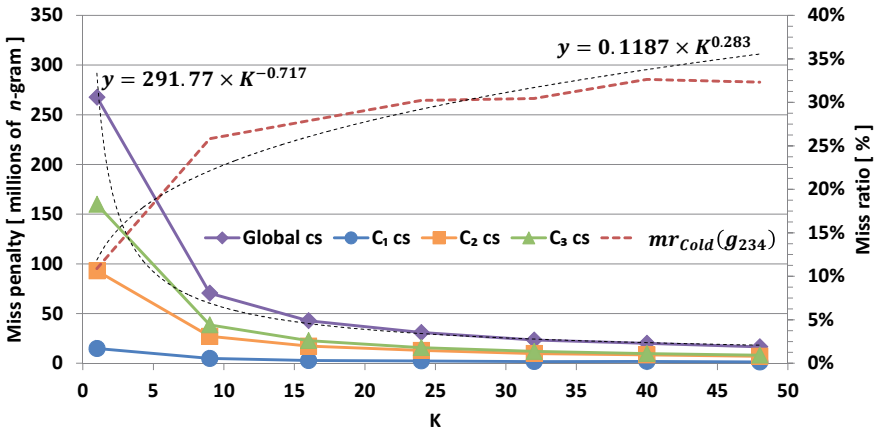
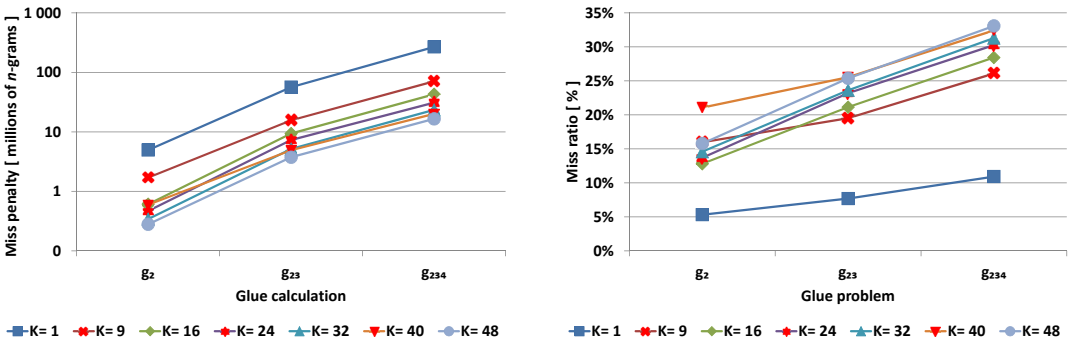


Figure 6.8: Per machine cold-start miss penalty (*Global cs* and individual $C_1\,cs$, $C_2\,cs$, $C_3\,cs$) and global cold-start miss ratio ($mr_{Cold}(g_{234})$) of the cache system (C_{1+2+3}) *versus* number of machines (K), for a *corpus* size of 466 Mw and fixed glue g_{234}. The X axis represents the number of machines, the left Y axis represents the number of distinct sub n-gram misses in millions of n-grams, and the right Y axis represents the cold-start miss ratio in percentage.

The evolution of the global cold-start miss penalty (*Global cs*) of the cache system and the global cold-start miss ratio (mr_{Cold}) as a function of the maximum n-gram size n of glue calculations in the range from g_2 up to g_{234} is shown in Figure 6.9, for the same fixed *corpus* size of 466 Mw when varying K from 1 to 48. The observed behaviors are similar to the ones for the single machine case.



a) *Global miss penalty* ***b)*** *Global miss ratio*

Figure 6.9: Per machine global cold-start miss penalty and global cold-start miss ratio of the cache system *versus* combined glue calculation, for a fixed *corpus* size of 466 Mw and different numbers of machines. The Y axis in subfigure a) represents the miss penalty in millions of n-grams and the Y axis in subfigure b) represents the miss ratio in percentage. In both subfigures the X axis represents the glue calculation.

6.4 *n*-gram Cache Evaluation with Warm-up

The cache hit ratio is increased by increasing the distinct sub n-gram repetitions for each glue calculation. It can also be increased by combining the calculation of the glues of n-grams, from 2 up to n, e.g., g_2, g_3 and g_4.

Sharing Common Distinct *n*-grams in the Local Caches. Figure 6.10 illustrates, for a combined glue g_{234}, how the sharing of common sub n-grams is achieved by having, in each machine, one controller instance with concurrent threads for calculating the glues for bigrams, trigrams and tetragrams.

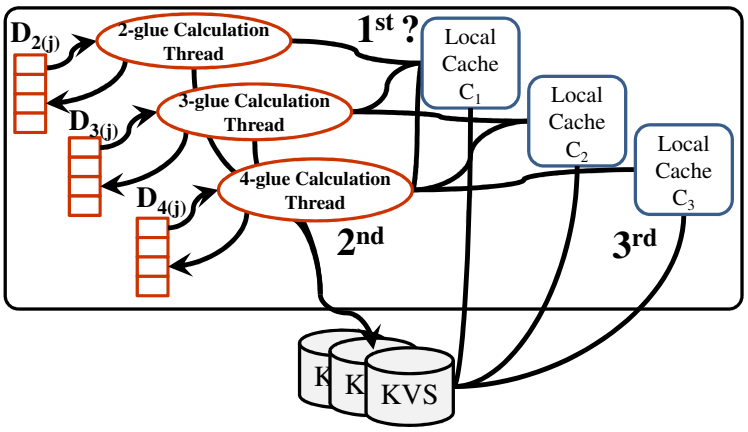


Figure 6.10: Calculate all glue combined.

Within each controller there is a thread for each instance of the glue calculation for the n-grams of each given size, that is, "2-glue calculation" for bigrams, "3-glue calculation" for trigrams, and so on up to the maximum n value considered for each combined glue

calculation. These threads calculate the glue of disjoint n-grams sets contained in each of the local partition tables $D_2(j)$, $D_3(j)$ and $D_4(j)$, which require to generate references to the sub n-grams included in those tables ($D_{1_{in}D_2}(j)$, $D_{1_{in}D_3}(j)$, $D_{1_{in}D_4}(j)$, $D_{2_{in}D_3}(j)$, and $D_{2_{in}D_4}(j)$), in order to obtaining the corresponding frequency counters, which are stored in the distributed KVS servers, or which are already in the cache. As there are common distinct sub n-grams, there are references shared by the three glue calculation threads (in this case use of maximum $n = 4$). The n-gram caches, associated to each controller, for each n-gram size, from 1 to 3 (in this example), denoted by C_1, C_2, and C_3, contribute to minimizing the number of remote accesses to those sub n-grams.

Analysis of the Sets of Shared Common Distinct *n*-grams. The number of shared sub n-gram references depends on K. For example, when considering cache C_1 for the combined glue g_{234}, the $D_{1_{in}D_n}$ sets are not disjoint, i.e., $D_{1_{in}D_2} \cap D_{1_{in}D_3} \cap D_{1_{in}D_4} \neq \emptyset$. When K=1 these sets are identical to D_1 which is the set of distinct unigrams in the entire *corpus*, so all distinct subunigrams references (D_1) are completely shared by the g_2, g_3 and g_4 calculations. So, we can first execute the glue g_2 calculation as a driver to warm-up cache C_1, and this ensures that there are no unigram misses at all, during the glue calculations g_3 and g_4. However, when $K > 1$, the sets of distinct subunigrams in each table are different, $D_{1_{in}D_2}(j) \neq D_{1_{in}D_3}(j) \neq D_{1_{in}D_4}(j)$, due to the way the distinct n-grams tables are distributed by hashing in the end of phase one. Nevertheless, there still are common subunigrams in the sets $D_{1_{in}D_2}(j)$, $D_{1_{in}D_3}(j)$ and $D_{1_{in}D_4}(j)$. It is desirable to identify such common subunigrams because they contribute to increasing the reutilization of cache items once they are fetched from the KVS servers, in a similar effect to the cooperative caching strategies.

To evaluate the impact of those shared sub n-grams, we conducted an empirical analysis for several *corpus* sizes and number of machines. As a result of this empirical study we confirmed that for K=1, $D_{1_{in}D_2} = D_{1_{in}D_3} = D_{1_{in}D_4} = D_1$, the distinct unigrams in the *corpus*; $D_{2_{in}D_3} = D_{2_{in}D_4} = D_2$, the distinct bigrams in the *corpus*; and so on. So by first running the glue g_2 (thus filling C_1 with all $D_1 in D_2$) we entirely achieve the effect of warming-up the cache C_1 with the required sub unigrams needed by the glue g_3 (that is $D_{1_{in}D_3}$) and also g_4 (that is $D_{1_{in}D_4}$). Similarly due to the equality of $D_{2_{in}D_3} = D_{2_{in}D_4}$ the cache C_2 is entirely warmed-up by running the calculation of the glue g_3 before starting the calculation of glue g_4. Note that when starting the calculation of glue g_4 the cache C_1 is already warm due to the previous running of the g_2 and g_3 glue calculations. The overall effect of the above strategy corresponds to greatly reducing the penalty due to the cold-start misses of all the caches for the n-grams 1 to $(n-2)$, when calculating the glue for n-grams of size n. As observed for the combined glue calculation g_{234}, the above warm-up effect contributes to a significant reduction of the global miss ratio of the cache system when compared to a no-cache configuration.

6.4.1 Cache Warm-up Efficiency

For $K > 1$, the sets of common sub n-grams referenced by the different glue calculations of bigrams, trigrams and tetragrams depend on the hashing method used in phase one for distributing the n-grams among the local partitions. In the following we present the results of an empirical analysis of the generated sets using four distinct hashing methods (presented on section 5.4.2 on page 84) that only differ in the part of the n-gram used as hash code: i) the hash code covers the entire n-gram (h_0); ii) the hash code covers the leftmost $(n-1)$ sub n-gram of the n-gram (h_1); iii) the hash code covers the leftmost and the rightmost unigrams (h_2); iv) the hash code covers the leftmost unigram (h_3).

Individual Cache Warm-up Efficiency for Isolated Glue g_n. The efficiency of the cache C_1 warming-up by the glue calculation g_2 regarding its impact upon the glue calculation g_3, is defined as follows:

$$\eta_{(C_{1_{WarmBy g_2}}, g_3)} = \frac{\left|D_{1_{in}D_2} \cap D_{1_{in}D_3}\right|}{\left|D_{1_{in}D_3}\right|} \tag{6.11}$$

where for simplicity, in the remaining of this section, the j^{th} index denoting each individual machine is omitted. In the numerator we consider the distinct subunigrams that were referenced both by the glue calculation g_2 and g_3 and in the denominator we consider the total number of distinct subunigrams referenced by the glue calculation g_3. Thus, this ratio gives the percentage of the unigrams which, having been put in the cache C_1 by glue g_2 are needed by glue g_3 (i.e., are the shared common distinct subunigrams to g_2 and g_3), over the total subunigrams needed by g_3.

Figure 6.11 shows the warm-up efficiency ($\eta_{(C_{1_{WarmBy g_2}}, g_3)}$) for a fixed *corpus* size as a function of K ($1 \leq K \leq 48$), for the four mentioned hash methods.

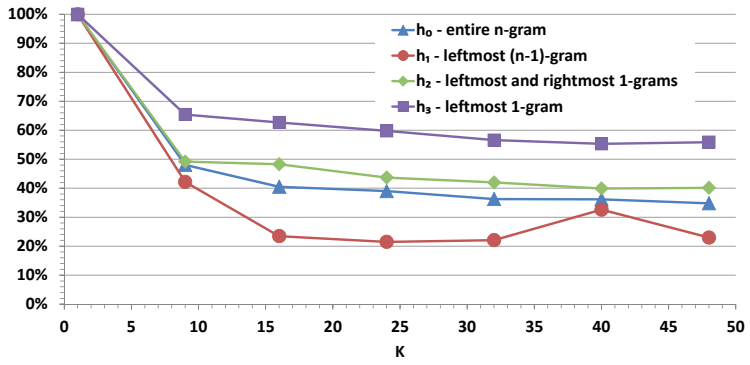


Figure 6.11: C_1 warming-up efficiency ($\eta_{(C_{1_{WarmBy g_2}}, g_3)}$) by glue g_2 when calculating glue g_3 as a function of the number of machines, for a *corpus* of 466 Mw and different hash methods. The Y axis represents the efficiency η in percentage and the X axis represents the number of machines.

The following conclusions can be taken:

- The efficiency of the cache C_1 warming-up varies only slightly with K, for K greater than 9, for any of the considered hash methods;
- The highest cache C_1 warming-up efficiency is obtained by the hashing method h_3;
- Similar behaviors were observed for cache C_1 warming-up efficiency ($\eta_{(C_{1_{WarmByg_2}}, g_4)}$) by glue g_2 when calculating glue g_4 as well as for cache C_2 warming-up efficiency ($\eta_{(C_{2_{WarmByg_3}}, g_4)}\eta$) by glue g_3 when calculating glue g_4.

The impact of the warm-up efficiency upon the warm-start miss ratio (mr_{Warm}) can be evaluated by analyzing the following expressions.

Let us consider the following definitions of the warm-start miss ratio of cache C_1 for the isolated glue calculation of trigrams, when C_1 is warmed-up by the glue calculation g_2, denoted by mr_{Warm} in expression (6.12), or when C_1 starts cold, denoted by mr_{Cold} in expression (6.13):

$$mr_{Warm}\left(C_{1_{WarmByg_2}}, g_3\right) = \frac{\left|D_{1_{in}D_3} \setminus \left(D_{1_{in}D_2} \cap D_{1_{in}D_3}\right)\right|}{2 \times |D_3|} = \frac{\left|D_{1_{in}D_3}\right| - \left|D_{1_{in}D_2} \cap D_{1_{in}D_3}\right|}{2 \times |D_3|} \tag{6.12}$$

$$mr_{Cold}\left(C_1, g_3\right) = \frac{\left|D_{1_{in}D_3}\right|}{2 \times |D_3|} \tag{6.13}$$

We obtain:

$$\frac{mr_{Warm}\left(C_{1_{WarmByg_2}}, g_3\right)}{mr_{Cold}\left(C_1, g_3\right)} = 1 - \eta\left(C_{1_{WarmByg_2}}, g_3\right) \tag{6.14}$$

Thus, the higher the efficiency $\eta\left(C_{1_{WarmByg_2}}, g_3\right)$, the lower the C_1 cache warm-start miss ratio, for each given cold-start miss ratio. Note that the ratio between $mr_{Warm}\left(C_{1_{WarmByg_2}}, g_i\right)$ and $mr_{Cold}(C_1, g_i)$ for $i > 2$ is given by $1 - \eta\left(C_{1_{WarmByg_2}}, g_i\right)$. For instance, with $\eta\left(C_{1_{WarmByg_2}}, g_3\right) \approx$ 60% the ratio of the warm miss ratio over the cold miss ratio is $mr_{Warm}/mr_{Cold} \approx 40\%$.

Global Cache System Warm-up Efficiency for Combined Glue $g_{n_1 \cdots n_2}$. In order to evaluate the overall result of the warming-up strategy, we define the global warm-up efficiency of the cache system formed by C_1 and C_2, caches, with C_1 warmed up by g_2 and C_2 warmed up by g_3, when calculating the combined glue g_{34}:

$$\eta\left(C_{1_{WarmByg_2}} \& C_{2_{WarmByg_3}}, g_{34}\right) = \eta_{12} = \frac{\left|D_{1_{in}D_2} \cap D_{1_{in}D_3}\right| + \left|D_{1_{in}D_2} \cap D_{1_{in}D_4}\right| + \left|D_{2_{in}D_3} \cap D_{2_{in}D_4}\right|}{\left|D_{1_{in}D_3}\right| + \left|D_{1_{in}D_4}\right| + \left|D_{2_{in}D_4}\right|} \tag{6.15}$$

where the numerator includes the sum of distinct subunigrams that were put in the cache C_1 by g_2 and needed by g_3, plus the distinct subunigrams that were put in the cache C_1 by g_2 and needed by g_4, plus the distinct subbigrams that were put in the cache C_2 by g_3 and needed by g_4; and the denominator is the sum of all distinct subunigrams needed

by g_3, plus all the distinct subunigrams needed by g_4, plus all the distinct subbigrams needed by g_4.

In this modeling we evaluate separately g_3 with C_1 warmed by g_2, from g_4 with C_1 warmed by g_2, although in the real implementation an incremental effect occurs, i.e., g_2 and g_3 jointly contribute to the warming of cache C_1 before g_4 runs (section 6.5.1.1, on page 161). Here the cache C_1 warm-up due to g_2 for g_3 usage is considered separately from the cache C_1 warm-up due to g_2 for g_4 usage. This is an approximation to the real warm-up strategy in which the effect of g_3 warm-up on cache C_1 adds upon the previous g_2 warm-up and this incremental warm-up effect is used by g_4.

Using the definitions of the individual warm-up efficiencies of cache C_1 for g_3 (named η_d), cache C_1 for g_4 (named η_e), and cache C_2 for g_4 (named η_f):

$$\eta_d = \eta\left(C_{1_{WarmByg_2}}, g_3\right); \quad \eta_e = \eta\left(C_{1_{WarmByg_2}}, g_4\right); \quad \eta_f = \eta\left(C_{2_{WarmByg3}}, g_4\right) \tag{6.16}$$

expression (6.15) simplifies to:

$$\eta\left(C_{1_{WarmByg_2}} \,\&\, C_{2_{WarmByg_3}}, g_{34}\right) = \eta_{12} = \frac{\eta_d \times \left|D_{1_{in}D_3}\right| + \eta_e \times \left|D_{1_{in}D_4}\right| + \eta_f \times \left|D_{2_{in}D_4}\right|}{\left|D_{1_{in}D_3}\right| + \left|D_{1_{in}D_4}\right| + \left|D_{2_{in}D_4}\right|} \tag{6.17}$$

Figure 6.12 shows the global warm-up efficiency (η_{12}) of the C_{1+2} cache system when K varies from 1 to 48, for a *corpus* of 466 Mw.

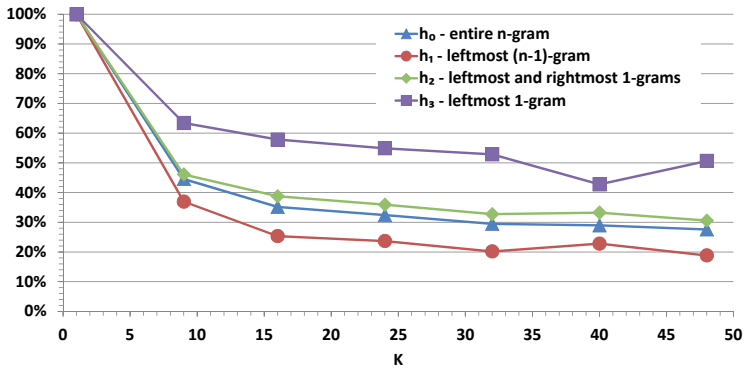


Figure 6.12: Global warm-up efficiency η_{12} of the cache system as a function of the number of machines, for a *corpus* of 466 Mw and different hash methods. The Y axis represents the efficiency in percentage and the X axis represents the number of machines.

The following conclusions can be taken:

- The global warm-up efficiency (η_{12}) exhibits a similar behavior as the individual cache warm-up efficiencies;
- The maximum global efficiency has a value of 100%, which occurs for $K = 1$, corresponding to $\eta_d = \eta_e = \eta_f = 100\%$; This efficiency reduces with K due to the corresponding reduction in C_1 and C_2 warming up efficiencies.

Expression (6.18) and Figure 6.13 represent the ratio of the global warm-start miss ratio (mr_{Warm}) divided by the global cold-start miss ratio (mr_{Cold}) for the entire cache system composed of C_{1+2+3}, during the glue calculation g_{234}:

$$\frac{mr_{Warm}\left(C_{C1_{WarmBy g2}+C2_{WarmBy g3}}, g_{234}\right)}{mr_{Cold}(C, g_{234})} = 1 - \frac{\eta_{12} \times \left(\left|D_{1_{in}D_3}\right| + \left|D_{1_{in}D_4}\right| + \left|D_{2_{in}D_4}\right|\right)}{\left(\left|D_{1_{in}D_2}\right| + \left|D_{2_{in}D_3}\right| + \left|D_{3_{in}D_4}\right|\right) + \left(\left|D_{1_{in}D_3}\right| + \left|D_{1_{in}D_4}\right| + \left|D_{2_{in}D_4}\right|\right)} \tag{6.18}$$

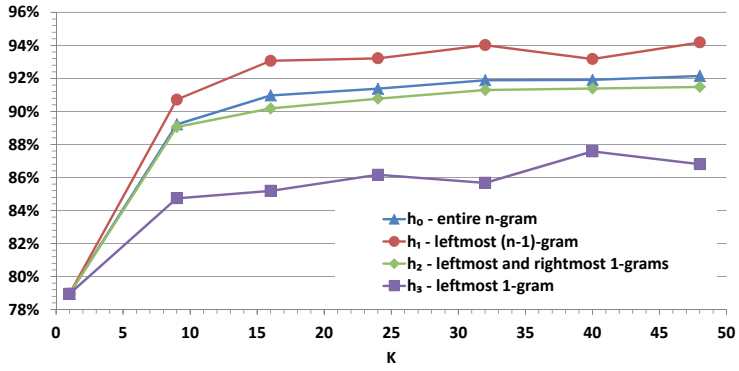


Figure 6.13: Warm-start miss ratio over cold-start miss ratio of the cache system as a function of the number of machines, for a *corpus* of 466 Mw and glue g_{234} and different hash methods. The Y axis represents the ratio mr_{Warm}/mr_{Cold} in percentage and the X axis represents the number of machines.

The following conclusions can be taken:

- For $K = 1$, when the warm-up efficiency (η_{12}) is 100%, the ratio mr_{Warm}/mr_{Cold} is only about 79%, which is due to fact that we are now accounting for all the misses for the evaluation of glue g_{234}, which includes the C_1 cache cold-start misses during g_2 calculation ($\left|D_{1_{in}D_2}\right|$), the C_2 cache cold-start misses during g_3 calculation ($\left|D_{2_{in}D_3}\right|$), and the C_3 cache cold-start misses during g_4 calculation ($\left|D_{3_{in}D_4}\right|$); This further highlights the need to take advantage of a more intensive cache reutilization, leading to a "incremental warm-up, read many" cache.

In order to effectively implement the above warming-up, we considered that the cache warming-up in each machine benefits from the sequential execution of the glue calculation g_2, g_3, etc., so this strategy was followed in the LocalMaxs Global method implementation. Furthermore, when the glue calculations in each machine proceed sequentially (g_2; g_3; g_4; ...), the cache C_1 warming-up efficiency is actually improved by an incremental warm-up effect, that is first by the glue calculation g_2, then by g_3, followed by g_4 etc. The implementation supports such incremental warming-up of all the individual caches C_i , $1 \leq i \leq (n-1)$ of the cache system. However, in the model estimates presented above, for

the warm-up cache behavior, for each cache C_i, we only considered separately the warm-up effect due to the glue $g_{(i+1)}$: For example, when the cache C_1 is warmed-up by the glue calculation g_2, we consider its impact upon the glue calculation g_3, and also separately consider the impact of g_2 upon the glue calculation g_4, but we did not model the impact due to the joint incremental warm-up of cache C_1 by g_3 (after executing g_2) upon g_4. As a result the model estimates lead to higher values of the miss penalties compared to the real execution results (presented in the following section 6.5).

6.4.2 Overall Estimation of Cache Benefits for a Single Machine ($K = 1$)

For calculating the glues g_2, g_{23}, g_{234}, $g_{2\cdots5}$ and $g_{2\cdots6}$ regarding the cache system we considered three modes of operation: i) No-cache: calculating the glues when all the sub n-gram references (all_{glue_gRef}) are remote; ii) Cold-start: calculating each glue g_2, g_3, g_4, g_5, g_6 separately, each starting with its corresponding caches empty; iii) Warm-start: calculating the glues g_2, g_3, g_4, g_5, g_6 in a strict sequence with warm-start caches, that is: first calculating the glue g_2; then calculating the glue g_3 yet still keeping the cache C_1 with the previous unigrams contents; calculating glue g_4 but also keeping caches C_1 and C_2 respectively with the previous unigrams and bigrams contents, and so on.

Global Cold-start Miss Ratio. Table 6.3 shows, for the glue calculations g_2, g_{23}, g_{234}, $g_{2\cdots5}$ and $g_{2\cdots6}$, the global cold-start miss ratio of the corresponding cache system, $mr_{Cold} = cs/nc$, where cs ($ColdStartMisses$) is the number of global cold-start misses, and nc ($NoCacheMisses$) is the total number of generated references, corresponding to the remote references in the no-cache system.

Table 6.3: Global miss ratios $ColdStart/NoCache = mr_{Cold}$ when $K = 1$. Miss ratio in percentage (%).

$\|C\|$ [Mw]	$mr_{Cold}=\frac{ColdStartMisses}{NoCacheMisses}=\frac{cs}{nc}$				
	g_2	g_{23}	g_{234}	$g_{2\cdots5}$	$g_{2\cdots6}$
2	9.31	14.71	19.42	23.29	26.39
4	8.57	13.56	18.14	22.00	25.14
9	7.99	12.54	16.95	20.77	23.95
18	7.54	11.63	15.85	19.61	22.79
36	7.23	10.86	14.86	18.53	21.70
73	6.98	10.13	13.91	17.48	20.62
140	6.83	9.54	13.09	16.56	19.66
245	6.75	9.08	12.44	15.80	18.86
491	6.71	8.58	11.69	14.91	17.90
982	6.76	8.12	10.97	14.05	**16.97**
Caches used	C_1	C_1, C_2	C_1, C_2, C_3	$C_1 \cdots C_4$	$C_1 \cdots C_5$

Global Warm-start Miss Ratio. Table 6.4 shows, for the glue calculations g_2, g_{23}, g_{234}, $g_{2\cdots5}$ and $g_{2\cdots6}$, the ratio of the global warm-start miss ratio to the global cold-start

miss ratio of the corresponding cache system, that is $mr_{Warm}/mr_{Cold} = ws/cs$, where cs ($ColdStartMisses$) is the number of global cold-start misses, and ws ($WarmStartMisses$) is the number of global warm-start misses out of the cache system.

Table 6.4: Global warm-start *versus* cold-start miss ratios $WarmStart/ColdStart$ when $K = 1$. Miss ratio in percentage (%).

$\|C\|$ [Mw]	$\frac{mr_{Warm}}{mr_{Cold}} = \frac{WarmStartMisses}{ColdStartMisses} = \frac{ws}{cs}$				
	g_2	g_{23}	g_{234}	$g_{2\cdots5}$	$g_{2\cdots6}$
2	100.00	86.43	68.74	54.37	44.03
4	100.00	87.23	70.09	55.60	45.00
9	100.00	87.89	71.38	56.84	46.00
18	100.00	88.42	72.60	58.07	47.02
36	100.00	88.78	73.70	59.27	48.04
73	100.00	89.09	74.79	60.50	49.11
140	100.00	89.27	75.73	61.63	50.13
245	100.00	89.37	76.48	62.59	51.00
491	100.00	89.42	77.35	63.76	52.11
982	100.00	89.35	78.16	64.94	**53.26**
Caches used	C_1	C_1, C_2	C_1, C_2, C_3	$C_1 \cdots C_4$	$C_1 \cdots C_5$

From the above tables we obtain the global warm-start miss ratio values (mr_{Warm}):

- The product of the values of each cell of Table 6.3, for the case $mr_{Cold} = cs/nc$, times each corresponding cell for the case ws/cs in Table 6.4 gives the global warm miss ratio $mr_{Warm} = ws/nc$, i.e., the reduction in the number of missed references from a no-cache to a warm-start cache system;

- For instance, for the 982 Mw *corpus* and glue calculation $g_{2\cdots6}$, with a $mr_{Cold} = 17\%$ (i.e., 16.97%, marked in bold in Table 6.3) and warm-start to cold-start ratio of 53% (i.e., 53.26%, marked in bold in Table 6.4), we get a global $mr_{Warm} = 0.17 \times 0.53 = 9\%$.

Evolution of Warm-Start Miss Ratio and Miss Penalty with the *Corpus* Size. Figure 6.14 shows, in a single machine scenario and glue g_{234}, the evolution of the global warm-start miss penalty of the cache system C_{1+2+3} (*Global ws*) and the individual warm-start miss penalties for the three caches C_1, C_2 and C_3, respectively, $C_1\,ws$, $C_2\,ws$ and $C_3\,ws$; and the global warm-start miss ratio of the cache system C_{1+2+3} ($mr_{Warm}(g_{234})$), as a function of the *corpus* size.

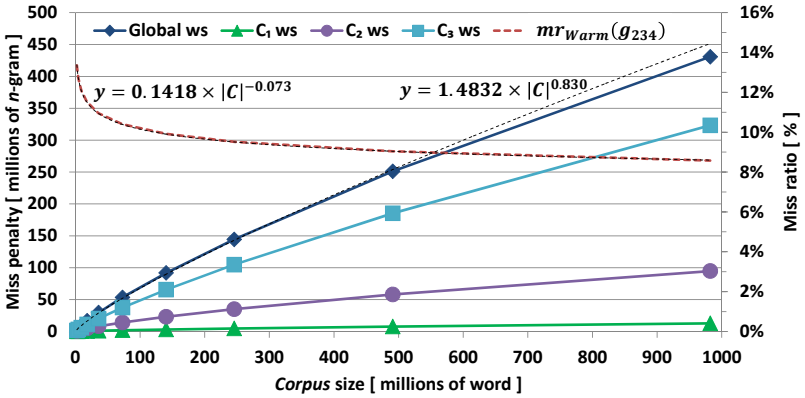


Figure 6.14: Warm-start miss penalties of the cache system C_{1+2+3} (*Global ws*) and of the individual caches ($C_1 ws$, $C_2 ws$ and $C_3 ws$) and global warm-start miss ratio ($mr_{Warm}(g_{234})$) when $K = 1$ for combined glue g_{234}. The left Y axis represents the warm-start miss penalties (millions of n-grams); the right Y axis represents the warm-start miss ratio (percentage); and the X axis represents the *corpus* size (millions of words).

The following conclusion can be taken:

- The evolution of the warm-start miss penalties (global of the cache system and individual for each cache) and the global warm-miss ratio of the cache system have a similar behavior as the cold-start penalties and cold-start miss ratio presented in Figure 6.4: i) The global warm-start miss ratio tends to decrease with the *corpus* size and stabilizes for larger sizes; ii) The warm-start miss penalties increase with the *corpus* size, for the observed values up to 982 Mw.

The comparison of the global miss penalty and the global miss ratio of the cache system C_{1+2+3} for glue g_{234} in the case of a warm-start against the case of a cold-start for different *corpus* sizes is presented in Figure 6.15.

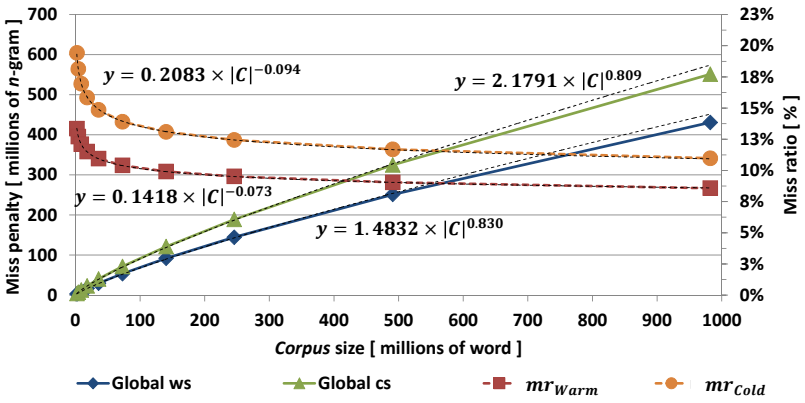


Figure 6.15: Single machine global miss penalty and global miss ratio — warm-start and cold-start — *versus corpus* size. The left Y axis represents the miss penalty (millions of n-grams); the right Y axis represents the miss ratio (percentage); and the X axis represents the *corpus* size (millions of words). The fitting expressions are shown next to each curve.

The following conclusions can be taken:

- The global warm-start miss penalty (*Global ws*) is less than the global cold-start miss penalty (*Global cs*); This is due to the warm-up effect of caches C_1 and C_2; In fact as a result of the glue calculation g_2 there are no subunigram misses observed during the glue calculation of g_3 and g_4; Also as a result of the glue calculation g_3 there are no subbigram misses observed during the glue calculation of g_4;

- Overall, for g_{234} the difference between the global cold-start and global warm-start miss penalties is equal to $2\times|D_1|+|D_2|$, because $Global\,cs = 3\times|D_1|+2\times|D_2|+|D_3|$ and $Global\,ws = |D_1|+|D_2|+|D_3|$; For other glue calculations, this difference increases with the maximum size of the n-grams for which the combined glue calculation is performed;

- For the combined glue calculation $g_{2\ldots n}$ the above difference is given by $\sum_{i=2}^{n-1}(n-i)\times|D_{i-1}|$ where the $|D_{i-1}|$ represents the reduction in the number of misses observed by cache C_{i-1}, each time it is used for each glue calculation from 2 up to n.

Evolution of Warm-Start Miss Ratio and Miss Penalty with the Glue Calculation. Figure 6.16 shows the global warm-start miss penalty *versus* the combined glue calculation of n-grams g_n when n varies from 2 up to 6, for different *corpus* sizes. The global warm-start miss penalty is given by $\sum_{i=1}^{n-1}|D_i|$ which corresponds to the sum of all distinct n-grams from unigrams up to $(n-1)$-grams. It exhibits a similar behavior as in the cold-start case.

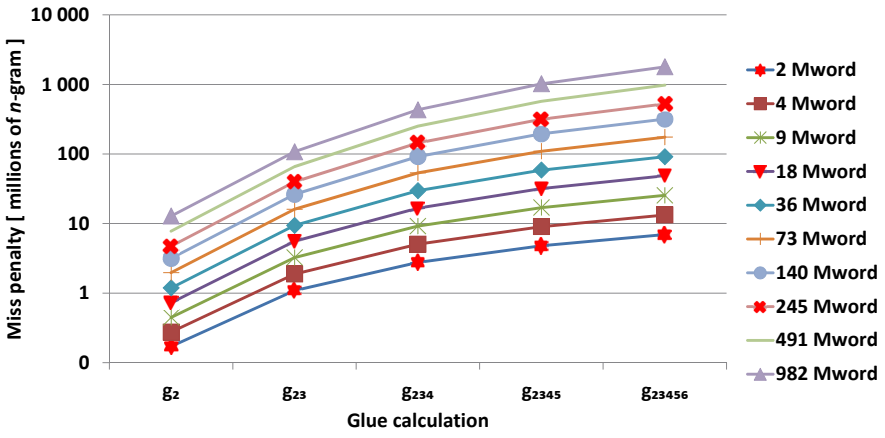


Figure 6.16: Single machine global warm-start miss penalty *versus* combined glue calculation, for different *corpus* sizes. The Y axis represents the global warm-start miss penalty in millions of n-grams and the X axis represents the glue calculation (g_2, g_{23}, ..., $g_{2\ldots6}$).

Figure 6.17 shows the global warm miss ratio *versus* the combined glue calculation. The warm-start miss ratio is given by:

$$mr_{Warm} = \frac{NumberWarmMisses}{\left|all_{glue_{g_2 \cdots g_n} Ref}\right|} \tag{6.19}$$

where $NumberWarmMisses = \sum_{i=1}^{n-1} |D_i|$ and $\left|all_{glue_{g_2 \cdots g_n} Ref}\right| = \sum_{i=2}^{n} 2 \times (i-1) \times |D_i|$. Thus:

$$mr_{Warm}(g_2 \cdots g_n) = \frac{|D_1| + \sum_{i=2}^{n-1} |D_i|}{2(n-1) \times |D_n| + \sum_{i=2}^{n-1} 2 \times (i-1) \times |D_i|} \tag{6.20}$$

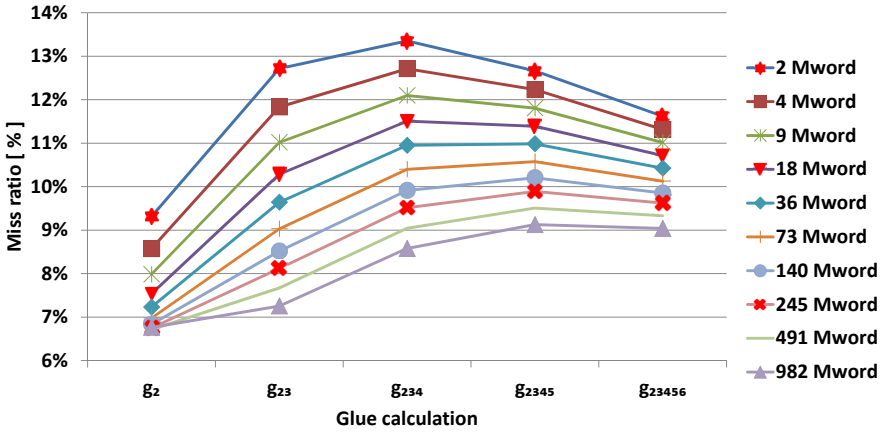


Figure 6.17: Single machine global warm-start miss ratio *versus* combined glue calculation. The Y axis represents the global warm-start miss ratio in percentage and the X axis represents the glue calculation (g_2, g_{23}, ..., $g_{2\cdots6}$).

The following conclusions can be taken:

- The global warm-start miss ratio first increases as n grows from 2 up to 4 for the *corpus* sizes up to 36 Mw, or it increases as n grows from 2 up to 5 for the other larger *corpus* sizes; and then decreases as n grows up to 6;
- The observed decrease of the global warm-start miss ratio is due to the greater influence of the D_n term in the denominator of expression (6.20) for the higher values of n (respectively, after 4 (tetragrams) in the case of smaller *corpora*, and after 5 (pentagrams) in the case of the larger *corpora*).

6.4.3 Overall Estimation of Cache Benefits for Multiple Machines ($K > 1$)

For simplicity of presentation in this section we focus on a fixed size *corpus* of 466 Mw, and consider a distinct number of machines, denoted by K: 1, 9, 16, 24, 32, 40, 48. Also we only consider a single case of a combined glue calculation, g_{234}.

Table 6.5 shows:

- The number of references to sub n-grams generated per machine (expressed in millions of n-grams), in the column labeled *nc* for the case of no-cache;

- The total number of global cold-start misses of the cache system C_{1+2+3} per machine, in column *cs* for the case of cold-start;

- The total number of global warm-start misses of the cache system per machine, in column ws_{h_1} for the case of warm-start and using the hash method h_1 (as presented in Figure 6.11) and the total number of warm-start misses per machine, in column ws_{h_3} for the case of warm-start and using the hash method h_3 (as presented in Figure 6.11);

- The ratios $mr_{Cold} = cs/nc$; for method h_1: $Warm_{misses}/Cold_{misses} = ws_{h_1}/cs$ and $mr_{Warm} = ws_{h_1}/nc$; for method h_3: $Warm_{misses}/Cold_{misses} = ws_{h_3}/cs$ and $mr_{Warm} = ws_{h_3}/nc$.

Table 6.5: Comparison of $No–cache/Cold–start/Warm–start$ cache system C_{1+2+3} for a *corpus* with 466 Mw for the combined glue g_{234} per machine. Miss ratio in percentage (%) and miss penalty in millions of n-grams.

K	nc	cs	ws_{h_1}	ws_{h_3}	cs/nc	ws_{h_1}/cs	ws_{h_1}/nc	ws_{h_3}/cs	ws_{h_3}/nc
1	2457	268	211	211	10.89	78.95	8.60	78.95	8.60
9	273	70	64	52	25.82	90.71	23.42	73.23	18.91
16	154	43	40	25	27.89	93.06	25.95	57.38	16.00
24	102	31	29	20	30.22	93.22	28.17	64.68	19.55
32	77	23	22	13	30.45	94.02	28.63	55.60	16.93
40	61	20	19	19	32.63	93.18	30.41	94.25	30.76
48	51	17	16	11	32.31	94.18	30.43	68.53	22.14

The following conclusions can be taken:

- The values in column *cs/nc* reflect the impact of the distribution of the distinct sub *n-grams* among the machines when the total number of machines increases, corresponding to an increase in the global cold-start miss ratio of the cache system per machine; However, that increase is moderate, for example when going from K=1 to 48, the global cold-start miss ratio increases only by a factor less than 4 (consistent with the power law behavior of the miss ratio proportional to K^{1-b}, with $0 < b < 1$, discussed in section 6.3.2, on page 142);

- The values in column *cs* reflect the global cold-start miss penalty of the cache system per machine when going from K=1 to 48, showing a reduction in the miss penalty by a factor close to 13 (consistent with the power law behavior of the miss penalty proportional to K^{-b}, with $0 < b < 1$, discussed in section 6.3.2);

- For any of the hashing methods, the values in columns ws_{h_1}/cs or ws_{h_3}/cs do not change significantly with increasing values of K, in agreement with the behavior of the cache warming-up efficiency (η) as observed in Figure 6.11, on page 147;

- For the two considered hashing methods in Table 6.5 the values in columns $mr_{Warm_{h_1}} = ws_{h_1}/nc$ and $mr_{Warm_{h_3}} = ws_{h_3}/nc$ are similar, although there is a reduction in the miss ratio for the hashing method h_3 due to its higher cache warming-up efficiency (Figure 6.11);
- When $K = 1$, i.e., with a single distinct n-gram table, for both methods h_1 and h_3, the global warm-start miss ratios (mr_{Warm}) are equal (8.6%); They are not zero because the global miss ratio that we are considering always includes the effect of the cold-start misses corresponding to the initial warming-up of the caches; Namely, for the glue calculation g_{234} the cache C_1 warming-up by g_2 includes all the distinct unigrams (D_1), the cache C_2 warming-up by g_3 includes all the distinct bigrams (D_2), and the cache C_3, starting empty, includes all the distinct trigrams (D_3) once the glue calculation g_4 is performed;
- For $K > 1$, the values in columns ws_{h_1} and ws_{h_3} reflect the global warm-start miss penalty per machine when going from K=1 to 48, showing a reduction in the miss penalty by a factor close to 13 for hashing method h_1, and by a factor close to 19 for hashing method h_3.

Note that in this global miss ratio calculation we always include all the observed n-gram misses, even the ones corresponding to the initial use of the cache from its cold-start state. This latter penalty will be progressively diminished as the cache system is repeatedly used by successive glue calculations which make references to already cached n-grams. In fact, the global warm-start miss ratio value would be much lower if we do not count the cold-start misses of each cache in the system, that is by considering a warm-start boundary for cache C_1 at the end of glue calculation of g_2, and a warm-start boundary for cache C_2 at the end of glue calculation of g_3, corresponding to not counting the C_1 and C_2 cold-start misses.

Overall Effect of the Warming-up upon the Global Miss Time Penalty. The above Table 6.5 only considers the total number of estimated misses. We can estimate their time impact by multiplying them by the average remote n-gram fetch time (t_{fetch}), which must be experimentally measured for each infrastructure. Assuming a t_{fetch} value of 10 microsecond per n-gram, we can obtain an indicative estimate of the time impact of the cache misses in each case, for example:

- Starting with the case of a single machine (K=1) without cache the estimated time for the remote fetch of the required 2,457 million of sub n-grams (see the upper left cell in Table 6.5) would be 6.83 hours;
- By increasing K to 48 machines still without any cache, the estimated time to remote fetch of the required 51 million of sub n-grams (see the lower left cell in Table 6.5) per machine would be 0.14 hours; This assumes that all machines are working in parallel and no network contention or server access conflicts occur;

- When considering a single machine with a cold-start cache system, the corresponding estimated time for the remote fetch of the required 268 million sub n-grams (see the cell under column labeled "cs" for the $K = 1$ row in Table 6.5) would be 0.74 hours;

- This time would be reduced to 0.05 hours for the remote fetch of the 17 million sub n-grams (see the cell under column labeled "cs" for the $K = 48$ row in Table 6.5) per machine when using 48 machines;

- However, when using a single machine with a cache warm-start strategy as proposed, the corresponding estimated time for the remote fetch of the required 211 million sub n-grams (see the cell under column labeled "ws_{h_3}" for the $K = 1$ row in Table 6.5) would be 0.59 hours;

- This time would be further reduced to 0.03 hours for the remote fetch of the 11 million sub n-grams (see the cell under column labeled "ws_{h_3}" for the $K = 48$ row in Table 6.5) per machine when using 48 machines.

In this example, the overall effect of going from a single machine without cache to 48 machines each one with a warm-up cache system, would amount to a reduction factor of 223 in the remote fetch of the sub n-grams per machine. In the latter case all the 48 machines are making the remote fetch requests in parallel.

The following figures give an overall view of the estimated behavior of the warm-start case, for a *corpus* with 466 Mw for the combined glue g_{234}:

- Figure 6.18 presents the evolution of the individual cache warm-start miss penalties ($C_1\,ws$, $C_2\,ws$ and $C_3\,ws$) and the global warm miss penalty of the cache system C_{1+2+3} ($Global\,ws$), and the global warm-start miss ratio (mr_{Warm}) as a function of K;

- Figure 6.19 compares the global warm-start with the global cold-start behavior of the cache system as a function of K.

- Figure 6.20 presents the evolution of the global warm-start miss penalty and the global warm-start miss ratio of the cache system as a function of the glue calculation, with a similar behavior as in the cold-start case.

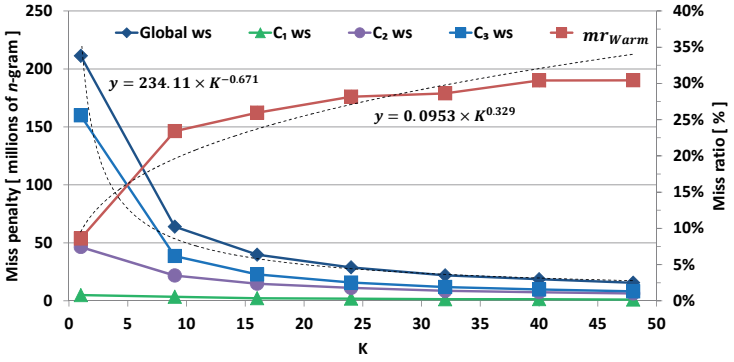


Figure 6.18: Per machine global warm-start miss penalty (*Global ws*, C_1 *ws*, C_2 *ws*, C_3 *ws*) and global warm-start miss ratio (mr_{Warm}) *versus* number of machines. The left Y axis represents the miss penalty in millions of *n*-grams; the right Y axis represents the miss ratio in percentage; and the X axis represents the number of machines.

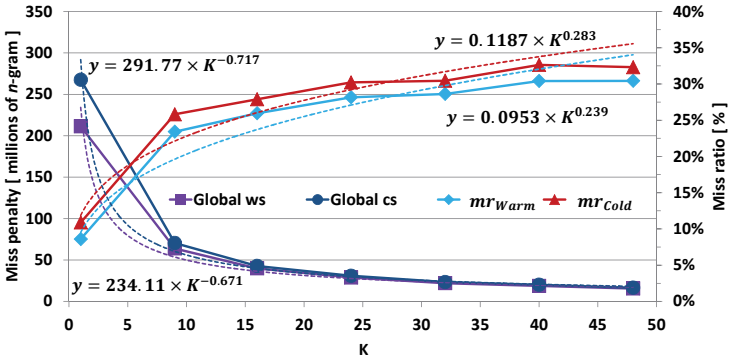
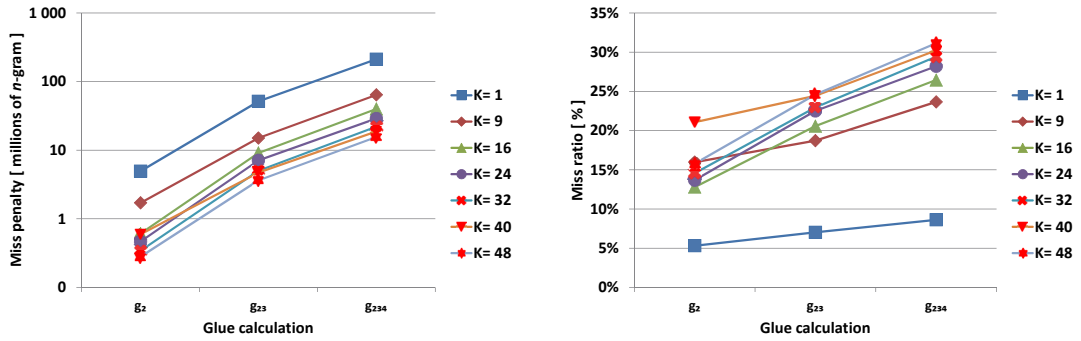


Figure 6.19: Per machine global miss penalty and global miss ratio — warm *versus* cold — as a function of the number of machines. The left Y axis represents the miss penalty in millions of *n*-grams; the right Y axis represents the miss ratio in percentage; and the X axis represents the number of machines.



a) *Global miss penalty* ***b)*** *Global miss ratio*

Figure 6.20: Per machine global warm-start miss penalty and global warm-start miss ratio *versus* combined glue calculation. The Y axis in subfigure a) represents the miss penalty in millions of *n*-grams and the Y axis in the subfigure b) represents the miss ratio in percentage. In both subfigures the X axis represents the glue calculation (g_2, g_{23}, ...).

6.5 Experimental Results in Real Execution Environments

We experimented with multiple runs of the Global method in the Lunacloud [Lun15] public cloud environment. We evaluated the extraction of relevant bigrams and trigrams from English *corpora* generated from Wikipedia [Wik16].

Each virtual machine used had similar characteristics: 4 CPU @ 1.5 GHz and a local disk storage of 40 GB and the same per machine main memory as described in Table 6.6. Each virtual machine included all the software components required: i) The Java classes for the Global method, the distributed in-memory KVS, and the AWARD workflow framework; ii) A JVM with version $\geq$ 1.7.X; iii) Linux Ubuntu with SSH access. The execution times shown are the average of 3 repeated runs for each experiment.

Here, we report on the results obtained in an environment (named "Environment B" as discussed in section 9.1.3.2 on page 292):

- Environment B was an improved implementation of the Global method with optimizations in the input processing and used an infinite cache system for all the *n*-gram sizes. This environment was used to process the following *corpus* sizes: 25, 227, 466, and 682 Mw, with distinct number of virtual machines (K): 1, 9, 16, 24, 32, 40, 48. For the considered configurations, for each value of K and *corpus* size, the virtual machines were configured with enough memory to ensure the behavior of an infinite cache system and the per machine memory configurations (in GB) used are presented in Table 6.6.

Table 6.6: Memory configurations (GB) used per machine in environment B

K	***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**
1	90			
9	85	85		
16	48	48	70	
24	32	32	49	85
32	24	24	37	64
40	19	19	29	51
48	16	16	25	43

As the *corpus* sizes increase it becomes impossible to execute the Global method in a single machine due to insufficient local memory. Thus all the considered experimental performed metrics are defined as relative to a minimum base number of machines necessary to run the algorithm for each *corpus* size.

6.5.1 *n*-gram Cache System Experimental Evaluation

In this evaluation we considered the glue calculations from g_2 up to g_{234} (which are required to evaluate the relevance of bigrams and trigrams) and we compared the cold-start and the warm-start approaches against each other, and also compared each of the approaches (cold-start and warm-start) against a no-cache configuration:

- The cold-start configuration corresponds to calculating each of the glues for bigrams, trigrams and tetragrams separately, each starting with its corresponding caches empty;

- The warm-start configuration corresponds to calculating each of the glues for bigrams, trigrams and tetragrams in a strict sequence, that is first calculating the glue for bigrams, then calculating the glues for trigrams (keeping cache C_1 with the previous unigrams contents), and finally calculating the glues for trigrams (keeping caches C_1 and C_2 respectively with the previous unigrams and bigrams contents);

- Note that the implementation enforces an incremental warm-up strategy for the caches; For the considered combined glue calculation g_{234} this implies that cache C_1 is first warmed by the glue calculation of bigrams followed by the warming-up due to the glue calculation of trigrams.

6.5.1.1 Real Execution Results

When executing the LocalMaxs Global method for the above mentioned ranges of *corpus* sizes and numbers of machines we considered different glue calculation cases: i) Isolated glue g_2; ii) Isolated glue g_3; iii) Combined glue g_{23}; iv) Isolated glue g_4; and v) Combined glue g_{234}. For each of the cases we present: i) The individual miss ratio of each involved cache; ii) For the combined glue calculations we present the global miss ratio of the entire cache system. During the execution of the algorithm, in each machine, the total number of generated references and the missed references to the local caches are registered. At the end of execution these values are collected into a log and used for calculation of the per machine miss ratios. These values are the averages among the K machines. In general, the miss ratio of an individual cache is obtained as follows:

$$mr = \frac{NumberObservedMisses}{NumberObservedReferences} \tag{6.21}$$

where $NumberObservedMisses$ is the number of n-grams that generate misses and the total number n-gram references generated are represented by $NumberObservedReferences$.

Tables 6.7 to 6.13 present the average values of the observed miss ratios among the multiple machines, for the different glue calculations, different values of K and the considered *corpus* sizes.

6.5.1.2 Isolated glue g_2 (C_1 with cold-start)

During the calculation of the isolated glue g_2, only cache C_1 is used. Figure 6.21 illustrates the set of distinct subunigrams in the $D_2(j)$ table.

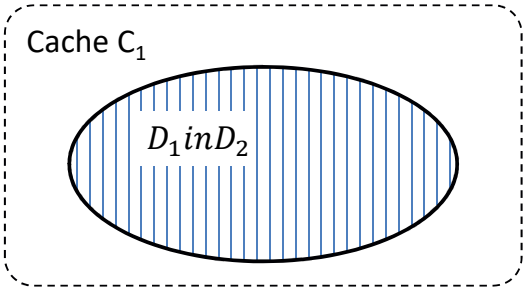


Figure 6.21: Cache misses for isolated glue g_2.

In this case the number of misses, numerator in expression (6.21), corresponds to the cold-start misses, i.e., the distinct subunigrams, $D_1 in_{D_2}(j)$, in each local partition $D_2(j)$. The total number of references, denominator in expression (6.21), corresponds to the total number of subunigrams in $D_2(j)$.

Table 6.7 shows the miss ratio, $mr_{Cold}(C_1, g_2)$, i.e., obtained from the observed measurements of the real execution for the individual cache C_1 with cold-start for the calculation of the isolated glue g_2 defined as:

$$mr_{Cold}(C_1, g_2, j) = \frac{\left|D_{1_{in}D_2}(j)\right|}{2 \times |D_2(j)|} \tag{6.22}$$

The values shown in Table 6.7 correspond to the average miss ratio among the K machines.

Table 6.7: Per machine miss ratio $mr_{Cold}(C_1, g_2)$ for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%).

K	$mr_{Cold}(C_1,g_2)$			
	***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**
1	7.00			
9	18.51	16.28		
16	21.16	18.23	15.56	
24	22.91	19.37	16.71	14.40
32	23.99	20.08	17.02	15.86
40	25.19	20.91	18.19	16.64
48	25.88	21.35	19.29	18.06

6.5.1.3 Isolated glue g_3 (C_1 warmed-up by g_2 and C_2 with cold-start)

During calculation of the isolated glue g_3 caches C_1 and C_2 are used. Glues g_2 and g_3 are executed successively in this ordering. Figure 6.22 illustrates, for cache C_2, the set of distinct subbigrams in the $D_3(j)$ table, and for cache C_1, the sets of distinct subunigrams in the $D_2(j)$ table and in the $D_3(j)$ table, with the aim to highlight the existence of common subunigrams shared by the glue calculation g_2 and g_3.

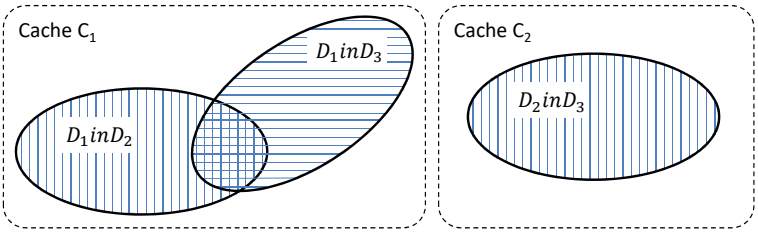


Figure 6.22: Cache misses for isolated glue g_3.

The missed references in cache C_1, numerator in expression (6.21), correspond to the cold-start misses when executing g_2 plus the misses when executing g_3. These latter misses correspond to the distinct subunigrams in the local partition $D_3(j)$ needed by glue g_3, except for the distinct subunigrams which were already previously loaded in cache C_1 by glue g_2. The set of missed references from cache C_1 during g_3 is given by:

$$Misses(C_1, g_3, j) = D_{1_{in}D_3}(j) \setminus \left(D_{1_{in}D_2}(j) \cap D_{1_{in}D_3}(j)\right) \tag{6.23}$$

The missed references for cache C_2 correspond to the cold-start misses when executing the glue g_3.

For both caches (C_1 and C_2) the total number of references during g_3, denominator in expression (6.21), corresponds to the total number of distinct sub n-grams (subunigrams in the case of cache C_1 and subbigrams in the case of cache C_2) in the $D_3(j)$ table.

Table 6.8 shows the miss ratios, $mr_{WarmedBy_{g_2}}(C_1, g_3)$ and $mr_{Cold}(C_2, g_3)$, obtained from the observed measurements of the real execution, for the individual caches C_1 and C_2 for the calculation of the isolated glue g_3 defined as:

$$mr_{WarmedBy_{g_2}}(C_1, g_3, j) = \frac{|Misses(C_1, g_3, j)|}{2 \times |D_3(j)|} \tag{6.24}$$

$$mr_{cold}(C_2, g_3, j) = \frac{\left|D_{2_{in}D_3}(j)\right|}{2 \times |D_3(j)|} \tag{6.25}$$

where $|Misses(C_1, g_3, j)|$ is defined in expression (6.23); $\left|D_{2_{in}D_3}(j)\right|$ is the total number of distinct subbigrams in the $D_3(j)$ table; and $|D_3(j)|$ is the total number of entries in the distinct trigrams table in the j machine, $1 \leq j \leq K$. The values shown in Table 6.8 correspond to the average miss ratio among the K machines.

Table 6.8: Per machine miss ratios $mr_{WarmedBy_{g_2}}(C_1, g_3)$ and $mr_{cold}(C_2, g_3)$ for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%).

K	$mr_{WarmedBy_{g_2}}(C_1,g_3)$				$mr_{cold}(C_2,g_3)$			
	***Corpus* size**				***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**	**25 Mw**	**227 Mw**	**466 Mw**	**682 Mw**
1	0.00				20.74			
9	5.16	3.84			48.30	39.84		
16	7.21	5.22	3.86		52.07	43.32	39.27	
24	8.52	6.08	4.56	3.25	54.57	45.81	41.81	38.89
32	9.66	6.80	4.92	3.78	55.93	47.18	42.99	40.58
40	10.44	7.29	5.60	4.25	57.24	48.57	44.64	41.56
48	11.23	7.78	6.23	4.86	58.00	49.40	47.15	43.41

6.5.1.4 Combined glue g_{23} (C_1 with cold-start for g_2 and with warm-start for g_3; and C_2 with cold-start for g_3)

For the combined glue g_{23} the set of missed references in cache C_1 during the execution of glue g_2 followed by glue g_3 is given by:

$$Misses(C_1, g_{23}, j) = D_{1_{in}D_2}(j) \cup \left(D_{1_{in}D_3}(j) \setminus \left(D_{1_{in}D_2}(j) \cap D_{1_{in}D_3}(j)\right)\right) \tag{6.26}$$

The missed references for cache C_2 correspond to the cold-start misses when executing the glue g_3.

Individual Cache Miss Ratios. Table 6.9 shows the miss ratios of the individual caches, $mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j)$ and $mr_{Cold}(C_2, g_{23}, j)$, obtained from the observed measurements of the real execution, for the individual caches C_1 and C_2 for the calculation of the combined glue g_{23} defined as:

$$mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j) = \frac{|Misses(C_1, g_{23}, j)|}{2 \times |D_2(j)| + 2 \times |D_3(j)|} \tag{6.27}$$

$$mr_{Cold}(C_2, g_{23}, j) = mr_{cold}(C_2, g_3, j) \tag{6.28}$$

where $Misses(C_1, g_{23}, j)$ is defined in expression (6.26). Expression (6.27) can be simplified to the following expression, based on the average values among the K machines:

$$\begin{aligned} mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j) = {} & \frac{2\times|D_2(j)|}{2\times|D_2(j)|+2\times|D_3(j)|} \times mr_{cold}(C_1, g_2, j) + \\ & \frac{2\times|D_3(j)|}{2\times|D_2(j)|+2\times|D_3(j)|} \times mr_{WarmedBy_{g_2}}(C_1, g_3, j) \end{aligned} \tag{6.29}$$

where $mr_{cold}(C_1, g_2, j)$ and $mr_{WarmedBy_{g_2}}(C_1, g_3, j)$ are the miss ratios from cache C_1 obtained, respectively, from expressions (6.22) and (6.24). We consider $|D_2(j)| = |D_2|/K$ and

$|D_3(j)| = |D_3|/K$. The miss ratio for cache C_2 is only due to the calculation of glue g_3 (expression (6.25)).

Table 6.9: Per machine miss ratios $mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23})$ and $mr_{Cold}(C_2, g_{23})$ for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%)

K	$mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1,g_{23})$				$mr_{Cold}(C_2,g_{23})$			
	***Corpus* size**				***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**	**25 Mw**	**227 Mw**	**466 Mw**	**682 Mw**
1	2.05				20.74			
9	9.08	6.95			48.30	39.84		
16	11.30	8.47	6.63		52.07	43.32	39.27	
24	12.74	9.40	7.43	5.82	54.57	45.81	41.81	38.89
32	13.86	10.11	7.78	6.56	55.93	47.18	42.99	40.58
40	14.77	10.69	8.57	7.10	57.24	48.57	44.64	41.56
48	15.53	11.17	9.32	7.90	58.00	49.40	47.15	43.41

Global Miss Ratio. Table 6.10 shows the global miss ratio for g_{23} of the entire cache system C_{1+2}, as obtained from the following expression:

$$mr(C_{1+2}, g_{23}, j) = \frac{2\times|D_2(j)|+2\times|D_3(j)|}{2\times|D_2(j)|+4\times|D_3(j)|} \times mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j) + \frac{2\times|D_3(j)|}{2\times|D_2(j)|+4\times|D_3(j)|} \times mr_{Cold}(C_2, g_{23}, j) \quad (6.30)$$

where $mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j)$ and $mr_{Cold}(C_2, g_{23}, j)$ are obtained, respectively, from expressions (6.29) and (6.28). We consider $|D_2(j)| = |D_2|/K$ and $|D_3(j)| = |D_3|/K$.

Table 6.10: Per machine global miss ratio $mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_{1+2}, g_{23})$ for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%)

K	$mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_{1+2},g_{23})$			
	***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**
1	9.79			
9	25.32	21.05		
16	28.18	23.41	20.76	
24	30.06	25.01	22.32	20.20
32	31.28	26.00	23.03	21.36
40	32.36	26.93	24.19	22.09
48	33.12	27.56	25.70	23.35

6.5.1.5 Isolated glue g_4 (C_1 warmed-up by g_2 and g_3, C_2 warmed-up by g_3 and C_3 with cold-start)

During calculation of the isolated glue g_4, caches C_1, C_2 and C_3 are used. Glues are executed successively in the order g_2, g_3 and g_4. Figure 6.23 shows: for cache C_1, the

intersections of the sets of distinct subunigrams in tables $D_2(j)$, $D_3(j)$ and $D_4(j)$; while for cache C_2, it shows the intersection of the sets of distinct subbigrams in tables $D_3(j)$ and $D_4(j)$; and for cache C_3, it just shows the set of distinct subtrigrams in table $D_4(j)$.

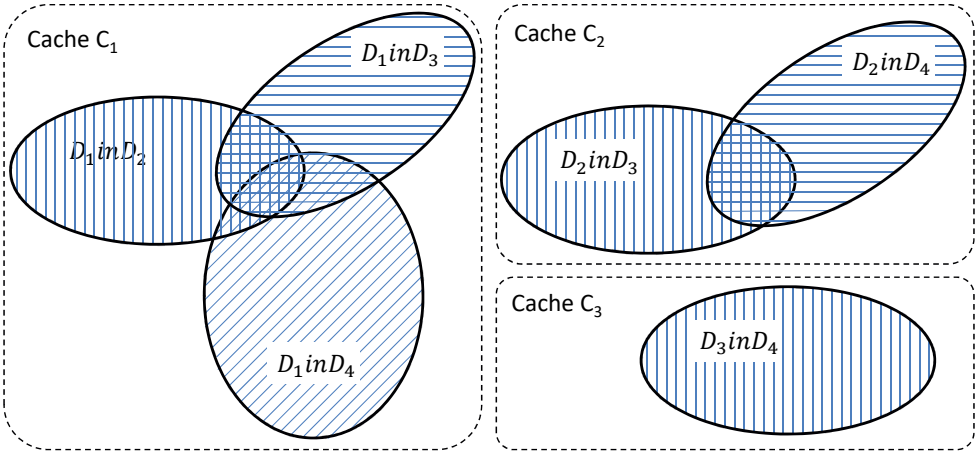


Figure 6.23: Cache misses for isolated glue g_4.

The missed references in cache C_1, numerator in expression (6.21), correspond to the cold-start misses when executing g_2 plus the misses when executing g_3 and g_4. These latter misses correspond to the distinct subunigrams in the $D_4(j)$ needed by g_4 except for the distinct subunigrams there were already previously loaded in cache C_1 by glues g_2 and g_3. In this way we capture the behavior that is obtained by the implementation concerning the incremental warm-up of an individual cache by successive glue calculations.

The set of missed references for cache C_1 during glue g_4 is given by:

$$Misses(C_1, g_4, j) = D_{1_{in}D_4}(j) \setminus \left[\left(D_{1_{in}D_2}(j) \cap D_{1_{in}D_4}(j)\right) \cup \left[\left(D_{1_{in}D_3}(j) \cap D_{1_{in}D_4}(j)\right) \setminus \left(D_{1_{in}D_2}(j) \cap D_{1_{in}D_3}(j) \cap D_{1_{in}D_4}(j)\right)\right]\right] \tag{6.31}$$

where $D_{1_{in}D_2}(j)$, $D_{1_{in}D_3}(j)$, and $D_{1_{in}D_4}(j)$ are the sets of distinct subunigrams respectively in $D_2(j)$, $D_3(j)$ and $D_4(j)$ tables in the j machine, $1 \leq j \leq K$.

The missed references for cache C_2 correspond to the cold-start misses when executing the glue g_3 plus the misses when executing g_4. Thus, the set of misses for cache C_2 during glue g_4 is given by:

$$Misses(C_2, g_4, j) = D_{2_{in}D_4}(j) \setminus \left(D_{2_{in}D_4}(j) \cap D_{2_{in}D_3}(j)\right) \tag{6.32}$$

where $D_{2_{in}D_3}(j)$ and $D_{2_{in}D_4}(j)$ are the sets of distinct subbigrams respectively in $D_3(j)$ and $D_4(j)$ tables in the j machine, $1 \leq j \leq K$.

For all the caches (C_1, C_2 and C_3) the total number of references, denominator in expression (6.21), corresponds to the total number of sub n-grams (subunigrams for

cache C_1, subbigrams for cache C_2, and subtrigrams for cache C_3) in each local partition $D_4(j)$.

Table 6.11 shows the miss ratios, $mr_{WarmedBy_{g_2\&g_3}}(C_1, g_4)$, $mr_{WarmedBy_{g_3}}(C_2, g_4)$ and $mr_{Cold}(C_3, g_4)$, obtained from the observed measurements of the real execution, for the individual caches C_1, C_2 and C_3 for the calculation of the isolated glue g_4 defined as:

$$mr_{WarmedBy_{g_2\&g_3}}(C_1, g_4, j) = \frac{|Misses(C_1, g_4, j)|}{2 \times |D_4(j)|} \tag{6.33}$$

$$mr_{WarmedBy_{g_3}}(C_2, g_4, j) = \frac{|Misses(C_2, g_4, j)|}{2 \times |D_4(j)|} \tag{6.34}$$

$$mr_{Cold}(C_3, g_4, j) = \frac{\left|D_{3_{in}D_4}(j)\right|}{2 \times |D_4(j)|} \tag{6.35}$$

where $|Misses(C_1, g_4, j)|$ is defined in expression (6.31); $|Misses(C_2, g_4, j)|$ is defined in expression (6.32); $|D_3 in D_4(j)|$ is the total number of distinct subtrigrams in the $D_4(j)$ table; and $|D_4(j)|$ is the total number of entries in the distinct tetragrams table in the j machine, $1 \leq j \leq K$. The values shown in Table 6.11 correspond to the average miss ratios among the K machines.

Table 6.11: Per machine miss ratios , $mr_{WarmedBy_{g_2\&g_3}}(C_1, g_4)$, $mr_{WarmedBy_{g_3}}(C_2, g_4)$ and $mr_{Cold}(C_3, g_4)$ for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%).

K	$mr_{WarmedBy_{g_2\&g_3}}(C_1,g_4)$				$mr_{WarmedBy_{g_3}}(C_2,g_4)$				$mr_{Cold}(C_3,g_4)$			
	***Corpus* size**				***Corpus* size**				***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**	**25 Mw**	**227 Mw**	**466 Mw**	**682 Mw**	**25 Mw**	**227 Mw**	**466 Mw**	**682 Mw**
1	0.00				0.00				35.21			
9	2.41	1.57			25.61	17.84			72.97	64.89		
16	3.45	2.15	1.44		33.15	23.35	19.65		76.12	67.77	64.32	
24	4.31	2.64	1.80	1.33	37.64	26.76	22.78	18.25	78.02	69.90	66.57	63.78
32	5.00	3.04	2.08	1.58	40.84	29.29	24.19	20.82	78.87	70.91	67.55	65.26
40	5.65	3.48	2.35	1.92	43.21	31.22	25.60	23.33	79.74	71.92	68.08	66.81
48	6.02	3.62	2.50	2.22	44.99	32.63	26.45	25.02	80.21	72.50	69.71	66.81

6.5.1.6 Combined glue g_{234} (C_1 with cold-start for g_2 and with warm-start for g_3 and g_4; C_2 with cold-start for g_3 and with warm-start for g_4; C_3 with cold-start for g_4)

Individual Cache Miss Ratios. Table 6.12 shows the individual cache miss ratios, for the calculation of the combined glue g_{234}, which are defined as follows:

For cache C_1:

$$mr_{ColdFor_{g2};WarmedBy_{g_2}For_{g3};WarmedBy_{g2\&g3}For_{g4}}(C_1, g_{234}, j) = \frac{2\times|D_2|+2\times|D_3|}{2\times|D_2|+2\times|D_3|+2\times|D_4|} \times mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j) + \frac{2\times|D_4|}{2\times|D_2|+2\times|D_3|+2\times|D_4|} \times mr_{WarmedBy_{g_2\&g_3}For_{g_4}}(C_1, g_4, j) \quad (6.36)$$

where $mr_{ColdFor_{g_2};WarmFor_{g_3}}(C_1, g_{23}, j)$ is obtained from expression (6.29), and from expression (6.33) we obtain $mr_{WarmedBy_{g_2\&g_3}For_{g_4}}(C_1, g_4, j)$.

For cache C_2:

$$mr_{ColdFor_{g3};WarmedBy_{g_3}For_{g4}}(C_2, g_{234}, j) = \frac{\left|D_{2_{in}D_3}(j)\right|+\left|D_{2_{in}D_4}(j)\right|-\left|\left(D_{2_{in}D_4}(j)\cap D_{2_{in}D_3}(j)\right)\right|}{2\times|D_3(j)|+2\times|D_4(j)|} \quad (6.37)$$

which simplifies to the following expression:

$$mr_{ColdFor_{g3};WarmedBy_{g_3}For_{g4}}(C_2, g_{234}, j) = \frac{2\times|D_3|}{2\times|D_3|+2\times|D_4|} \times mr_{Cold}(C_2, g_3, j) + \frac{2\times|D_4|}{2\times|D_3|+2\times|D_4|} \times mr_{WarmedBy_{g_3}}(C_2, g_4, j) \quad (6.38)$$

For cache C_3:

$$mr_{Cold}(C_3, g_{234}, j) = mr_{Cold}(C_3, g_4, j) \quad (6.39)$$

where $mr_{Cold}(C_2, g_3, j)$ and $mr_{WarmedBy_{g_3}}(C_2, g_4, j)$ are obtained from expressions (6.25) and (6.34), and $mr_{Cold}(C_3, g_4, j)$ is obtained from expression (6.35). The values shown in Table 6.12 correspond to the average miss ratios among the K machines.

Table 6.12: Per machine miss ratios $mr_{WarmedBy_{g2\&g3}}(C_1, g_{234})$, $mr_{WarmedBy_{g_3}}(C_2, g_{234})$ and $mr_{Cold}(C_3, g_{234})$ for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%).

K	$mr_{WarmedBy_{g2\&g3}}(C_1,g_{234})$				$mr_{WarmedBy_{g_3}}(C_2,g_{234})$				$mr_{Cold}(C_3,g_{234})$			
	***Corpus* size**				***Corpus* size**				***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**	**25 Mw**	**227 Mw**	**466 Mw**	**682 Mw**	**25 Mw**	**227 Mw**	**466 Mw**	**682 Mw**
1	1.02				8.57				35.21			
9	5.74	3.96			34.99	26.10			72.97	64.89		
16	7.37	4.96	3.63		40.97	30.85	26.68		76.12	67.77	64.32	
24	8.52	5.65	4.18	3.21	44.63	33.92	29.60	25.60	78.02	69.90	66.57	63.78
32	9.42	6.19	4.49	3.66	47.07	36.01	30.93	27.86	78.87	70.91	67.55	65.26
40	10.20	6.69	4.98	4.09	49.01	37.74	32.42	29.83	79.74	71.92	68.08	66.81
48	10.76	6.98	5.38	4.59	50.37	38.93	33.87	31.57	80.21	72.50	69.71	66.81

Global Cache Miss Ratio. Table 6.13 shows the global miss ratio for g_{234} of the entire cache system C_{1+2+3} in each machine, as obtained from the following expression:

$$mr(C_{1+2+3}, g_{234}, j) =$$

$$\begin{aligned}
&\frac{2\times|D_2(j)|+2\times|D_3(j)|+2\times|D_4(j)|}{2\times|D_2(j)|+4\times|D_3(j)|+6\times|D_4(j)|} \times mr_{ColdFor_{g2};WarmedBy_{g_2}For_{g3};WarmedBy_{g_2\&g_3}For_{g_4}}(C_1, g_{234}, j) + \\
&\frac{2\times|D_3(j)|+2\times|D_4(j)|}{2\times|D_2(j)|+4\times|D_3(j)|+6\times|D_4(j)|} \times mr_{ColdFor_{g3};WarmedBy_{g_3}For_{g4}}(C_2, g_{234}, j) + \\
&\frac{2\times|D_4(j)|}{2\times|D_2(j)|+4\times|D_3(j)|+6\times|D_4(j)|} \times mr_{Cold}(C_3, g_{234}, j)
\end{aligned} \tag{6.40}$$

where $mr_{ColdFor_{g2};WarmedBy_{g_2}For_{g3};WarmedBy_{g_2\&g_3}For_{g_4}}(C_1, g_{234}, j)$ is obtained from expression (6.36), $mr_{ColdFor_{g3};WarmedBy_{g_3}For_{g_4}}(C_2, g_{234}, j)$ from expression (6.37) and $mr_{Cold}(C_3, g_{234}, j)$ from expression (6.39). We also consider $|D_2(j)| = |D_2|/K$, $|D_3(j)| = |D_3|/K$ and $|D_4(j)| = |D_4|/K$.

Table 6.13: Per machine global miss ratio for g_{234} of the entire cache system C_{1+2+3}, with cache C_1 (cold-start for g_2 and warm-start for g_3 and g_4), cache C_2 (cold-start for g_3 and warm-start for g_4) and cache C_3(cold-start) for *corpus* sizes ranging from 25 up to 682 Mw and different values of K. Miss ratios in percentage (%).

K	$mr_{ColdFor_{g_2};WarmFor_{g_3\&g_4}}(C_{1+2+3}, g_{234})$			
	***Corpus* size**			
	25 Mw	**227 Mw**	**466 Mw**	**682 Mw**
1	11.03			
9	30.65	25.85		
16	34.18	28.64	26.15	
24	36.40	30.52	27.96	25.53
32	37.85	31.73	28.79	26.88
40	39.06	32.80	29.66	28.13
48	39.90	33.48	30.73	28.97
Avg.	32.72	30.50	28.66	27.37

Evolution of the Miss Ratios and Miss Penalties for Combined Glue g_{234}. The following figures (obtained from the above presented tables) give an overall view of the observed behavior of the warm-start case for the combined glue g_{234}.

Figure 6.24 presents the evolution of the global miss ratio ($mr_{Warm}(C_{1+2+3}, g_{234})$) of the cache system C_{1+2+3} as a function of K, for different *corpus* sizes. The values were obtained as averages over the K machines.

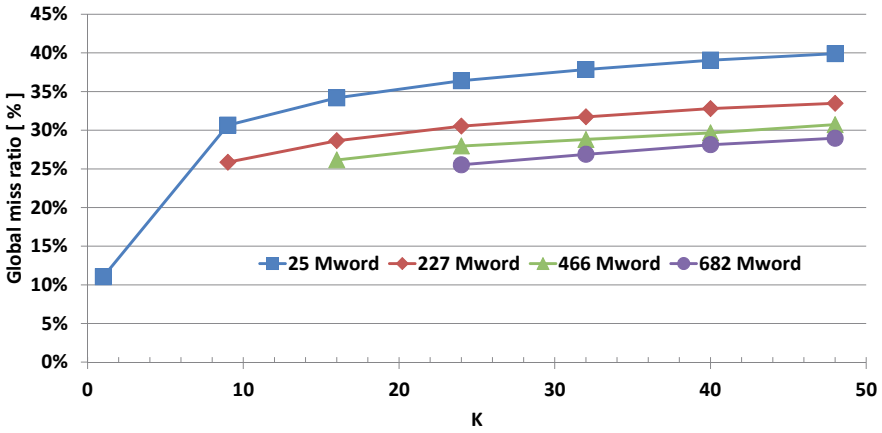
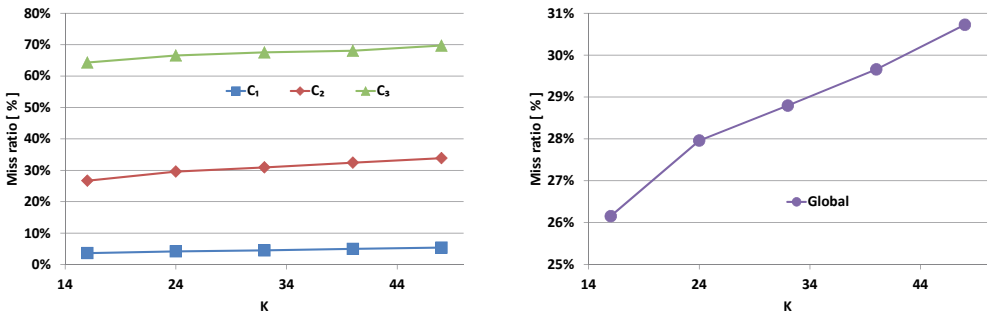


Figure 6.24: Per machine global miss ratio *versus* number of machines for different *corpus* sizes and g_{234}. The Y axis represents the global miss ratio in percentage and the X axis represents the number of machines.

Figure 6.25 presents a finer detail of the global miss ratio ($mr_{Warm}(C_{1+2+3}, g_{234})$) in terms of its component individual cache miss ratios (C_1, C_2 and C_3) for a fixed *corpus* size of 466 Mw.



a) *Miss ratio of individual caches* ***b)*** *Global miss ratio (C_{1+2+3})*

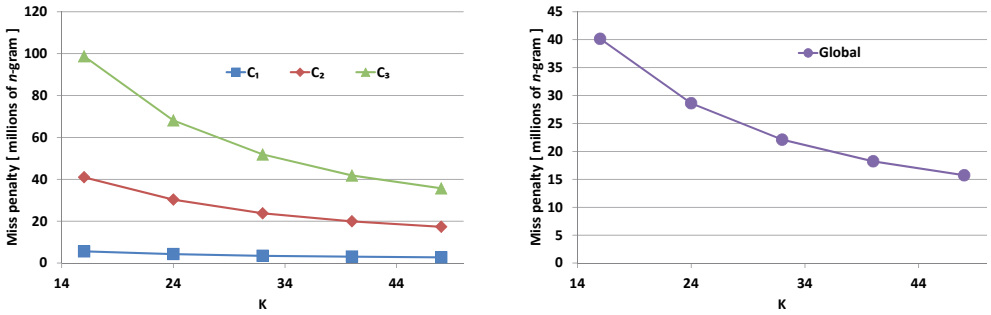
Figure 6.25: Per machine miss ratios (individual caches C_1, C_2 and C_3 and Global C_{1+2+3}) *versus* number of machines for glue g_{234} and a *corpus* size of 466 Mw. The Y axis represents the miss ratio in percentage and the X axis represents the number of machines.

Figure 6.24 and 6.25 show:

- The per machine warm-start miss ratios, both global and individual, increase with K, which is consistent with the model estimates;
- The per machine global miss ratio $mr_{Warm}(C_{1+2+3}, g_{234})$ presents an average value around 30% for the observed *corpus* range up to 682 Mw, when the number of machines goes up to 48, while the per machine individual cache miss ratios exhibit significantly different average values, e.g., for the *corpus* of 466 Mw: around 5% for C_1, around 30% for C_2, and around 70% for C_3; The differences observed between the individual cache values are due to the fact that both C_1 and C_2 benefit from the

warming-up by the previous glue calculations (g_2 for C_1 and g_3 for C_2), while cache C_3, only used for g_4, starts cold; Besides, the set $D_{3_{in}D_4}$ of the distinct subtrigrams that determine the misses in cache C_3, is much larger than the corresponding sets of distinct sub n-grams in caches C_1 and C_2, and a significant percentage of distinct trigrams in $D_{3_{in}D_4}$ are singletons in this *corpus* size range, originating lesser cache C_3 reutilization.

Figure 6.26 presents the evolution of the global miss penalty (*Global ws*) of the cache system C_{1+2+3} and the corresponding individual cache miss penalties (C_1 *ws*, C_2 *ws*, C_3 *ws*) for a fixed *corpus* size of 466 Mw, and Figure 6.27 presents the evolution of the global miss ratio ($mr_{Warm}(C_{1+2+3}, g_{234})$) and of the global miss penalty (*Global ws*) for the cache system C_{1+2+3}, as a function of the glue calculation for different values of K.



a) *Miss penalties of individual caches* ***b)*** *Global miss penalty*

Figure 6.26: Per machine miss penalties (individual caches C_1, C_2 and C_3 and Global C_{1+2+3}) for glue g_{234} and a *corpus* size of 466 Mw. The Y axis represents the miss penalty in millions of n-grams an the X axis represents the number of machines.

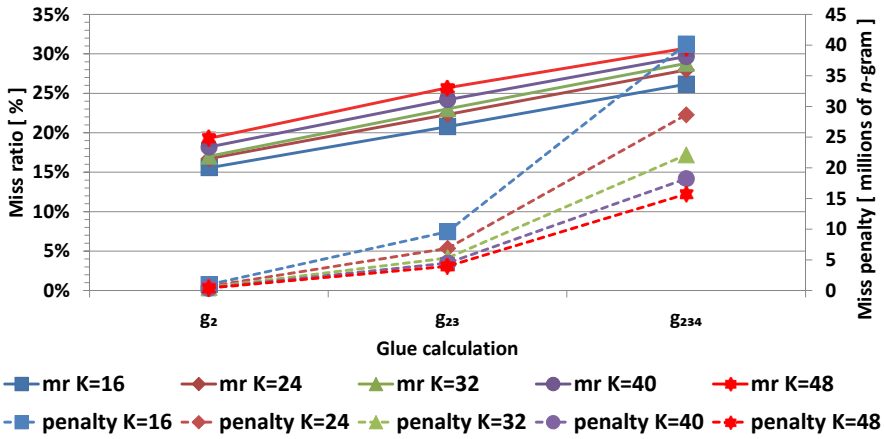


Figure 6.27: Per machine global miss ratio and miss penalty *versus* combined glue calculations for a *corpus* size of 466 Mw. The left Y axis represents the global miss ratio in percentage; the right Y axis represents the global miss penalty in millions of n-grams; and the X axis represents the glue calculation (g_2, g_{23} and g_{234}).

The observed behaviors concerning the miss ratios and miss penalties as a function of K, *corpus* size and glue calculations, are overall consistent with the model estimates.

6.5.1.7 Impact of the *n*-gram Cache in Phase Two Execution Time

In the results presented in the previous sections, it was shown how the inclusion of an infinite dynamic cache in the implementation of phase two of the Global method contributes to reduce the global miss ratio. In this section we present experimental results that show the impact of such cache system upon the execution time of phase two of the Global method.

Table 6.14 presents the measured phase two execution times for the warm-start ($T_2(ws)$) and the cold-start ($T_2(cs)$) cases, and their ratio ($T_2(ws)/T_2(cs)$). These values correspond to the measurements obtained from the real execution of phase two in "Environment B" for a *corpus* size of 466 Mw and the combined glue g_{234}, with the number of machines ranging from 16 to 48. The presented values are averaged among all the machines.

Table 6.14: Measured phase two total execution times (in minutes): warm-start *versus* cold-start, and time ratio, for different numbers of machines and a fixed *corpus* size of 466 Mw and g_{234}.

K	$T_2(ws)$	$T_2(cs)$	$T_2(ws)/T_2(cs)$
16	36.60	38.92	0.94
24	22.13	23.90	0.93
32	17.27	18.94	0.91
40	15.64	17.35	0.90
48	14.10	15.97	0.88

By using expression (5.33) the phase two execution times in each machine j can be approximated by:

$$T_2(j) \simeq T_{Glue}(j) + T_{comm}(j) \tag{6.41}$$

This expression applies both to the warm-start (ws) and no-cache (nc) cases.

We observe that $T_{Glue}(j) \ll T_{comm}(j)$. For example, for the no-cache case the ratio $T_{Glue}(j)/T_{comm_{nc}}(j) \approx \left(N_n(phase2) \times t_{g_n}\right)/\left(all_{glue_{2\cdots6}Ref} \times t_{fetch}\right)$, ranges from $0.14 \times t_{g_n}/t_{fetch}$ to $0.10 \times t_{g_n}/t_{fetch}$ when we go from the *corpus* of 1.6×10^7 to a *corpus* of 1.3×10^{14} words (as in Table 5.11, on page 111), and assuming a ratio $t_{g_n}/t_{fetch} \approx 1/20$ (as in expression (5.47), on page 126), the above ratio varies from 0.007 to 0.005 in the above *corpus* range. So, expression (6.41) can be approximated by:

$$T_{2_{nc}}(j) \simeq T_{comm_{nc}}(j) \tag{6.42}$$

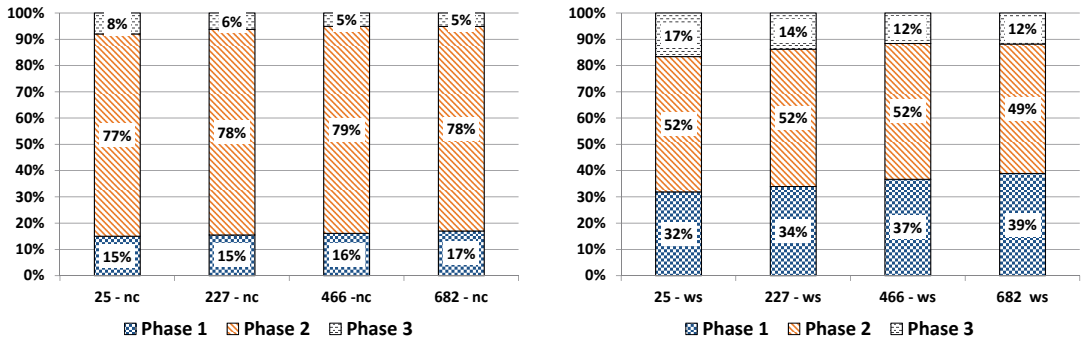
Regarding the warm-start case the above approximation remains valid when the miss ratio (mr_{ws}) is in the order of 30% as happened in this experiment, as presented in Table 6.15 showing the measured miss ratio values for the warm-start case.

Table 6.15: Measured phase two warm-start miss ratio (in percentage) for different numbers of machines and a fixed *corpus* size of 466 Mw and g_{234}.

K	mr_{ws}
16	26%
24	28%
32	29%
40	30%
48	30%

Thus, the ratio $T_2(ws)/T_2(nc)$ can be approximated by $T_{comm_{ws}}/T_{comm_{nc}} \approx mr_{ws}$.

Under the above approximations and using the values in Tables 6.14, 6.15 and 6.16 (presented ahead) we obtained the following percentages of the weight of each individual phase time with respect to the total execution time in the no-cache configuration (Figure 6.28-a)). In Figure 6.28-b) we show the corresponding weights for the phase times in the warm-start case.



a) *No-cache* ***b)*** *Warm-start cache*

Figure 6.28: Average distribution of the Global method per-phase execution times for the *corpora* 25, 227, 466 and 682 Mw, over K from 1 to 48 machines, for glue g_{234}, in the no-cache *versus* the warm-start cache.

From the above we conclude that the percentage of the phase two execution time with respect to the total execution time of the Global method is reduced from an average value of 78% in the configuration without cache to an average value of 51% for the warmed-up cache system.

Let us consider the total execution time ($T_{Total}(.)$) of the Global method algorithm as the sum of the execution times of its phases in the cases of warm-start (ws) and no-cache (nc) configurations:

$$T_{Total}(ws) = T_1 + T_2(ws) + T_3 \tag{6.43}$$

$$T_{Total}(nc) = T_1 + T_2(nc) + T_3 \tag{6.44}$$

As the execution times of phase one (T_1) and three (T_3) are unchanged when including a cache (recall we are only using a cache in phase two), we obtain the following ratio of the total execution time with cache over the execution time without cache:

$$\frac{T_{Total}(ws)}{T_{Total}(nc)} = \frac{T_2(ws)}{T_2(nc)} \cdot \frac{p_{2inTotal}(nc)}{p_{2inTotal}(ws)} \tag{6.45}$$

where $p_{2inTotal}(nc) = T_2(nc)/T_{Total}(nc)$ represents the ratio of the phase two execution time *versus* the total execution time without cache, and $p_{2inTotal}(ws) = T_2(ws)/T_{Total}(ws)$ represents the ratio of the phase two execution time *versus* the total execution time when using a warm-start cache.

From the above measurements (Figure 6.28) $p_{2inTotal}(nc) = 0.78$ and $p_{2inTotal}(ws) = 0.51$ for the conducted experiments. From Table 6.13 we obtain the value of 0.3 for the ratio $T_2(ws)/T_2(nc)$, leading to a ratio of $T_{Total}(ws)/T_{Total}(nc) = 0.45$, which corresponds to a reduction of 55% in the total execution time of the Global method due to the warm-cache in this range of *corpus* sizes and machine numbers. This is illustrated in Figure 6.29.

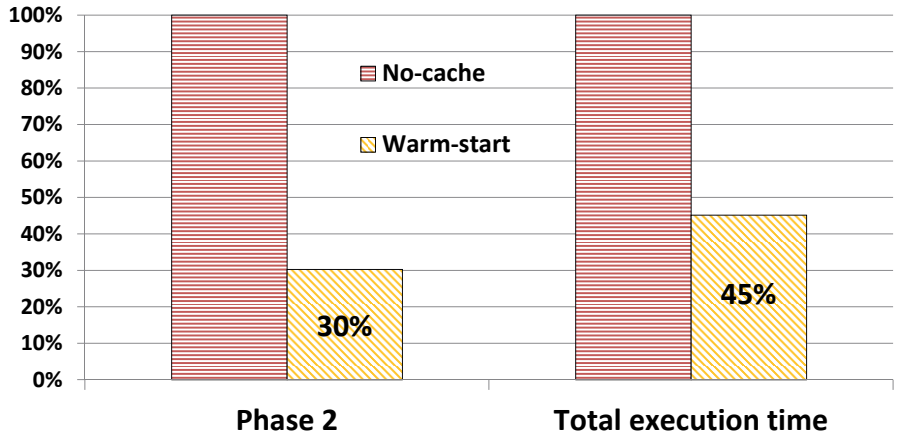


Figure 6.29: Phase two execution time reduction *versus* total execution time reduction.

6.5.2 Phase Two Relative Speedup and Relative Sizeup Evaluation

Table 6.16 shows the measured values of the execution time of phase two and of the total execution time of the Global method in "Environment B", for relevant expressions evaluation re_{23} of bigrams and trigrams with different *corpus* sizes.

Table 6.16: Phase two and total Global method execution times for re_{23} in environment B, in minutes.

	K	*Corpus* size			
		25 Mw	227 Mw	466 Mw	682 Mw
Phase 2	**1**	19.15			
	9	3.66	32.20		
	16	2.64	18.81	36.60	
	24	1.91	13.25	22.13	33.26
	32	1.72	10.06	17.27	24.13
	40	1.55	8.33	15.64	19.42
	48	1.52	8.29	14.10	16.49
Total of phases 1, 2 and 3	**1**	40.10			
	9	6.78	61.64		
	16	5.01	35.38	69.53	
	24	3.70	25.48	42.69	63.97
	32	3.22	19.12	33.71	47.26
	40	3.05	16.19	31.17	40.39
	48	3.02	15.90	26.82	35.84

Figure 6.30 shows the phase two execution time behavior, corresponding to Table 6.16, *versus* the number of machines (K) for the considered *corpus* sizes.

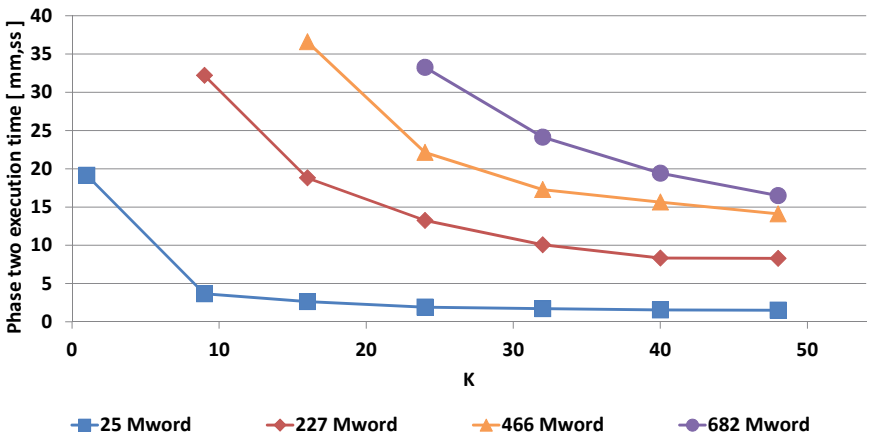


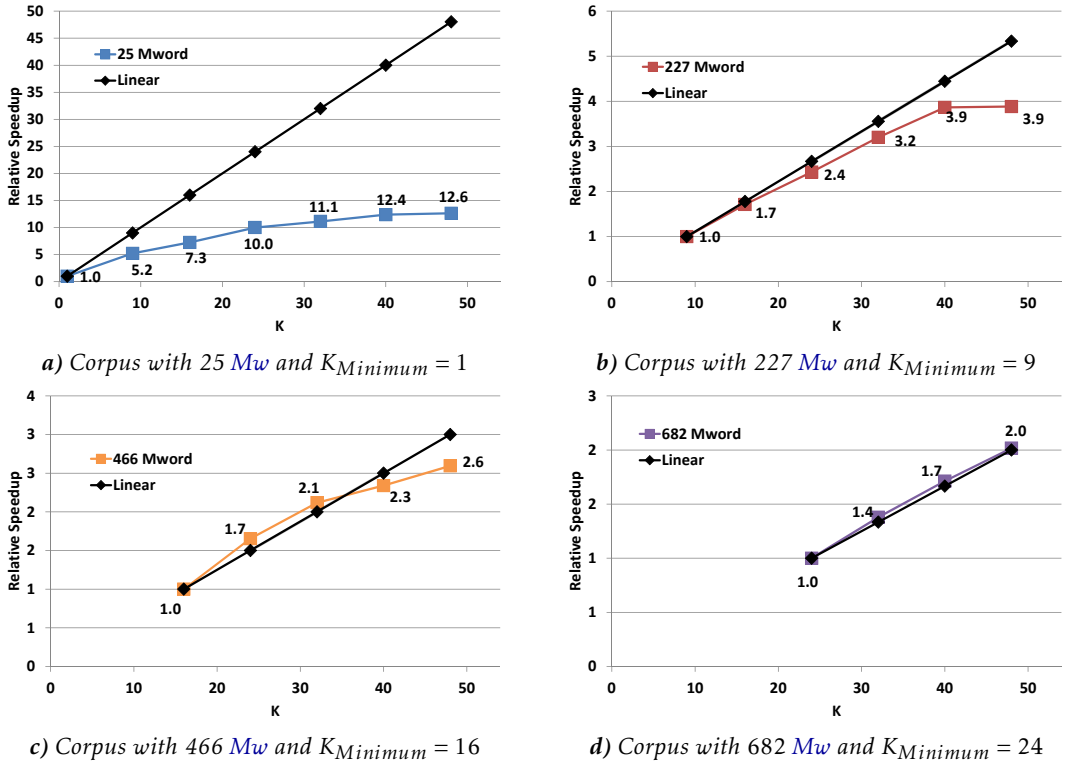
Figure 6.30: Execution time for phase two as a function of K for re_{23}. The Y axis represents the execution time in minutes and the X axis represents the number of machines.

For any fixed *corpus* size in the experimented range (except for the *corpus* of 25 Mw) the phase two execution time is almost proportional to $1/K$ thus leading to an almost linear relative speedup.

In the following we present the results corresponding to the relative performance metrics, when going from K_1 to K_2 machines. For each *corpus* size K_1 is the minimal number of machines required to execute phase two and it is the value used as the reference in relation to which the relative performance is evaluated. For example, in the case

of relative speedup $Sp_{K_1 \to K}(K) = \frac{T(K_1)}{T(K)}$ and the corresponding values of this ratio are presented for all the K values in the experimented range of machine numbers.

Relative (fixed problem size) Speedup. Figure 6.31 shows the relative speedup (Definition 2.5, on page 20) for phase two *versus* the number of machines (K), for re_{23} with different *corpus* sizes. The lines with the filled lozenge (⧫) represent the linear (ideal) relative speedup. The lines with a filled square (■) represent the obtained relative speedup.



a) Corpus with 25 Mw and $K_{Minimum} = 1$

b) Corpus with 227 Mw and $K_{Minimum} = 9$

c) Corpus with 466 Mw and $K_{Minimum} = 16$

d) Corpus with 682 Mw and $K_{Minimum} = 24$

Figure 6.31: Relative (fixed problem size) speedup for phase two (strong scaling) as function of K for different *corpus* sizes. The Y axis represents the relative speedup and the X axis represents the number of machines.

Relative Efficiency. For any fixed *corpus* size in the experimented range (except for the *corpus* of 25 Mw), and as a result of the almost linear relative speedup, the relative efficiency (Definition 2.6, on page 21) is close to 100% and stays almost constant with K. Figure 6.32 shows the relative efficiency for phase two.

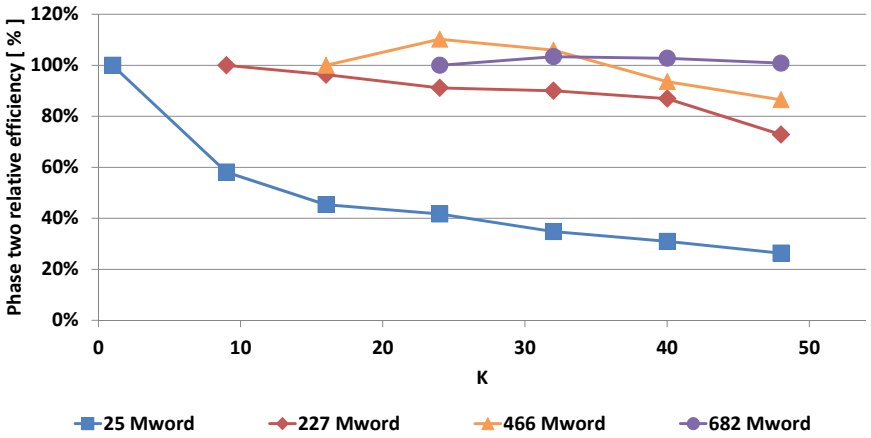


Figure 6.32: Relative efficiency for phase two as function of K for different *corpus* sizes. The Y axis represents the relative efficiency in percentage and the X axis represents the number of machines.

Relative Sizeup. Figure 6.33 shows the relative sizeup based on Definition 2.7, on page 21, for different configurations of *corpus* sizes and numbers of machines, as indicated next to each point in the curves of Figure 6.33.

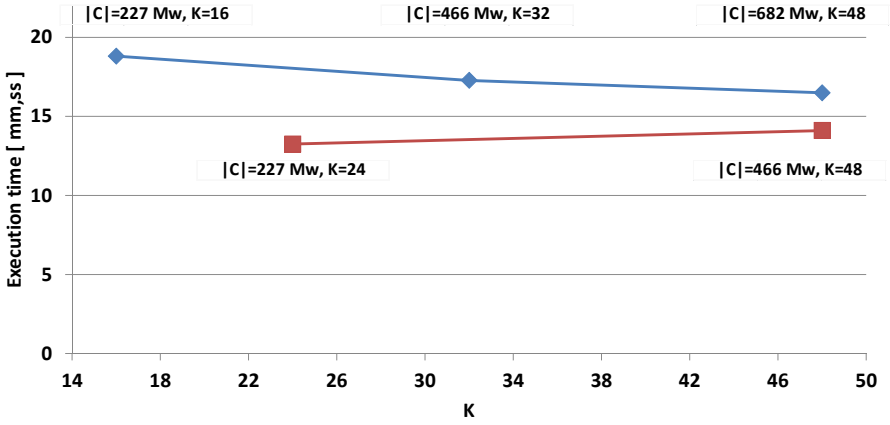


Figure 6.33: Relative sizeup (weak scaling) for phase two as function of K for different *corpus* sizes. The Y axis represents the relative sizeup and the X axis represents the number of machines.

As shown in figure, the execution time remains practically constant when the *corpus* size and the number of machines are scaled by the same factor. This indicates that the relative sizeup is almost linear with K. So phase two shows a good relative weak scaling behavior.

Load Balancing in Phase Two. The load balancing among the multiple machines in phase two was evaluated for the considered ranges of *corpus* sizes and number of machines. The following definition of a load balancing metric (LB) was considered, relating the

average execution time among the machines to the maximum execution time observed in the slowest machine:

$$LB = \frac{Average\{T_2(j),\ j:1\cdots K\}}{max\{T_2(j),\ j:1\cdots K\}} = \left(\left(\sum_{j=1}^{K} T_2(j)\right)/K\right)/max(T_2(j),\ j:1\cdots K) \tag{6.46}$$

In all experiments, all the machines exhibited similar phase two completion times, corresponding to a good load balancing. This is consistent with the even work partitioning among the machines and the equal partitioning of the distinct *n*-gram tables among the KVS servers.

As an example, Figure 6.34 shows the phase two per machine execution time ($T_2(j)$) for the *corpus* of 466 Mw when using 24 machines, when using the workflow shown in Figure 5.3 on page 80. The corresponding value of *LB* was 93%.

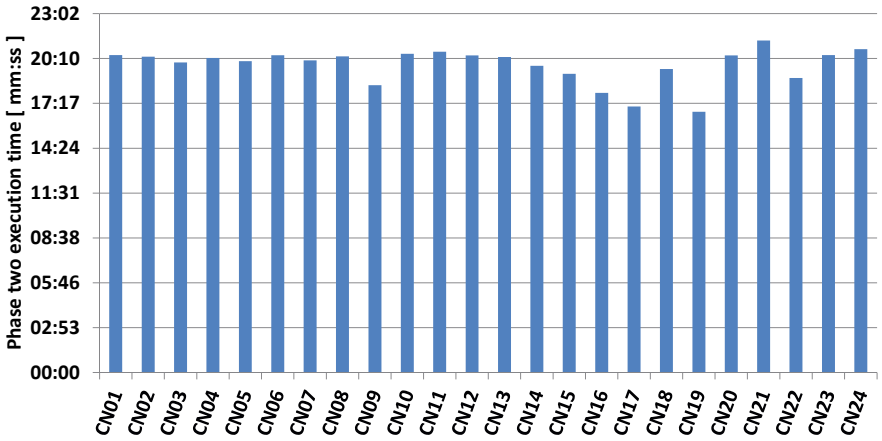


Figure 6.34: Phase two per machine execution time

The observed behaviors of the phase two real execution times and the corresponding speedup, efficiency, sizeup and load balancing metrics were obtained in multiple experiments conducted, in different times (2015 and 2016), in public cloud environments, with consistent results [GSC15; GSC16].

6.6 Chapter Summary

For the range of experimentally analyzed *corpora* up to 1 Gw and for a range of machines up to 54 virtual machines, we have observed that phase two exhibits almost linear relative speedup and sizeup. We also observed that, in this *corpus* size range, phase two (T_2) is responsible for a significant component of the total execution time.

In chapter 9 we extend this analysis by estimating the behavior of the Global method in all its three phases with *corpus* sizes beyond 1 Gw covering the entire *corpus* size range up to the *plateaux*, using the theoretical model (presented in chapter 3) as a basis.

The significant communication overhead of phase two arises due to the need to fetch the remote sub *n-grams* for glue calculation, which justifies the design of an n-gram cache system in each machine, with one cache for each n-gram size C_1, ..., C_{n-1}, with the objective of reducing the above communication penalties, and achieving lower values for the miss ratio (mr).

When using an infinite dynamic cache system, the following behavior was experimentally observed for the above *corpus* and machine ranges and was consistently explained by the developed analytical performance model, regarding the per machine global miss ratio for the glue calculation in phase two:

- It increases with the number of machines (K);
- It decreases with the *corpus* size;
- It increases with the glue calculation complexity (maximum n-gram size).

Also, the per machine global miss time penalty of the cache system decreases with K according to a power law.

In the experiments with *corpus* from 25 to 682 Mw and from 1 to 48 machines, the measured global miss ratio of the cache system C_{1+2+3} during phase two for glue g_{234} was in average around 30%, which is still a significant value leading to low efficiency values, compared to an ideal sequential machine with limited memory. However, we should note that, on one hand, the above value of the global miss ratio decreases for *corpus* sizes larger than 1 Gw, and on the other hand, the obtained values reflect the cold-start misses in all the (considered infinite) caches.

In the following two chapters we consider alternative and complementary cache designs that achieve further reduction in the miss ratio and miss time penalty values.

Chapter 7

Static Cache Prefetching in LocalMaxs Based on Fixed Frequency Accumulation Sets

This chapter proposes a static prefetching strategy for the LocalMaxs Global method n-gram cache system. This is based on the concept of FAset presented in chapter 3.

When using a dynamic cache only, even being of infinite capacity, as discussed in chapters 5 and 6, the cold-start misses can not be ignored, and they are responsible for a significant overhead in the execution of the LocalMaxs Global method. However, once the n-gram data responsible for those cold-start misses are fetched into the cache and the cache is warmed-up, the above overhead tends to decrease for any future access references that can be requested by the executing algorithms. Thus, it is possible to mitigate this overhead by increasing the reutilization of the n-gram caches through the execution of computations that rely on the previously fetched n-gram data. In chapter 6 we discussed a warming-up strategy for the on-demand dynamic cache and showed the improvements to the cache efficiency that were achieved within a single LocalMaxs Global method execution instance through the evaluation of combined glue calculations for multiple n-gram sizes. Although not pursued in this thesis, the above effect can be further augmented by executing other LocalMaxs algorithm instances or other relevance metric calculations that also share and reuse the n-gram cache contents.

In this chapter we propose a different, albeit complementary, strategy centered on the n-gram static cache prefetching based on the concept of Fixed Frequency Accumulation Set, denoted as FAset (chapter 3), and evaluate its impact upon the cache miss ratio and miss penalty. This strategy tries to reduce the impact of the cold-start misses by

hiding their miss time penalty through the static prefetching of the needed n-gram data, overlapped with phase one and two computations.

This chapter is organized in 4 sections. A static cache system based on the concept of FAset is presented on section 7.1. Section 7.2 analyzes a cache system composed of a static cache followed by a dynamic on-demand cache. Experimental results are presented in section 7.3. This chapter ends with a summary (section 7.4).

7.1 Static Cache Prefetching

In section 7.1.1 we present the basic definitions regarding the static cache prefetching. An analysis of the global hit ratio of the static cache is presented in section 7.1.2. The FAset concept applied to the LocalMaxs Global method is discussed in section 7.1.3. The design of an algorithm for searching the minimal cost FAset to achieve a given target global static cache hit ratio is described in section 7.1.4. The behavior of the FAset membership in a multiple machine configuration is discussed in section 7.1.5.

7.1.1 Definitions

Figure 7.1 illustrates the input reference stream ($all_{glue_g Ref}$) of an n-gram cache system and the corresponding output stream of cache misses. The input reference stream corresponds to the sequence of distinct sub n-gram access requests that are generated by the glue calculations of LocalMaxs Global method execution during phase two, as described in chapters 5 and 6. We now consider an approach where each individual n-gram cache C_i with $1 \leq i \leq (n-1)$, where n is the maximum n-gram size for calculating the n-gram glues, is loaded in advance with a precomputed set of n-gram data that are defined by the FAset concept.

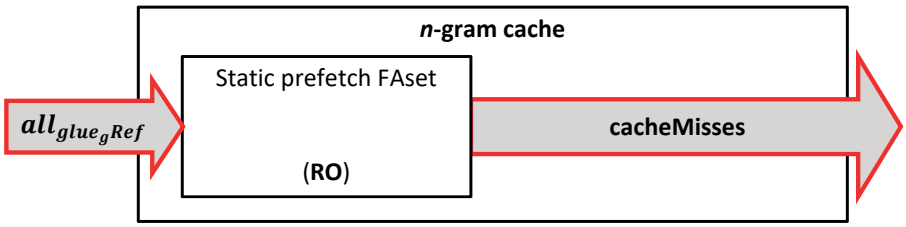


Figure 7.1: Diagram of a single static cache system.

In order to illustrate this, let us first consider the example of an isolated glue calculation, for example g_2 regarding the bigrams in the D_2 n-gram table. The LocalMaxs method requires access to all the subunigrams occurrences (in a total amount of $2 \times |D_2|$) contained in those distinct bigrams. Those subunigrams correspond to the input references made to the unigrams cache C_1. Our goal is to be able to identify with anticipation the FAset that should be loaded on cache C_1, i.e., the set of distinct subunigrams whose

accumulated sum of frequencies of occurrences within the bigrams of the D_2 table ensures a desired hit ratio ($h_{RO_{C_1}}$) for the unigrams cache. The elements within the FAset are a selected subset of the elements in the set $D_{1_{in}D_2}$ as defined in chapter 5. The notation used ($h_{RO_{C_1}}$) hints the intended meaning as a preloaded cache that remains unchanged (Read Only — **RO**) during execution, thus named a static cache.

The degree of time overlapping of this prefetching activity with other useful computations determines the degree of reduction in the miss time penalty when compared to an on-demand fetch strategy. On the other hand, in the case of LocalMaxs Global method, at the end of phase one we know which are all the required sub n-grams and their frequencies of occurrences in the *corpus*, so we can use this knowledge in order to load the n-gram cache with anticipation, with the guarantee that those n-grams will be necessary to the computation.

When we consider the case of a combined glue calculation, for example g_{234}, the n-gram cache is composed of three individual caches (as presented in Figure 7.2) for unigrams, bigrams and trigrams. Although in the following we use g_{234} an example, all the analysis can be generalized to any other glue calculation case. In Figure 7.2 the unigrams cache is denoted by $C_{1_{RO}}$, the bigrams cache is denoted by $C_{2_{RO}}$, and the trigrams cache is denoted by $C_{3_{RO}}$, but for a matter of simplicity, in this section we denote the unigrams, bigrams and trigrams caches respectively by C_1, C_2 and C_3.

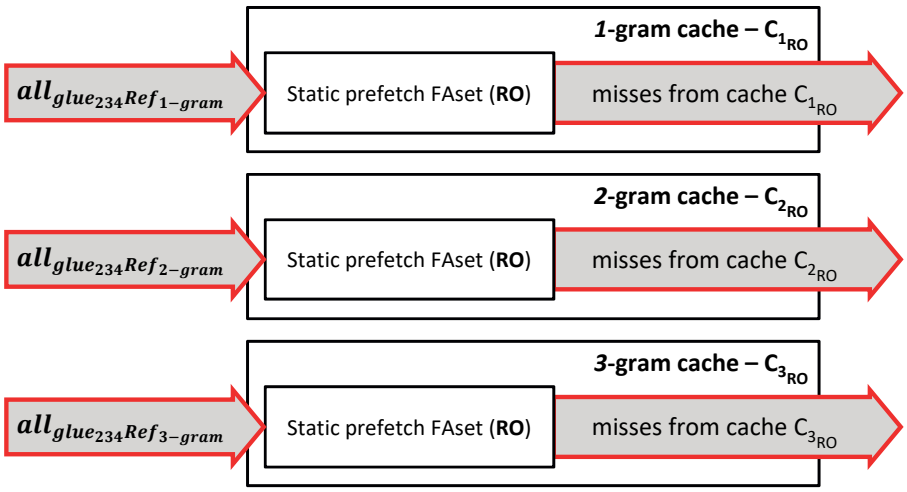


Figure 7.2: Diagram of a static cache system for glue g_{234}.

In this case the input reference stream (all_{glue_gRef}) is composed of unigram occurrences ($all_{glue_{234}Ref_{1-gram}}$), bigram occurrences ($all_{glue_{234}Ref_{2-gram}}$) and trigram occurrences ($all_{glue_{234}Ref_{3-gram}}$), which are split respectively among the unigram, bigram and trigram caches. For example, in the case of the unigrams cache the reference stream is composed of the subunigram occurrences in each entry of the tables D_2, D_3 and D_4. For the bigrams cache the reference stream is composed of the subbigram occurrences in each entry of the

tables D_3 and D_4. For the trigrams cache the reference stream is composed of the subtrigram occurrences in each entry of the table D_4. Their total numbers of input references are as follows:

$$\left|all_{glue_{234}}Ref_{1\text{–}gram}\right| = 2\times|D_2| + 2\times|D_3| + 2\times|D_4| \tag{7.1}$$

$$\left|all_{glue_{234}}Ref_{2\text{–}gram}\right| = 2\times|D_3| + 2\times|D_4| \tag{7.2}$$

$$\left|all_{glue_{234}}Ref_{3\text{–}gram}\right| = 2\times|D_4| \tag{7.3}$$

$$\left|all_{glue_{234}}Ref\right| = \left|all_{glue_{234}}Ref_{1\text{–}gram}\right| + \left|all_{glue_{234}}Ref_{2\text{–}gram}\right| + \left|all_{glue_{234}}Ref_{3\text{–}gram}\right| = 2\times|D_2| + 4\times|D_3| + 6\times|D_6| \tag{7.4}$$

For each individual cache, from C_1 to C_3, we calculate the corresponding FAset leading to a desired individual hit ratio ($h_{RO_{C_1}}$, $h_{RO_{C_2}}$ and $h_{RO_{C_3}}$). The following definitions apply to each individual cache where the corresponding FAset should be considered, and $freq_{in\ all_{glue}Ref_{i\text{–}gram}}(ng)$ corresponds (in the case of glue g_{234}) to the frequency of unigrams, bigrams and trigrams in the respective input references streams ($all_{glue}Ref_{i\text{–}gram}$, $1 \le i \le 3$), where ng denotes a distinct n-gram in the FAset.

$$h_{RO} = \frac{\left|all_{glue_gRef}\right| - |cacheMisses|}{\left|all_{glue_gRef}\right|} \tag{7.5}$$

$$mr_{RO} = 1 - h_{RO} = \frac{|cacheMisses|}{\left|all_{glue_gRef}\right|} \rightarrow |cacheMisses| = \left|all_{glue_gRef}\right| \times (1 - h_{RO}) \tag{7.6}$$

$$\left|all_{glue_gRef}\right| - |cacheMisses| = \sum_{ng \in FAset} freq_{in\ all_{glue}Ref_{i\text{–}gram}}(ng) = numberHits \tag{7.7}$$

7.1.2 Analysis of the Global Hit Ratio (h_{RO})

There are different combinations of triples $\left\langle h_{RO_{C_1}}, h_{RO_{C_2}}, h_{RO_{C_3}}\right\rangle$ leading to the same h_{RO}. We are interested in the triple which leads to the minimum cost in terms of cache size for a given h_{RO}, i.e., the total number of n-gram entries in the three caches (C_1, C_2 or C_3).

For each cache we must select a set of distinct n-grams that leads to the desired value of the corresponding cache hit ratio. As shown in Figure 3.14 on page 52 the number of hits in each cache can be ensured by the accumulated sum of frequencies of the distinct n-grams which are selected for the corresponding cache FAset. As previously discussed, although the populations of unigram, bigram, and trigram occurrences in a *corpus* are

approximately equal to the total *corpus* size, the corresponding number of distinct unigrams, bigrams, and trigrams in the *corpus* are significantly different from one another. Namely, $|D_1| \ll |D_2| \ll |D_3|$.

This is related to the fact, for example, that the distinct unigrams exhibit in average a much higher repetition factor in their numbers of occurrences than the distinct bigrams and the same applies regarding the distinct trigrams. Thus, in average the distinct unigrams have higher individual contributions to the hit ratio of the cache system when compared with the distinct bigrams and the distinct trigrams. In the following we discuss a strategy for selecting the distinct n-grams to be placed in each individual cache by giving priority to the ones with the minimum space cost and higher contributions to the hit ratio.

7.1.2.1 n-gram Coverages

The different caches (C_1,C_2 or C_3) have different contributions to the global hit ratio (h_{RO}) of the cache system for the combined glue g_{234} depending on the weights of the associated coverages, i.e., for the combined glue calculation g_{234} the unigrams coverage is given by $Coverage_{1\text{–}gram} = \left|all_{glue_{234}}Ref_{1\text{–}gram}\right| / \left|all_{glue_{234}}Ref\right|$, the bigrams coverage is given by $Coverage_{2\text{–}gram} = \left|all_{glue_{234}}Ref_{2\text{–}gram}\right| / \left|all_{glue_{234}}Ref\right|$, and the trigrams coverage is given by $Coverage_{3\text{–}gram} = \left|all_{glue_{234}}Ref_{3\text{–}gram}\right| / \left|all_{glue_{234}}Ref\right|$.

$$h_{RO}(g_{234}) = Coverage_{1\text{–}gram} \times h_{RO_{C_1}} + Coverage_{2\text{–}gram} \times h_{RO_{C_2}} + Coverage_{3\text{–}gram} \times h_{RO_{C_3}} = \frac{\left|all_{glue_{234}}Ref_{1\text{–}gram}\right|}{\left|all_{glue_{234}}Ref\right|} \times h_{RO_{C_1}} + \frac{\left|all_{glue_{234}}Ref_{2\text{–}gram}\right|}{\left|all_{glue_{234}}Ref\right|} \times h_{RO_{C_2}} + \frac{\left|all_{glue_{234}}Ref_{3\text{–}gram}\right|}{\left|all_{glue_{234}}Ref\right|} \times h_{RO_{C_3}} \tag{7.8}$$

Figure 7.3 shows the evolution of the n-gram coverages for unigrams, bigrams and trigrams along the entire range of *corpus* sizes from 2 Mw up to the *plateau* regions.

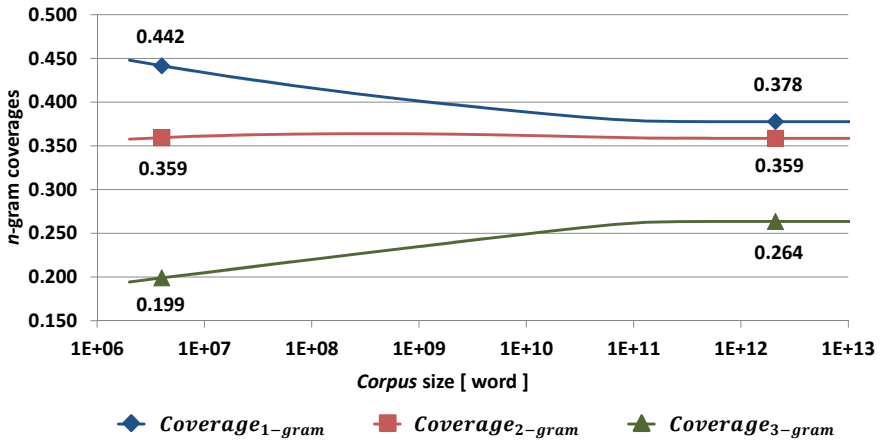


Figure 7.3: Individual n-gram coverages *versus corpus* sizes. The Y axis represents the n-gram coverages and the X axis represents the *corpus* size in words.

These values were obtained by applying the distinct numbers of n-grams given by the theoretical model presented in chapter 3 to the n-gram coverages in expression (7.8). The n-gram coverages exhibit only a very slight variation with the *corpus* size. Also, the total number of unigram references is greater than the total number of bigram references which is greater than the total number of trigram references. As expected, the unigrams coverage is higher than the bigrams coverage, which is higher than the trigrams coverage.

Table 7.1 presents detailed experimental data in the range of *corpus* sizes up to 13 Gw in the case of glue g_{234}. These values are consistent with the values of the distinct numbers of n-grams generated by the theoretical model presented in chapter 3. The columns labeled $|Ref_{n\text{–}gram}|$ represent the references to unigrams ($|all_{g_{234}} Ref_{1\text{–}gram}|$), bigrams ($|all_{g_{234}} Ref_{2\text{–}gram}|$) and trigrams ($|all_{g_{234}} Ref_{3\text{–}gram}|$); and the columns $Cov_{n\text{–}gram}$ represent the coverages of unigrams ($Coverage_{1\text{–}gram}$), bigrams ($Coverage_{2\text{–}gram}$) and trigrams ($Coverage_{3\text{–}gram}$).

Table 7.1: Distinct numbers of n-grams and coverages of unigrams up tetragrams for different *corpus* sizes up to 1 Gw. The values of distinct and n-gram references are in millions.

$\|C\|$ [Mw]	$\|D_1\|$	$\|D_2\|$	$\|D_3\|$	$\|D_4\|$	$\|Ref_{1\text{–}gram}\|$	$\|Ref_{2\text{–}gram}\|$	$\|Ref_{2\text{–}gram}\|$	$Cov_{1\text{–}gram}$	$Cov_{2\text{–}gram}$	$Cov_{3\text{–}gram}$
13	0.5	3.4	7.7	10.4	42.9	36.1	20.7	0.430	0.362	0.208
25	0.8	5.8	14.1	20.0	79.8	68.1	39.9	0.425	0.363	0.213
64	1.6	11.9	31.4	47.5	181.5	157.8	95.0	0.418	0.363	0.219
128	2.6	19.9	56.5	89.9	332.6	292.8	179.7	0.413	0.364	0.223
256	4.3	33.2	100.8	168.6	605.2	538.8	337.3	0.409	0.364	0.228
511	7.1	54.8	177.8	313.7	1092.6	982.9	627.4	0.404	0.364	0.232
1023	11.8	89.2	309.2	575.6	1948.1	1769.6	1151.2	0.400	0.363	0.236

7.1.2.2 Influence of the Individual Hit Ratios ($h_{RO_{C_i}}$) Upon h_{RO}

From expression (7.8) the different caches (C_1, C_2 or C_3) have different contributions to the global h_{RO} depending on the weights of the associated coverages ($Coverage_{1\text{–}gram} > Coverage_{2\text{–}gram} > Coverage_{3\text{–}gram}$). Thus, the effective contribution of each cache C_i is given by an effective individual hit ratio:

$$h'_{RO_{C_i}} = Coverage_{i\text{–}gram} \times h_{RO_{C_i}} \tag{7.9}$$

where $h_{RO_{C_i}}$ is the individual cache C_i miss ratio calculated as $h_{RO_{C_i}} = \frac{hits\,Cache\,C_i}{|all_{glue_{g_{234}}\,Ref_{i\text{–}gram}}|}$, while $h'_{RO_{C_i}} = \frac{hits\,Cache\,C_i}{|all_{glue_{g_{234}}\,Ref}|}$ with $hits\,Cache\,C_i$ corresponding to the number of hits in the cache C_i.

As a result, if we wanted to ensure the same value of $h'_{RO_{C_i}}$ for all individual caches $1 \leq i \leq 3$, then we would have to provide higher values of $h_{RO_{C_3}}$ than $h_{RO_{C_2}}$ and higher values of $h_{RO_{C_2}}$ than $h_{RO_{C_1}}$, in the corresponding proportions of the n-gram coverages

($Coverage_{3\text{–}gram}/Coverage_{2\text{–}gram}$, and $Coverage_{2\text{–}gram}/Coverage_{1\text{–}gram}$). However, the assumption of keeping equal values of $h'_{RO_{C_i}}$ will lead to higher space cost in terms of the FAsets to be loaded in the trigrams cache comparing to the bigrams cache and to the unigrams cache.

In the search for the triple of values of the individual $h_{RO_{C_i}}$ that leads to a desired value of the global h_{RO}, we find multiple alternative solutions. In fact, for each desired h_{RO} value there is a tridimensional space defined by points such as $\langle h_{RO_{C_1}}, h_{RO_{C_2}}, h_{RO_{C_3}} \rangle$ which could be exhaustively searched for all admissible values of $h_{RO_{C_i}}$. This would lead to a set of scenarios, where for each given *corpus* size, all the possible values of $h_{RO_{C_1}}$, $h_{RO_{C_2}}$ and $h_{RO_{C_3}}$ would be considered, and the corresponding values of h_{RO} would be determined.

Figure 7.4 illustrates a subset of such scenarios corresponding to a *corpus* size around 1.3 Tw (where the distinct numbers of unigrams, bigrams and trigrams have reached the *plateau* regions — chapter 3) where for a fixed value of $h_{RO_{C_3}} = 30\%$, the values of $h_{RO_{C_1}}$ and $h_{RO_{C_2}}$ vary from 5% to 100% in steps of 5%.

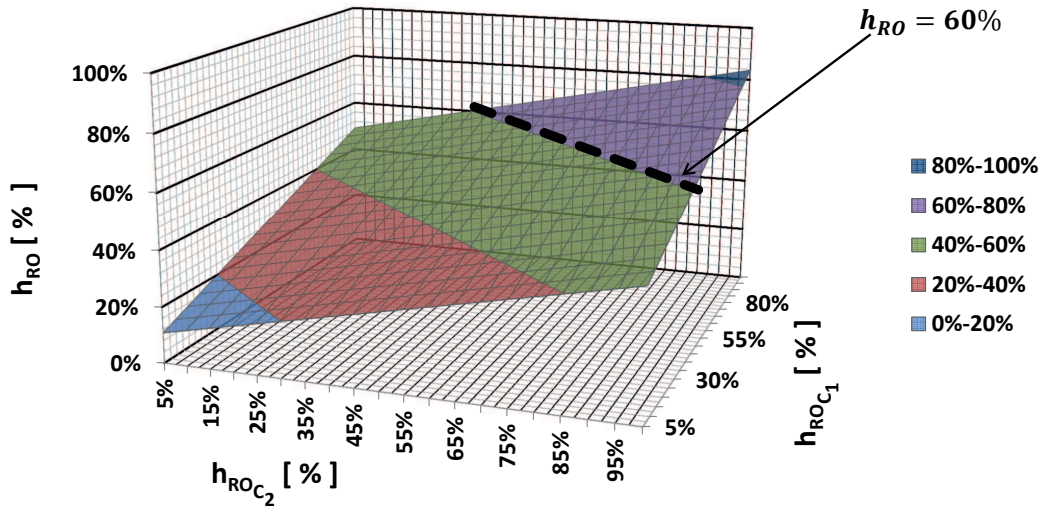


Figure 7.4: Static global hit ratio for the *plateaux* — $h_{RO_{C_3}} = 30\%$. The vertical axis represents the global hit ratio and the horizontal axes represent the individual hit ratio of the unigrams cache ($h_{RO_{C_1}}$) and the bigrams cache ($h_{RO_{C_2}}$).

The following observations are made:

- The heavily dotted line in Figure 7.4 marks all the points $\langle h_{RO_{C_1}}, h_{RO_{C_2}}, 30\% \rangle$ that lead to the same value of $h_{RO} = 60\%$, for example the points $\langle 100\%, 30\%, 30\% \rangle$ and $\langle 40\%, 100\%, 30\% \rangle$;
- However, the above two points have different space costs.

The criterion used to select the best combination of the values $\langle h_{RO_{C_1}}, h_{RO_{C_2}}, h_{RO_{C_3}} \rangle$ for a desired h_{RO} in the case of g_{234} is to choose the point that will lead to the minimum space

cost in terms of the sum of the FAset sizes for the considered caches.

7.1.2.3 Influence of the *Corpus* Size upon h_{RO}

When considering a fixed point of values $\langle h_{RO_{C_1}}, h_{RO_{C_2}}, h_{RO_{C_3}} \rangle$, and varying the *corpus* size, the global static cache hit ratio exhibits a very slight variation as shown in Figure 7.5 for different selected points.

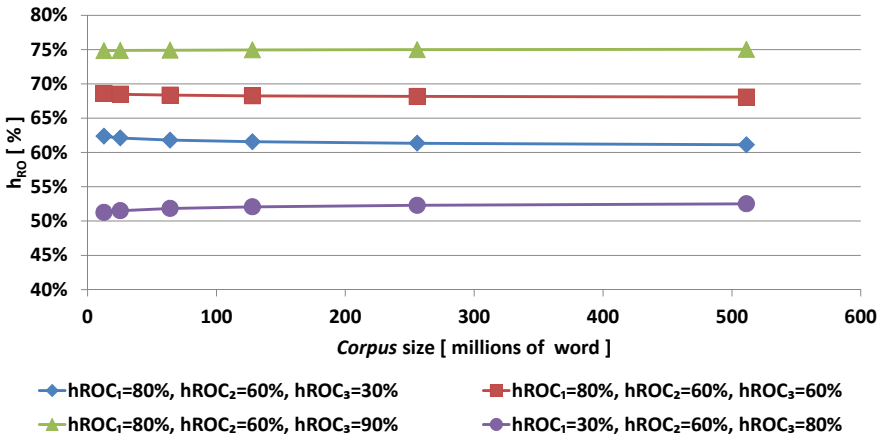


Figure 7.5: Global hit ratio *versus corpus* size. The Y axis represents the global hit ratio in percentage and the X axis represents the *corpus* size in millions of words.

The almost constancy of h_{RO} with the *corpus* size for each fixed point is directly related to the behavior of the n-gram coverages with the *corpus* size as presented in Figure 7.3.

7.1.3 FAset for LocalMaxs

In section 7.1.3.1 we present the FAset definitions in the context of LocalMaxs Global method. An example of the usage of the FAset concept is presented in section 7.1.3.2.

7.1.3.1 FAset Definitions

In the following each FAset is defined similarly to the Definition 3.1, on page 52, except for the definition of the frequency of the n-grams which now corresponds to the frequency of the references to the sub n-grams contained in the n-gram tables that are required by phase two of the LocalMaxs Global method algorithm.

Definition 7.1: FAset for LocalMaxs Global method for Isolated Glue

When calculating the isolated glue g_{n_m}, the hit ratio (h) ensured by a FAset ($FA\left(n, g_{n_m}, h\right)$) for cache C_n corresponding to a given n-gram of size n using LocalMaxs Global method algorithm is the percentage of the cumulative sum of frequencies of the leftmost and rightmost sub n-grams of size n that occur as leftmost or rightmost sub n-grams within the n-grams contained in the distinct n_m-gram table (D_{n_m}), with respect to the total number of references to the leftmost and rightmost sub n-grams of size n ($\left|all_{glue_{n_m} Ref_{n\text{-}gram}}\right|$) — Definition 2.1 on page 15 — in the above mentioned n_m-gram table (D_{n_m}), and is defined such as:

$$h\left(n, FA, g_{n_m}\right) = \frac{\sum_{ng \in FA} freq_{in D_{n_m}}(ng)}{\left|all_{glue_{n_m} Ref_{n\text{-}gram}}\right|}, \qquad 1 \leq n \leq n_m - 1, \qquad n_m \geq 2$$

$$minimal\ subset\ FA\left(n, g_{n_m}, h\right) \subset D_{n_{in} D_{n_m}}$$

where ng denotes a distinct n-gram that is included in the FAset, $freq_{in D_{n_m}}(ng)$ is the frequency of the n-gram ng in the D_{n_m} table, and $D_{n_{in} D_{n_m}}$ is the set of distinct leftmost and rightmost sub n-grams of size n in the D_{n_m} table. For the LocalMaxs Global method algorithm $\left|all_{glue_{n_m} Ref_{n\text{-}gram}}\right|$ is equal to $2 \times \left|D_{n_m}\right|$.

For a given n-gram size n, the FAset to ensure a target hit ratio (h), when evaluating the isolated glue of the n-grams of size n_m, is defined by counting the frequencies of occurrences of the leftmost and rightmost sub n-grams of size n within the table (D_{n_m}) of distinct n_m-grams .

For the calculation of the combined glue $g_{n_1..n_2}$, i.e., calculating all the glues from g_{n_1} up to including g_{n_2} the following definition applies.

Definition 7.2: FAset for LocalMaxs Global method for Combined Glue

When calculating the combined glue $g_{n_1..n_2}$, the hit ratio ensured by a FAset for cache C_n corresponding to a given n-gram size n using LocalMaxs Global method algorithm is the percentage of the cumulative sum of frequencies of the distinct sub n-grams of size n that occur (as leftmost or rightmost sub n-grams) within the n-grams contained in the tables from the distinct n_1-gram table (D_{n_1}) up to and including the distinct n_2-gram table (D_{n_2}), with respect to the total number of references to all leftmost and rightmost sub n-grams of size n ($\left|all_{glue_{n_1...n_2}Ref_{n\text{-}gram}}\right|$) — Definition 2.1 on page 15 — found within the n-grams contained in the union of the n-gram tables from n_1-gram up to and including the n_2-gram table, and is defined as:

$$h\left(n, FA, g_{n_1..n_2}\right) = \frac{\sum_{ng \in FA} freq_{in(D_{n_1}...\cup...D_{n_2})}(ng)}{\left|all_{glue_{n_1...n_2}Ref_{n\text{-}gram}}\right|}, \qquad 1 \le n \le n_2 - 1, \qquad n_2 > n_1 \ge 2$$

$$minimal\ subset\ FA\left(n, g_{n_1...n_2}, h\right) \subset D_{n_{in}(D_{n_1}...\cup...D_{n_2})}$$

where ng denotes a distinct n-gram that is included in the FAset, $freq_{in(D_{n_1}...\cup...D_{n_2})}(ng)$ is the frequency of the n-gram ng in the $D_{n_1}... \cup ...D_{n_2}$ tables, and $D_{n_{in}(D_{n_1}...\cup...D_{n_2})}$ is the set of distinct sub n-grams in $D_{n_1}... \cup ...D_{n_2}$. For the LocalMaxs Global method algorithm $\left|all_{glue_{n_1...n_2}Ref_{n\text{-}gram}}\right|$ is equal to $2 \times \left|D_{n_1}\right| + ... + 2 \times \left|D_{n_2}\right|$.

The above definitions also apply to the multiple machines case ($K > 1$) where the sets FAset and $D_{n_{in}D_{n_m}}$ or $D_{n_{in}(D_{n_1}...\cup...D_{n_2})}$ correspond to the per machine local partitions of n-gram tables (labeled by the machine index j, $1 \le j \le K$).

7.1.3.2 An Example

As an example consider the following Tables 7.2 and 7.3 with an excerpt of the data of unigrams and bigrams for an hypothetical *corpus* and assume that we want a FAset $FA(1, g_2, 70\%)$ for the unigrams with a hit ratio of $h_{RO_1} = 70\%$ for the calculation of glue g_2.

Table 7.2: Hypothetical table of distinct unigrams for an hypothetical *corpus*.

***n*-gram**	**Global frequency in the *corpus***
the	15034
that	8383
expect	165
comes	156
show	49
summer	34

Table 7.3: Hypothetical table of distinct bigrams for an hypothetical *corpus*. Frequency counts omitted for simplicity.

***n*-gram**
expect the
expect that
show that
that comes
the summer

When building the FAset $FA(1, g_2, 70\%)$, corresponding to the FAset of unigrams within the bigrams, we first need to sort the unigrams in the set $D_{1_{in}D_2}$ according to their decreasing frequencies in the D_2 table. In this example:

$$\{\langle that, 3\rangle, \langle the, 2\rangle, \langle expect, 2\rangle, \langle comes, 1\rangle, \langle show, 1\rangle, \langle summer, 1\rangle\} \tag{7.10}$$

By including the unigram "that" in the FAset we obtain a $h_{RO_1} = 3/10 = 30.0\%$, by including the unigrams "that" and "the" we obtain a $h_{RO_1} = (3+2)/10 = 50.0\%$. So to obtain at least an hit ratio of 70% we need to include in the FAset the unigrams "that", "the", and "expect", which lead to an hit ratio $h_{RO_1} = (3+2+2)/10 = 70\%$. The resulting FAset would be $FA(1, g_2, 70\%) = \{the, that, expect\}$. As result, the static unigrams cache would be loaded with the following contents: $\{\langle the, 15034\rangle, \langle that, 8383\rangle, \langle expect, 165\rangle\}$.

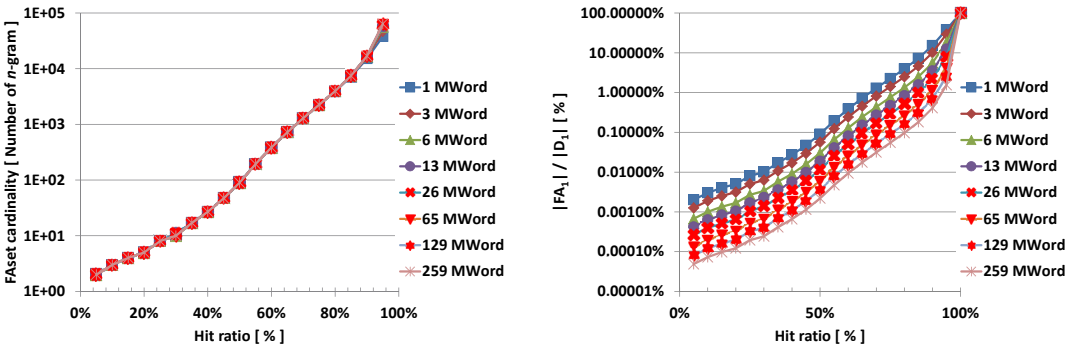
7.1.4 Algorithm for Searching the Minimal Cost FAset

In this section we present the design of an algorithm for searching the minimal cost FAset to achieve a given target global static cache hit ratio. Section 7.1.4.1 analyzes the evolution of the FAset size as a function of the individual cache hit ratio for the isolated glue calculations. The algorithm is presented in section 7.1.4.2 and an example is shown in section 7.1.4.3. The influence of the *n*-gram distribution over the FAset is discussed in section 7.1.4.4. An analysis of the influence of excluding the *n*-gram singletons for the FAset is presented in section 7.1.4.5.

7.1.4.1 FAset Sizes as a Function of the $h_{RO_{C_i}}$ for the Isolated Glues

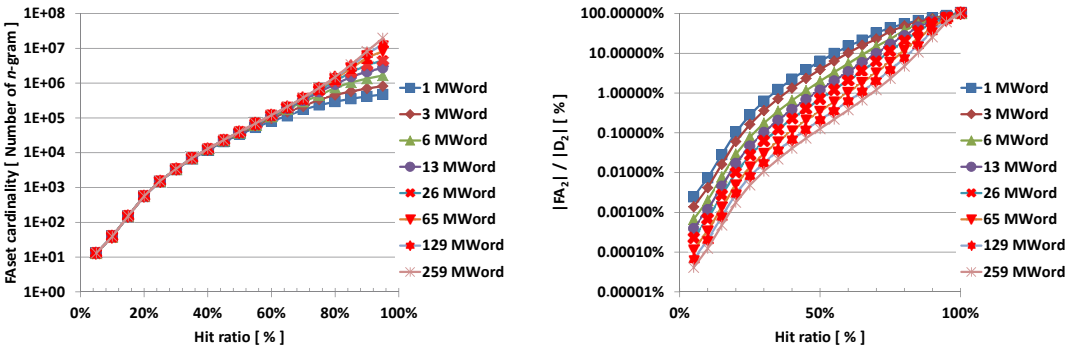
In this section we analyze the value of the individual cache hit ratio in the following cases: $h_{RO_{C_1}}$ for the unigrams when calculating the isolated glue g_2; $h_{RO_{C_2}}$ for the bigrams when calculating the isolated glue g_3; and $h_{RO_{C_3}}$ for the trigrams when calculating the isolated glue g_4. For each case we evaluate the cardinality of the corresponding individual FAset, $FA\left(1, g_2, h_{RO_{C_1}}\right)$, $FA\left(2, g_3, h_{RO_{C_2}}\right)$ and $FA\left(3, g_4, h_{RO_{C_3}}\right)$, when the individual hit ratios ($h_{RO_{C_i}}$, with $1 \leq i \leq 3$) range from 5% to 95% in steps of 5%, as represented in the X axis of all figures in this section.

Individual FAset Sizes. In Figures 7.6 to 7.8 the absolute values of the FAset cardinalities (for unigrams, bigrams and trigrams) and the ratio of the FAset size over the number of the distinct n-grams of the same size (Y axis) are shown for a range of *corpus* sizes from 1 Mw to 259 Mw.



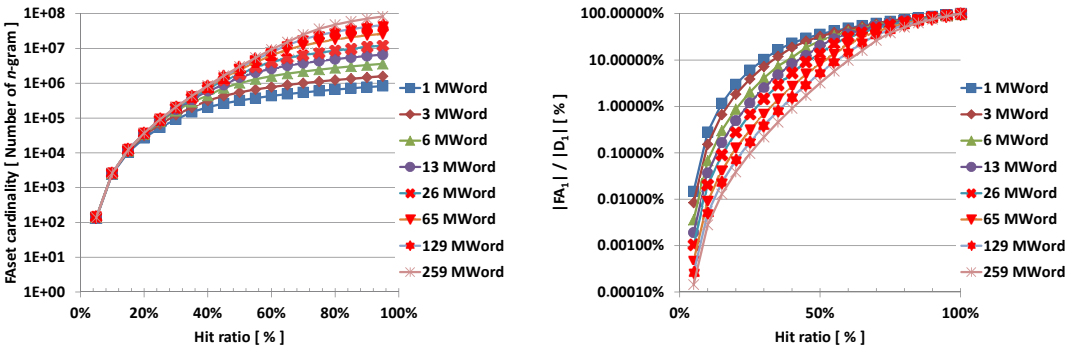
a) Unigrams $FA\left(1, g_2, h_{RO_{C_1}}\right)$ *versus* $h_{RO_{C_1}}$ *b) Ratio* $\left|FA\left(1, g_2, h_{RO_{C_1}}\right)\right| / |D_1|$ *versus* $h_{RO_{C_1}}$

Figure 7.6: Unigrams FAset for English ($D_{1_{in}D_2}$) for different *corpus* sizes.



a) Bigrams $FA\left(2, g_3, h_{RO_{C_2}}\right)$ *versus* $h_{RO_{C_2}}$ *b) Ratio* $\left|FA\left(2, g_3, h_{RO_{C_2}}\right)\right| / |D_2|$ *versus* $h_{RO_{C_2}}$

Figure 7.7: Bigrams FAset for English ($D_{2_{in}D_3}$) for different *corpus* sizes.



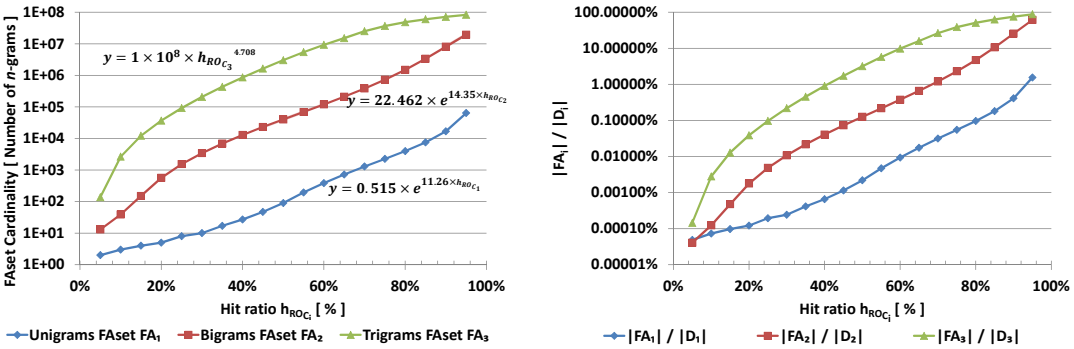
a) Trigrams $FA\left(3,g_4,h_{ROC_3}\right)$ *versus* h_{ROC_3} *b) Ratio* $\left|FA\left(3,g_4,h_{ROC_3}\right)\right|/|D_3|$ *versus* h_{ROC_3}

Figure 7.8: Trigrams FAset for English ($D_{3_{in}D_4}$) for different *corpus* sizes.

In all the three cases we observe:

- The cardinality of the individual FAset increases as the individual cache hit ratio gets higher;
- The FAset membership reaches the total number of distinct n-grams of the corresponding size (D_1, D_2 or D_3) for $h_{ROC_i} = 100\%$.

Comparing the FAset Sizes. Figure 7.9 shows the FAset sizes and the ratios of FAset sizes over the total numbers of distinct n-grams, for the cases of unigrams, bigrams and trigrams.



a) FAset cardinality versus hit ratio *b) Ratio* $\left|FA\left(i,g_n,h_{ROC_i}\right)\right|/|D_i|$ *versus hit ratio*

Figure 7.9: FAset cardinality and cost as a function of the individual static cache hit ratio h_{ROC_i}, for a fixed *corpus* size of 259 Mw.

The following observations can be made:

- From Figure 7.9-a) we conclude that for the same values of h_{ROC_1} =h_{ROC_2} =h_{ROC_3} (X axis) the required FAset size (Y axis) of the trigrams cache is much higher than the one required by the bigrams cache, which is also higher than the one for the unigrams cache;

- ✦ Also, for the same values of $h_{RO_{C_i}}$, the FAset for the unigrams cache represents a much lower proportion of distinct unigrams with regard to the total number of distinct unigrams ($\left|FA\left(1, g_2, h_{RO_{C_1}}\right)\right|/|D_1|$) than the corresponding proportion for the bigrams cache($\left|FA\left(2, g_3, h_{ROC_2}\right)\right|/|D_2|$) as well as than the one for the trigrams cache ($\left|FA\left(3, g_4, h_{RO_3}\right)\right|/|D_3|$), as shown in Figure 7.9-b). This is due to the higher average repetition factor of unigrams ($|C|/|D_1|$) compared to the repetition factor of bigrams ($(|C|-1)/|D_2|$), which is itself higher than the repetition factor of trigrams ($(|C|-2)/|D_3|$);

FAset Efficiencies. The efficiency of a given FAset for a given glue calculation can be defined by a metric that relates the number of hits in the static cache to the FAset size. In fact, the unigrams show a ratio of average number of hits ($\#Hits$) per FAset elements, $\#Hits_1/\left|FA\left(1, g_2, h_{RO_{C_1}}\right)\right|$, where $\#Hits_1 = h_{RO_{C_1}} \times \left|all_{g_2} Ref_{1\text{–}gram}\right|$; the bigrams show a corresponding ratio of $\#Hits_2/\left|FA\left(2, g_3, h_{RO_{C_2}}\right)\right|$, where $\#Hits_2 = h_{RO_{C_2}} \times \left|all_{g_3} Ref_{2\text{–}gram}\right|$; and the trigrams show a ratio given by $\#Hits_3/\left|FA\left(3, g_4, h_{RO_{C_3}}\right)\right|$, where $\#Hits_3 = h_{RO_{C_3}} \times \left|all_{g_4} Ref_{3\text{–}gram}\right|$. This is illustrated in Figure 7.10.

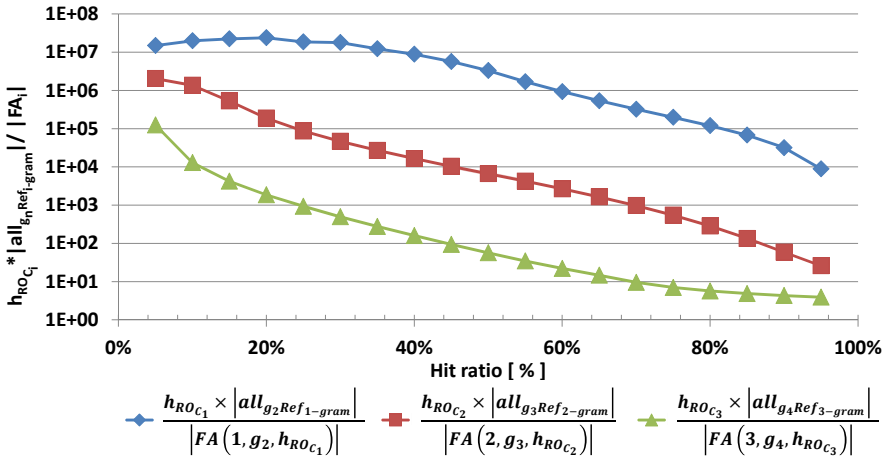


Figure 7.10: FAset efficiencies — $\#Hits_i/|FA_i|$ — for a fixed *corpus* size of 259 Mw. The Y axis represents the individual FAset efficiency and the X axis represents the individual cache hit ratio in percentage.

Due to the above mentioned differences between the average repetition factors of unigrams, bigrams and trigrams, the unigrams FAset efficiency is higher than the efficiencies of bigrams or trigrams.

7.1.4.2 Minimal Cost FAset Algorithm

Objective. For a given *corpus* size and, for example, for LocalMaxs glue g_{234}, what is the minimum cost solution for the combination of $h(1, FA_1, g_{234})$, $h(2, FA_2, g_{234})$ and $h(3, FA_3, g_{234})$ to obtain a target global h_{RO}? We define absolute cost of a solution as:

$$Cost(g_{234}) = \frac{|FA|}{h_{RO}} = \frac{|FA_1| + |FA_2| + |FA_3|}{h_{RO}} \tag{7.11}$$

Rationale of the Algorithm Design. For each n-gram size n, $1 \leq n \leq 3$, consider a stack consisting of buckets. Each bucket is a subset, with variable size, of the total number of distinct n-grams that are totally ordered according to their decreasing frequencies of occurrences within the corresponding sets $D_{n_{in}(D_{n_1} \ldots \cup \ldots D_{n_2})}$, which are, respectively, for unigrams ($n = 1$) the set $D_{1_{in}(D_2 \ldots \cup \ldots D_4)}$, for bigrams ($n = 2$) the set $D_{2_{in}(D_3 \ldots \cup \ldots D_4)}$, and for trigrams ($n = 3$) the set $D_{3_{in}D_4}$. The n-grams with equal frequency are ordered according to their lexicography ordering.

Figure 7.11 shows an example of the bucket stacks for unigrams, bigrams and trigrams for the case of combined glue g_{234}. For each stack ($1 \leq n \leq 3$) its buckets are totally ordered: For example, in the case of the unigrams stack, the last element within the first bucket $b_{1\text{–}gram_{s_1=1}}$ is higher in the ordering than the first element within the following (second) bucket $b_{1\text{–}gram_{s_1=2}}$, and so on. For each n-gram size n, the label s_n denotes the index (starting at 1) of the current top of the n-grams stack. The corresponding value of the FAset already accumulated in the stack $b_{n\text{–}gram_{s_n}}$ is denoted as $FA\left(b_{n\text{–}gram_{s_n}}\right)$, which for simplicity is shortened to FA_n, for n-grams of size n.

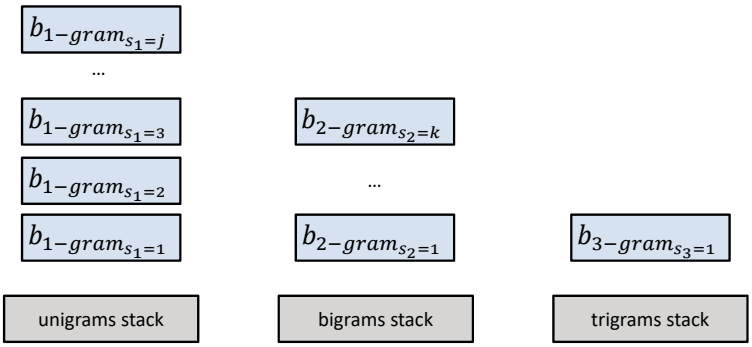


Figure 7.11: Bucket stacks for combined glue g_{234}.

The Minimal Cost FAset algorithm proceeds in a sequence of steps i, from 1 up to the final step, which corresponds to having reached a configuration of n-gram stacks ensuring the desired target h_{RO} with the minimal cost global FAset FA.

In each step the algorithm gathers a new bucket of a given n-gram size among a set of candidate buckets. The candidate buckets are organized into subsets, one subset for each n-gram size, and all the buckets within each subset are totally ordered.

The selected bucket is the one, among the first buckets in each subset, which satisfies a minimum cost criterion as explained in the following. For each n-gram size, each bucket is responsible for adding a fixed increment to the $h'_{RO_{C_n}}$ (expression (7.9)) that corresponds to the contribution from the n-gram stack C_n to the global hit ratio h_{RO} (expression (7.8)).

The increment to the hit ratio ($h'_{RO_{C_n}}$) requires an increment in the FAset FA_n of the corresponding n-gram stack C_n. For all the steps and all the n-gram stacks, the algorithm

initially considers fixed and equal increments in each individual hit ratio ($h_{RO_{C_n}}$) equal to Δh_n for all n-gram sizes, which defines the granularity of the individual hit ratio increments that are considered in the sequence of the algorithm evaluations. However, in fact, in order to ensure integer increments for the FAset cardinalities, the corresponding increments in Δh_n may be slightly above the initially considered Δh_n value, thus they can be slightly different among the distinct n-grams stack sizes. Besides, there is a minimum value in the individual hit ratio increment to ensure that the FAset increment is at least one.

Thus, at each point in the algorithm execution, the set of buckets that were already pushed into each stack of each n-gram size n, determines the effective contribution of this stack to the corresponding hit ratio ($h'_{RO_{C_n}}$). This value is given by the following expression:

$$h'_{RO_{C_n}}\left(b_{n\text{–}gram_{s_n}}\right) = sizeOfStack\left(b_{n\text{–}gram_{s_n}}\right) \times Coverage_{n\text{–}gram} \times \Delta h_n \tag{7.12}$$

where, at each step of the algorithm execution, $h'_{RO_{C_n}}\left(b_{n\text{–}gram_{s_n}}\right)$ denotes the current value of $h'_{RO_{C_n}}$ hit ratio from the n-gram stack $b_{n\text{–}gram_{sn}}$, and $sizeOfStack\left(b_{n\text{–}gram_{s_n}}\right)$ is the number of elements, i.e., buckets, on the n-gram bucket stack $b_{n\text{–}gram_{s_n}}$.

For example, in Figure 7.11, the illustrated configuration of stacks corresponds to a contribution of $j \times Coverage_{1\text{–}gram} \times \Delta h_n$ from the unigrams stack, a contribution of $k \times Coverage_{2\text{–}gram} \times \Delta h_n$ from the bigrams stack, and a contribution of $1 \times Coverage_{3\text{–}gram} \times \Delta h_n$ from the trigrams stack. This configuration shows the current states of the stacks after $j + k + 1$ steps, which corresponds to an overall h_{RO} of the entire cache system given by $\left(j \times Coverage_{1\text{–}gram} + k \times Coverage_{2\text{–}gram} + 1 \times Coverage_{3\text{–}gram}\right) \times \Delta h_n$.

Description. The FAset increments are defined as follows:

$$\begin{aligned} \Delta FA\left(b_{n\text{–}gram_{s_n}}\right) &= \left|FA\left(b_{n\text{–}gram_{s_n}}\right)\right|, \quad \text{if} \quad s_n = 1 \\ \Delta FA\left(b_{n\text{–}gram_{s_n}}\right) &= \left|FA\left(b_{n\text{–}gram_{s_n}}\right)\right| - \left|FA\left(b_{n\text{–}gram_{s_{n-1}}}\right)\right|, \quad \text{if} \quad s_n > 1 \end{aligned} \tag{7.13}$$

where s_n is the current size of the n-gram stack, i.e., s_n is equal to the current value $sizeOfStack\left(b_{n\text{–}gram_{s_n}}\right)$.

However, the corresponding FAset increments are not equal among all the stacks. Some buckets contribute with higher incremental costs than others, in the sense that they require more FAset additional elements than other buckets.

At each step i of the algorithm execution, the representation of each individual n-gram stack is given by the following expression:

$$S_{n\text{–}gram_{s_n}}(i) = \left(\left\langle b_{n\text{–}gram_1}, b_{n\text{–}gram_2}, b_{n\text{–}gram_3}, \cdots, b_{n\text{–}gram_{s_n}}\right\rangle, i\right) \tag{7.14}$$

Thus, for each n-grams stack and glue evaluation g:

$$Cost\left(S_{n\text{–}gram_{s_n}}, g\right) = Cost\left(b_{n\text{–}gram_{s_n}}, g\right) \tag{7.15}$$

where the absolute cost $Cost\left(b_{n\text{–}gram_{s_n}}, g\right)$ is given by:

$$Cost\left(b_{n\text{–}gram_{s_n}}, g\right) = \frac{\left|FA\left(b_{n\text{–}gram_{s_n}}\right)\right|}{h'_{RO_{C_n}}\left(b_{n\text{–}gram_{s_n}}\right)} \tag{7.16}$$

$$\left|FA\left(S_{n\text{–}gram_{s_n}}\right)\right| = \left|FA\left(b_{n\text{–}gram_{s_n}}\right)\right| = \sum_{l=1}^{s_n} \Delta FA\left(b_{n\text{–}gram_l}\right) \tag{7.17}$$

$$h'_{RO_{C_n}}\left(S_{n\text{–}gram_{s_n}}\right) = h'_{RO_{C_n}}\left(b_{n\text{–}gram_{s_n}}\right) = s_n \times Coverage_{n\text{–}gram} \times \Delta h_n \tag{7.18}$$

The global configuration of all the n-gram stacks is given by the tuple:

$$S(i) = \left(S_{1\text{–}gram_{s_1}}(i), S_{2\text{–}gram_{s_2}}(i), S_{3\text{–}gram_{s_3}}(i), \cdots, S_{(n\text{–}1)\text{–}gram_{s_{(n-1)}}}(i)\right) \tag{7.19}$$

where n is the maximum n-gram size for the glue calculation, e.g., $n = 4$ in the case of glue g_{234} (Figure 7.11). The corresponding global FAset ensured by a given global configuration $S(i)$ is:

$$FA(S(i)) = FA\left(S_{1\text{–}gram_{s_1}}(i)\right) \cup FA\left(S_{2\text{–}gram_{2_2}}(i)\right) \cup \cdots \cup FA\left(S_{(n\text{–}1)\text{–}gram_{s_{(n-1)}}}(i)\right) \tag{7.20}$$

Initially all the stacks start empty and the algorithm goal is to build a triple of FAset (FA_1, FA_2 and FA_3) that collectively ensures a target global hit ratio (h_{RO}). In the initial state, for each n-gram size, there is an ordered list of candidate buckets waiting to be selected.

At each step the algorithm chooses for expansion one of the stacks, by inspecting the head buckets in all the stack lists, and choosing the one that contributes to the minimal incremental cost, and adds this selected bucket to the top of the corresponding stack. The incremental cost of adding a Δh_n to one of the stacks with the corresponding FAset increment is given by:

$$\Delta Cost\left(b_{n\text{–}gram_{s_n}}\right) = \frac{\Delta FA\left(b_{n\text{–}gram_{s_n}}\right)}{Coverage_{n\text{–}gram} \times \Delta h_n} \tag{7.21}$$

The absolute cost of an individual n-gram stack with s_n buckets is given by:

$$Cost\left(b_{n\text{–}gram_{s_n}}\right) = \frac{\left|FA\left(b_{n\text{–}gram_{s_n}}\right)\right|}{h'_{RO_{C_n}}\left(b_{n\text{–}gram_{s_n}}\right)} = \frac{\sum_{l=1}^{s_n} \Delta Cost\left(b_{n\text{–}gram_l}\right)}{s_n} \tag{7.22}$$

At the end of each step the algorithm evaluates the current global hit ratio (h_{RO}) that is determined by the current tops of the stacks, for example $b_{1\text{–}gram_{s_1=j}}$, $b_{2\text{–}gram_{s_2=k}}$, and $b_{3\text{–}gram_{s_3=1}}$ in the case of Figure 7.11, and it terminates in case the current global hit ratio

is equal or greater than the target hit ratio. The FAset for each n-gram size is defined by the contents of the accumulated buckets in each corresponding stack.

Given a global stack configuration $S(i)$ at step i the corresponding value of the global hit ratio is:

$$\begin{aligned} h'(S(i)) = h_{RO} = h'_{RO_{C_1}}\left(S_{1\text{–}gram_{s_1}}(i)\right) + \\ h'_{RO_{C_2}}\left(S_{2\text{–}gram_{s_2}}(i)\right) + \cdots + \\ h'_{RO_{C_{n-1}}}\left(S_{(n-1)\text{–}gram_{s_{(n-1)}}}(i)\right) \end{aligned} \tag{7.23}$$

$$Cost(S(i)) = \frac{|FA(S(i))|}{h'(S(i))} \tag{7.24}$$

This leads to the configuration of minimal cost to ensure a target h_{RO} (the minimal FAset). The correctness of the algorithm is ensured, in the sense of achieving a minimum cost solution, by choosing a small enough increment in the hit ratio to guarantee that in each step during the algorithm execution there are no minimum cost alternatives left out. This would be achieved if the increment in each step would correspond to adding a single element to the selected FAset candidate, i.e., each bucket would have size one. However, that approach would lead to a huge number of algorithm steps due to the large sizes of the FAset considered. For example for a *corpus* with 466 Mw, in a single machine case and for combined glue g_{234}, this would lead to a number of steps equal to $216\,107\,531 = |D_1| + |D_2| + |D_3|$, for h_{RO} equal to the maximum possible value of 100%. To make the algorithm implementation feasible, we considered small increments in the hit ratio at each step, in the order of 5%. In this case for the same *corpus* of 466 Mw the algorithm only requires 60 steps to find a solution.

7.1.4.3 Minimal Cost FAset Algorithm — An example

Evolution of the Global Hit Ratio and FAset Size. Figure 7.12 shows, for a *corpus* size of 466 Mw and for glue g_{234}, the evolution of the global hit ratio (h_{RO}) from 0% to 100%, represented in the Y axis and the corresponding ordered list (from left to right) of the buckets (represented in the X axis) as selected by the algorithm. The notation used for the bucket identifiers is: `BU` for unigrams, `BB` for bigrams and `BT` for the trigrams. The associated number represents the selection ordering on the corresponding stack. In this case, for the considered increment in the individual hit ratio $\Delta h_n = 5\%$, there are a total of 20 buckets for each n-gram stack corresponding to a total of 60 buckets, although for clarity the X axis only shows half of their identifiers.

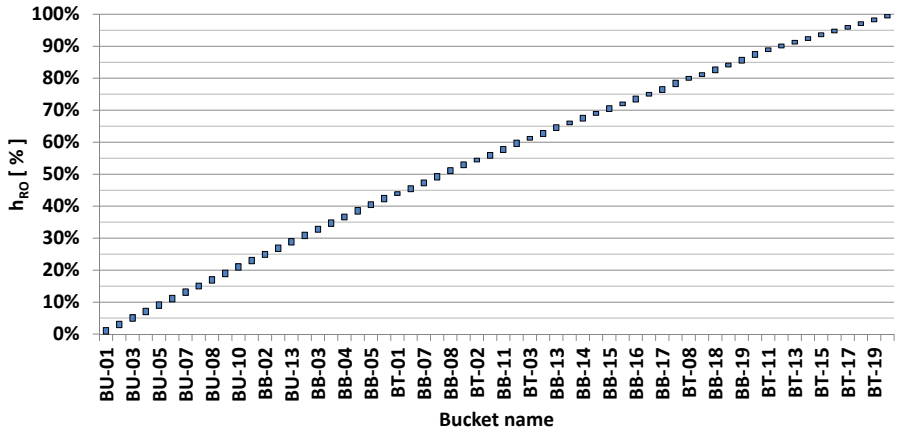


Figure 7.12: FAset algorithm — Evolution of the global hit ratio for a *corpus* of 466 Mw and glue g_{234}.

The last bucket selected leads to the total FAset which includes all the distinct unigrams, bigrams and trigrams of the analyzed *corpus*, corresponding to a global h_{RO} = 100%.

Figure 7.13 shows the evolution of the size of the total FAset in the same conditions as above.

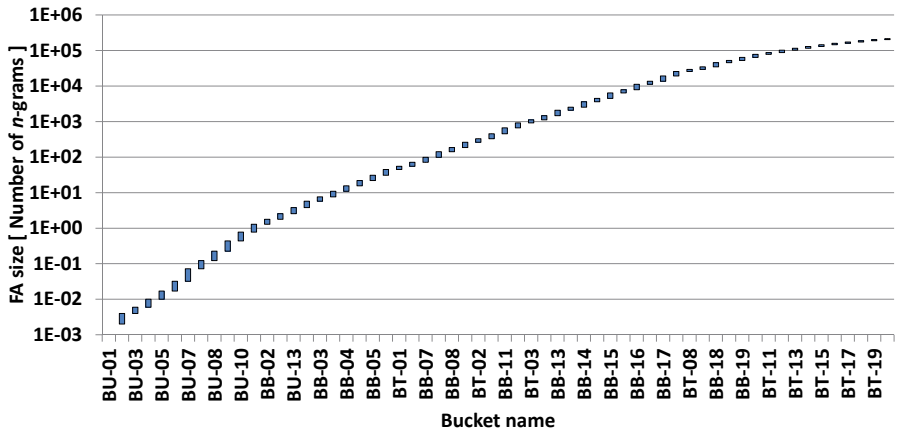


Figure 7.13: FAset algorithm — Ordered list of selected buckets for a *corpus* of 466 Mw and glue g_{234}. The Y axis represents the cardinality of the FAset and the X axis represents the ordered list of selected buckets.

Examples of Different Global Hit Ratio Targets. The algorithm calculates the triple $\langle h_{RO_{C_1}}, h_{RO_{C_2}}, h_{RO_{C_3}} \rangle$ with the minimum cost that corresponds to each target h_{RO}. Table 7.4 shows 3 examples for the h_{RO} targets of 70%, 80% and 90% and the corresponding values, obtained by the algorithm, for $h_{RO_{C_1}}$, $h_{RO_{C_2}}$ and $h_{RO_{C_3}}$, for a *corpus* of 466 Mw in the case of glue g_{234}.

Table 7.4: Algorithm results for a *corpus* of 466 Mw and combined glue g_{234}. FAset cardinalities are expressed in number of n-grams.

	Target h_{RO}		
	70%	**80%**	**90%**
Obtained h_{RO}	71.36 %	80.51 %	90.65 %
$\|FA\|$	6 371 858	27 610 360	96 726 260
$\|FA\|/\|FA_{max}\|$	3.02 %	13.07 %	45.78 %
$h_{RO_{C_1}}$	95.00 %	100.00 %	100.00 %
$\|FA_1\|$	325 143	4 941 629	4 941 629
$\|FA_1\|/\|FA_{1_{max}}\|$	6.58 %	100.00 %	100.00 %
$h_{RO_{C_2}}$	75.00 %	85.00 %	100.00 %
$\|FA_2\|$	4 440 834	13 340 884	46 470 946
$\|FA_2\|/\|FA_{2_{max}}\|$	9.56 %	28.71 %	100.00 %
$h_{RO_{C_3}}$	25.00 %	40.00 %	60.00 %
$\|FA_3\|$	1 605 881	9 327 847	45 313 685
$\|FA_3\|/\|FA_{3_{max}}\|$	1.00 %	5.83 %	28.34 %

We observe the following:

- The obtained values of the global hit ratio (h_{RO}) are slightly above the target ones due to the above mentioned integer approximation of the FAset value;
- For each n-gram size, the value of $FA_{n_{max}}$ corresponds to a value of 100% for $h_{RO_{C_n}}$; Note that due to the coverages ($Coverage_{n\text{–}gram}$) of the n-grams population for each n-gram size relative to the total number of n-gram references, the maximum effective contribution to the global hit ratio (h_{RO}) from each of the individual n-gram caches C_1, C_2, and C_3 are, respectively, $h'_{RO_{C_1}} = h_{RO_{C_1}} \times 0.404 = 40.4\%$, $h'_{RO_{C_2}} = h_{RO_{C_2}} \times 0.363 = 36.3\%$, and $h'_{RO_{C_3}} = h_{RO_{C_3}} \times 0.233 = 23.3\%$, when $h_{RO_{C_1}} = h_{RO_{C_2}} = h_{RO_{C_3}}$=100%. The values of coverages of the unigrams, bigrams and trigrams are, respectively, 0.404, 0.363 and 0.233 obtained as in Table 7.1, for the 466 Mw *corpus*.
- The table illustrates that, as a result of the higher priority given by the algorithm to the smaller n-gram sizes, the recommended individual hit ratio for the unigrams ($h_{RO_{C_1}}$) is higher than the one for bigrams ($h_{RO_{C_2}}$) and for trigrams ($h_{RO_{C_3}}$).

Global FAset Size. Figure 7.14 shows the variation of the size of the global FAset for a combined glue calculation g_{234} *versus* the global hit ratio (h_{RO}) for a *corpus* size of 466 Mw. The size of the global FAset corresponds to the sum of the sizes of the FAset for unigrams, bigrams and trigrams.

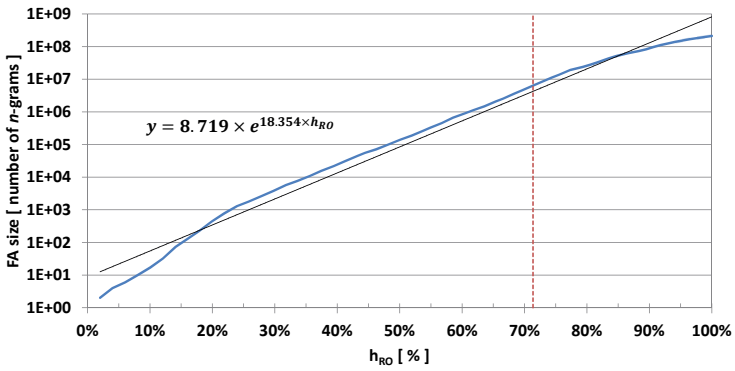


Figure 7.14: FAset algorithm — Global size FA *versus* h_{RO} for *corpus* of 466 Mw. The Y axis represents the FAset size and the X axis represents the global hit ratio in percentage. The dotted vertical line represents an example of a target h_{RO} value near 70%.

We observe the following:

- The overall trend of the global FAset size as a function of h_{RO} is approximated by the exponential function presented in the figure;
- Due to the Zipf-like distribution of n-gram frequencies, which are progressively decreasing as we move from the higher frequency n-gram ranks to the lower frequency ranks, there is a need to include higher proportions of n-grams into the FAset as we increase the target h_{RO}. Thus, as we increase the required target h_{RO} the FAset membership grows quickly until the entire set of distinct n-grams are included in the FAset.

Individual FAsets Sizes. Figure 7.15 shows the variation of the size of the individual FAsets for a combined glue calculation g_{234} *versus* the global hit ratio (h_{RO}) for a *corpus* size of 466 Mw.

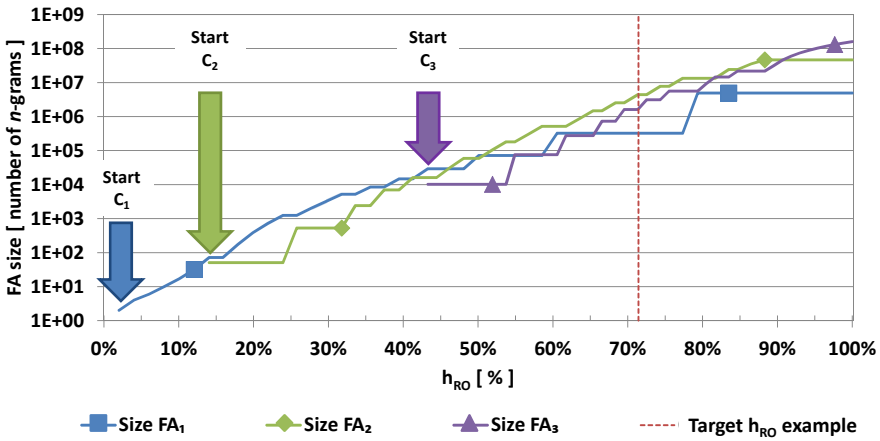


Figure 7.15: FAset algorithm — Sizes FA_1, FA_2 and FA_3 *versus* h_{RO} for a *corpus* of 466 Mw. The Y axis represents the size of the individual FAsets and the X axis represents the global hit ratio in percentage.

The figure illustrates how the algorithm identifies distinct segments when increasing h_{RO}, where each individual n-gram cache is called for contribution:

- The first segment is only due to cache C_1 contribution, followed by a cache C_2 segment contribution, and another C_1 contribution, and so on;
- Note that the first segment contribution from cache C_3 (marked "Start C_3") is only requested for higher values of h_{RO}.

Global FAset Cost. Figure 7.16 shows the variation of the cost (ratio $|FA|/h_{RO}$) for a combined glue calculation g_{234} *versus* the global hit ratio (h_{RO}) for a *corpus* size of 466 Mw.

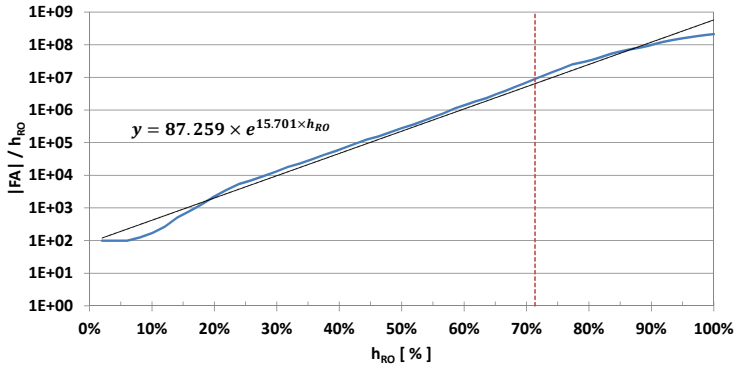


Figure 7.16: FAset algorithm — Ratio $|FA|/h_{RO}$ *versus* h_{RO} for a *corpus* of 466 Mw. The Y axis represents the cost and the X axis represents the global hit ratio in percentage.

This figure illustrates the following points:

- The FAset membership required per unit increment of h_{RO} grows as the target h_{RO} increases;
- As the algorithm starts by first selecting the n-grams with higher frequencies for each n-gram size, then as further new FAset elements are required for increasing values of h_{RO}, their contribution is progressively lower (according to the Zipf-like distribution), leading to an overall increase in the $|FA|/h_{RO}$ cost.

7.1.4.4 Influence of the *n*-grams Frequency Distribution in the FAset

The FAsets considered by the algorithm and for instance illustrated in the cases of Table 7.4 take into account all distinct sub n-gram frequencies of occurrences in the set of distinct n-grams tables.

Note that when $K = 1$ the sets of distinct subunigrams are equal for any of the glue calculations g_2, g_3, g_4, g_{23} or g_{234}: $D1_{in}D_2$=$D1_{in}D_3$=$D1_{in}D_4$=$D1_{in(D_2 \cup D_3)}$=$D1_{in(D_2 \cup D_3 \cup D_4)}$=$D_1$. The same applies to bigrams, trigrams and so on. However, each individual sub n-gram contributes with different frequencies of occurrences within the distinct n-gram tables

involved on each glue calculation. In the case of combined glue g_{234}, regarding the subunigrams the above mentioned tables correspond to $D_2 \cup D_3 \cup D_4$; for subbigrams it is $D_3 \cup D_4$; and for subtrigrams it is D_4. In the case of combined glue $g_{2...6}$, regarding the subunigrams the above mentioned tables would be $D_2 \cup D_3 \cup D_4 \cup D_5 \cup D_6$; for subbigrams it is $D_3 \cup D_4 \cup D_5 \cup D_6$; for subtrigrams it is $D_4 \cup D_5 \cup D_6$; and so on.

Thus, a given FAset membership consisting of the same distinct sub n-grams, has a different impact depending on the n-gram tables where the sub n-gram occurrences are counted.

An Example. For g_2 one can build a unigram FAset to ensure a target $h_{RO_{C_1}} = 80\%$ and compare that unigram FAset with the unigram FAset required to obtain the same hit ratio for the glue g_{234}. We expect that the latter unigram FAset would be smaller than the former, because the average unigram efficiency in terms of number of hits per FAset element increases:

$$0.8 \times 2 \times |D_2| = numberHitsByFAset_{D_1 inD_2} \tag{7.25}$$

$$0.8 \times 2 \times (|D_2| + |D_3| + |D_4|) = numberHitsByFAset_{D_1 in(D_2 \cup D_3 \cup D_4)} \tag{7.26}$$

$$\frac{0.8 \times 2 \times |D_2|}{\left|FAset_{D_1 inD_2}\right|} = efficiencyHitPerFAsetElement_{D_1 inD_2} \tag{7.27}$$

$$\frac{0.8 \times 2 \times (|D_2| + |D_3| + |D_4|)}{\left|FAset_{D_1 in(D_2 \cup D_3 \cup D_4)}\right|} = efficiencyHitPerFAsetElement_{D_1 in(D_2 \cup D_3 \cup D_4)} \tag{7.28}$$

By dividing expression (7.27) by expression (7.28) we obtain:

$$\frac{efficiencyHitPerFAsetElement_{D_1 inD_2}}{efficiencyHitPerFAsetElement_{D_1 in(D_2 \cup D_3 \cup D_4)}} = \frac{0.8 \times 2 \times |D_2|}{\left|FAset_{D_1 inD_2}\right|} \times \frac{\left|FAset_{D_1 in(D_2 \cup D_3 \cup D_4)}\right|}{0.8 \times 2 \times (|D_2| + |D_3| + |D_4|)} = \frac{|D_2|}{|D_2| + |D_3| + |D_4|} \times \frac{\left|FAset_{D_1 in(D_2 \cup D_3 \cup D_4)}\right|}{\left|FAset_{D_1 inD_2}\right|} \tag{7.29}$$

FAset Ranking Histograms. Figure 7.17 shows the frequencies of occurrences of the unigrams included in the FAsets for the following cases:

- $h_{RO_{C_1}}\left(1, FAset_{D_{1_{in}D_2}}, g_2\right)$;
- $h_{RO_{C_1}}\left(1, FAset_{D_{1_{in}D_3}}, g_3\right)$;
- $h_{RO_{C_1}}\left(1, FAset_{D_{1_{in}D_4}}, g_4\right)$;
- $h_{RO_{C_1}}\left(1, FAset_{D_{1_{in}(D_2 \cup D_3)}}, g_{23}\right)$;

- $h_{RO_{C_1}}\left(1, FAset_{D_{1_{in}(D_2 \cup D_3 \cup D_4)}}, g_{234}\right)$.

all obtained for a *corpus* with 466 Mw and targeting the same fixed value of the corresponding $h_{RO_{C_1}} = 80\%$.

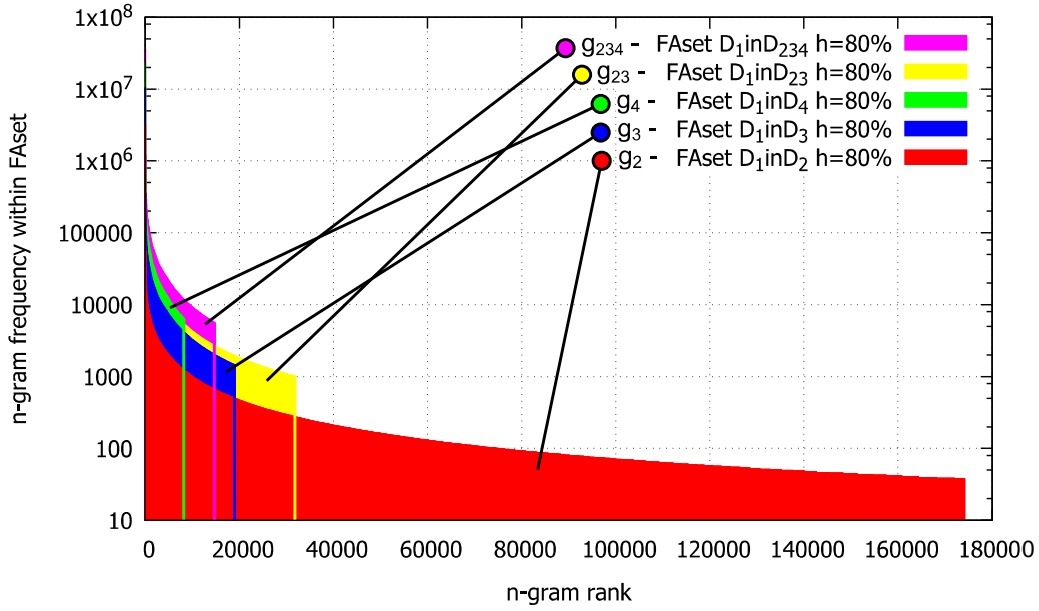


Figure 7.17: FAset unigram histograms for a *corpus* of 466 Mw and $h_{RO_{C_1}} = 80\%$. The Y axis represents the n-gram frequency in the sets $D_{1_{in}D_m}$ and the X axis represents the n-gram rank.

Figure 7.17 shows:

- The number of elements in $FAset_{D_1 inD_2}$ is much greater than the number of elements in $FAset_{D_1 in(D_2 \cup D_3 \cup D_4)}$, for equal target hit ratio $h_{RO_{C_1}} = 80\%$;

- As expected, the number of hits per FAset element in the case of $FAset_{D_1 in(D_2 \cup D_3 \cup D_4)}$ is greater than in the case of $FAset_{D_1 inD_2}$, also shown in expression (7.29).

Similar results can be derived for larger n-gram sizes, as illustrated in Figure 7.18 for bigrams for the same *corpus* size as above, both for a *corpus* of 466 Mw.

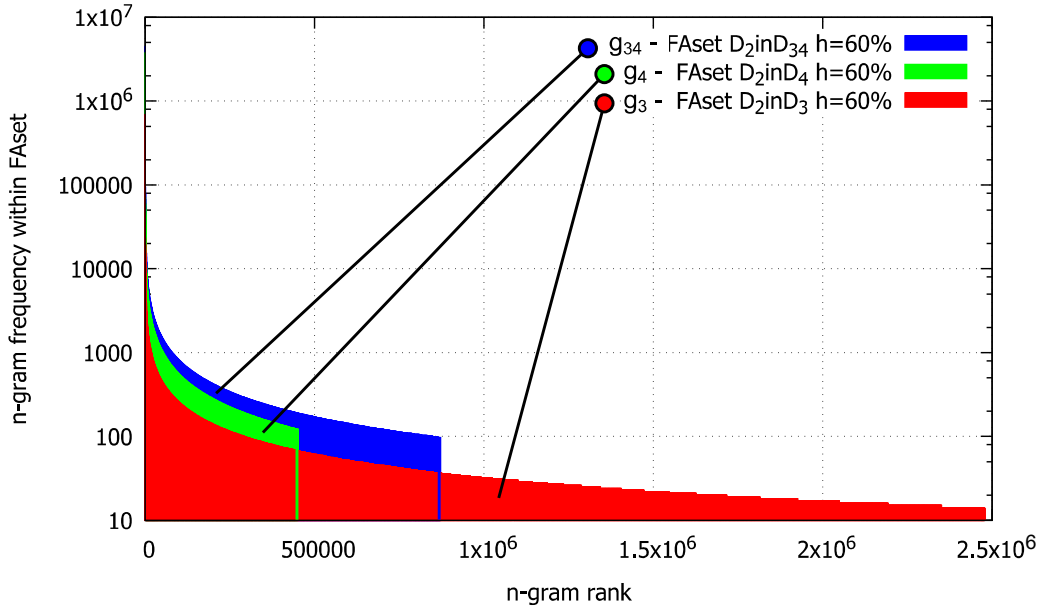


Figure 7.18: FAset bigram histograms for a *corpus* of 466 Mw and $h_{RO_{C_2}} = 60\%$. The Y axis represents the n-gram frequency in the sets $D_{2_{in}D_m}$ and the X axis represents the n-gram rank.

Figure 7.19 shows the frequencies of occurrences of the trigrams included in the FAset $FAset_{D_{3_{in}D_4}}$, obtained for a *corpus* with 466 Mw.

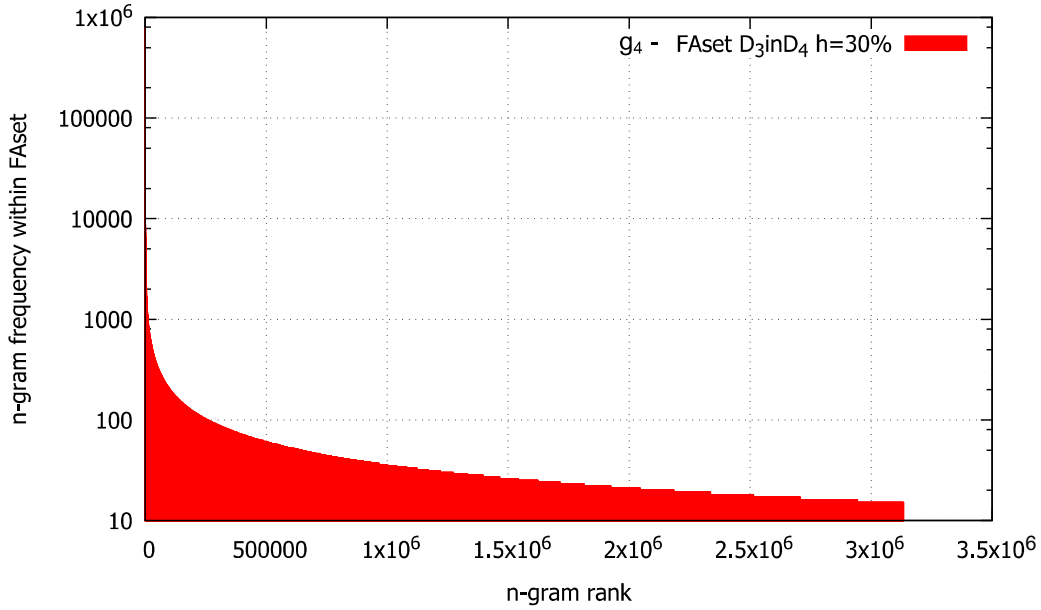


Figure 7.19: FAset trigram histograms for a *corpus* of 466 Mw and $h_{RO_{C_3}} = 30\%$. The Y axis represents the n-gram frequency in the sets $D_{3_{in}D_m}$ and the X axis represents the n-gram rank.

Approximations to the FAset. For larger *corpus* sizes and higher n-gram sizes the FAset calculations performed by the Minimal Cost FAset algorithm may become computationally heavy due to the sizes of the distinct n-gram tables. In this context, an approximation to the FAset membership based on a subset of distinct n-gram tables can be considered. For example, building the FAset for unigrams based on a subset of $D_{1_{in}D_2}$ only, which just corresponds to the FAset needed for the isolated glue g_2; building the FAset for bigrams based on a subset of $D_{2_{in}D_3}$ only, which just corresponds to the FAset needed for the isolated g_3, etc. Although this would lead to higher FAset sizes than the ones obtained by the Minimal Cost algorithm, the base target hit ratio considered for each n-gram size and isolated glue calculation will be further increased due to the successive and combined glue calculations.

An even more lightweight approach would be building the FAsets by including all the distinct sub n-grams found within the corresponding distinct n-gram tables, following the ordering of their decreasing efficiencies (section 7.1.4.1), i.e., starting with unigrams, then bigrams, etc. However, in this case, it would be wasteful to include also the singleton n-grams, as discussed in the following section.

7.1.4.5 Excluding the Singleton *n*-grams from the FAset

In Table 7.4 the set $FA_{n_{max}}$ ($1 \leq n \leq 3$) corresponds to a value of $h_{RO_{C_n}} = 100\%$, and includes all the distinct n-grams of size n. However, in some *corpus* size ranges, a subset of those n-grams are singletons, occurring only once in the *corpus*. For example, for the English *corpora* considered, there are singleton unigrams, or bigrams, or trigrams, ..., or hexagrams, for all the *corpus* sizes smaller than values around 1 to 10 Tw (section 3.6 on page 48).

Thus, those singletons should not be included in the FAset that will be used to load a static cache. So, we are interested in building FAsets that only include nonsingleton n-grams. In the following we evaluate the influence of excluding the singleton n-grams in the case of calculating the combined glue g_{234}.

Analysis of the FAset for Cache C_1. We first consider the isolated glue calculation g_2 and the maximum hit ratio achieved by cache C_1 when $FA_{NonSing}\left(1, g_2, h_{RO_{C_1}}\right)$ includes all nonsingleton unigrams occurring in the *corpus*:

$$h_{RO_{C_1}}\left(g_2, FA_{NonSing}\right) = \frac{\sum\limits_{ng \in FA_{NonSing}} freq_{inD_2}(ng)}{\left|all_{glue_{g_2}Ref_{1\text{-}gram}}\right|} = 1 - \frac{|S_1|}{|D_2|} \tag{7.30}$$

where ng is a distinct nonsingleton unigram, S_1 is the set of singleton unigrams in the *corpus*, D_2 is the set of distinct bigrams in the *corpus*, and $freq_{inD_2}(ng)$ is the frequency of ng in D_2.

Apart from the first and the last unigram occurrences in the *corpus*, each singleton unigram in the *corpus* occurs twice in the D_2 table, namely, it occurs exactly within two

distinct bigrams. For example, a substring in the *corpus* such as *"a b c a b"* maps into a D_2 table containing the following distinct bigrams: {*"a b"*,*"b c"*,*"c a"*}, where it is clear that the singleton unigram *"c"* appears only in two bigrams. Let us consider all the input unigram references to cache C_1 for glue g_2, that is $\left|all_{glue_{g_2}Ref_{1\text{–}gram}}\right| = 2 \times |D_2|$. This can be decomposed into a first parcel, representing the unigrams FAset ($FA_{NonSing}$), which considers only the existing nonsingleton unigrams in the *corpus* and accumulating all their frequencies of occurrences in the D_2 table; and a second parcel which accounts for the singleton unigrams occurrences in the D_2 table, that is $2 \times |S_1|$.

The following expressions may be derived using a similar reasoning for the isolated glues g_3 and g_4:

$$h_{RO_{C_1}}\left(g_3, FA_{NonSing}\right) = \frac{\sum\limits_{ng \in FA_{NonSing}} freq_{inD_3}(ng)}{\left|all_{glue_{g_3}Ref_{1\text{–}gram}}\right|} = 1 - \frac{|S_1|}{|D_3|} \tag{7.31}$$

$$h_{RO_{C_1}}\left(g_4, FA_{NonSing}\right) = \frac{\sum\limits_{ng \in FA_{NonSing}} freq_{inD_4}(ng)}{\left|all_{glue_{g_4}Ref_{1\text{–}gram}}\right|} = 1 - \frac{|S_1|}{|D_4|} \tag{7.32}$$

where $freq_{inD_3}(ng)$ and $freq_{inD_4}(ng)$ are, respectively, the frequencies of the nonsingleton unigram ng in D_3 and D_4.

Likewise, for the combined glue g_{234}:

$$h_{RO_{C_1}}\left(g_{234}, FA_{NonSing}\right) = \frac{\sum\limits_{ng \in FA_{NonSing}} freq_{in(D_2 \cup D_3 \cup D_4)}(ng)}{\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right|} = 1 - \frac{3 \times |S_1|}{|D_2| + |D_3| + |D_4|} \tag{7.33}$$

where $freq_{in(D_2 \cup D_3 \cup D_4)}(ng)$ is the frequency of the nonsingleton unigram ng in the union of D_2, D_3 and D_4 tables.

The above expression is obtained as follows:

$$\frac{\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right|}{\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right|} = \frac{\sum\limits_{ng \in FA_{NonSing}} freq_{in(D_2 \cup D_3 \cup D_4)}(ng) + (2 \times |S_1| + 2 \times |S_1| + 2 \times |S_1|)}{\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right|} \tag{7.34}$$

leading to:

$$1 = \frac{\sum\limits_{ng \in FA_{NonSing}} freq_{in(D_2 \cup D_3 \cup D_4)}(ng)}{\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right|} + \frac{6 \times |S_1|}{\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right|} \tag{7.35}$$

where $\left|all_{glue_{g_{234}}Ref_{1\text{–}gram}}\right| = 2 \times |D_2| + 2 \times |D_3| + 2 \times |D_4|$.

The above hit ratios are shown in Figure 7.20 as a function of the *corpus* size.

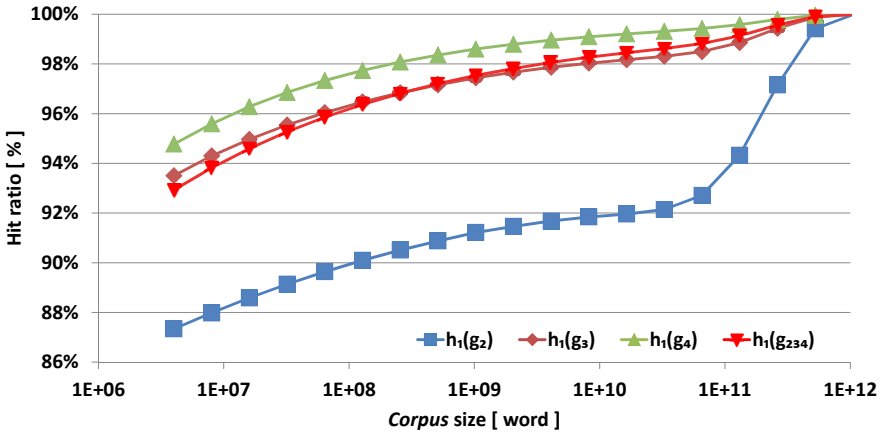


Figure 7.20: Cache C_1 hit ratios for nonsingletons FAset. The Y axis represents the hit ratio in percentage and the X axis represents the *corpus* size in words.

In all the cases of the isolated glue calculations, from g_2 to g_4, the total number of unigram misses out of the static cache C_1, preloaded with the above FAset ($FA_{NonSing}$), is the same and equal to $2 \times |S_1|$. Thus:

- For each fixed *corpus* size and its associated nonsingleton unigrams FAset, the corresponding cache C_1 hit ratios increase when going from g_2 to g_4;
- The hit ratios grow monotonically with the *corpus* size as the population of the nonsingleton unigrams becomes dominant, and reach 100% in the *plateau* regions as soon as the singletons disappear; Of course this requires enough memory capacity to include the growing FAset memberships that follow the same behavior as the nonsingletons (Figure 3.10 on page 49).

Apparently, from the above analysis, one could be led to conclude that the curves in Figure 7.20 show the absolute maximum hit ratios that could be achieved by a static C_1 cache in the above conditions where the entire population of singleton unigrams is strictly treated as cache misses.

However, when we consider the nature of the above unigram misses, we know that they correspond to singleton unigrams in the *corpus*, thus they have a frequency of occurrence of 1 in the *corpus*. By recalling that the LocalMaxs glue calculation is based on the knowledge of the global frequencies of occurrence of n-grams in the *corpus*, one can automatically infer that any of the distinct n-grams associated to the above cache misses has a frequency of 1, without requiring any further processing, that is the cache plays the role of filtering the singletons. Thus, it has an equivalent behavior of a cache with a 0% miss ratio. This assumes that the prefetching of the nonsingleton n-grams in the FAset takes place without any overhead, namely that is performed in complete time overlap with other computations.

Analysis of the FAsets for Caches C_2 and C_3. From a similar analysis, we derive the corresponding hit ratio expressions for static caches C_2 and C_3 when loaded with non-singleton FAsets exclusively. Considering that the singleton bigrams in the *corpus* (S_2), except for the first and the last ones in the *corpus*, have a total number of occurrences equal to $2 \times |S_2|$ as subbigrams in each of the distinct n-gram tables D_3 or D_4, we obtain the following expressions:

$$h_{RO_{C_2}}\left(g_3, FA_{NonSing}\right) = \frac{\sum_{ng \in FA_{NonSing}} freq_{inD_3}(ng)}{\left|all_{glue_{g_3} Ref_{2\text{–}gram}}\right|} = 1 - \frac{|S_2|}{|D_3|} \tag{7.36}$$

$$h_{RO_{C_2}}\left(g_4, FA_{NonSing}\right) = \frac{\sum_{ng \in FA_{NonSing}} freq_{inD_4}(ng)}{\left|all_{glue_{g_4} Ref_{2\text{–}gram}}\right|} = 1 - \frac{|S_2|}{|D_4|} \tag{7.37}$$

The following expression for cache C_2 and glue g_{234} is derived similarly to expression (7.33):

$$h_{RO_{C_2}}\left(g_{234}, FA_{NonSing}\right) = \frac{\sum_{ng \in FA_{NonSing}} freq_{in(D_3 \cup D_4)}(ng)}{\left|all_{glue_{g_{234}} Ref_{2\text{–}gram}}\right|} = 1 - \frac{2 \times |S_2|}{|D_3| + |D_4|} \tag{7.38}$$

where $freq_{in(D_3 \cup D_4)}(ng)$ is the frequency of the nonsingleton bigram ng in the union of D_3 and D_4 tables.

Considering that the singleton trigrams in the *corpus* (S_3), except for the first and the last ones in the *corpus*, have a total number of occurrences equal to $2 \times |S_3|$ as subtrigrams in the distinct n-gram table D_4, we obtain the following expression:

$$h_{RO_{C_3}}\left(g_4, FA_{NonSing}\right) = \frac{\sum_{ng \in FA_{NonSing}} freq_{inD_4}(ng)}{\left|all_{glue_{g_4} Ref_{3\text{–}gram}}\right|} = 1 - \frac{|S_3|}{|D_4|} \tag{7.39}$$

where $freq_{inD_4}(ng)$ is the frequency of the nonsingleton trigram ng in the D_4 table.

The same conclusions as above can be drawn concerning the interpretation of the singleton bigram misses and the singleton trigram misses.

Global Hit Ratio for the Cache System for Combined Glues. Finally the global hit ratios for the cache system, for the combined glues g_{234} and $g_{2\ldots6}$, when the FAsets consist entirely of all nonsingleton n-grams, can be derived as follows. For example, for cache system C_{1+2+3} and glue g_{234} we obtain the following:

$$h_{RO_{C_{1+2+3}}}\left(g_{234}, FA_{NonSing_{1,2,3}}\right) \times \left|all_{glue_{g_{233}}Ref}\right| = \sum_{ng_1 \in FA_{NonSing_1}} freq_{in(D_2 \cup D_3 \cup D_4)}(ng_1) + \sum_{ng_2 \in FA_{NonSing_2}} freq_{in(D_3 \cup D_4)}(ng_2) + \sum_{ng_3 \in FA_{NonSing_3}} freq_{in(D_4)}(ng_3) \tag{7.40}$$

where ng_1, ng_2, and ng_3 denote, respectively, a nonsingleton unigram, a nonsingleton bigram, and a nonsingleton trigram, included in the corresponding ($FA_{NonSing_1}$, $FA_{NonSing_2}$ or $FA_{NonSing_3}$) FAsets.

$$\begin{aligned} &h_{RO_{C_{1+2+3}}}\left(g_{234}, FA_{NonSing_{1,2,3}}\right) \times \left|all_{glue_{g_{233}}Ref}\right| = \\ &\left|all_{glue_{g_{233}}Ref_{1-gram}}\right| \times \left(1 - \frac{6 \times |S_1|}{\left|all_{glue_{g_{234}}Ref_{1-gram}}\right|}\right) + \\ &\left|all_{glue_{g_{233}}Ref_{2-gram}}\right| \times \left(1 - \frac{4 \times |S_2|}{\left|all_{glue_{g_{234}}Ref_{2-gram}}\right|}\right) + \\ &\left|all_{glue_{g_{233}}Ref_{3-gram}}\right| \times \left(1 - \frac{2 \times |S_3|}{\left|all_{glue_{g_{234}}Ref_{3-gram}}\right|}\right) \end{aligned} \tag{7.41}$$

$$\begin{aligned} &h_{RO_{C_{1+2+3}}}\left(g_{234}, FA_{NonSing_{1,2,3}}\right) \times \left|all_{glue_{g_{233}}Ref}\right| = \\ &\left(\left|all_{glue_{g_{233}}Ref_{1-gram}}\right| + \left|all_{glue_{g_{233}}Ref_{2-gram}}\right| + \left|all_{glue_{g_{233}}Ref_{3-gram}}\right|\right) - \\ &(6 \times |S_1| + 4 \times |S_2| + 2 \times |S_3|) \end{aligned} \tag{7.42}$$

leading to:

$$h_{RO_{C_{1+2+3}}}\left(g_{234}, FA_{NonSing_{1,2,3}}\right) = 1 - \frac{6 \times |S_1| + 4 \times |S_2| + 2 \times |S_3|}{2 \times |D_2| + 4 \times |D_3| + 6 \times |D_4|} \tag{7.43}$$

Likewise, for cache system $C_{1+\cdots+5}$ and glue $g_{2\cdots6}$ we obtain the following:

$$h_{RO_{C_{1+\cdots+5}}}\left(g_{2\cdots6}, FA_{NonSing_{1,\cdots,5}}\right) = 1 - \frac{10 \times |S_1| + 8 \times |S_2| + 6 \times |S_3| + 4 \times |S_4| + 2 \times |S_5|}{2 \times |D_2| + 4 \times |D_3| + 6 \times |D_4| + 8 \times |D_5| + 10 \times |D_6|} \tag{7.44}$$

Figure 7.21 shows the corresponding curves as a function of the *corpus* size.

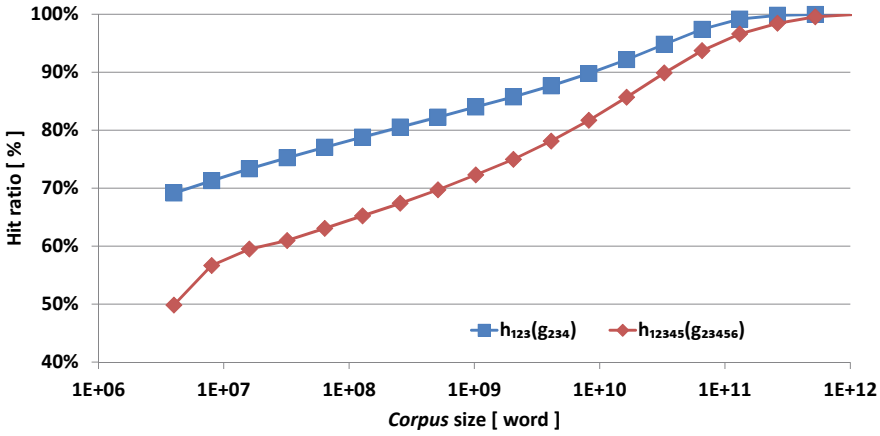


Figure 7.21: Global hit ratio for nonsingletons FAset for combined glues g_{234} and $g_{2\cdots6}$. The Y axis represents the hit ratio in percentage, and the X axis represents the *corpus* size in words.

7.1.5 Multiple Machines Case — Evolution of FAset *versus* K

The above discussion and results presented in sections 7.1.1 to 7.1.4 can be applied to an individual machine, in a single or a multiple machines configuration. In this latter case, the local machine tables ($D_i(j)$) and the local static cache C_{RO_i} must be considered for the local FAset analysis in each machine.

In the following we analyze the evolution of the FAset membership when varying the number of machines (K).

We expect that the FAset membership per machine, as a function of K, will exhibit a similar behavior as the evolution of the number of distinct sub n-grams in each machine (section 5.4.3 on page 87). This is due to the fact that the FAset is a minimal subset of the distinct sub n-grams set, whose membership criterion follows the decreasing order of frequencies of the n-gram occurrences. Thus, the per machine distribution of those n-grams is influenced by the repetition of occurrences of the most intensively used n-grams among the different machines.

Similarly to the analysis of the distribution of the number of distinct n-grams we followed an empirical approach concerning the per machine FAset membership. The following results were obtained by executing the phase one of the LocalMaxs Global method, suitably extended to calculate the FAset for a given *corpus* size and glue calculation. An evaluation of the impact of these extensions upon the execution time is included in section 7.3.1, on page 231.

The results were obtained in a multiple machine environment and different numbers of machines were considered. We used different *corpus* sizes, namely, 64, 128, 256 and 511 Mw, and evaluated the per machine FAset membership for unigrams, bigrams and trigrams, when considering the isolated glue calculations respectively g_2, g_3 and g_4. From Figure 7.10, on page 194, the efficiency of the FAset for unigrams is much greater than for

bigrams and for trigrams, and the efficiencies for the bigrams and trigrams FAset show a significant decrease with the value of the corresponding individual hit ratio. Thus, it is more reasonable, concerning the FAset size, to achieve higher values of the individual hit ratio for the unigrams than the trigrams. So, in these experiments, we considered the following values for the individual hit ratio to be ensured by the FAsets in each machine: $h_{RO_{C_1}} = 80\%$ for the isolated glue g_2, $h_{RO_{C_2}} = 60\%$ for the isolated glue g_3, and $h_{RO_{C_3}} = 30\%$ for the isolated glue g_4. As an example, Table 7.4 shows that when considering a target global hit ratio (h_{RO}) of 70% and the combined glue g_{234} the individual hit ratios indicated by the Minimal Cost FAset algorithm are $h_{RO_{C_1}} = 95\%$, $h_{RO_{C_2}} = 75\%$ and $h_{RO_{C_3}} = 25\%$.

Evolution of the Individual FAsets as a Function of K. The following results show, in Figures 7.22 to 7.24, the evolution of the individual FAset sizes per machine, as a function of the number of machines (K = 3, 6, 9, 12, 18) for the cases of unigrams, bigrams and trigrams, when considering, respectively, the isolated glue calculations g_2, g_3 and g_4. The values shown are average values of the per machine FAset cardinality over the K machines.

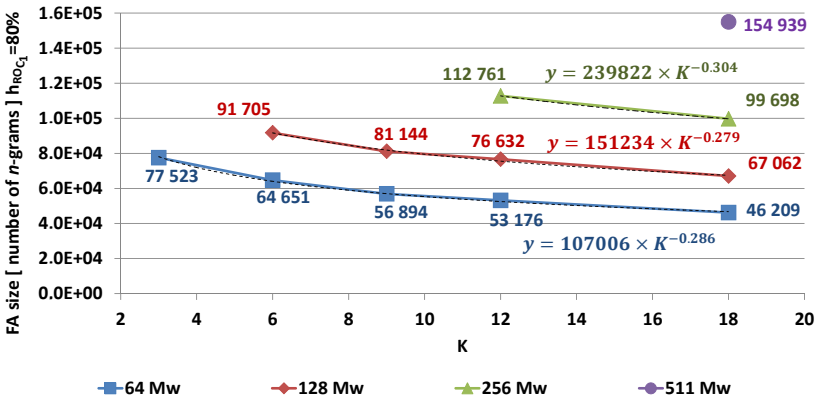


Figure 7.22: Per machine FAset for unigrams, glue g_2 and different *corpus* sizes *versus* K.

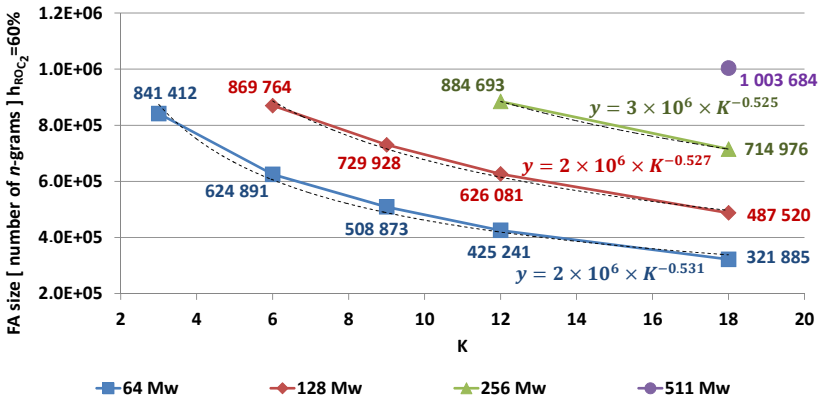


Figure 7.23: Per machine FAset for bigrams, glue g_3 and different *corpus* sizes *versus* K.

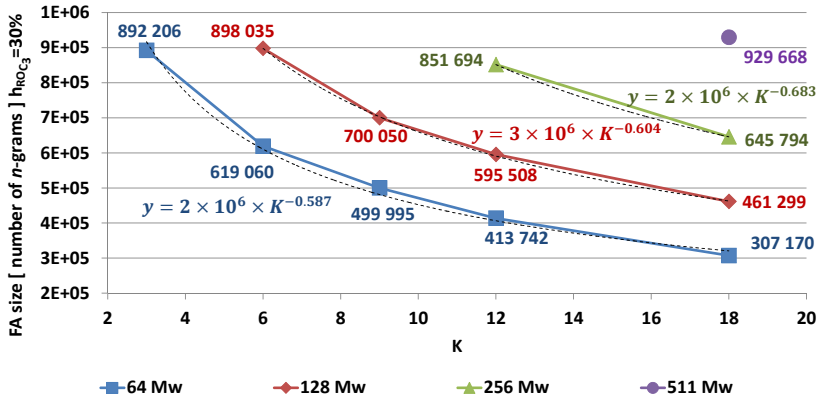


Figure 7.24: Per machine FAset for trigrams, glue g_4 and different *corpus* sizes *versus* K.

From the above figures we conclude:

- The individual per machine FAset curves exhibit a similar trend as the curves for the number of distinct n-grams $D_{i_{in}D_n}(j)$ in the local $D_n(j)$ tables as a function of the number of machines, also following a power law with a negative exponent (section 5.4.3 on page 87);
- Similar behaviors occur for the global FAset of the cache system in each machine when considering the combined glue evaluations.

Evolution of the Static Cache Miss Penalties as a Function of K. Figure 7.25 shows the per machine individual cache miss penalties obtained as averages over the K machines for fixed values of $h_{RO_{C_1}} = 80\%$, $h_{RO_{C_2}} = 60\%$, and $h_{RO_{C_3}} = 30\%$, respectively for the calculation of the isolated glues g_2, g_3 and g_4. For each static cache C_i, with $1 \leq i \leq 3$, the individual cache miss penalty ($miss_i(j)$) for the glue calculation $g_{(i+1)}$ is given by the following expression:

$$miss_i(j) = 2 \times \left|D_{(i+1)}(j)\right| \times \left(1 - h_{RO_{C_i}}(j)\right) \tag{7.45}$$

where j is the machine number ($1 \leq j \leq K$), $D_{(i+1)}(j)$ corresponds to the table of distinct n-grams of size $(i+1)$ in the j machine, and $h_{RO_{C_i}}(j)$ is the corresponding individual cache (C_i) hit ratio.

The following remarks can be taken:

- The average per machine individual cache C_i miss penalty for glue $g_{(i+1)}$ varies with $1/K$ due to being proportional to the distinct n-gram table $D_{(i+1)}(j)$ size. As discussed in chapter 5, the distinct n-grams tables are equally partitioned among K machines, thus $\left|D_{(i+1)}(j)\right|$ is approximately equal to $\left|D_{(i+1)}\right|/K$;
- In the cases where the FAset includes all the nonsingleton n-grams, excluding all the singletons, then we note that, although from the static cache point of view there

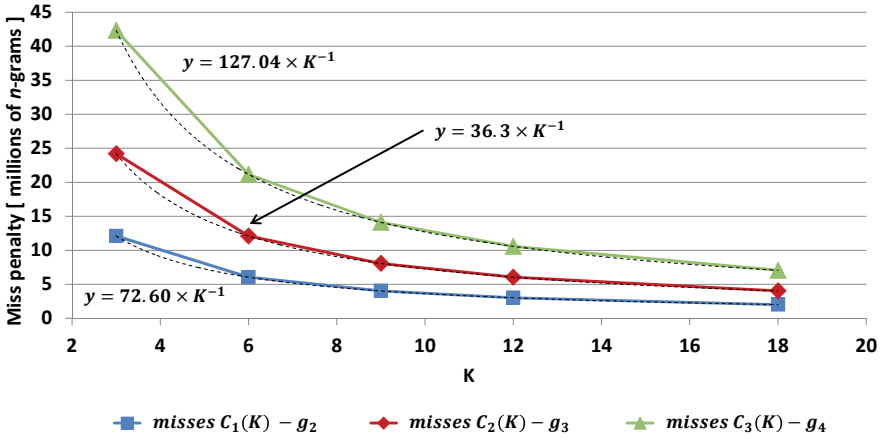


Figure 7.25: Per machine average individual static cache misses for a *corpus* of 64 Mw *versus* K. The Y axis represents the miss penalty in millions of n-grams and the X axis represents the number of machines.

are still misses, they really do not involve any overhead from the point of view of the outside of the cache system because they were clearly identified as singletons, so they do not require a remote fetch of their n-gram frequency data.

In summary, for a scenario where h_{RO} is kept constant when going from $K = 1$ to $K > 1$, the miss penalty of the static cache decreases as $1/K$. For the same scenario, for each n-gram size, the FAset per machine is proportional to K^{-b}, $0 < b < 1$.

7.2 Static plus Dynamic Cache

If it is feasible to design a static cache with enough memory capacity to contain a preloaded FAset that, for each *corpus* size includes all nonsingleton n-grams of size n existing in the *corpus*, then all detected cache misses correspond to singleton n-grams in the *corpus*. In this case the cache system is able to identify those singletons and can automatically associate a frequency of occurrence of 1 to each of those n-grams, which is what is required by the LocalMaxs glue calculation. Thus, this has the equivalent effect of a cache with a zero miss ratio.

If the above assumption does not hold, then the static cache FAset is only able to include a subset of all the nonsingleton n-grams of each size n in the *corpus*, thus the detected cache misses correspond to the remaining nonsingleton n-grams plus all the singleton n-grams. In this case, the cache system is unable to automatically identify the singleton n-gram misses, unlike the former case.

Furthermore, in this latter case, the nonsingletons that were not included in the FAset exhibit multiple occurrences in the *corpus* and also in the input cache references stream ($all_{glue_g Ref}$, e.g., the set $D_2 \cup D_3 \cup D_4$ in the case of glue g_{234}): indeed each nonsingleton is referenced two or more times in that stream, depending on its pattern of occurrences within the *corpus*. Each singleton n-gram in the *corpus*, on the other hand, occurs twice

(as sub n-gram) in the input cache reference stream as explained above (section 7.1.4.5), except for the first and the last ones in the *corpus*.

As discussed in chapter 6, a dynamic cache with infinite capacity only generates the first occurrence misses corresponding to the number of distinct n-grams occurring in the input cache reference stream. Thus, a dynamic cache can be applied to the static cache output miss stream with the result of filtering out all the repetitions in the static cache misses. This has the overall effect of reducing the miss ratio of a composite cache (C_{RO+RW}) that includes a static subcache (C_{RO}) plus a dynamic subcache (C_{RW}), when compared to a single static cache. However, with an infinite dynamic cache this solution will require the same memory capacity as a single static cache loaded with a FAset containing all the distinct n-grams. The required memory can be reduced by using a dynamic cache with finite capacity that allows keeping the n-grams with the higher priorities according to a certain criterion, such as the decreasing order of their frequencies of occurrences, although this solution will imply an additional overhead due to the finite capacity misses.

Concerning the singletons, in a first approach, discussed in this chapter, there is no distinction established by the cache system among the nonsingleton and singleton n-grams appearing in the static cache output miss stream, with the consequence that the dynamic cache (C_{RW}) will also generate one miss for each singleton n-gram. The filtering of any existing singleton n-grams will lead to a further reduction in the overall miss ratio of the cache system. A strategy for filtering the singleton n-grams using Bloom filters is presented in chapter 8, which can be used to improve a single dynamic cache system (as the one in chapter 6) or a composite static plus dynamic cache system (as discussed in this chapter).

In section 7.2.1 we discuss the modeling of the static plus dynamic cache. In section 7.2.2 we discuss the single machine case. Finally, in section 7.2.3 we discuss the multiple machines case.

7.2.1 Definitions

The following expressions (7.46) to (7.49) define the hit and miss ratios of a composite n-gram cache including a static plus a dynamic cache (Figure 7.26).

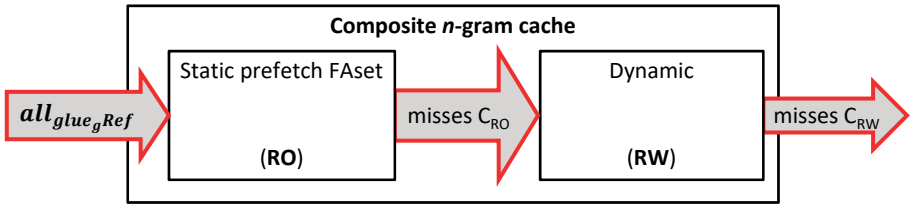


Figure 7.26: Diagram of a composite static plus dynamic cache system.

The individual static cache hit ratio (h_{RO}) and the corresponding miss ratio (mr_{RO}) are:

$$h_{RO} = \frac{\left|all_{glue_g}Ref\right| - |missesC_{RO}|}{\left|all_{glue_gRef}\right|} = 1 - mr_{RO} \tag{7.46}$$

where $missesC_{RO}$ denotes the set of static cache misses.

The individual dynamic cache hit ratio (h_{RW}) and the corresponding miss ratio (mr_{RW}) are:

$$h_{RW} = \frac{|missesC_{RO}| - |missesC_{RW}|}{|missesC_{RO}|} = 1 - mr_{RW} \tag{7.47}$$

where $missesC_{RW}$ denotes the set of dynamic cache misses.

The global hit ratio (h_{RO+RW}) of the composite static plus dynamic cache is:

$$h_{RO+RW} = \frac{\left|all_{glue_g}Ref\right| - |missesC_{RW}|}{\left|all_{glue_gRef}\right|} = 1 - \frac{|missesC_{RW}|}{\left|all_{glue_gRef}\right|} = 1 - mr_{RO+RW} \tag{7.48}$$

and the corresponding miss ratio is:

$$mr_{RO+RW} = mr_{RO} \times mr_{RW} = \frac{|missesC_{RO}|}{\left|all_{glue_gRef}\right|} \times \frac{|missesC_{RW}|}{|missesC_{RO}|} = \frac{|missesC_{RW}|}{\left|all_{glue_gRef}\right|} \tag{7.49}$$

From expression (7.49) we conclude that $mr_{RO+RW} \leq mr_{RO}$ and $mr_{RO+RW} \leq mr_{RW}$. The above definitions apply to an individual composite cache, for a given n-gram size, as well as to a global cache system made of a set of individual n-gram static plus dynamic caches for several values of n (Figure 7.27).

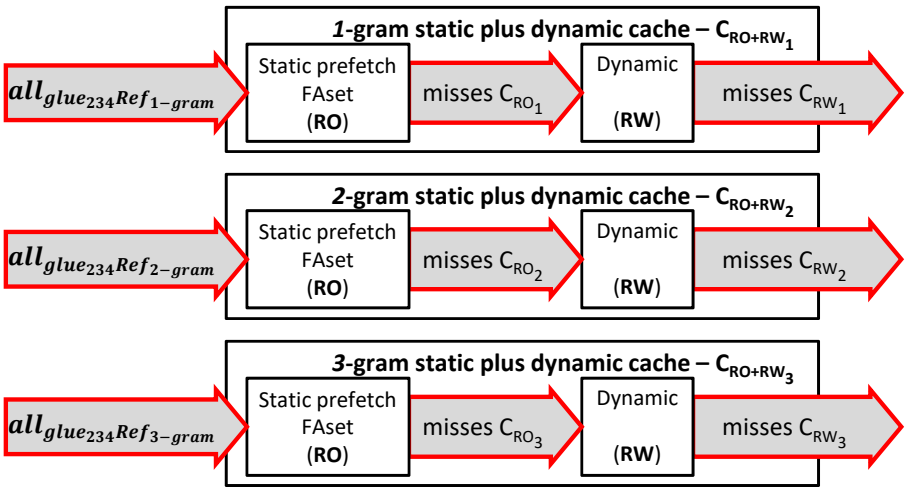


Figure 7.27: Diagram of a static plus dynamic cache system for glue g_{234}.

For an individual static plus dynamic cache C_n with n-gram size of n, the individual cache miss ratio for the isolated glue $g_{(n+1)}$ is given by:

$$mr_{(RO+RW)}\left(C_n, g_{(n+1)}\right) = \frac{\left|D_{n_{in}D_{(n+1)}}\right| - \left|FA\left(n, g_{n+1}, h_{RO_{C_n}}\right)\right|}{2 \times \left|D_{(n+1)}\right|} \tag{7.50}$$

Assuming that the included dynamic caches C_{RW_i} have infinite capacity we can calculate the global miss ratio of the static plus dynamic cache system (C_{1+2+3}), for the combined glue g_{234} as:

$$\begin{aligned} mr_{(RO+RW)}\left(C_{1+2+3}, g_{234}\right) = & \frac{\left|D_{1_{in}D_{234}}\right| - \left|FA\left(1, g_{234}, h_{RO_{C_1}}\right)\right|}{2\times|D_2|+4\times|D_3|+6\times|D_4|} + \\ & \frac{\left|D_{2_{in}D_{34}}\right| - \left|FA\left(2, g_{234}, h_{RO_{C_2}}\right)\right|}{2\times|D_2|+4\times|D_3|+6\times|D_4|} + \\ & \frac{\left|D_{3_{in}D_4}\right| - \left|FA\left(3, g_{234}, h_{RO_{C_3}}\right)\right|}{2\times|D_2|+4\times|D_3|+6\times|D_4|} \end{aligned} \tag{7.51}$$

where D_{234} represents the union of the tables of distinct bigrams (D_2), trigrams (D_3) and tetragrams (D_4); D_{34} represents the union of the tables of distinct trigrams (D_3) and tetragrams (D_4); and $D_{1_{in}D_{234}}$, $D_{2_{in}D_{34}}$ and $D_{3_{in}D_4}$ represent, respectively, the corresponding distinct subunigrams, subbigrams and subtrigrams in the above tables.

7.2.2 Single Machine Case ($K = 1$)

Static plus Dynamic Cache Size. Figure 7.28 shows the cardinalities of the individual static cache FAsets and the sizes of the corresponding individual dynamic caches, for each n-gram static plus dynamic cache of size n, $1 \leq n \leq 3$, for the glue calculation g_{234} and a fixed *corpus* size of 466 Mw, when the corresponding individual static hit ratio $h_{RO_{C_n}}$ varies from 5% to 95%. The sets of distinct n-grams D_1, D_2, D_3 and D_4 were calculated by running phase one of the LocalMaxs Global method to generate the n-gram tables. Then the static cache FAsets were calculated by varying the static hit ratio, for each individual cache (C_1, C_2 and C_3), using Definitions 7.1 and 7.2. The corresponding values $|D_i| - \left|FA\left(i, g, h_{RO_{C_i}}\right)\right|$ were obtained to model the total number of first occurrence misses of the dynamic cache, i.e., its size in terms of n-gram entries.

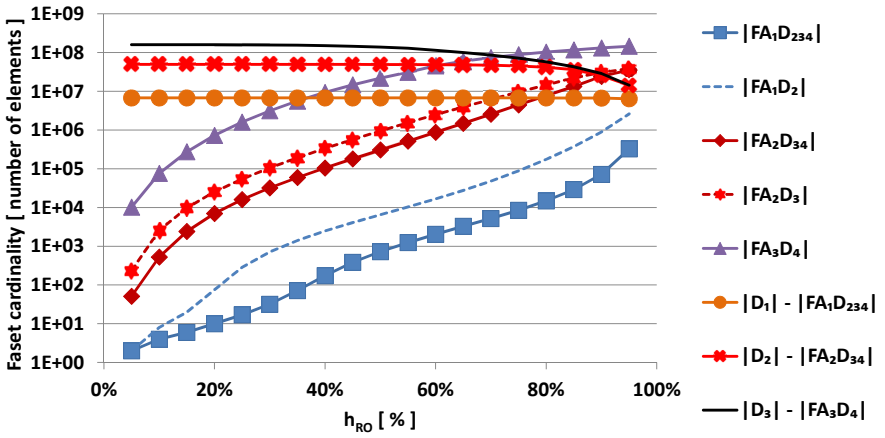


Figure 7.28: FAset cardinalities, dynamic cache sizes involved in the global miss ratio for a *corpus* size of 466 Mw, and glues g_2, g_3, g_4 and g_{234}.

In Figure 7.28 we use a shortened notation for the FAsets. The X axis represents the hit ratio of the individual static caches ($h_{RO_{C_n}}$) and the Y axis represents the FAset cardinalities of the following sets: FA_1D_{234}=$FA\left(1, g_{234}, h_{RO_{C_1}}\right)$; FA_2D_{34}=$FA\left(2, g_{234}, h_{RO_{C_2}}\right)$; FA_3D_4=$FA\left(3, g_{234}, h_{RO_{C_3}}\right)$; FA_1D_2=$FA\left(1, g_2, h_{RO_{C_1}}\right)$; and FA_2D_3=$FA\left(2, g_3, h_{RO_{C_2}}\right)$; and the sizes of each dynamic cache C_{RW_n} ($|D_1|-|FA_1D_{234}|$; $|D_2|-|FA_2D_{34}|$; and $|D_3|-|FA_3D_4|$).

For each given static cache and a fixed static hit ratio, the FAset size when calculating a combined glue g_{234} is lower than the FAset size when calculating an isolated glue. For instance, in the case of g_2 for cache C_1 we have $\left|FA\left(1, g_2, h_{RO_{C_1}}\right)\right| \geq \left|FA\left(1, g_{234}, h_{RO_{C_1}}\right)\right|$ for all considered values of $h_{RO_{C_1}}$; in the case of g_3 for cache C_2 we have $\left|FA\left(2, g_3, h_{RO_{C_2}}\right)\right| \geq \left|FA\left(2, g_{234}, h_{RO_{C_2}}\right)\right|$ for all considered values of $h_{RO_{C_2}}$.

The dynamic cache size for unigrams varies only slightly for all the values of $h_{RO_{C_1}}$. Indeed, the relative deviation between the lowest and highest values of the dynamic cache size for unigrams and glue g_{234} in the range $5\% \leq h_{RO_{C_1}} \leq 95\%$ is:

$$\frac{\left(|D_1|-|FA_1D_{234}|_{\left(h_{RO_{C_1}}=5\%\right)}\right)-\left(|D_1|-|FA_1D_{234}|_{\left(h_{RO_{C_1}}=95\%\right)}\right)}{|D_1|-|FA_1D_{234}|_{\left(h_{RO_{C_1}}=5\%\right)}} \approx 5\% \qquad (7.52)$$

The size of the dynamic cache C_2 shows a relative deviation between its lowest and highest values that stays within 5% in the range $5\% \leq h_{RO_{C_2}} \leq 70\%$. For the dynamic cache C_3, its size has a relative deviation between its lowest and highest values that stays within 6% in the range $5\% \leq h_{RO_{C_3}} \leq 40\%$. In both of these two cases this behavior is related to the steeper slopes (Figure 7.28) of the corresponding FAset sizes for values of $h_{RO_{C_2}} > 70\%$ and $h_{RO_{C_3}} > 40\%$, corresponding to the need of including more and more low frequency n-grams into the FAset.

Global Miss Ratio of the Static plus Dynamic Cache System. Figures 7.29 to 7.31 show the global miss ratio of the cache system C_{1+2+3} (denoted in the Y axis by mr_{RO+RW}) corresponding to Figure 7.27, obtained from expression (7.51) for a *corpus* of 466 Mw for the glue g_{234}. The values of $h_{RO_{C_1}}$ and $h_{RO_{C_2}}$ (the horizontal axes) vary from 5% up to 100% in steps of 5%. Regarding the value of $h_{RO_{C_3}}$ three scenarios were considered: $h_{RO_{C_3}} = 30\%$, $h_{RO_{C_3}} = 60\%$ and $h_{RO_{C_3}} = 90\%$.

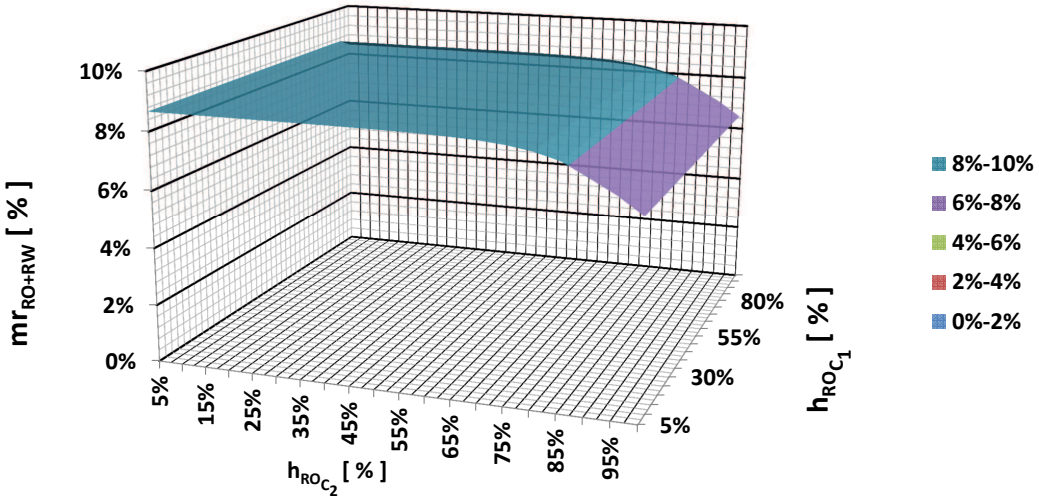


Figure 7.29: Global miss ratio for a *corpus* of 466 Mw and glue g_{234} — $h_{RO_{C_3}} = 30\%$.

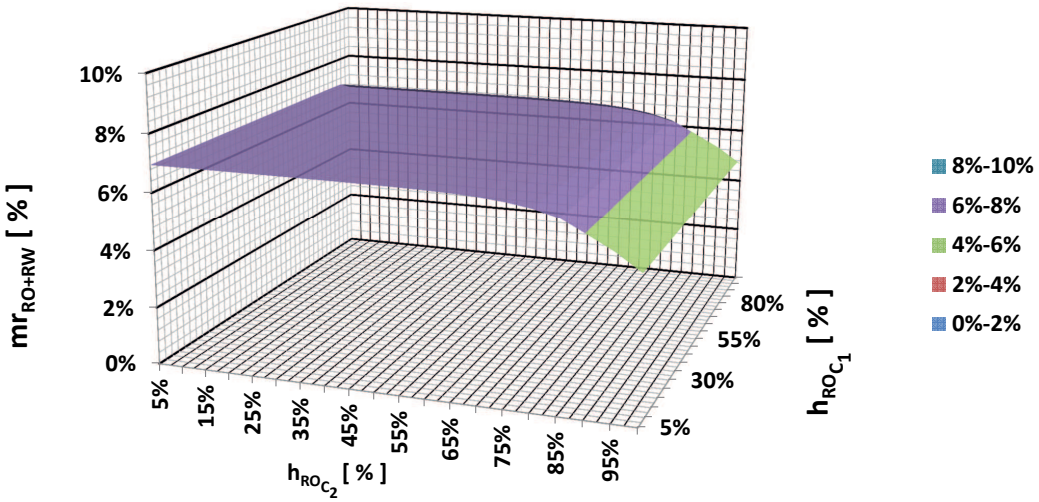


Figure 7.30: Global miss ratio for a *corpus* of 466 Mw and glue g_{234} — $h_{RO_{C_3}} = 60\%$.

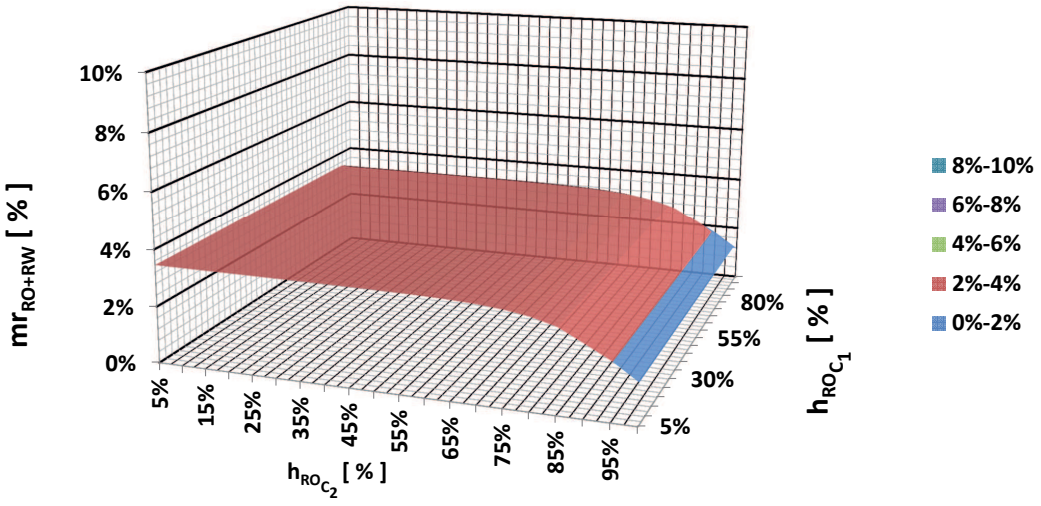


Figure 7.31: Global miss ratio for a *corpus* of 466 Mw and glue g_{234} — $h_{RO_{C_3}}$ = 90%.

The following conclusions can be taken:

- Due to the almost constancy of the cache miss penalty of each individual static plus dynamic cache C_1, C_2 and C_3 (Figure 7.28) in the above specified ranges of static hit ratios and as we are considering a fixed *corpus* size, the corresponding individual cache miss ratios are also almost constant; So, it is expected that the global miss ratio of the cache system C_{1+2+3} exhibits a similar behavior when the static hit ratios ($h_{RO_{C_1}}$, $h_{RO_{C_2}}$ and $h_{RO_{C_3}}$) vary;

- For each of the above figures with a fixed $h_{RO_{C_3}}$, the global miss ratio stays approximately unchanged as long as $h_{RO_{C_3}}$ is below 70% for all values of $h_{RO_{C_1}}$; When we go from $h_{RO_{C_3}}$=30% (Figure 7.29) to $h_{RO_{C_3}}$=60% (Figure 7.30) there is a visible reduction in the global miss ratio due to the influence of the reduction in the dynamic cache C_3 miss ratio, where $h_{RO_{C_3}}$ is already above the limit of 40% of its constancy range. This reduction in the global miss ratio is more significant when we go from $h_{RO_{C_3}}$=60% (Figure 7.30) to $h_{RO_{C_3}}$=90% (Figure 7.31).

7.2.3 Multiple Machines Case ($K > 1$)

In section 7.2.3.1 we describe the execution environment and experimental scenarios used to support the analysis of the static plus dynamic cache system in the multiple machines case. The analysis of the per machine individual cache miss ratio in the case of isolated glues and combined glues is presented in section 7.2.3.2 and 7.2.3.3. Section 7.2.3.4 presents an analysis regarding the per machine global miss ratio in the case of the combined glue g_{234}.

7.2.3.1 Execution Environment and Experimental Scenarios

In order to evaluate experimentally the behavior of the static and dynamic cache, we performed several runs of the Global method in phases one and two, under the Amazon EC2 public cloud environment. We considered 4 distinct *corpus* sizes (Table 7.5) with different numbers of machines K, ranging from 3 up to 18. For each case the isolated and combined glues were evaluated, and the numbers of hits and misses of each individual cache were measured, to obtain the corresponding hit and miss ratios.

Table 7.5: Number of distinct n-grams for the *corpora* analyzed. *Corpus* size is in million words.

***Corpus* [Mw]**	$\|D_1\|$	$\|D_2\|$	$\|D_3\|$	$\|D_4\|$
64	1 591 515	11 855 860	31 413 238	47 477 297
128	1 747 440	12 147 602	37 795 214	59 821 749
256	4 316 177	33 195 356	100 792 182	168 630 935
511	7 129 215	54 834 546	177 770 692	313 683 177

In all cases the hit ratio of each individual static cache was configured as follows, where the FAsets were calculated according to the approximation discussed in section 7.1.4.4, on page 202:

- For cache C_1 we imposed a base value for the static cache hit ratio $h_{RO_{C_1}}(g_2) = 80\%$ targeted at the calculation of the isolated glue g_2, and we obtained the corresponding FAset $\left(FA_1, g_2, h_{RO_{C_1}}\right)$. This same FAset (once loaded into the $C_{1_{RO}}$ cache) was reused for the calculation of the glue g_3 and also for the glue g_4;

- For cache C_1 the dynamic part ($C_{1_{RW}}$) was initially empty (cold-start) for the calculation of g_2, but it was left warmed-up for the calculation of the following glue g_3, which contributed to an incremental warming-up of the $C_{1_{RW}}$ cache that was then used for the calculation of glue g_4;

- For cache C_2 we imposed a base value for the static cache hit ratio $h_{RO_{C_2}}(g_3) = 60\%$ targeted at the calculation of the isolated glue g_3, and we obtained the corresponding FAset $\left(FA_2, g_3, h_{RO_{C_2}}\right)$. This same FAset (once loaded into the $C_{2_{RO}}$ cache) was reused for the calculation of the glue g_4;

- For cache C_2 the dynamic part ($C_{2_{RW}}$) was initially empty (cold-start) for the calculation of g_3, but it was left warmed-up for the calculation of the following glue g_4;

- For cache C_3 we imposed a base value for the static cache hit ratio $h_{RO_{C_3}}(g_4) = 30\%$ targeted at the calculation of the isolated glue g_3, and we obtained the corresponding FAset $\left(FA_3, g_4, h_{RO_{C_3}}\right)$, that was loaded in $C_{3_{RO}}$ cache before calculating glue g_4.

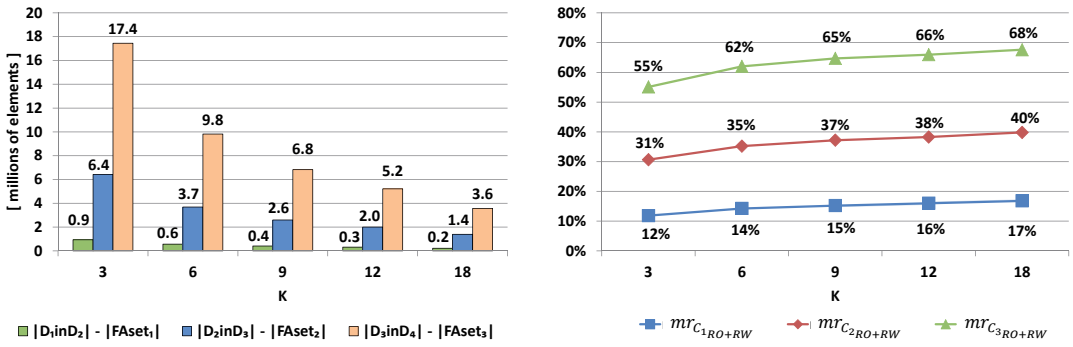
7.2.3.2 Analysis of the Individual Miss Ratios for Isolated Glues

Figure 7.32 shows the per machine numbers of misses and the individual miss ratios of each static plus dynamic cache C_1, C_2 and C_3 when the number of machines (K) varies from 3 up to 18, when considered separately, respectively, for the evaluation of the isolated glues g_2, g_3 and g_4. We considered the average values of the number of misses and miss ratios over the K machines. The per machine static hit ratios of the individual caches were fixed using the values $h_{RO_{C_1}} = 80\%$, $h_{RO_{C_2}} = 60\%$, and $h_{RO_{C_3}} = 30\%$. The *corpus* size ranges from 64 Mw up to 511 Mw, respectively in Figure 7.32-a) to 7.32-d). In the figure we use a shortened notation for the FAsets: $FAset_1{=}FA\left(1, g_2, h_{RO_{C_1}}\right)$, $FAset_2{=}FA\left(2, g_3, h_{RO_{C_2}}\right)$ and $FAset_3{=}FA\left(3, g_4, h_{RO_{C_3}}\right)$. These FAset values can be used to configure the size of the dynamic cache in order to obtain an infinite cache behavior.

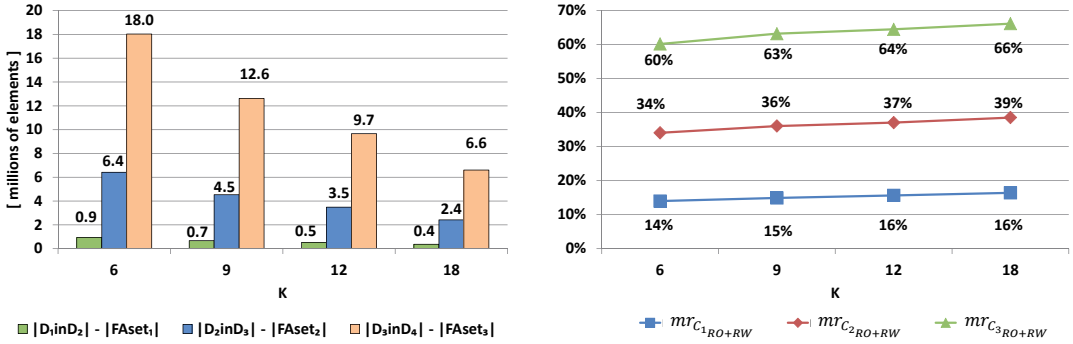
From the figure we can observe:

- The per machine individual miss ratio tends to increase with K, with a similar trend as the individual miss ratio of a single dynamic cache (as presented in chapter 6). This is due to the fact that we keep the static hit ratio value fixed ($h_{RO_{C_i}}$) and $mr_{(RO+RW)_i} = mr_{RO_i} \times mr_{RW_i}$;

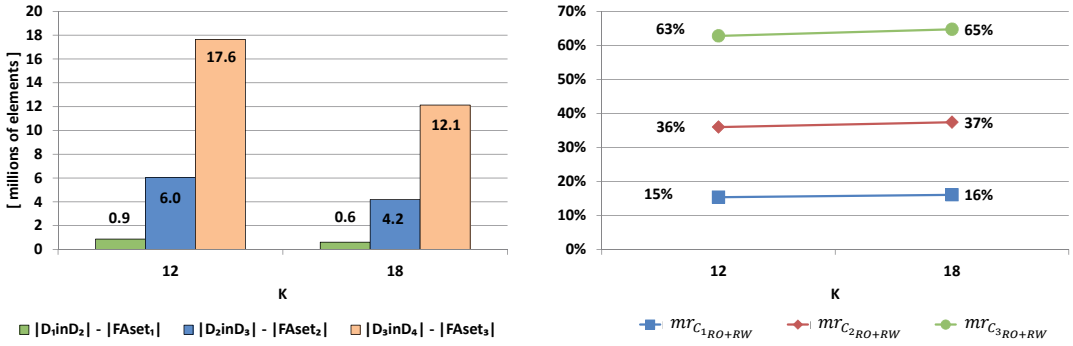
- However, the rate of growth of the per machine individual miss ratio for the static plus dynamic cache when K increases, is lower than the corresponding rate of growth of the miss ratio for a single dynamic cache. First, we recall that the reason why the per machine miss ratio of a dynamic cache grows with K is due to the fact that the n-grams that have higher frequencies of occurrences in the *corpus* tend to be referenced from multiple partitions of the n-gram tables, so the number of those n-grams per machine is proportional to $1/K^b$, $0 < b < 1$, instead of $1/K$. Then, in the case of the static plus dynamic cache, the above mentioned n-grams tend to belong to the FAset, thus they are filtered out from the static cache output miss stream which is forwarded to the downstream dynamic cache. As a consequence, this latter stream exhibits a lower average n-gram repetition factor than the input stream in the case of a single dynamic cache, so it is less sensitive to the increase of the number of machines.



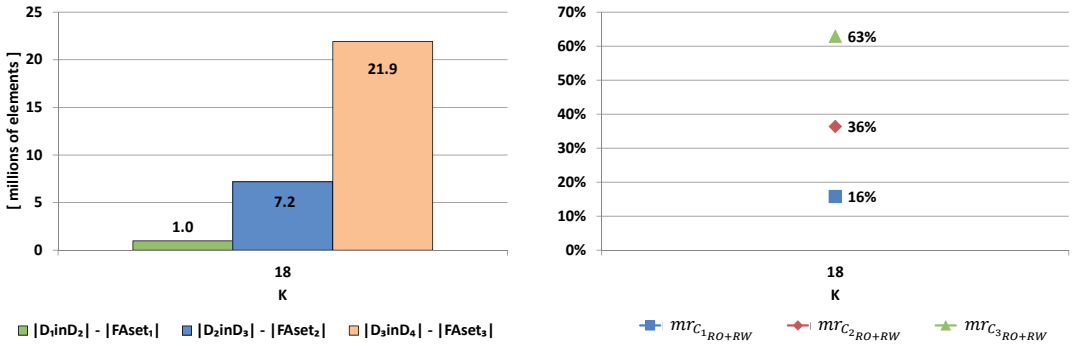
a) Number of misses and individual miss ratios for C_1, C_2 and C_3 with a corpus size of 64 Mw



b) Number of misses and individual miss ratios for C_1, C_2 and C_3 with a corpus size of 128 Mw



c) Number of misses and individual miss ratios for C_1, C_2 and C_3 with a corpus size of 256 Mw



d) Number of misses and individual miss ratios for C_1, C_2 and C_3 with a corpus size of 511 Mw

Figure 7.32: Per machine numbers of misses and individual miss ratios for $C_{1_{(RO+RW)}}$, $C_{2_{(RO+RW)}}$ and $C_{3_{(RO+RW)}}$ *versus* K with $h_{RO_{C_1}} = 80\%$, $h_{RO_{C_2}} = 60\%$, and $h_{RO_{C_3}} = 30\%$.

7.2.3.3 Analysis of the Individual Miss Ratios for Combined Glues

For each *corpus* size a Table like 7.6 shows the obtained per machine FAset sizes for different numbers of machines (K). Note that as explained above, each static cache $C_{1_{RO}}$, $C_{2_{RO}}$ and $C_{3_{RO}}$ was loaded with the same FAset contents, respectively, $\left(FA_1, g_2 h_{RO_{C1}}\right)$, $\left(FA_2, g_3, h_{RO_{C2}}\right)$ and $\left(FA_3, g_4, h_{RO_{C3}}\right)$ for all the cases, where the values of FAset considered were determined according to the *corpus* sizes, in order to ensure a fixed static cache hit ratio, which was fixed in all experiments: $h_{RO_{C_1}}(g_2) = 80\%$, $h_{RO_{C_2}}(g_3) = 60\%$, and $h_{RO_{C_3}}(g_4) = 30\%$.

Table 7.6: Per machine FAset sizes (in number of n-grams) *versus* K for a *corpus* of 64 Mw.

K	$\left\|\left(FA_1, g_2, h_{RO_{C_1}}\right)\right\|$	$\left\|\left(FA_2, g_3, h_{RO_{C_2}}\right)\right\|$	$\left\|\left(FA_3, g_4, h_{RO_{C_3}}\right)\right\|$
3	77 523	814 412	892 206
6	64 651	624 891	619 060
9	56 894	508 873	499 995
12	53 176	425 241	413 742
18	46 209	321 885	307 170

Similar tables for other *corpus* sizes, namely 128, 256 and 511 Mw are presented in Appendix C.

Each dynamic cache was warmed-up by the first glue calculation, i.e., g_2 for $C_{1_{RW}}$ and g_3 for $C_{2_{RW}}$.

Cache $C_{1_{RO}}$. The value of $h_{RO_{C_1}}(g_2)$=80% is enforced by the associated FAset $(FA_1, g_2, h_{RO_{C_1}})$, which was used to evaluate glue g_2. The remaining glues g_3 and g_4 were evaluated using the cache $C_{1_{RO}}$ loaded with the same FAset $(FA_1, g_2, h_{RO_{C_1}})$. Then, the obtained values $h_{RO_{C_1}}(g_3)$ and $h_{RO_{C_1}}(g_4)$ were experimentally measured.

Cache $C_{2_{RO}}$. The value of $h_{RO_{C_2}}(g_3)$=60% is enforced by the FAset $(FA_2, g_3, h_{RO_{C_2}})$, which was used to calculate glue g_3. The remaining glue g_4 was calculated using the cache $C_{2_{RO}}$ loaded with the same FAset $(FA_2, g_3, h_{RO_{C_2}})$. The obtained value $h_{RO_{C_2}}(g_4)$ was experimentally measured.

Cache $C_{3_{RO}}$. The value of $h_{RO_{C_3}}(g_4)$=30% is enforced by the FAset $(FA_3, g_4, h_{RO_{C_3}})$, which was used to calculate glue g_4.

Observed Values. For each *corpus* size and glue calculation the observed numbers of hits from each static and dynamic cache were collected, and were divided by the total numbers of input references ($\left|all_{glue_g Ref_{i\text{-}gram}}\right|$), as shown in Table 7.7, where $hits_{RO_{C_i}}$ denotes the total number of hits observed from the static cache C_{RO_i}; and $hits_{RW_{C_i}}$ denotes the total number of hits observed from the dynamic cache C_{RW_i}. Then, the total number of hits in

each static plus dynamic cache was obtained as $hits_{RO_{C_i}} + hits_{RW_{C_i}}$ allowing to calculate the corresponding total number of misses ($\left|misses_{(RO+RW)_{C_1}}\right|$), which led to the values of the individual miss ratios of each static plus dynamic cache, as shown in the last three lines of the table.

Table 7.7 shows the average of the measured values over the considered number of machines ($K = 18$) for a *corpus* size of 511 Mw.

Table 7.7: Average number of hit and miss ratio values in percentage (%) isolated glues and a *corpus* of 511 Mw with 18 machines.

Set	Hit and miss ratio percentages	g_2	g_3	g_4
A	$hits_{RO_{C_1}} / \left\|all_{glue_g Ref_{1\text{–}gram}}\right\|$	80.00	84.80	90.02
	$hits_{RO_{C_2}} / \left\|all_{glue_g Ref_{2\text{–}gram}}\right\|$		60.00	54.98
	$hits_{RO_{C_3}} / \left\|all_{glue_g Ref_{3\text{–}gram}}\right\|$			30.00
B	$hits_{RW_{C_1}} / \left\|all_{glue_g Ref_{1\text{–}gram}}\right\|$	4.21	10.42	7.88
	$hits_{RW_{C_2}} / \left\|all_{glue_g Ref_{2\text{–}gram}}\right\|$		3.54	20.58
	$hits_{RW_{C_3}} / \left\|all_{glue_g Ref_{3\text{–}gram}}\right\|$			5.70
C	$\left\|misses_{(RO+RW)_{C_1}}\right\| / \left\|all_{glue_g Ref_{1\text{–}gram}}\right\|$	15.79	4.78	2.10
	$\left\|misses_{(RO+RW)_{C_2}}\right\| / \left\|all_{glue_g Ref_{2\text{–}gram}}\right\|$		36.46	24.44
	$\left\|misses_{(RO+RW)_{C_3}}\right\| / \left\|all_{glue_g Ref_{3\text{–}gram}}\right\|$			64.30

- The column labeled "g_2" corresponds to the case of calculating the isolated glue g_2 with the static cache $C_{1_{RO}}$ loaded with the FAset $\left(FA_1, g_2, h_{RO_{C_1}}\right)$ and the dynamic cache $C_{1_{RW}}$ with cold-start;
- The column labeled "g_3" corresponds to the case of calculating the isolated glue g_3 with the static cache $C_{1_{RO}}$ loaded with the same FAset as previously, the static cache $C_{2_{RO}}$ loaded with the FAset $\left(FA_2, g_3, h_{RO_{C_2}}\right)$, the dynamic cache $C_{1_{RW}}$ warmed-up by the glue calculation g_2, and the dynamic cache $C_{2_{RW}}$ with cold-start;
- The column labeled "g_4" corresponds to the case of calculating the isolated glue g_4 with the static caches $C_{1_{RO}}$ and $C_{2_{RO}}$ loaded with the same FAsets as previously, the static cache $C_{3_{RO}}$ loaded with the FAset $\left(FA_3, g_4, h_{RO_{C_3}}\right)$, the dynamic cache $C_{1_{RW}}$ warmed-up by the glue calculation g_2 followed by the glue calculation g_3, the dynamic cache $C_{2_{RW}}$ warmed-up by the glue calculation g_3, and the dynamic cache $C_{3_{RW}}$ with cold-start.

For ease of reference, the lines in Table 7.7 are labeled in three sets: **A**, **B** and **C**. The first three lines (set **A**) present the average values of the individual static cache hit ratios ($h_{RO_{C_i}}$). The next three lines (set **B**) present the average values of percentage of observed hits out of the dynamic cache with respect to the number of references in the input of the cache system (i.e., $2 \times |D_2|/K$ for the column g_2, $2 \times |D_3|/K$ for the column g_3, and $2 \times |D_4|/K$ for the column g_4). The last three lines (set **C**) present the average values of the individual miss ratio ($mr_{(RO+RW)_{C_i}}$) of the static plus dynamic cache for each case. The following two comments are in order:

(1) Per Machine Static Cache Hit Ratios. For cache C_1 we observe that, for this *corpus* size and number of machines, the measured average individual static cache hit ratios for the glues g_3 and g_4 are higher than the base value of 80% imposed for the glue g_2. This is due to the much higher accumulated concentration of frequencies of occurrences of unigrams which are members of the considered FAset, for example, in the case of g_3 *versus* g_2 when comparing their contribution to the numerator in expression (7.54) with respect to the numerator in expression (7.53), in such a way that it even compensates the increase observed in the denominators.

$$h_{RO_{C_1}}(g_2) = \frac{\sum\limits_{ng \in \left(FA_1, g_2, h_{RO_{C_1}}\right)} freq_{inD_1 inD_2(j)}(ng)}{2 \times \frac{|D_2|}{K}} \tag{7.53}$$

$$h_{RO_{C_1}}(g_3) = \frac{\sum\limits_{ng \in \left(FA_1, g_2, h_{RO_{C_1}}\right)} freq_{inD_1 inD_3(j)}(ng)}{2 \times \frac{|D_3|}{K}} \tag{7.54}$$

However for cache C_2, for this *corpus* size and number of machines, the above mentioned increase in the accumulation of frequencies in the numerators of the corresponding static cache hit ratio expressions (7.55 and 7.56), in the case of bigrams, is not enough to compensate the increase in the denominators. So for this *corpus* size, the C_2 static cache hit ratio did not increase when going from g_3 to g_4.

$$h_{RO_{C_2}}(g_3) = \frac{\sum\limits_{ng \in \left(FA_2, g_3, h_{RO_{C_2}}\right)} freq_{inD_2 inD_3(j)}(ng)}{2 \times \frac{|D_3|}{K}} \tag{7.55}$$

$$h_{RO_{C_2}}(g_4) = \frac{\sum\limits_{ng \in \left(FA_2, g_3, h_{RO_{C_2}}\right)} freq_{inD_2 inD_4(j)}(ng)}{2 \times \frac{|D_4|}{K}} \tag{7.56}$$

(2) Per Machine Static plus Dynamic Cache Miss Ratios. The entries of the set **C** in Table 7.7 represent the individual miss ratio of the static plus dynamic cache $C_{(RO+RW)_i}$,

with $1 \leq i \leq 3$, which was calculated from the measured values according to the following expression:

$$\left|mr_{(RO+RW)_{C_i}}\right| = \frac{\left|all_{glue_g Ref_{i-gram}}(j)\right| - hits_i}{\left|all_{glue_g Ref_{i-gram}}(j)\right|} = \left(1 - h_{(RO+RW)_{C_i}}\right) \tag{7.57}$$

where for cache $C_{(RO+RW)_i}$, $\left|all_{glue_g Ref_{i-gram}}(j)\right|$ is the total number of input references of n-grams of size i; $hits_i = hits_{RO_{C_i}} + hits_{RW_{C_i}}$ is the total number of hits observed on the cache $C_{(RO+RW)_i}$ (static plus dynamic); and $h_{(RO+RW)_{C_i}}$ is equal to the sum of the parcels "$hits_{RO_{C_i}} / \left|all_{glue_g Ref_{i-gram}}\right|$" and "$hits_{RW_{C_i}} / \left|all_{glue_g Ref_{i-gram}}\right|$".

Note that, when going from g_2 to g_3 and from g_3 to g_4, the individual hit ratio of the static plus dynamic cache $C_{(RO+RW)_1}$ increases, i.e., it goes from 84.21% = 80.00% + 4.21% for the "g_2" case, to the value of 95.22% = 84.80% + 10.42% for the "g_3" case, and to the value 97.90% = 90.02% + 7.88% for the "g_4" case. This happens even if the "$hits_{RW_{C_i}} / \left|all_{glue_g Ref_{i-gram}}\right|$" decreases when going from "g_3" to "g_4", but this reduction in the proportion of hits is related to the previous increase in the proportion of hits observed in the static cache in the same case of going from "g_3" to "g_4". The observed misses in cache $C_{(RO+RW)_1}$ in this case are related to the unigrams which were not captured either by the static cache or by the warmed-up dynamic cache, due to being most likely among the most rarely occurring unigrams, and as such they typically have a minor contribution to the efficiency of the cache concerning the number of generated hits per element. Note that the values presented above reflect the ones presented in Table 7.7. An identical analysis can be made for cache $C_{(RO+RW)_2}$.

Similar tables for other *corpus* sizes, namely 64, 128 and 256 Mw are presented in Appendix C.

Combined glue g_{234}. Under the above assumptions the expression for the per machine individual hit ratio ($h_{RO_{C_1}}(g_{234})$) of the static cache C_{RO_1} for the combined glue g_{234}, can be calculated as:

$$\begin{aligned} h_{RO_{C_1}}(g_{234}) = {} & h_{RO_{C_1}}(g_2) \times \tfrac{2\times|D_2|}{2\times|D_2|+2\times|D_3|+2\times|D_4|} + \\ & h_{RO_{C_1}}(g_3) \times \tfrac{2\times|D_3|}{2\times|D_2|+2\times|D_3|+2\times|D_4|} + \\ & h_{RO_{C_1}}(g_4) \times \tfrac{2\times|D_4|}{2\times|D_2|+2\times|D_3|+2\times|D_4|} \end{aligned} \tag{7.58}$$

The per machine individual hit ratio ($h_{RO_{C_2}}(g_{234})$) of the static cache C_{RO_2} for the combined glue g_{234}, is given by:

$$h_{RO_{C_2}}(g_{234}) = h_{RO_{C_2}}(g_3) \times \frac{2\times|D_3|}{2\times|D_3|+2\times|D_4|} + h_{RO_{C_2}}(g_4) \times \frac{2\times|D_4|}{2\times|D_3|+2\times|D_4|} \tag{7.59}$$

Concerning the static cache C_{RO_3}, we have imposed a value of 30% as explained:

$$h_{RO_{C_3}}(g_{234}) = h_{RO_{C_3}}(g_4) = 30\% \tag{7.60}$$

By using the above expressions, the corresponding values of the hit and miss ratios of each static plus dynamic cache for the above case of a *corpus* size of 511 Mw and 18 machines are shown in Table 7.8.

Table 7.8: Per machine average number of hit and miss ratio values in percentage (%) for combined glue g_{234} and a *corpus* of 511 Mw and 18 machines.

Hit and miss ratio percentages	$\mathbf{g_{234}}$
$h_{RO_{C_1}}$	87.32
$h_{RO_{C_2}}$	56.80
$h_{RO_{C_3}}$	30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g} Ref_{1\text{–}gram} \rvert$	8.34
$hits_{RW_{C_2}} / \lvert all_{glue_g} Ref_{2\text{–}gram} \rvert$	14.42
$hits_{RW_{C_3}} / \lvert all_{glue_g} Ref_{3\text{–}gram} \rvert$	5.70
$mr_{(RO+RW)}(C_1, g_{234})$	4.35
$mr_{(RO+RW)}(C_2, g_{234})$	28.79
$mr_{(RO+RW)}(C_3, g_{234})$	64.30

7.2.3.4 Analysis of the Per Machine Global Miss Ratio

The global miss ratio of a cache system ($C_{(RO+RW)_{1+2+3}}$) with individual static plus dynamic caches is defined by expression (7.51), on page 217. In the multiple machines case ($K > 1$), the expression still applies to each machine j: $1 \cdots K$, and all the elements in that expression become indexed with j.

Figure 7.33 shows the global view of the miss related contributions of each individual static plus dynamic cache to the global miss ratio of the cache system for the calculation of the combined glue g_{234}, following the same strategy regarding the FAsets as in section 7.2.3.3.

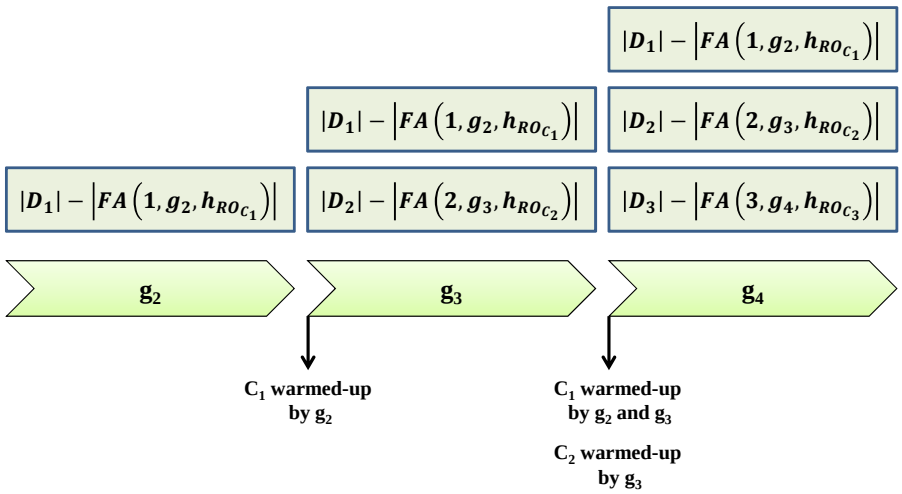


Figure 7.33: Contribution of each individual static plus dynamic cache to the global miss ratio of the cache system $C_{(RO+RW)_{1+2+3}}$ for glue g_{234}.

From the above Table 7.7, which only shows the cache miss ratios of each composite static plus dynamic cache for isolated glues, it is possible to derive the global miss ratio ($mr_{(Ro+RW)_{(C_{1+2+3})}}(g_{234})$) of the entire cache system composed of C_1, C_2 and C_3, when used for the calculation of the combined glue g_{234}, under the same assumptions, i.e., using the same FAsets as defined above for the static caches, and performing the glue calculations in the sequence g_2 followed by g_3, followed by g_4, with the incremental warming-up of the C_1 and C_2 dynamic caches.

$$\left|Misses_{g_{234}}(j)\right| = \left|Misses_{C_1}(j)\right| + \left|Misses_{C_2}(j)\right| + \left|Misses_{C_3}(j)\right| = \left|all_{g_{234}}Ref\,(j)\right| \times mr_{C_{g_{234}}}(j) \quad (7.61)$$

$$\begin{aligned}\left|Misses_{C_1}(j)\right| &= \left|Misses_{C_{1_{g_2}}}(j)\right| + \left|Misses_{C_{1_{g_3}}}(j)\right| + \left|Misses_{C_{1_{g_4}}}(j)\right| \\ &= 2 \times |D_2\,(j)| \times mr_{C_{1_{g_2}}}(j) + 2 \times |D_3\,(j)| \times mr_{C_{1_{g_3}}}(j) + 2 \times |D_4\,(j)| \times mr_{C_{1_{g_4}}}(j)\end{aligned} \quad (7.62)$$

where $mr_{C_{1_{g_2}}}(j) = mr_{(RO+RW)_{C_{1_{g_2}}}}(j)$, $mr_{C_{1_{g_3}}}(j) = mr_{(RO+RW)_{C_{1_{g_3}}}}(j)$ and $mr_{C_{1_{g_4}}}(j) = mr_{(RO+RW)_{C_{1_{g_4}}}}(j)$ are the individual miss ratios of the cache C_1 ($C_{1_{(RO+RW)}}$), respectively, for glues g_2, g_3 and g_4.

$$\begin{aligned}\left|Misses_{C_2}(j)\right| &= \left|Misses_{C_{2_{g_3}}}(j)\right| + \left|Misses_{C_{2_{g_4}}}(j)\right| \\ &= 2 \times |D_3\,(j)| \times mr_{C_{2_{g_3}}}(j) + 2 \times |D_4\,(j)| \times mr_{C_{2_{g_4}}}(j)\end{aligned} \quad (7.63)$$

where $mr_{C_{2_{g_3}}}(j) = mr_{(RO+RW)_{C_{2_{g_3}}}}(j)$ and $mr_{C_{2_{g_4}}}(j) = mr_{(RO+RW)_{C_{2_{g_4}}}}(j)$ are the individual miss ratios of the cache C_2 ($C_{2_{(RO+RW)}}$), respectively, for glues g_3 and g_4.

$$\left|Misses_{C_3}(j)\right| = \left|Misses_{C_{3_{g_4}}}(j)\right| = 2 \times |D_4\,(j)| \times mr_{C_{3_{g_4}}}(j) \quad (7.64)$$

where $mr_{C_{3_{g4}}}(j) = mr_{(RO+RW)_{C_{3_{g4}}}}(j)$ is the individual miss ratio of the cache C_3 ($C_{3_{(RO+RW)}}$) for glue g_4.

$$mr(g_{234}) = \frac{2\times|D_2(j)|}{2\times|D_2(j)|+4\times|D_3(j)|+6\times|D_4(j)|} \times mr_{C_{1_{g_2}}} + \frac{2\times|D_3(j)|}{2\times|D_2(j)|+4\times|D_3(j)|+6\times|D_4(j)|} \times \left(mr_{C_{1_{g3}}} + mr_{C_{2_{g3}}}\right) + \frac{2\times|D_4(j)|}{2\times|D_2(j)|+4\times|D_3(j)|+6\times|D_4(j)|} \times \left(mr_{C_{1_{g4}}} + mr_{C_{2_{g4}}} + mr_{C_{3_{g4}}}\right) \quad (7.65)$$

Table 7.9 and Figure 7.34 show the evolution of the per machine global miss ratio for the entire cache system $C_{(RO+RW)_{(1+2+3)}}$ when calculating the glue g_{234}, for the *corpus* presented in Table 7.5 when the number of machines varies from 3 up to 18.

Table 7.9: Per machine global miss ratio of the $C_{(RO+RW)_{(1+2+3)}}$ cache system for glue g_{234} for different *corpus* sizes *versus* K. Miss ratio in percentage (%).

***Corpus* [Mw]**	$K = 3$	$K = 6$	$K = 9$	$K = 12$	$K = 18$
64	11.11	12.29	13.16	13.81	14.76
128		12.37	13.05	13.47	14.19
256				13.21	13.81
511					13.58

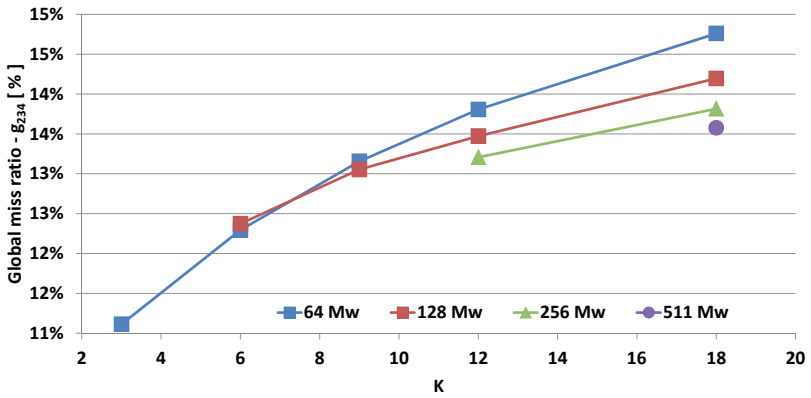


Figure 7.34: Per machine global miss ratio of the C_{1+2+3} cache system, composed of individual static plus dynamic caches, for glue g_{234} for different *corpus* sizes *versus* K. The Y axis represents the average values of the global miss ratio for the C_{1+2+3} cache system per machine and the X axis represents the number of machines.

The following conclusions can be taken:

- For each *corpus* size the per machine global miss ratio of the cache system increases when the number of machines increases, but the amplitude of its variation is small, around 1% – 4%;
- For each fixed K, the per machine global miss ratio decreases with the *corpus* size;

✦ The average value of the per machine global miss ratio for g_{234} over the considered *corpus* sizes and range of machine numbers is around 13% – 14%.

7.3 Real Execution Times

In this section we present the experimental results concerning the execution of phase two of the LocalMaxs Global method using a cache system composed of multiple individual static plus dynamic caches. Since the usage of an n-gram cache system only has impact on the execution of phase two, in this section we just present the execution times for phase two. Section 7.3.1 presents the overheads derived from the building and loading of the FAset in each machine, and section 7.3.2 presents the total execution time for phase two and the relative speedup and sizeup.

7.3.1 FAsets Building and Prefetching Times

FAset Construction. When using a static cache the building of the FAset can be performed as soon as the n-gram counting and final aggregation has ended, for each n-gram size, in phase one. This is illustrated in Figure 7.35.

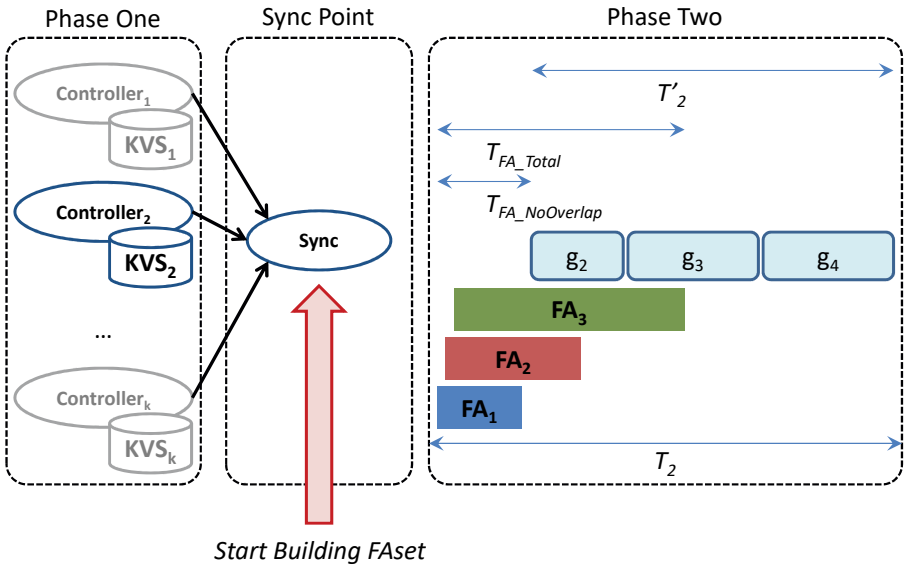


Figure 7.35: Loading FAset in overlap with the glue calculation.

On the other hand, phase two can only start when the FAset with the n-grams used by the first glue operation is ready. We recall that the glue calculations are performed in a strict sequence by each machine, in the ordering g_2, g_3, g_4, and so on. So, for example, to start calculation of g_2 only the unigrams FAset needs to be ready. We use the sync task, named by "Sync" box in Figure 7.35, to send a message to each KVS server to start the building of the FAsets based on their local n-gram tables. In the current implementation of the Global method the building of the FAsets is performed asynchronously with respect

to the beginning of phase two. Internally, for each FAset associated to a given n-gram size n, each server creates one thread responsible for the following actions:

1. Obtain the tables $D_{n_1..n_2}(j)$, that are local to this server, with the distinct n-grams of size n_1 up to n_2;

2. Identify all the leftmost and rightmost sub n-grams of size n, which correspond to the set $D_{n_{in}(D_{n_1}...\cup...D_{n_2})}(j)$, and count their frequencies in this set;

3. Sort the sub n-grams in the set $D_{n_{in}(D_{n_1}...\cup...D_{n_2})}(j)$ according to the values of their frequencies;

4. Sum the frequencies of the sub n-grams until the desired static cache hit ratio is reached. This may correspond to the application of the Minimal Cost FAset algorithm;

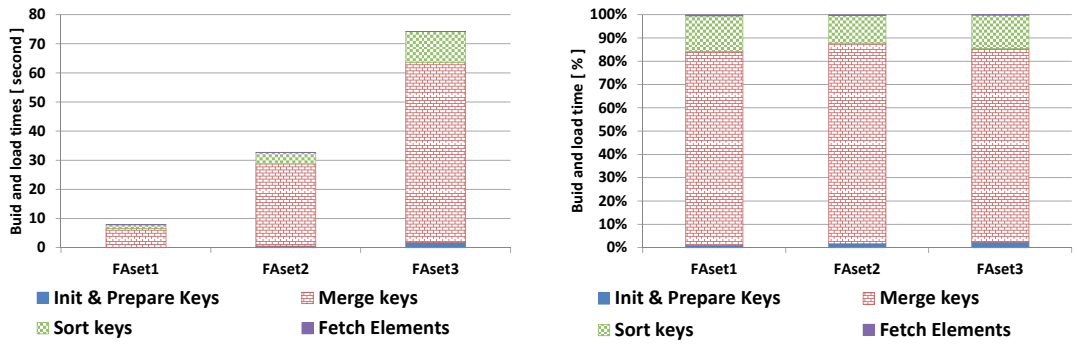
5. For each element in the FAset define a pair in the form $\langle key, value \rangle$ where key is the sub n-gram identification and $value$ is the global frequency of the sub n-gram in the *corpus*, that must be fetched from the n-gram tables; This is the only step that requires remote communication with the KVS servers.

In the initialization of phase two, each controller in each machine requests the loading of the required FAsets to its colocated KVS server. As soon as the unigrams FAset is loaded, the calculation of glue g_2 begins in the corresponding machine, while the other FAsets are being prepared and loaded.

In all the conducted experiments, from the analysis of the logs, when calculating the glue g_{234} with the static hit ratios $h_{RO_{C_1}} = 80\%$, $h_{RO_{C_2}} = 60\%$, and $h_{RO_{C_3}} = 30\%$, for the *corpora* presented in Table 7.5, on page 221, and when varying the number of machines from 3 up to 18, the controllers only waited for the unigram FAsets before starting the glue calculations.

Depending on the target static cache hit ratios defined by the bigram and trigram FAsets, and for larger FAset sizes other intervals of no overlapped execution may occur, respectively before starting g_3 or g_4.

Figure 7.36 shows as an example, the average building times of the individual FAset (unigrams $FAset_1$ for isolated glue g_2, bigrams $FAset_2$ for isolated glue g_3 and trigrams $FAset_3$ for isolated glue g_4), for a *corpus* with 511 Mw when using 18 machines in the Amazon EC2 infrastructure. Figure 7.36-a) shows the absolute time and Figure 7.36-b) shows the relative percentage for each step involved.



a) Absolute *b) Relative*

Figure 7.36: Per machine individual FAset building times (in second) for a *corpus* of 511 Mw.

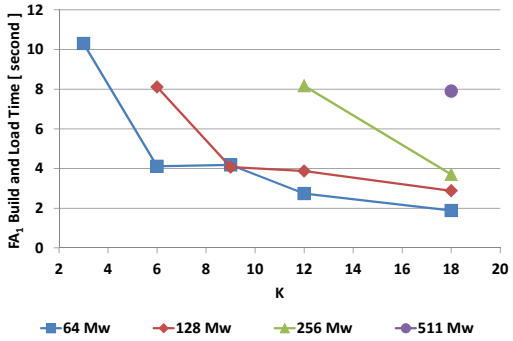
From all the steps involved in the building of the FAsets the "Merge keys" step, corresponding to the identification of all the leftmost and rightmost sub n-grams and counting their frequencies, takes almost 80% of the total time needed to build and load the FAset.

FAset Building Times. Table 7.10 (and Figure 7.37) show the average times (in seconds) to build the unigrams, bigrams and trigrams individual FAsets, respectively, for glues g_2, g_3 and g_4.

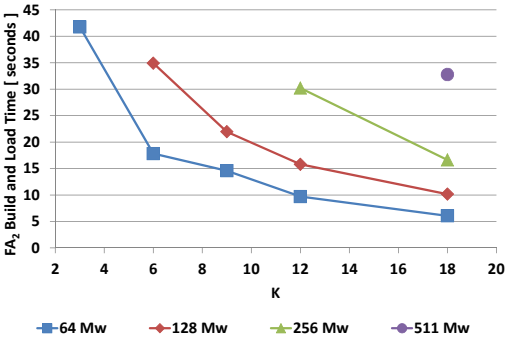
Table 7.10: Per machine FAset average building time for unigrams, bigrams and trigrams as a function of K for different *corpus* sizes. Values in seconds.

n-gram	*Corpus* [Mw]	$K = 3$	$K = 6$	$K = 9$	$K = 12$	$K = 18$
unigrams	**64**	10.30	4.11	4.18	2.74	1.88
	128		8.11	4.07	3.87	2.88
	256				8.17	3.71
	511					7.91
bigrams	**64**	41.77	17.82	14.59	9.70	6.06
	128		34.91	21.96	15.80	10.16
	256				30.22	16.63
	511					32.76
trigrams	**64**	77.25	32.79	27.35	19.34	13.31
	128		66.56	43.38	31.36	20.61
	256				61.55	38.85
	511					74.29

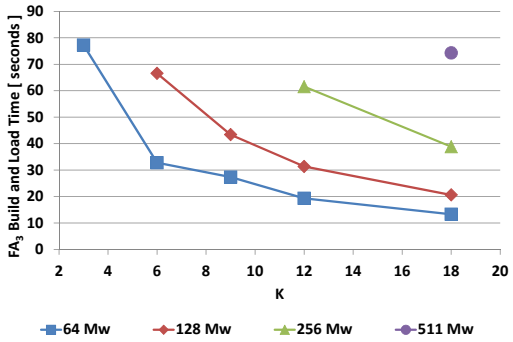
The individual FAset building times for unigrams, bigrams and trigrams for the isolated glues g_2, g_3 and g_4, are respectively presented in 7.37-a), 7.37-b) and 7.37-c).



a) Unigrams FAset for g_2, $h_{RO_{C_1}}$ = 80%



b) Bigrams FAset for g_3, $h_{RO_{C_2}}$ = 60%



c) Trigrams FAset for g_4, $h_{RO_{C_4}}$ = 30%

Figure 7.37: Per machine FAset average building and loading time *versus* the number of machines for different *corpus* sizes. The Y axis represents the FAset building time in seconds and the X axis represents the number of machines.

The time needed to build each individual FAset in each machine decreases with the number of machines.

FAset Building Overheads. The execution time for phase two (T_2) in each machine j when considering the usage of a static cache system (or a static one followed by a dynamic one) can be decomposed in two components as follows:

$$T_2(j) = T_{FA_{NoOverlap}}(j) + T'_2(j) \tag{7.66}$$

where $T_{FA_{set-NoOverlap}}(j)$ is the time to build and load the FAset that cannot be overlapped with glue calculations, and $T'_2(j)$ is the time spent in all the operations during the glue calculation by each machine (Figure 7.35).

From equation (7.66) we can define the total overhead in each machine j due to the time spent in the preparation of the FAset that cannot be overlapped with other useful computations, and which represents idle time from the point of view of the glue calculation:

$$Overhead_{FA}(j) = \frac{T_{FA_{NoOverlap}}(j)}{T_2(j)} \tag{7.67}$$

Also, we define the FAset $NoOverlapRatio_{FA}$ that measures the percentage of FAset building and loading time that occurs without overlapped execution ($T_{FA_{NoOverlap}}$) with the glue calculation, with respect to the total FAset building and loading time (T_{FA}), as illustrated in Figure 7.35 by the unigrams FAset line.

$$NoOverlapRatio_{FA}(j) = \frac{T_{FA_{NoOverlap}}(j)}{T_{FA}(j)} \tag{7.68}$$

Table 7.11 shows the average of $T_2(j)$ times (in minutes) among all the machines, for the *corpora* presented in Table 7.5 when calculating the glue g_{234} and the number of machines ranging from 3 up to 18. The times presented in the table below were collected from the logs produced during the execution of the experiments, obtained by instrumentation internal to each controller, without considering the overheads due to the launching of the workflow nodes (which are included in the absolute times presented in Table 7.12).

Table 7.11: Per machine average time of phase two — T_2 . Values in minutes.

***Corpus* [Mw]**	**$K = 3$**	**$K = 6$**	**$K = 9$**	**$K = 12$**	**$K = 18$**
64	14.23	7.59	5.90	4.20	2.96
128		15.52	10.53	7.87	5.26
256				15.93	10.18
511					21.41

Using the values in Tables 7.10 and 7.11 the $Overhead_{FA}$ and $NoOverlapRatio_{FA}$ metrics were calculated. For the ranges of *corpus* sizes and machine numbers considered, the obtained $Overhead_{FA}$ was around 1% and the $NoOverlapRatio_{FA}$ stayed around 13%.

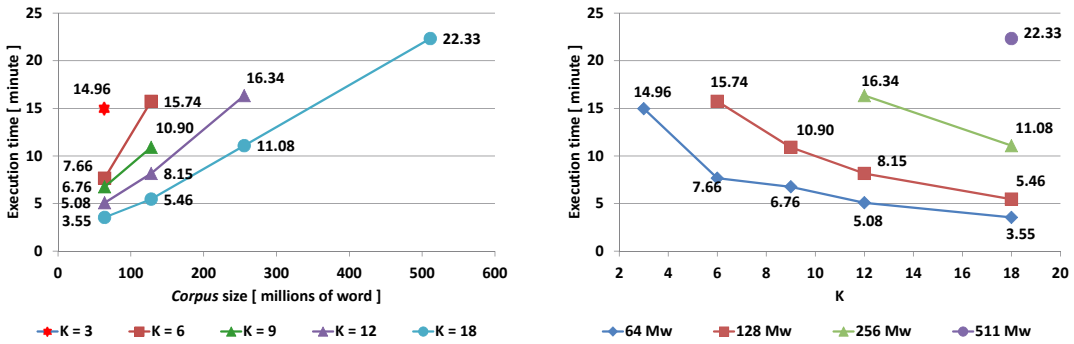
7.3.2 Execution Times of Phase Two with a Static plus Dynamic Cache

Table 7.12 shows the absolute execution time of phase two for the *corpora* presented in Table 7.5 when calculating glue g_{234} and the number of machines varying from 3 to 18 in a configuration where each machine has a cache system made of multiple individual static plus dynamic caches. In the measurements of these times we used the tools from the workflow framework, which report the execution time of phase two but also include the launching and initialization times of the controllers. As these latter components were not captured by the instrumentation of the Global method implementation that was used to measure the above presented time (T_2), the times presented in Table 7.12 are slightly higher than the ones presented above in Table 7.11.

Table 7.12: Per machine average execution time (minutes) for phase two.

Corpus [Mw]	$K = 3$	$K = 6$	$K = 9$	$K = 12$	$K = 18$
64	14.96	7.66	6.76	5.08	3.55
128		15.74	10.90	8.15	5.46
256				16.34	11.08
511					22.33

The values presented in Table 7.12 are pictured in the curves of Figure 7.38, which represent the phase two execution time as a function of the *corpus* size and as a function of K.

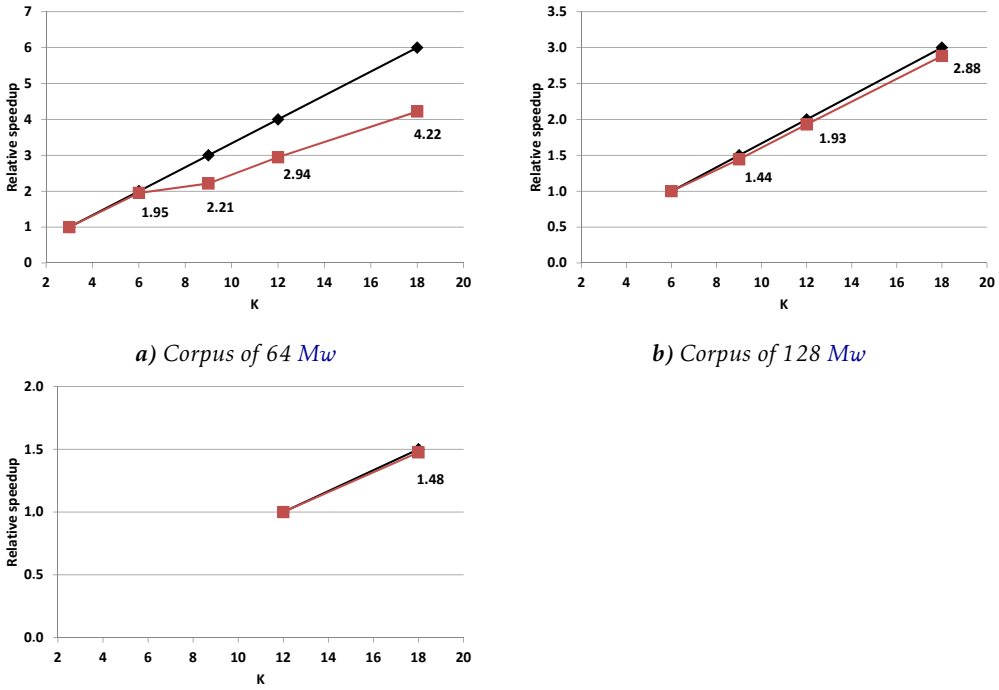


a) *Execution time versus corpus size for different K*

b) *Execution time versus K for different corpus sizes*

Figure 7.38: Phase two average execution time. The Y axis represents the execution time for phase two in minutes. In subfigure -a) the X axis represents the *corpus* size in millions of words and in subfigure -b) the X axis represents the number of machines.

For each fixed *corpus* size, the observed execution times decrease with K, in an evolution that corresponds to an almost relative linear speedup, as evidenced in Figure 7.39, where the lines with the filled lozenge (⧫) represent the linear (ideal) relative speedup and the lines with a filled square (■) represent the obtained relative speedup. For each *corpus* size, we consider the value of K_1 corresponding to the minimal number of machines required to execute phase two and this value is the value used as the reference in relation to which the relative speedup is evaluated: $Sp_{K_1 \to K}(K) = \frac{T(K_1)}{T(K)}$ and the corresponding values of this ratio are presented for all the K values in the experimented range of machine numbers.



a) *Corpus of 64 Mw*

b) *Corpus of 128 Mw*

c) *Corpus of 256 Mw*

Figure 7.39: Phase two relative speedup. The Y axis represents the relative speedup and the X axis represents the number of machines.

7.4 Chapter Summary

A static cache system preloaded with a FAset containing all the nonsingleton n-grams presents an equivalent global miss ratio of zero, as long as the cache FAset prefetching is totally overlapped with other useful computations.

However, even without considering the above best case, and not having discriminated the singleton n-grams in the experiments reported in this chapter, the use of a static plus dynamic cache (for combined glue g_{234}), with individual static cache hit ratios $h_{RO_{C_1}}$ = 80%, $h_{RO_{C_2}}$ = 60% and $h_{RO_{C_3}}$ = 30% led to global miss ratio values for the cache system $C_{(RO+RW)_{(1+2+3)}}$ around 11% - 14% for similar ranges of *corpus* sizes and machine numbers where a single dynamic cache system $C_{RW_{(1+2+3)}}$ only showed global miss ratio values about 30% (chapter 6). Even if the prefetching overlap was not complete in the conducted experiments, the $NoOverlapRatio$ (expression 7.68) stayed within relatively low values (around 13%).

When using a static cache system we can keep its global miss ratio constant by ensuring the same fixed h_{RO} value in each machine cache when we vary the number of machines or the *corpus* size, by only adjusting the size of the FAset.

For the range of *corpus* sizes and machines considered, the use of a static plus dynamic cache in the Global method led to almost linear relative speedup and sizeup for phase two.

Chapter 8

Filtering n-gram Singletons in LocalMaxs Using Bloom Filters

This chapter extends the n-gram cache system with Bloom filters.

In this chapter we discuss the impact of filtering the singleton n-grams from the input stream of the dynamic cache system that was introduced in chapter 6. We present an implementation of an n-gram dynamic cache with Bloom filters to exclude the singleton n-grams out of the cache input stream. We analyze its effect in the reduction of the miss ratio and the cache size in the entire range of *corpus* sizes.

We also propose an extension to the composite static plus dynamic cache presented in chapter 7 to include a Bloom filter and evaluate its overall miss ratio and cache size.

This chapter is organized in 4 sections. Section 8.1 introduces an extension to the LocalMaxs Global method implementation to filter the singleton n-grams out of the dynamic cache system. Section 8.2 extends the dynamic cache model introduced in chapter 6 and presents experimental results of the extended Global method implementation. Section 8.3 extends the static plus dynamic cache model introduced in chapter 7. The chapter summary is presented in section 8.4.

8.1 Filtering Singletons Using Bloom Filters

In the dynamic cache system proposed in chapter 6 for the LocalMaxs Global method, the impact of the cold-start misses upon the miss ratio was evaluated, and we identified the cache size necessary to ensure the infinite cache assumption for the entire range of *corpus* sizes. Both of the above dimensions are directly determined by the total number of the distinct n-grams in the *corpus* for each n-gram size i ($|D_i|$), which can be decomposed in two sets according to their total numbers of occurrences in the *corpus*: The singletons

and the nonsingletons. From the analysis presented in chapter 3 we concluded that the sizes of those sets depend on the *n*-gram size and vary significantly along the *corpus* sizes range.

As, from the point of view of a cache design, it is wasteful to include the singletons into the cache, we now propose an extension to the dynamic cache that excludes, from the cache input reference stream, all the occurrences of singleton *n*-grams existing in the *corpus*, and show the resulting reduction in the cache miss ratio and size, as a function of the *corpus* size.

For the range of *corpora* analyzed experimentally up to 1 Gw, the number of singletons is significant when compared with the number of nonsingletons. However, as the percentage of singletons changes along the *corpus* sizes ranges, an estimation of the impact of using Bloom filters up to the *plateau corpus* regions is also presented in this chapter.

In this section we describe the approach used to filter the singleton *n*-grams using Bloom filters. Section 8.1.1 briefly describes the concept of Bloom filter and section 8.1.2 shows the use of Bloom filters in the LocalMaxs Global method.

8.1.1 Introduction to Bloom Filters

Bloom filters [Blo70] can be used to test whether an element is or not a member of a set. Internally a Bloom filter has a bit vector and a set of hash functions. When an item is inserted into the filter all the hash functions are applied to the item and the corresponding bits in the vector are set. False positive matches are possible, but false negatives are not. In other words, a query returns either: **definitely not in set** or **possibly in set**. As an example Figure 8.1[1] shows a graphical representation of a Bloom filter trained with animals names, such as {"dog", "cat", "rabbit", "chicken", "lion", "tiger"}, tested with something that is not an animal name, in this case {"car"}.

[1]Generated using application available in https://www.jasondavies.com/bloomfilter/

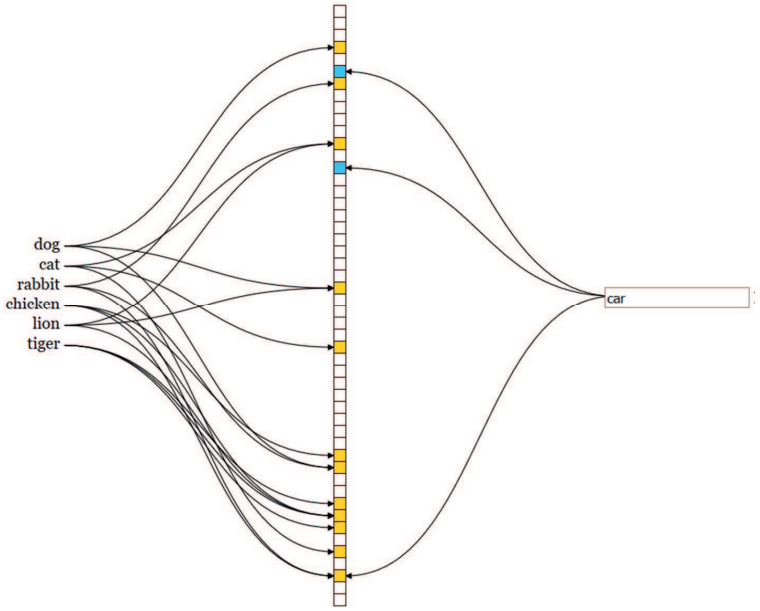


Figure 8.1: Example of a Bloom filter trained with animal names.

In this case two bits in the Bloom filter map are not set, meaning that the word "car" is not an animal name.

8.1.2 Implementing Bloom Filters for LocalMaxs Global Method

In this approach we trained the Bloom filter with the nonsingletons in the *corpus* as shown in Figure 8.2. For the majority of singletons the Bloom filter will say **definitely not in set** — one of the bits will be zero (0). A minority of singletons will say **possibly in set** — all bits are one (1) — these are false positives. All the nonsingletons will say **possibly in set**.

Each KVS server in the architecture described in chapter 4 generates a Bloom filter for each n-gram table, i.e., for the distinct n-grams of each size. During phase one of the Global method when the KVS servers update the distinct n-gram frequencies in the *corpus*, if an n-gram frequency is greater than 1 the corresponding Bloom filter (for the corresponding n-gram size) on the KVS server is trained with that n-gram (Figure 8.2). At the end of phase one, after the KVS servers have received all the n-gram frequencies, the Bloom filters in each KVS server were trained with all the n-grams that are not singletons.

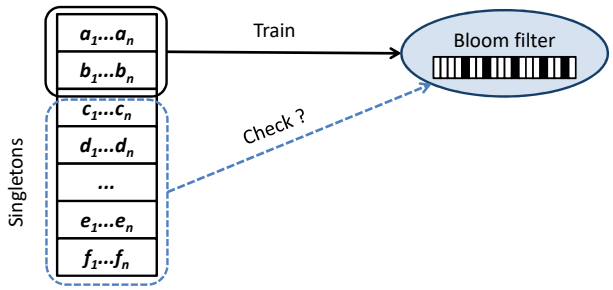


Figure 8.2: Training Bloom filter and checking n-gram membership.

During phase two, each controller gets a copy of the Bloom filters trained by the KVS servers. While calculating the glue each controller must access the needed sub *n*-gram data. Figure 8.3 shows the actions performed when checking for the presence and fetching the sub *n*-gram data: i) In the Bloom filter; ii) In the cache system; iii) Fetched from the corresponding KVS server.

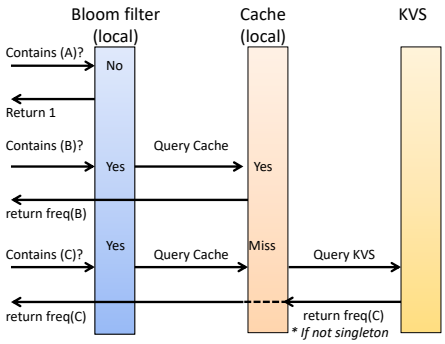


Figure 8.3: Actions executed to fetch the sub *n*-gram data during glue calculation: if (!A) elseif (!B) else (C). Adapted from [BFR11].

Listing 8.1 shows an excerpt of the code executed by the controllers to check/fetch a sub *n*-gram.

Listing 8.1: Pseudo code to fetch sub *n*-grams

```
1  private Record getSubNGramValue(String w) {
2    int idx = hash(w) % getNumberKVSServers();
3    if ( bfilters[ idx ] == null ) {
4      bfilter[ idx ]=loadFilter( idx );
5    }
6    if ( !bfilters[ idx ].contains( w ) ) {
7      return new Record( 1, w);
8    }
9    else {
10     if ( cache.contains( w ) {
11       return cache.getValue( w );
12     }
13     else {
14       Record r = KVSs[ idx ].getRecord( w );
15       if ( r.frequency!=1 ) {
16         cache.put( r );
17       }
18       return r;
19     }
20   }
21 }
```

8.2 Extending the n-gram Dynamic Cache with Bloom Filters

In this section we describe how the n-gram dynamic cache was extended using Bloom filters. Section 8.2.1 introduces a set of metrics used to evaluate the impact of filtering the singleton n-grams. A derivation of the values of the singleton and nonsingleton n-grams using the theoretical model presented in chapter 3, is presented in section 8.2.2. Sections 8.2.3 and 8.2.4 analyze the impact of filtering the singleton n-grams, respectively, for the single machine and multiple machines cases.

8.2.1 Metrics Used

In order to quantify the impact of filtering singletons we define the following metrics.

Definition 8.1: Singleton Filter Ratio

This is the ratio of the total number of singleton n-grams of size i in the *corpus* over the total number of distinct n-grams of that size in the *corpus*:

$$SF_i = \frac{|S_i|}{|D_i|}$$

where $|S_i|$ is the total number of singleton n-grams of size i in the *corpus* and $|D_i|$ is the total number of distinct n-grams of size i in the *corpus*.
Likewise, we define the global singleton filter ratio ($SF_{All(1\cdots n)_{in}C}$) as the ratio of the total number of singletons ($\left|S_{All(1\cdots n)_{in}C}\right|$) over the total number of distinct n-grams ($\left|D_{All(1\cdots n)_{in}C}\right|$) for sizes from 1 up to n:

$$SF_{All(1\cdots n)_{in}C} = \frac{\left|S_{All(1\cdots n)_{in}C}\right|}{\left|D_{All(1\cdots n)_{in}C}\right|}$$

Definition 8.2: Nonsingleton Filter Ratio

This is a complementary metric to the above, and is defined as the ratio of the total number of distinct nonsingleton n-grams of size i in the *corpus* over the total number of distinct n-grams of that size in the *corpus*:

$$NSF_i = \frac{|NS_i|}{|D_i|}$$

$$NSF_i = \frac{|D_i| - |S_i|}{|D_i|}$$

$$NSF_i = 1 - SF_i$$

where $|NS_i|$ is the total number of distinct nonsingleton n-grams of size i in the *corpus*, and $|D_i|$, $|S_i|$ and SF_i are as in Definition 8.1.
Likewise, we define the global nonsingleton filter ratio ($NSF_{All(1\cdots n)_{in}C}$) as the ratio of the total number of distinct nonsingletons ($\left|NS_{All(1\cdots n)_{in}C}\right|$) over the total number of distinct n-grams ($\left|D_{All(1\cdots n)_{in}C}\right|$) for sizes from 1 up to n:

$$NSF_{All(1\cdots n)_{in}C} = \frac{\left|NS_{All(1\cdots n)_{in}C}\right|}{\left|D_{All(1\cdots n)_{in}C}\right|}$$

Please recall that, as mentioned in section 3.6, on page 48, when using the term "nonsingleton" we intend to mean "distinct nonsingleton".

8.2.1.1 Impact of Filtering Singletons upon the Cache Size, Miss Ratio, Granularity and Efficiency

In this section we evaluate the impact of filtering the singleton n-grams upon the size of the n-gram cache system, the miss ratio, and the phase two granularity and efficiency.

Cache Size

In a system using a Bloom filter followed by a dynamic cache the total memory space occupied is given by the sum of the Bloom filter memory space and the cache system memory space. In the following analysis we only consider the cache memory space for each n-gram size i ($CacheSizeWithBF$ and $CacheSizeWithoutBF$, in terms of number of entries). In section 8.2.4.2, on page 260, we experimentally evaluate the proportion of the Bloom filter space regarding the total memory space.

$$Cache\ Size\ BF\ Factor = \frac{CacheSizeWithBF}{CacheSizeWithoutBF} = \frac{|D_i| - |S_i|}{|D_i|} = NSF_i = 1 - SF_i \qquad (8.1)$$

Miss Ratio

The number of misses of a dynamic cache system with infinite capacity corresponds to the total number of distinct n-grams ($|D_i|$) in the input reference stream. By using a Bloom filter we can exclude the misses due to the singleton n-grams ($|S_i|$).

$$Miss\ Ratio\ BF\ Factor = \frac{MissRatioWithBF}{MissRatioWithoutBF} = \frac{|D_i| - |S_i|}{|D_i|} = NSF_i = 1 - SF_i \tag{8.2}$$

Granularity

The phase two computation-to-communication granularity G of the LocalMaxs Global method as defined in expression (5.34), on page 110, measures the ratio of the phase two computation time over the communication time, and it can be increased if we reduce the miss ratio. Filtering the singleton n-grams out of the cache system is a way to reduce the miss ratio. Thus, the granularity ratio when including Bloom filters (G_{BF}) is related to the granularity G without Bloom filters as follows:

$$Granularity\ BF\ Factor = \frac{G_{BF}}{G} = \frac{mr(C, g_n)}{mr_{BF}(C, g_n)} = \frac{1}{NSF_{All(1\cdots(n-1))_{in}C}} \tag{8.3}$$

where for a given *corpus* C and glue g_n, $mr(C, g_n)$ is the global miss ratio of the cache system without Bloom filters and $mr_{BF}(C, g_n)$ is the global miss ratio with Bloom filters, and $NSF_{All(1\cdots(n-1))_{in}C}$ is given by Definition 8.2. The above expression results directly from the definition of phase two granularity (expression(5.34), on page 110) and the expression (8.2).

Thus, the increase in the granularity for phase two when filtering the singleton n-grams is proportional to the ratio $mr(C, g_n)/mr_{BF}(C, g_n)$. As $NSF_{All(1\cdots(n-1))_{in}C} \leq 1$ then $G_{BF} \geq G$.

Efficiency

The efficiency of the LocalMaxs Global method implementation relative to an ideal sequential machine, as a function of the granularity G, is given by:

$$E_0 = \frac{1}{1 + \frac{1}{G}} \tag{8.4}$$

Thus, for phase two, the ratio of the efficiency without Bloom filter (E_0) over the efficiency with Bloom filter (E_{BF}) is given by:

$$Efficieny\ BF\ Factor = \frac{E_{BF}}{E_0} = \frac{\frac{1}{1+\frac{1}{G_{BF}}}}{\frac{1}{1+\frac{1}{G}}} = \frac{G+1}{G_{BF}+1} \times \frac{G_{BF}}{G} = \frac{G+1}{G_{BF}+1} \times \frac{1}{NSF_{All}} = \frac{G+1}{G+NSF_{All}} \tag{8.5}$$

8.2.2 Evolution of Singletons and Nonsingletons

Figures 8.4 to 8.9, and corresponding Tables 8.1 to 8.6, show the evolution of the total numbers of distinct ($|D_i|$), singleton ($|S_i|$) and nonsingleton ($|NS_i|$) n-grams *versus* the *corpus* size for unigrams up to hexagrams.

The numbers of distinct and singleton n-grams were obtained by applying the theoretical model of the distribution of n-grams presented in chapter 3. The number of nonsingleton n-grams of size i is equal to $|NS_i| = |D_i| - |S_i|$. The figures also show the singleton ratio (SF_i) according to Definition 8.1.

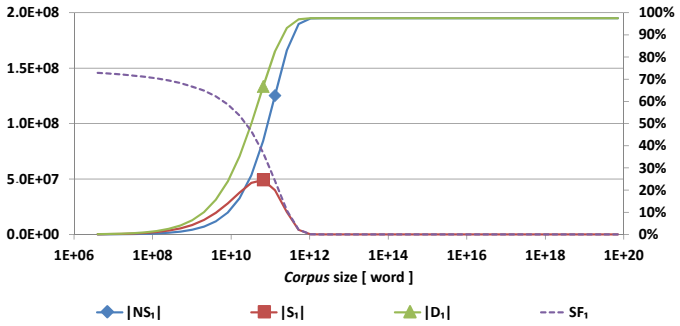


Figure 8.4: Distinct (D_1 in unigrams), singletons (S_1 in unigrams), nonsingletons (NS_1 in unigrams) and SF_1 *versus corpus* size for unigrams. The left Y axis represents D_1, S_1 and NS_1 in number of unigrams; the right Y axis represents SF_1 in percentage and the X axis represents the *corpus* size in words.

Table 8.1: Distinct (D_1 in unigrams), singletons (S_1 in unigrams), nonsingletons (NS_1 in unigrams) and SF_1 *versus corpus* size (in words) for unigrams.

$\|C\|$	$\|D_1\|$	$\|S_1\|$	$\|NS_1\|$	SF_1
$\mathbf{1.6 \times 10^7}$	6.94×10^5	4.75×10^5	2.18×10^5	69%
$\mathbf{1.3 \times 10^8}$	3.06×10^6	2.04×10^6	1.01×10^6	67%
$\mathbf{1.0 \times 10^9}$	1.28×10^7	8.09×10^6	4.69×10^6	63%
$\mathbf{1.6 \times 10^{10}}$	7.04×10^7	3.59×10^7	3.46×10^7	51%
$\mathbf{1.3 \times 10^{11}}$	1.65×10^8	3.79×10^7	1.27×10^8	23%
$\mathbf{1.0 \times 10^{12}}$	1.95×10^8	1.42×10^5	1.95×10^8	0%
$\mathbf{1.7 \times 10^{13}}$	1.95×10^8	0.00×10^0	1.95×10^8	0%
$\mathbf{1.3 \times 10^{14}}$	1.95×10^8	0.00×10^0	1.95×10^8	0%

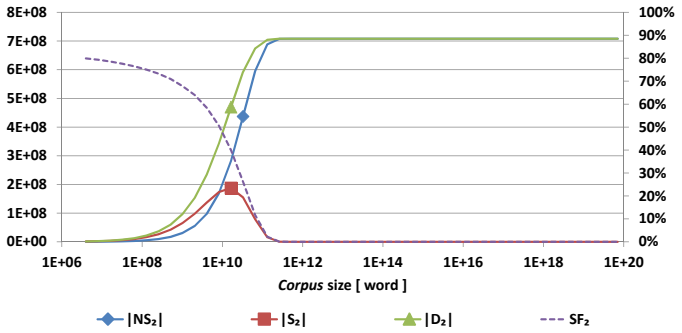


Figure 8.5: Distinct (D_2 in bigrams), singletons (S_2 in bigrams), nonsingletons (NS_2 in bigrams) and SF_2 *versus corpus* size for bigrams. The left Y axis represents D_2, S_2 and NS_2 in number of bigrams; the right Y axis represents SF_2 in percentage and the X axis represents the *corpus* size in words.

Table 8.2: Distinct (D_2 in bigrams), singletons (S_2 in bigrams), nonsingletons (NS_2 in bigrams) and SF_2 *versus corpus* size (in words) for bigrams.

$\|C\|$	$\|D_2\|$	$\|S_2\|$	$\|NS_2\|$	SF_2
$\mathbf{1.6 \times 10^7}$	4.38×10^6	3.27×10^6	1.11×10^6	75%
$\mathbf{1.3 \times 10^8}$	2.17×10^7	1.55×10^7	6.23×10^6	71%
$\mathbf{1.0 \times 10^9}$	9.70×10^7	6.25×10^7	3.45×10^7	64%
$\mathbf{1.6 \times 10^{10}}$	4.70×10^8	1.78×10^8	2.92×10^8	38%
$\mathbf{1.3 \times 10^{11}}$	7.04×10^8	1.53×10^7	6.89×10^8	2%
$\mathbf{1.0 \times 10^{12}}$	7.08×10^8	0.00×10^0	7.08×10^8	0%
$\mathbf{1.7 \times 10^{13}}$	7.08×10^8	0.00×10^0	7.08×10^8	0%
$\mathbf{1.3 \times 10^{14}}$	7.08×10^8	0.00×10^0	7.08×10^8	0%

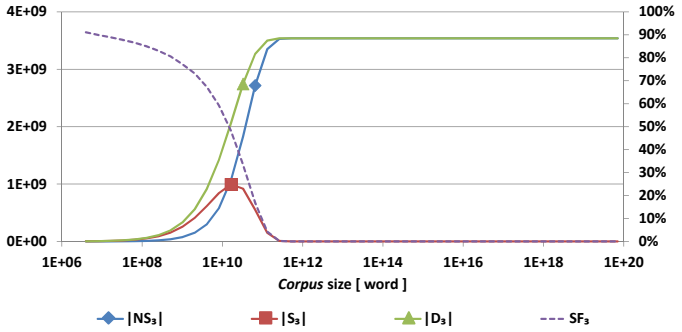


Figure 8.6: Distinct (D_3 in trigrams), singletons (S_3 in trigrams), nonsingletons (NS_3 in trigrams) and SF_3 *versus corpus* size for trigrams. The left Y axis represents D_3, S_3 and NS_3 in number of trigrams; the right Y axis represents SF_3 in percentage and the X axis represents the *corpus* size in words.

Table 8.3: Distinct (D_3 in trigrams), singletons (S_3 in trigrams), nonsingletons (NS_3 in trigrams) and SF_3 *versus corpus* size (in words) for trigrams.

$\|C\|$	$\|D_3\|$	$\|S_3\|$	$\|NS_3\|$	SF_3
$\mathbf{1.6\times10^{7}}$	9.94×10^{6}	8.40×10^{6}	1.54×10^{6}	85%
$\mathbf{1.3\times10^{8}}$	6.09×10^{7}	4.92×10^{7}	1.17×10^{7}	81%
$\mathbf{1.0\times10^{9}}$	3.33×10^{8}	2.44×10^{8}	8.89×10^{7}	73%
$\mathbf{1.6\times10^{10}}$	2.06×10^{9}	9.45×10^{8}	1.12×10^{9}	46%
$\mathbf{1.3\times10^{11}}$	3.50×10^{9}	1.41×10^{8}	3.36×10^{9}	4%
$\mathbf{1.0\times10^{12}}$	3.54×10^{9}	0.00×10^{0}	3.54×10^{9}	0%
$\mathbf{1.7\times10^{13}}$	3.54×10^{9}	0.00×10^{0}	3.54×10^{9}	0%
$\mathbf{1.3\times10^{14}}$	3.54×10^{9}	0.00×10^{0}	3.54×10^{9}	0%

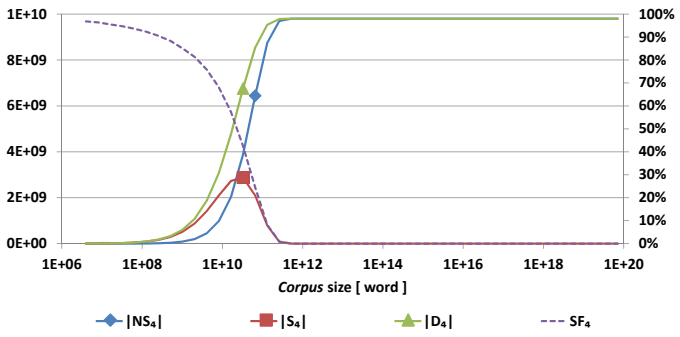


Figure 8.7: Distinct (D_4 in tetragrams), singletons (S_4 in tetragrams), nonsingletons (NS_4 in tetragrams) and SF_4 *versus corpus* size for tetragrams. The left Y axis represents D_4, S_4 and NS_4 in number of tetragrams; the right Y axis represents SF_4 in percentage and the X axis represents the *corpus* size in words.

Table 8.4: Distinct (D_4 in tetragrams), singletons (S_4 in tetragrams), nonsingletons (NS_4 in tetragrams) and SF_4 *versus corpus* size (in words) for tetragrams.

$\|C\|$	$\|D_4\|$	$\|S_4\|$	$\|NS_4\|$	SF_4
$\mathbf{1.6\times10^{7}}$	1.35×10^{7}	1.22×10^{7}	1.25×10^{6}	91%
$\mathbf{1.3\times10^{8}}$	9.51×10^{7}	8.34×10^{7}	1.17×10^{7}	88%
$\mathbf{1.0\times10^{9}}$	6.08×10^{8}	4.91×10^{8}	1.17×10^{8}	81%
$\mathbf{1.6\times10^{10}}$	4.77×10^{9}	2.60×10^{9}	2.18×10^{9}	54%
$\mathbf{1.3\times10^{11}}$	9.54×10^{9}	7.47×10^{8}	8.79×10^{9}	8%
$\mathbf{1.0\times10^{12}}$	9.80×10^{9}	5.79×10^{1}	9.80×10^{9}	0%
$\mathbf{1.7\times10^{13}}$	9.80×10^{9}	0.00×10^{0}	9.80×10^{9}	0%
$\mathbf{1.3\times10^{14}}$	9.80×10^{9}	0.00×10^{0}	9.80×10^{9}	0%

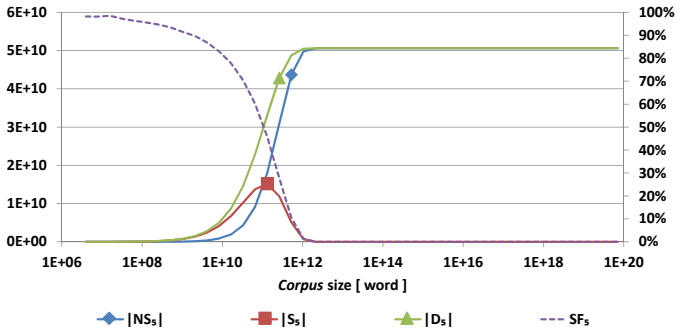


Figure 8.8: Distinct (D_5 in pentagrams), singletons (S_5 in pentagrams), nonsingletons (NS_5 in pentagrams) and SF_5 *versus corpus* size for pentagrams. The left Y axis represents D_5, S_5 and NS_5 in number of pentagrams; the right Y axis represents SF_5 in percentage and the X axis represents the *corpus* size in words.

Table 8.5: Distinct (D_5 in pentagrams), singletons (S_5 in pentagrams), nonsingletons (NS_5 in pentagrams) and SF_5 *versus corpus* size (in words) for pentagrams.

$\|C\|$	$\|D_5\|$	$\|S_5\|$	$\|NS_5\|$	SF_5
$\mathbf{1.6 \times 10^7}$	1.50×10^7	1.40×10^7	9.69×10^5	94%
$\mathbf{1.3 \times 10^8}$	1.13×10^8	1.03×10^8	1.05×10^7	91%
$\mathbf{1.0 \times 10^9}$	7.97×10^8	6.93×10^8	1.05×10^8	87%
$\mathbf{1.6 \times 10^{10}}$	8.70×10^9	6.43×10^9	2.27×10^9	74%
$\mathbf{1.3 \times 10^{11}}$	3.31×10^{10}	1.44×10^{10}	1.88×10^{10}	43%
$\mathbf{1.0 \times 10^{12}}$	5.05×10^{10}	6.51×10^8	4.98×10^{10}	1%
$\mathbf{1.7 \times 10^{13}}$	5.06×10^{10}	0.00×10^0	5.06×10^{10}	0%
$\mathbf{1.3 \times 10^{14}}$	5.06×10^{10}	0.00×10^0	5.06×10^{10}	0%

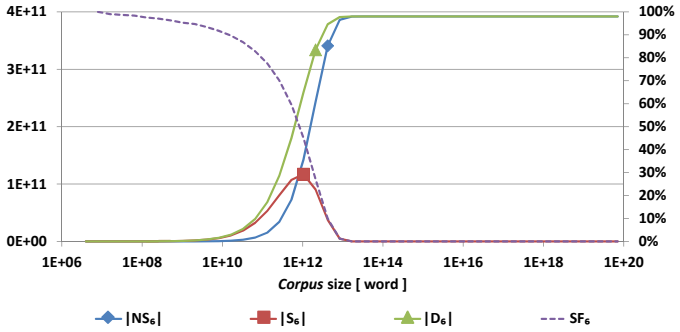


Figure 8.9: Distinct (D_6 in hexagrams), singletons (S_6 in hexagrams), nonsingletons (NS_6 in hexagrams) and SF_6 *versus corpus* size for hexagrams. The left Y axis represents D_6, S_6 and NS_6 in number of hexagrams; the right Y axis represents SF_6 in percentage and the X axis represents the *corpus* size in words.

Table 8.6: Distinct (D_6 in hexagrams), singletons (S_6 in hexagrams), nonsingletons (NS_6 in hexagrams) and SF_6 *versus corpus* size (in words) for hexagrams.

$\|C\|$	$\|D_6\|$	$\|S_6\|$	$\|NS_6\|$	SF_6
$\mathbf{1.6\times10^{7}}$	1.62×10^{7}	1.46×10^{7}	1.62×10^{6}	90%
$\mathbf{1.3\times10^{8}}$	1.20×10^{8}	1.11×10^{8}	8.80×10^{6}	93%
$\mathbf{1.0\times10^{9}}$	8.95×10^{8}	8.10×10^{8}	8.50×10^{7}	91%
$\mathbf{1.6\times10^{10}}$	1.18×10^{10}	1.01×10^{10}	1.75×10^{9}	85%
$\mathbf{1.3\times10^{11}}$	6.86×10^{10}	5.05×10^{10}	1.81×10^{10}	74%
$\mathbf{1.0\times10^{12}}$	2.59×10^{11}	1.11×10^{11}	1.48×10^{11}	43%
$\mathbf{1.7\times10^{13}}$	3.92×10^{11}	6.19×10^{7}	3.92×10^{11}	0%
$\mathbf{1.3\times10^{14}}$	3.92×10^{11}	0.00×10^{0}	3.92×10^{11}	0%

The following aspects can be pointed out common to all the n-gram sizes:

- The number of singletons shows an increasing trend following the number of distinct n-grams as new distinct n-grams appear with increasing *corpus* sizes and, in the case of the English *corpora* analyzed, it reaches a maximum for *corpus* sizes in the range from 10 Gw to 1 Tw depending on the n-gram size;
- However, there is a monotonic decrease of the singleton ratio SF_i, which becomes zero when the *plateau* regions are reached.

Figure 8.10 shows the variation of the nonsingleton ratio (NSF_i), for unigrams up to hexagrams, when the *corpus* size ranges from 16 Mw up to 130 Tw.

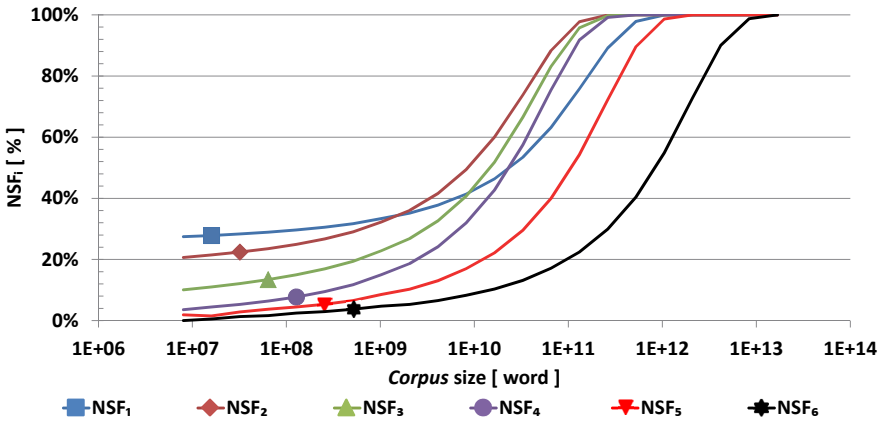


Figure 8.10: NSF_i *versus corpus* size. The Y axis represents the NSF_i in percentage and the X axis represents the *corpus* size in words.

The following observations can be taken:

- Except for the unigrams case, for a fixed *corpus* size the nonsingleton ratio decreases with the n-gram size, i.e., there is a higher proportion of singletons among the distinct n-grams for the larger n-gram sizes than for smaller sizes;

- ✦ The singletons of the smaller n-gram sizes start disappearing first, i.e., for smaller *corpus* sizes thresholds, than the singletons of the higher n-gram sizes.

In the following we analyze the impact of a Bloom filter upon a dynamic cache in the cases of a single machine (section 8.2.3) and multiple machines (section 8.2.4).

8.2.3 Single Machine Case ($K = 1$)

In this section we present a model for the Bloom filter plus dynamic cache that allows us to estimate the cache size and miss ratio metrics. The size of the cache system is discussed in section 8.2.3.1. Section 8.2.3.2 analyzes the impact of using Bloom filters upon the miss ratio and miss penalty. An analysis regarding how the filtering of singletons affects the granularity and efficiency is presented in section 8.2.3.3.

8.2.3.1 Cache Size

The effect of the Bloom filter upon the dynamic cache size is directly determined by the evolution of the sub n-gram singletons that are illustrated in Figure 8.11 for the cases of the combined glue g_{234} and $g_{2\cdots6}$, which are, respectively the singletons from unigrams up to trigrams, and the singletons from unigrams up to pentagrams.

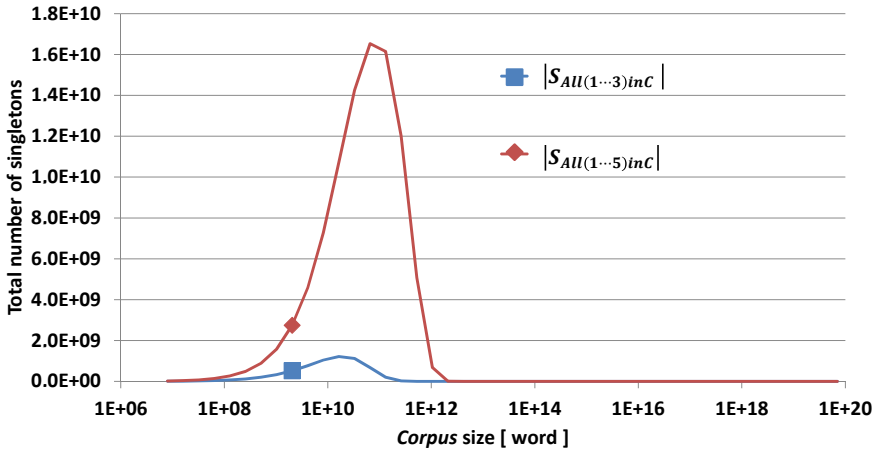


Figure 8.11: Sub n-gram singletons evolution for $K = 1$. The Y axis represents the total number of sub n-gram singletons and the X axis represents the *corpus* size in words.

The size of the cache system is given by the sum of the sizes of the individual caches. In a scenario where all the singleton n-grams are kept within the cache system the size of each individual cache corresponds to the number of distinct n-grams ($|D_i|$). When the singleton n-grams are filtered out of the individual caches, the size of each individual cache is equal to the nonsingleton n-grams $|NS_i| = |D_i| - |S_i|$. Thus, the following expressions give the size of the entire cache system, for the combined glues g_{234} and $g_{2\cdots6}$ in the two considered scenarios: without filtering the singletons (expressions (8.6) and (8.8)) and when filtering the singletons (expressions (8.7) and (8.9)).

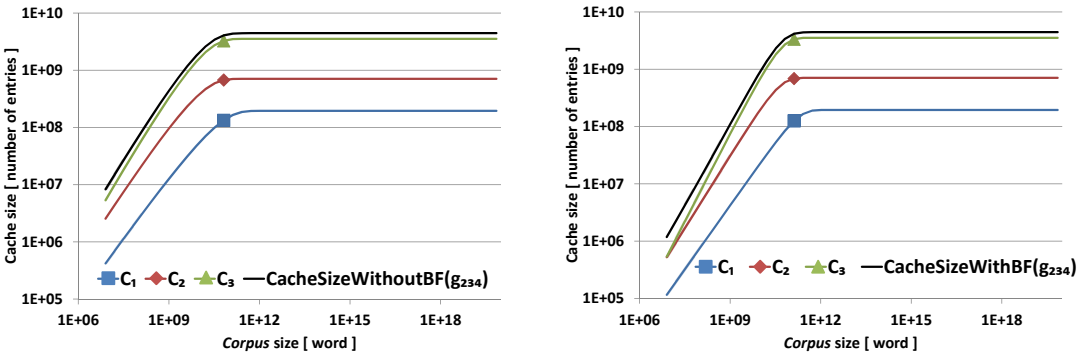
$$CacheSizeWithoutBF(g_{234}) = |D_1| + |D_2| + |D_3| = \left|D_{All(1\cdots3)inC}\right| \tag{8.6}$$

$$CacheSizeWithBF(g_{234}) = |NS_1| + |NS_2| + |NS_3| = \left|NS_{All(1\cdots3)inC}\right| \tag{8.7}$$

$$CacheSizeWithoutBF(g_{2\cdots6}) = |D_1| + |D_2| + |D_3| + |D_4| + |D_5| = \left|D_{All(1\cdots5)inC}\right| \tag{8.8}$$

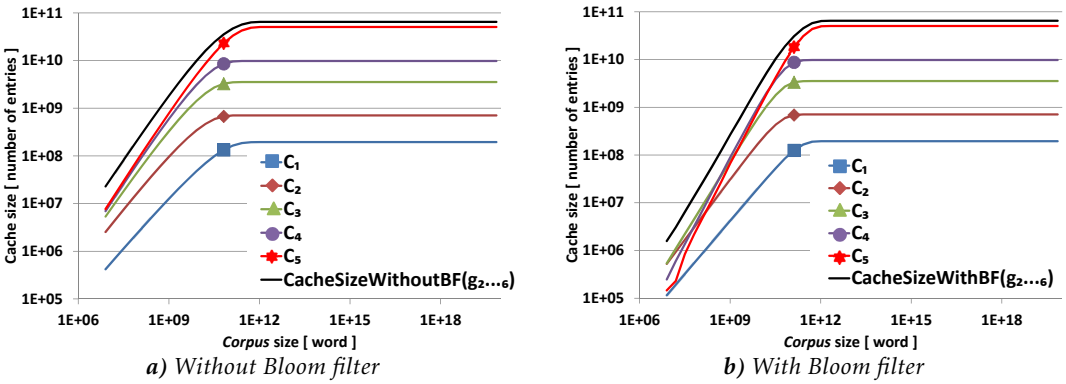
$$CacheSizeWithBF(g_{2\cdots6}) = |NS_1| + |NS_2| + |NS_3| + |NS_4| + |NS_5| = \left|NS_{All(1\cdots5)inC}\right| \tag{8.9}$$

Figures 8.12 and 8.13 show the evolution of the total number of n-gram cache entries, with and without Bloom filter, for the combined glues g_{234} and glue $g_{2\cdots6}$, along the entire *corpus* range up to the *plateaux*.



a) Without Bloom filter *b) With Bloom filter*

Figure 8.12: Number of cache entries *versus corpus* size for combined glue g_{234}, for $K = 1$. The Y axis represents the cache size in number of n-gram entries for the individual caches C_1, C_2, C_3, and for the global C_{1+2+3} cache system. The X axis represents the *corpus* size in words.



a) Without Bloom filter *b) With Bloom filter*

Figure 8.13: Number of cache entries *versus corpus* size for combined glue $g_{2\cdots6}$, for $K = 1$. The Y axis represents the cache size in number of n-gram entries for the individual caches C_1, C_2, C_3, C_4, C_5, and for the global $C_{1+2+3+4+5}$ cache system. The X axis represents the *corpus* size in words.

In the *plateau* regions the Bloom filters have no effect and the cache size is determined by the number of distinct (nonsingleton) n-grams of the *corpus* in the *plateaux*.

In order to evaluate the memory cache size reduction due to the Bloom filter we analyze the nonsingleton ratio. The global nonsingleton ratio in the case of cache system ($C_{1+2+\cdots+(n-1)}$) for a glue g_n where n is the maximum n-gram size is shown in Figure 8.14 in the case of combined glues g_{234} and $g_{2\cdots6}$ when the *corpus* size varies up to the *plateau* regions.

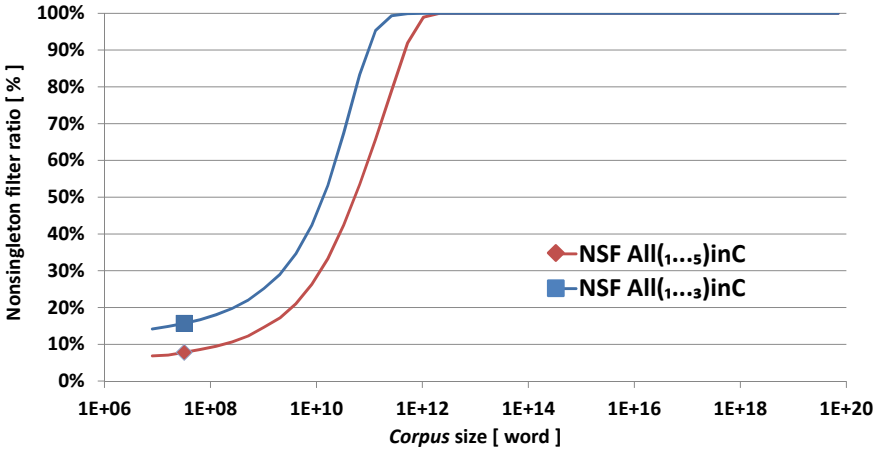


Figure 8.14: Nonsingleton ratio for cache sizes for combined glues g_{234} and $g_{2\cdots6}$, for $K = 1$. The Y axis represents the nonsingleton ratio in percentage and X axis represents the *corpus* size in words.

As the size of the cache system with Bloom filter is $NSF_{All} \times CacheSizeWithoutBF$ we observe that its minimum size occurs for the smaller *corpus* sizes, corresponding to a size reduction (when going from a system without Bloom filter to a system with Bloom filter) around 80% for g_{234} and 90% for $g_{2\cdots6}$. However, as the nonsingletons increase (with increasing *corpus* sizes), the cache size reduction due to the Bloom filter tends to decrease, reaching a reduction around 50% for *corpus* sizes around 16 Gw for g_{234} and around 65 Gw for $g_{2\cdots6}$, in the case of the English *corpora* analyzed.

8.2.3.2 Miss Ratio and Miss Penalty

Taking into account the existence of a nonnull probability of false positives associated to the implementations of the Bloom filters, the number of singleton n-grams identified by the Bloom filter is given by:

$$\left|S_i^{'}\right| = |S_i| - |FalsePositive_i| \tag{8.10}$$

where $S_i^{'}$ is the set of true positives given by the Bloom filter, i.e., for which the reply is "not present"; S_i is the real set of singleton n-grams in the *corpus*; $FalsePositive_i$ is the set of false positive singleton n-grams identified by the Bloom filter. From this expression we can estimate the miss ratio by using the value of $|S_i|$ obtained from each *corpus* data and by taking into account the Bloom filter false positive probability (denoted as

FalsePositiveProbability and shortened *FPP*) intrinsic to the Bloom filter implementation. Thus, the miss ratio of the dynamic cache, in the case of warmed-up cache for the combined glue $g_{2\cdots6}$ is given by the following expressions which were obtained from expression (6.20 on page 155) by subtracting the number of observed singletons given by the Bloom filter:

$$\begin{aligned}
mr_{(BF+RW)}(C_1, g_{2\cdots6}) &= \frac{\left(|D_1| - \left|S_1'\right|\right)}{2|D_2|+2|D_3|+2|D_4|+2|D_5|+2|D_6|} \\
mr_{(BF+RW)}(C_2, g_{2\cdots6}) &= \frac{\left(|D_2| - \left|S_2'\right|\right)}{2|D_3|+2|D_4|+2|D_5|+2|D_6|} \\
mr_{(BF+RW)}(C_3, g_{2\cdots6}) &= \frac{\left(|D_3| - \left|S_3'\right|\right)}{2|D_4|+2|D_5|+2|D_6|} \\
mr_{(BF+RW)}(C_4, g_{2\cdots6}) &= \frac{\left(|D_4| - \left|S_4'\right|\right)}{2|D_5|+2|D_6|} \\
mr_{(BF+RW)}(C_5, g_{2\cdots6}) &= \frac{\left(|D_5| - \left|S_5'\right|\right)}{2|D_6|}
\end{aligned} \tag{8.11}$$

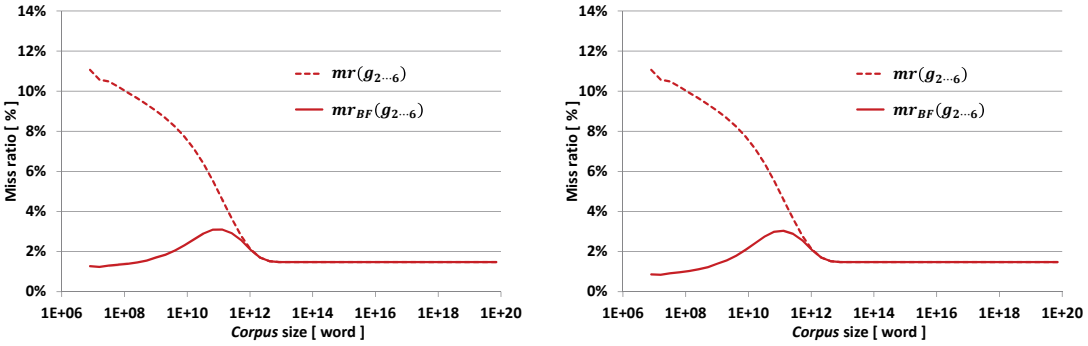
$$mr_{(BF+RW)}(C_{1+\cdots+5}, g_{2\cdots6}, FPP) = \frac{\left|D_{All(1\cdots5)inC}\right|}{\left|all_{glue_{g_2\cdots g_6}Ref}\right|} - \frac{\left|S'_{All(1\cdots5)inC}\right|}{\left|all_{glue_{g_2\cdots g_6}Ref}\right|} \tag{8.12}$$

and expression (8.12) simplifies to:

$$mr_{(BF+RW)}(C_{1+\cdots+5}, g_{2\cdots6}, FPP) = mr_{RW}(C_{1+\cdots+5}, g_{2\cdots6}) - \frac{\left|S'_{All(1\cdots5)inC}\right|}{\left|all_{glue_{g_2\cdots g_6}Ref}\right|} \tag{8.13}$$

where $\left|S'_{All(1\cdots5)inC}\right| = \sum_{i=1}^{5}\left|S_i'\right| = \left|S_{All(1\cdots5)inC}\right| - \sum_{i=1}^{5}|FalsePositive_i|$.

Global Miss Ratio. Figure 8.15 shows the variation of the global warm-start miss ratio, for different Bloom filter false positive probabilities, in the case of the combined glue $g_{2\cdots6}$, when the *corpus* size varies up to the *plateau* regions and using a single machine ($K = 1$). Figure 8.15-a) and Figure 8.15-b) correspond to the scenarios where the Bloom filter false positive probability values are, respectively, 0% and 1%. In both subfigures the dotted line represents the global warm-start miss ratio without Bloom filters.



a) *False positive probability (fpp)* =0% ***b)*** *False positive probability (fpp)* =1%

Figure 8.15: Global miss ratio *versus corpus* size for glue $g_{2\cdots6}$ with Bloom filters, for $K = 1$. The Y axis represents the miss ratio in percentage and the X axis represents the *corpus* size in words.

Table 8.7 shows the corresponding values of the global warm-start miss ratio with Bloom filters, for the curves presented in Figure 8.15, when the *corpus* size ranges from 16 Mw up to 130 Tw.

From the figure and the table it is clear that such low value of false positive probability has a negligible effect upon the results.

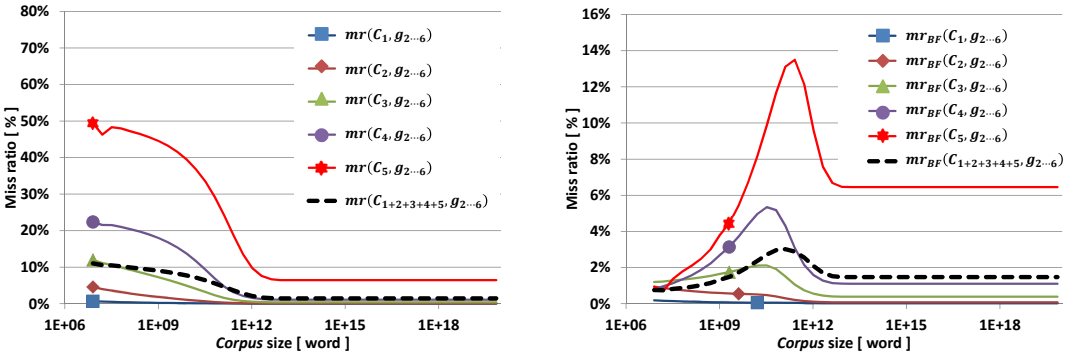
Table 8.7: Global miss ratio ($mr_{BF}(C_{1+2+3+4+5}, g_{2\cdots6}, FPP)$) *versus corpus* size (in words) for glue $g_{2\cdots6}$ with Bloom filters, for $K = 1$. Miss ratios in percentage (%).

$\|C\|$	$mr(g_{2\cdots6})$	$mr_{BF}(g_{2\cdots6}$,FPP=0%)	$mr_{BF}(g_{2\cdots6}$,FPP=1%)
$\mathbf{1.6\times10^{7}}$	10.57	0.75	0.84
$\mathbf{1.3\times10^{8}}$	9.93	0.94	1.03
$\mathbf{1.0\times10^{9}}$	9.02	1.32	1.40
$\mathbf{1.6\times10^{10}}$	7.12	2.37	2.42
$\mathbf{1.3\times10^{11}}$	4.60	3.02	3.04
$\mathbf{1.0\times10^{12}}$	2.11	2.08	2.08
$\mathbf{1.7\times10^{13}}$	1.47	1.47	1.47
$\mathbf{1.3\times10^{14}}$	1.47	1.47	1.47

The following conclusions are obtained:

- As soon as the total number of singleton *n*-grams is zero (in the *plateau* regions), the effect of the Bloom filter becomes null;
- However, in the regions of the *corpus* sizes where the total number of singleton *n*-grams is a significant percentage of the total number of distinct *n*-grams (in the regions of the *corpus* size from 8 Mw until around 130 Gw), the result of using a Bloom filter is a reduction that leads the global warm-start miss ratio from the interval of values between 11% and 5% (in the case of no Bloom filter) to the interval of values between 1% and 3% when using a Bloom filter.

Individual Miss Ratios. Figure 8.16 shows the comparison of the global cache system $C_{1+\cdots+5}$ miss ratio and the individual warm-start cache miss ratios for the individual caches C_1 to C_5, in the cases without and with Bloom filters, respectively, in Figures 8.16-a) and 8.16-b), for the combined glue $g_{2\cdots6}$, when the *corpus* size varies up to the *plateau* regions.



a) *Individual and global miss ratio without Bloom filter*

b) *Individual and global miss ratio with Bloom filter and FPP = 0%*

Figure 8.16: Global and individual miss ratios *versus corpus* size for glue $g_{2\cdots6}$, for $K = 1$. The Y axis represents the miss ratio in percentage and the X axis represents the *corpus* size in words.

Table 8.8 shows the corresponding values of the individual cache warm-start miss ratio without Bloom filters, for the curves presented in Figure 8.16-a), when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 8.8: Individual cache warm-start miss ratio *versus corpus* size (in words) for glue $g_{2\cdots6}$ without Bloom filters, for $K = 1$. Miss ratios in percentage (%).

$\|C\|$	$mr(C_1,g_{2\cdots6})$	$mr(C_2,g_{2\cdots6})$	$mr(C_3,g_{2\cdots6})$	$mr(C_4,g_{2\cdots6})$	$mr(C_5,g_{2\cdots6})$
$\mathbf{1.6\times10^{7}}$	0.59	4.02	11.14	21.57	46.27
$\mathbf{1.3\times10^{8}}$	0.37	2.79	9.28	20.40	47.21
$\mathbf{1.0\times10^{9}}$	0.23	1.84	7.23	17.98	44.56
$\mathbf{1.6\times10^{10}}$	0.13	0.86	4.08	11.63	36.76
$\mathbf{1.3\times10^{11}}$	0.07	0.31	1.57	4.69	24.17
$\mathbf{1.0\times10^{12}}$	0.03	0.11	0.55	1.58	9.72
$\mathbf{1.7\times10^{13}}$	0.02	0.08	0.39	1.11	6.45
$\mathbf{1.3\times10^{14}}$	0.02	0.08	0.39	1.11	6.45

Table 8.9 shows the values of the individual cache warm-start miss ratio with Bloom filters (with false positive probability = 0%), for the curves presented in Figure 8.16-b), when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 8.9: Individual cache warm-start miss ratio *versus corpus* size (in words) for glue $g_{2...6}$ with Bloom filters for a false positive probability $FPP = 0\%$, for $K = 1$. Miss ratios in percentage (%).

$\|C\|$	$mr_{BF}(C_1, g_{2...6})$	$mr_{BF}(C_2, g_{2...6})$	$mr_{BF}(C_3, g_{2...6})$	$mr_{BF}(C_4, g_{2...6})$	$mr_{BF}(C_5, g_{2...6})$
$\mathbf{1.6 \times 10^7}$	0.16	0.86	1.23	0.97	0.71
$\mathbf{1.3 \times 10^8}$	0.11	0.70	1.39	1.57	2.11
$\mathbf{1.0 \times 10^9}$	0.08	0.59	1.65	2.69	3.80
$\mathbf{1.6 \times 10^{10}}$	0.06	0.52	2.11	4.97	8.15
$\mathbf{1.3 \times 10^{11}}$	0.05	0.30	1.51	4.30	13.12
$\mathbf{1.0 \times 10^{12}}$	0.03	0.11	0.55	1.58	9.59
$\mathbf{1.7 \times 10^{13}}$	0.02	0.08	0.39	1.11	6.45
$\mathbf{1.3 \times 10^{14}}$	0.02	0.08	0.39	1.11	6.45

The following conclusions can be taken:

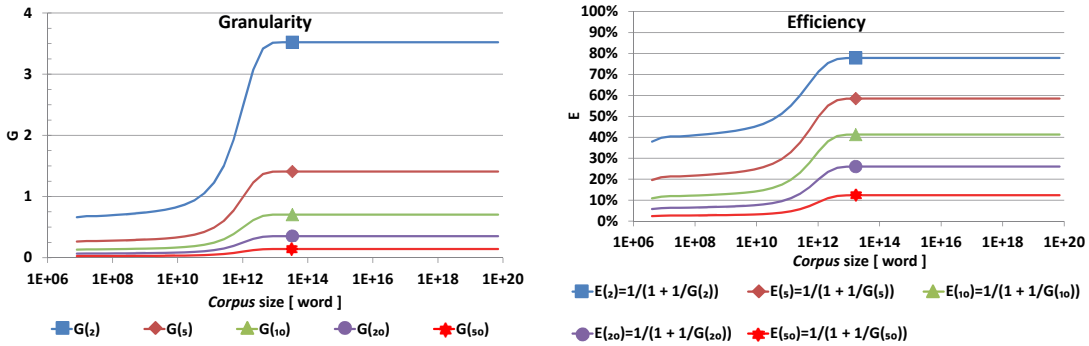
- The figures and tables illustrate that the reduction of the individual cache miss ratio due to including Bloom filters is higher for the larger n-grams sizes, due to their lower NSF_i ratio;

- For each n-gram size i, the individual cache C_i miss penalty reduces in the same proportion as the NSF_i ratio when it goes from $|D_i|$ (infinite dynamic cache without Bloom filter) to $|D_i - S_i|$ (with Bloom filter).

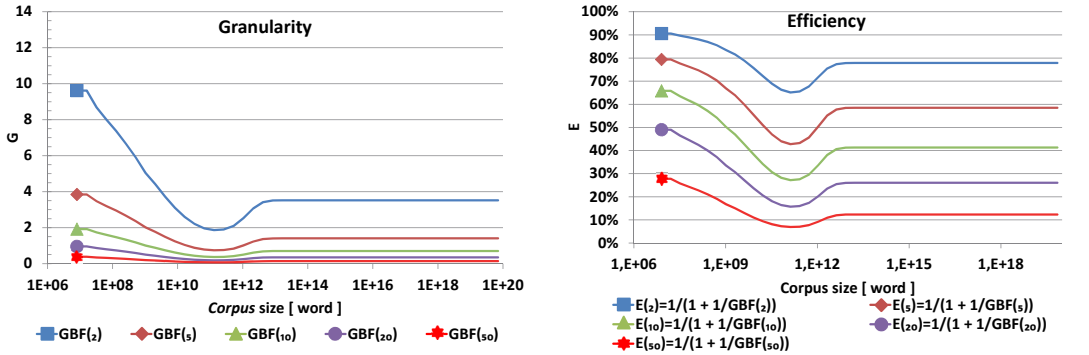
8.2.3.3 Granularity and Efficiency

In order to quantify the overheads due to remote communications regarding the n-gram misses that occur in a single machine with a limited memory, we consider the granularity of the phase two computation component *versus* the communication component, e.g., for $K = 1$ is $G(K)_{K=1} = T_{0_2}/T_{comm_2}$. We also consider the corresponding efficiency of phase two in a single machine with limited memory (defined in relation to an ideal single sequential machine with unlimited memory), which is defined by $E_0(K)_{K=1} = T_{0_2}/T_2$, where $T_2 = T_{0_2} + T_{comm_2}$, so $E_0(K)_{K=1} = 1/\ (1 + 1/G(K))$.

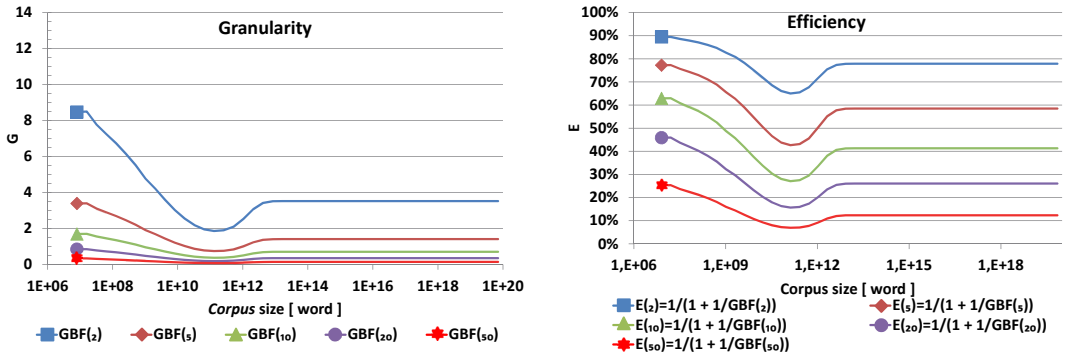
Figure 8.17 shows the evolution of the granularity and efficiency for phase two when filtering the singleton n-grams, for different Bloom filter false positive probabilities, in the case of the combined glue $g_{2...6}$, when the *corpus* size varies up to the *plateau* regions and using a single machine ($K = 1$). Figure 8.17-a) corresponds to a scenario without filtering singletons. Figure 8.17-b) and Figure 8.17-c) correspond to the scenarios where the Bloom filter false positive probability values are, respectively, 0% and 1%.



a) *Without Bloom filters*



b) *False positive probability* = 0%



c) *False positive probability* = 1%

Figure 8.17: Granularity and efficiency *versus corpus* size for glue $g_{2...6}$ and for different time ratios t_{fetch}/t_{g_n}, for $K = 1$. The Y axis represents the granularity or the efficiency (in percentage); the X axis represents the *corpus* size in words.

In Figure 8.17 the parameter i in the granularities and efficiencies ($G(i)$, $GBF(i)$ and $E(i)$) represents the ratio t_{fetch}/t_{g_n} with values: 2, 5, 10, 20 and 50. The following conclusions can be taken:

- The granularity values in the case of using Bloom filters (G_{BF}) are always higher than the values without Bloom filters (G) for each fixed *corpus* size all along the entire *corpus* range and for any value i of the ratio t_{fetch}/t_{g_n}; Even when the granularity with Bloom filter reaches its minimum value, it is still higher than in the case without Bloom filter;

- The granularity values without or with Bloom filters become equal in the *plateau*

regions;

- The efficiency due to the use of Bloom filters is significantly higher than in the case without Bloom filter, except in the *plateau* regions, where they are equal; while, for example when the ratio t_{fetch}/t_{g_n} is 20, without Bloom filter the efficiency ranges from about 6% until 26% in the plateaux, on the other hand, the efficiency with Bloom filter ranges from about 50% for the smaller *corpus*, goes through a minimum about 15% (for a *corpus* size where the efficiency without Bloom filter is about 10%), and then raises to the value of 26% in the plateau regions.

8.2.4 Multiple Machines Case ($K > 1$)

Following the same empirical approach as in the previous studies of the single dynamic cache and the static plus dynamic cache, we analyze the behavior of the multiple machines case when having Bloom filters by considering real execution experimental scenarios based on the extended implementation of the Global method with Bloom filters.

In this section we present the experimental evaluation of the dynamic cache when filtering the singleton n-grams using Bloom filters. Section 8.2.4.1 describes the execution environment. A discussion regarding the cache size when filtering the singleton n-grams is presented in section 8.2.4.2. An analysis of the miss ratio is presented in section 8.2.4.3. Section 8.2.4.4 presents the execution time of phase two when filtering the singleton n-grams.

8.2.4.1 Execution Environment

In order to evaluate experimentally the impact of filtering the singleton n-grams in the miss ratio of a dynamic cache, we executed the three phases of the LocalMaxs Global method, under the Lunacloud public cloud environment, using virtual machine instances with 64 GB of RAM and 4 vCPU, in two scenarios: without Bloom filters and using Bloom filters. We considered one *corpus* of 64 Mw and used 10 machines in order to evaluate the relevance $re_{2...5}$. In both scenarios we used a dynamic cache with infinite capacity. However, in the first scenario the singletons were not excluded from the input reference stream of the cache system. In the second scenario the singletons were filtered out using the approach described in section 8.1.2, on page 241. In this latter case the false probability of the Bloom filters was configured to be 0.1%.

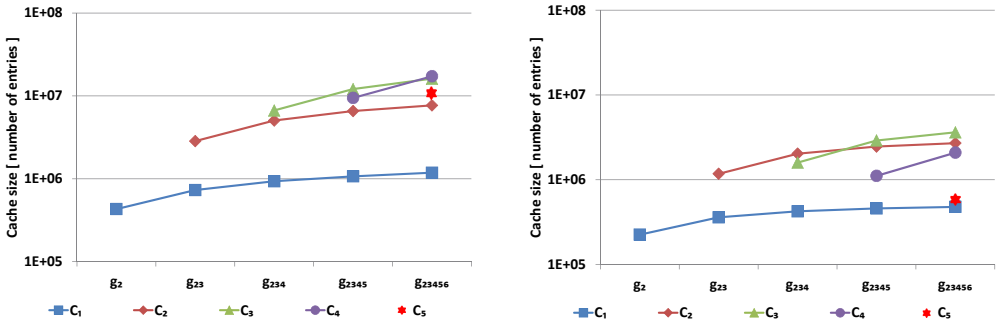
During the execution of the Global method each controller in each machine collects a set of measures such as the number of hits on each cache, the size of the input reference stream and the time spent in each phase of the algorithm. The values of those measures were saved in a log file (one per controller) and used to build the figures and tables presented in this section. Table 8.10 shows the average values, among the 10 machines, of the total numbers of distinct, singleton and nonsingleton n-grams; and the singleton ratio for the *corpus* of 64 Mw.

Table 8.10: Per machine average numbers of distinct (D_i), singleton (S_i) and nonsingleton (NS_i) n-grams, and singleton ratio (SF_i), for unigrams up to hexagrams, in a *corpus* of 64 Mw, and $K = 10$.

n-gram	$\|D_i\|$	$\|S_i\|$	$\|NS_i\|$	SF_i
unigrams	159 152	108 128	51 024	0.679
bigrams	1 185 850	879 058	306 793	0.741
trigrams	3 145 314	2 675 141	470 174	0.851
tetragrams	4 762 784	4 402 363	360 421	0.924
pentagrams	5 578 882	5 375 450	203 432	0.964
hexagrams	5 927 909	5 815 854	112 055	0.981

8.2.4.2 Cache Size

Figure 8.18 shows the average cache sizes without and with Bloom filters (in number of n-gram entries), among the $K = 10$ machines, of the individual caches (C_1 to C_5) when calculating all the glues from g_2 to $g_{2...6}$ and Table 8.11 shows the corresponding values.



a) Without Bloom filters *b) With Bloom filters*

Figure 8.18: Per machine sizes of cache C_1 up to C_5 *versus* glues g_2 up to $g_{2...6}$, for a *corpus* of 64 Mw, $K = 10$, . The Y axis represents the size of the caches in number of n-grams in each cache and the X axis represents the glue calculation cases.

Table 8.11: Per machine average of observed individual cache sizes (number of n-gram entries) *versus* the glue calculations (g_2 to $g_{2...6}$) without and with Bloom filters for a *corpus* of 64 Mw and $K = 10$.

	Cache C_i	g_2	g_{23}	g_{234}	$g_{2...5}$	$g_{2...6}$
Without BF	C_1	430525	730773	933287	1070356	1183152
	C_2		2844088	5045987	6564460	7688336
	C_3			6666551	12096946	16157159
	C_4				9460808	17243293
	C_5					10784184
With BF	C_1	225078	359793	423840	458381	478252
	C_2		1175286	2029502	2463026	2701654
	C_3			1591956	2903140	3624770
	C_4				1110130	2088832
	C_5					579423

Experimental evaluation of the Bloom filter memory space

Previously when evaluating the cache size we only considered its number of entries, given by $CacheSizeWithoutBF$ or $CacheSizeWithBF$ depending on not having or having a Bloom filter. In order to evaluate the cache size in number of bytes we consider the entry size ($EntrySize$), which is the average number of bytes occupied by each cache entry (n-gram identifier plus its global frequency in the *corpus*) for the corresponding glue calculation (g_2 up to $g_{2...6}$). The $EntrySize$ parameter is calculated using expression (9.1), on page 275.

The memory space of the cache in bytes is given by:

$$MemorySpace_{CacheWithoutBF} = CacheSizeWithoutBF \times EntrySize \tag{8.14}$$

$$MemorySpace_{CacheWithBF} = CacheSizeWithBF \times EntrySize \tag{8.15}$$

However, when using Bloom filters, it is also necessary to evaluate the memory space occupied by the Bloom filter in bytes, which is denoted as $MemorySpace_{BF}$. Thus, in a composite cache with a Bloom filter plus an individual cache ($BF + CacheWithBF$), the total memory space in bytes is given by:

$$\begin{aligned} MemorySpace_{(BF+CacheWithBF)} = MemorySpace_{BF} + MemorySpace_{CacheWithBF} = \\ MemorySpace_{CacheWithBF}\left(1 + \frac{MemorySpace_{BF}}{MemorySpace_{CacheWithBF}}\right) \end{aligned} \tag{8.16}$$

In order to compare the above total memory space with the memory space of a cache system without Bloom filter, we obtain the following expression:

$$\frac{MemorySpace_{BF+CacheWithBF}}{MemorySpace_{CacheWithoutBF}} = NSF_{All} \times \left(1 + \frac{MemorySpace_{BF}}{MemorySpace_{CacheWithBF}}\right) \tag{8.17}$$

In the following we present an experimental evaluation of this latter aspect for the case of a *corpus* 64 Mw and a configuration with $K = 10$ machines. In this context, the above expressions defining all the above quantities related to the numbers of distinct (D_i), singleton (S_i) and nonsingleton (NS_i) n-grams, and the corresponding ratios (SF_i and NSF_i) are considered in a per machine basis, being labeled with the machine index j. The same applies to the above expressions concerning the memory space.

Figure 8.19 shows the per machine $NSF_{All}(j)$ ratio, which only considers the ratio between the cache sizes with and without Bloom filters, and the total memory space ratio that considers the cache size and the Bloom filter memory space, with and without Bloom filter as defined by expression (8.17).

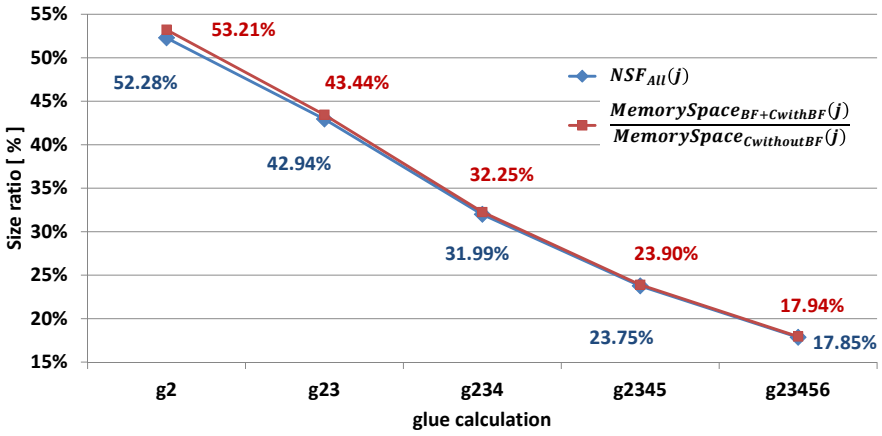


Figure 8.19: Per machine nonsingleton filter ratio for the cache system for a *corpus* of 64 Mw and $K = 10$, when the glue calculation varies from g_2 to $g_{2...6}$. The Y axis represents the size ratio in percentage and the X axis represents the glue calculations (from glue g_2 to $g_{2...6}$).

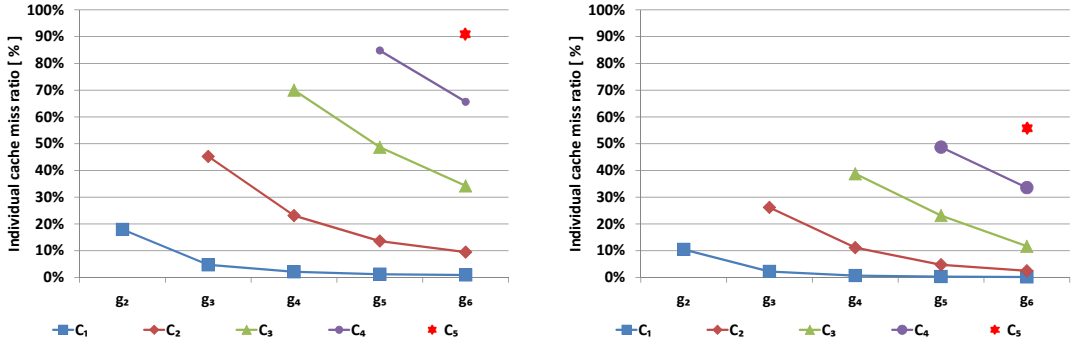
From the Figure 8.19 we conclude:

- The additional memory space due to the Bloom filter map represents only a very small fraction of the total memory space, in fact below 1% (in the case of this experiment), and decreasing with the maximum n-gram size considered (in the glue calculations);

- By filtering the singleton n-grams the total number of entries in each cache is reduced; In the conducted experiment the total amount of memory space per machine needed by the LocalMaxs Global method was significantly reduced for the *corpus* size of 64 Mw, ranging from a reduction of about 50% for the glue g_2 to over 80% for the combined glue $g_{2...6}$.

8.2.4.3 Miss Ratio

Per Machine Individual Cache Miss Ratios. We also evaluated experimentally the per machine individual cache miss ratio for the *corpus* size of 64 Mw with a configuration of $K = 10$ machines. Figure 8.20 shows the per machine average individual cache (C_1 to C_5) miss ratios without and with Bloom filters and Table 8.12 shows the corresponding average values among the K machines, when calculating the isolated glues g_2 to g_6.



a) Without Bloom filters *b) With Bloom filters*

Figure 8.20: Per machine individual cache miss ratios *versus* the isolated glue (g_2 to g_6) without and with Bloom filters for a *corpus* of 64 Mw and $K = 10$. The Y axis represents the miss ratio in percentage and the X axis represents the isolated glue calculation.

Table 8.12: Per machine average of observed individual cache miss ratios *versus* the isolated glue (g_2 to g_6) without and with Bloom filters for a *corpus* of 64 Mw and $K = 10$. Miss ratios percentage (%).

	Cache C_i	g_2	g_3	g_4	g_5	g_6
Without BF	C_1	17.94	4.74	2.12	1.23	0.95
	C_2		45.22	23.12	13.61	9.48
	C_3			69.99	48.67	34.25
	C_4				84.80	65.65
	C_5					90.97
With BF	C_1	10.47	2.24	0.71	0.33	0.19
	C_2		26.17	11.16	4.75	2.50
	C_3			38.74	23.13	11.66
	C_4				48.69	33.57
	C_5					55.89

Per Machine Global Miss Ratio of the Cache System for Combined Glue. Using expression (6.40), presented on page 169, and the values of Table 8.12 we can derive the global miss ratio of the cache systems C_{1+2+3} and $C_{1+\cdots+5}$, respectively, for the combined glues g_{234} and $g_{2\cdots6}$ in the two cases: without Bloom filters and with Bloom filters. These values are presented in Table 8.13.

Table 8.13: Per machine obtained global miss ratios of the cache systems for combined glues g_{234} and $g_{2\cdots6}$ for a *corpus* of 64 Mw and $K = 10$. Miss ratios in percentage.

Miss ratio	**A —Without Bloom filter**	**B — With Bloom filter**
$mr(g_{234})$	29.04	15.75
$mr(g_{2\cdots6})$	35.98	18.82

Figure 8.21 is based on Figure 6.24, presented on page 170, complemented with the values of the global miss ratio for the combined glue g_{234} ($mr(g_{234})$) presented in Table

8.13 (points labeled "A —Without Bloom filters" and "B —With Bloom filters"). The four curves corresponding to the other corpus sizes from 25 Mw to 682 Mw, although corresponding to the case without Bloom filter, are included in this figure for indicative purposes. The behavior when using Bloom filters for those *corpora* is expected to be similar to the case of 64 Mw, depending on the corresponding values of the NSF ratio.

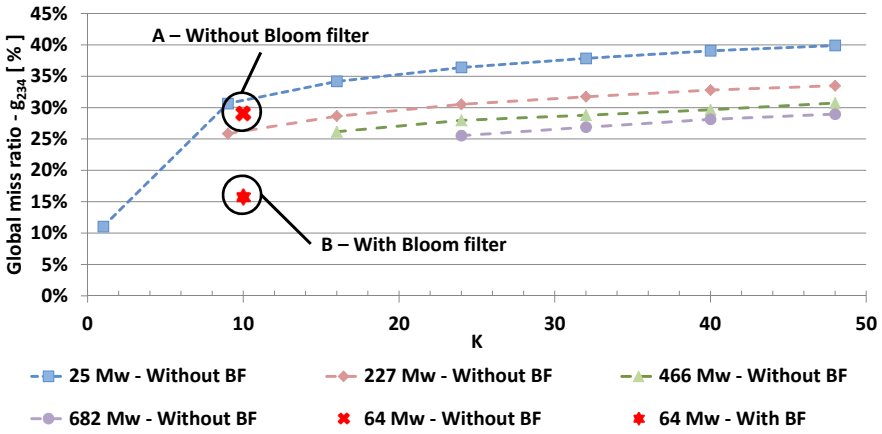


Figure 8.21: Per machine global miss ratio of the cache system C_{1+2+3} with and without Bloom filter, for a *corpus* of 64 Mw, $K = 10$ and combined glue g_{234}. The Y axis represents the global miss ratio in percentage and the X axis represents the number of machines.

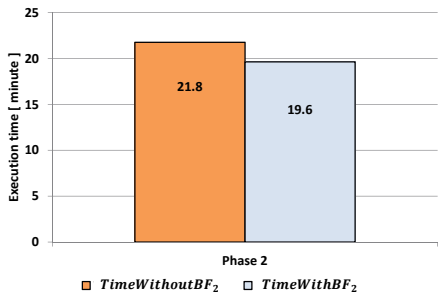
The reported experimental results refer to a single *corpus* size and number of machines, and show a significant reduction in the global miss ratio due to using Bloom filters: around 50% in both the combined glues considered. The behavior of Bloom filters for other *corpus* sizes is determined by the evolution of the NSF_{All} ratio as discussed in section 8.2.1.1 on page 244.

8.2.4.4 Execution Time

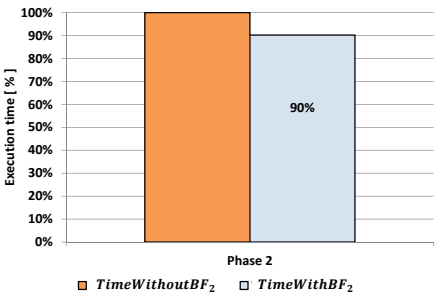
In this section we only describe the execution time for phase two due to the fact that filtering the singleton n-grams will only reduce the miss ratio of the dynamic cache used during phase two. Figure 8.22 shows the observed execution time of phase two needed to find relevant expressions of bigrams up to pentagrams for a *corpus* of 64 Mw when using 10 machines (in the execution environment described in section 8.2.4.1).

The following conclusions can be taken:

- Filtering singletons led to a 10% reduction in the phase two execution time for this experiment;
- This provides the experimental confirmation, for the case of a *corpus* of 64 Mw with 10 machines, that the expected reduction in the execution time in phase two, which is dominated by the miss time penalty, is equal to the NSF ratio. Thus, for other *corpus* sizes it should follow the same behavior as determined by the NSF dependence on the *corpus* size.



a) Absolute phase two execution time

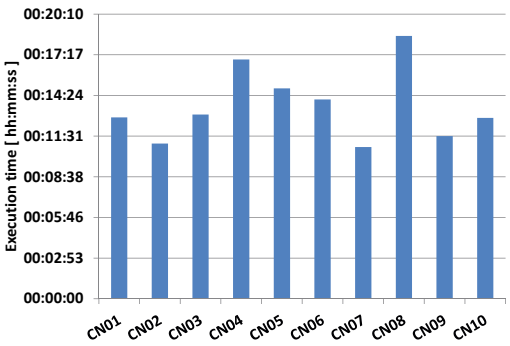


b) Phase two execution time ratio: $\frac{TimeWithBF_2}{TimeWithoutBF_2}$

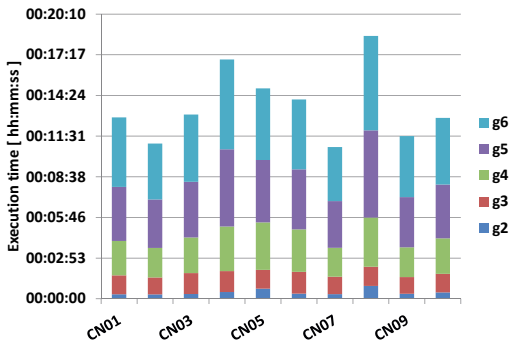
Figure 8.22: Phase two execution time for a *corpus* of 64 Mw, $K = 10$ and combined glue $g_{2...6}$. Subfigure a) represents the execution time (in minutes), and subfigure b) represents the execution time ratio, defined as $TimeWithBF_2/TimeWithoutBF_2$. The darker column in the left of each subfigure represents the phase two execution time without using Bloom filters ($TimeWithoutBF_2$) and the lighter column in the right represents the phase two execution time when using Bloom filters ($TimeWithBF_2$).

Bloom Filters Loading Time. The phase two execution time presented above, when the singleton n-grams are filtered out, already includes the time spent by the controllers to load the Blooms filters from the KVS servers. From the analysis of the execution logs for the execution of the LocalMaxs Global method in phase two for the 64 Mw *corpus* and 10 machines, the time taken to load the Bloom filters, averaged over the 10 machines, was less than 1 second, which in this experiments corresponds to 0.08% of the execution time for phase two, thus it appears to be negligible.

Phase Two Load Balancing with Bloom Filters. Figure 8.23 shows the detailed per machine phase two execution time ($T_2(j)$), among the $K = 10$ machines (individually named from CN_{01} up to CN_{10}). Figure 8.23-a) shows the absolute execution times and Figure 8.23-b) shows the per machine fractions of time taken in average for each isolated glue calculation from g_2 up to g_6.



a) Total



b) Detailed

Figure 8.23: Detailed per machine execution time for phase two using Bloom filters for a *corpus* of 64 Mw and combined glue $g_{2...6}$ and $K = 10$ machines (from CN_{01} up to CN_{10}).

Figure 8.23 shows that there is a good load balancing among all the machines. By using the load balancing metric expression (6.46), on page 178, the corresponding value of the load balancing metric was 73%.

8.3 Extending the Static plus Dynamic Cache with Bloom Filters

As discussed in chapter 7, using a static plus dynamic cache in the case of a system with limited memory capacity allows to reduce the number of cache misses because the dynamic cache contributes to filtering the repetitions of the nonsingleton n-grams that appear as misses out of the static cache, thus leading to a lower or equal miss ratio when compared to a static cache for each given static cache hit ratio. We recall that in the limited memory case, when it is not possible to include all the nonsingleton n-grams in the static cache FAset, the resulting static cache output misses include the remaining nonsingleton occurrences (that are out of the FAset) and all the singleton n-grams.

In this section we show that it is still possible to further reduce the number of cache misses of a static plus dynamic cache system, by removing the singleton n-grams that otherwise will appear as first occurrence misses in the dynamic cache. This reduction can be achieved by integrating a Bloom filter between the static and the dynamic caches, as illustrated in Figure 8.24. This has a similar effect as the inclusion of a Bloom filter in a dynamic cache as discussed in section 8.2.

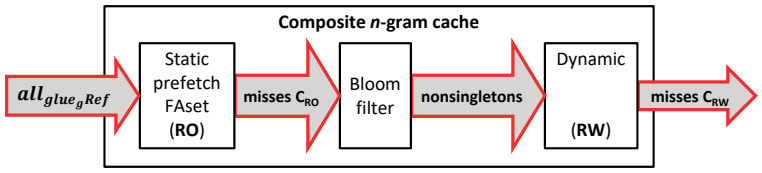


Figure 8.24: Diagram of a static plus dynamic plus Bloom filter cache system.

Cache Size. While for each n-gram size i, the static plus dynamic cache has a size equal to the total number of distinct n-grams ($|D_i|$), the size of a static plus dynamic cache with Bloom filters is equal to $|D_i| - |S_i|$, where $|S_i|$ is the total number of singletons of that size, assuming that the memory space occupied by the Bloom filter bitmap can be neglected. Thus, this reduction on the cache memory size has the same behavior as the number of singleton n-grams as a function of the *corpus* size (Figure 8.11 on page 251).

Global Miss Ratio. The global miss ratio of a static plus dynamic cache system with a Bloom filter for the combined glue g_{234}, assuming the Bloom filter false positive probability is zero, is given by:

$$mr_{(RO+BF+RW)}(C_{1+\cdots+3}, g_{234}) = \frac{|D_{1_{in}D_{234}}| - \left|FA\left(1, g_{234}, h_{RO_{C_1}}\right)\right| - |S_{1_{in}D_{234}}|}{\left|all_{glue_{g_2\cdots g_4}Ref}\right|} + \frac{|D_{2_{in}D_{34}}| - \left|FA\left(2, g_{234}, h_{RO_{C_2}}\right)\right| - |S_{2_{in}D_{34}}|}{\left|all_{glue_{g_2\cdots g_4}Ref}\right|} + \frac{|D_{3_{in}D_4}| - \left|FA\left(3, g_{234}, h_{RO_{C_3}}\right)\right| - |S_{3_{in}D_4}|}{\left|all_{glue_{g_2\cdots g_4}Ref}\right|} \quad (8.18)$$

where D_{234} is the union of the tables of distinct bigrams (D_2), trigrams (D_3) and tetragrams (D_4); $\left|S_{1_{in}D_{234}}\right|$ is the number of singleton unigrams in the *corpus* that appear in D_{234}; D_{34} is the union of the tables of distinct trigrams (D_3) and tetragrams (D_4); $\left|S_{2_{in}D_{34}}\right|$ is the number of singleton bigrams in the *corpus* that appear in D_{34}; and $\left|S_{3_{in}D_4}\right|$ is the number of singleton trigrams in the *corpus* that appear in D_4. If using multiple machines ($K > 1$) we should consider the per machine values.

The above expression simplifies to:

$$mr_{(RO+BF+RW)}(C_{1+\cdots+3}, g_{234}) = mr_{(RO+RW)}(C_{1+\cdots+3}, g_{234}) - \frac{\left|S_{1_{in}D_{234}}\right| + \left|S_{2_{in}D_{34}}\right| + \left|S_{3_{in}D_4}\right|}{\left|all_{glue_{g_2\cdots g_4}Ref}\right|} \quad (8.19)$$

where $mr_{(RO+RW)}(C_{1+\cdots+3}, g_{234})$ is the global miss ratio of the static plus dynamic cache system without Boom filter.

Figure 8.25 shows the decrement of the global miss ratio value due to the inclusion of a Bloom filter in a static plus dynamic cache system for the cases of the combined glues g_{234} and $g_{2\cdots6}$. This corresponds to the second parcel in the right side of expression (8.19).

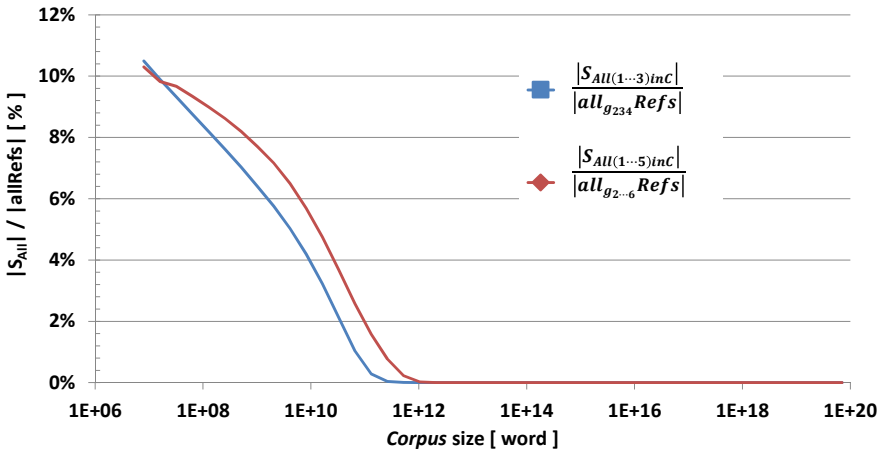


Figure 8.25: Global miss ratio decrement by filtering the singleton n-grams in the case of the combined glues g_{234} and $g_{2\cdots6}$. The Y axis represents the global miss ratio decrement, defined as the percentage of the total number of singletons relative to the total number of input stream references and the X axis represents the *corpus* size in words.

The decrement of the global miss ratio value goes from about 10% for *corpus* sizes around 8 Mw and decreases gradually with increasing *corpus* sizes until it becomes null for *corpus* sizes in the *plateau* regions.

8.4 Chapter Summary

For any of the phase two metrics, cache size, miss ratio and execution time, in the case of a dynamic cache (C_{RW}), the corresponding values when using Bloom filters ($C_{(BF+RW)}$) are given by multiplying the values without Bloom filters (C_{RW}) by the NSF_{All} ratio. Thus, they show a reduction equal to $1 - NSF_{All} = SF_{All}$, that is determined by the singleton ratio. When we consider the variation of the *corpus* size across the entire range up to the *plateau* regions, we observe that the effect of using a Bloom filter diminishes progressively as the singleton ratio SF_{All} tends to zero.

In the case of a static plus dynamic cache ($C_{(RO+RW)}$), the effect of introducing a Bloom filter ($C_{(RO+BF+RW)}$) is also determined by the same NSF_{All} ratio.

Part V

Global Evaluation

Chapter 9

Evaluation of the Parallel and Distributed LocalMaxs Implementations

Global evaluation of the parallel and distributed LocalMaxs implementations.

In this chapter we present an overall evaluation of the Global method (in section 9.1) followed by the description of an alternative method, named the Local method, for implementing LocalMaxs (section 9.2). This new method is based on a completely independent application of the LocalMaxs method to separate *corpus* partitions by using multiple machines, where each machine generates a set of candidate relevant expressions, which are then combined into a global solution. In section 9.3 we present a comparison between the Global and the Local methods concerning their performance and their precision and recall. The chapter summary is presented in section 9.4.

9.1 Global Method

This section addresses issues concerning an evaluation of the memory space (presented in section 9.1.1), an overall discussion of the cache configuration alternatives used in phase two of the Global method (presented in section 9.1.2), and the performance of the Global method in all its phases (presented in section 9.1.3).

9.1.1 Analysis of the Memory Space

We first present a simplified analytical model of the memory space consumption of the Global method that allows us to understand the trends of the memory requirements for the entire range of *corpus* sizes up to the *plateaux*. As this model does not capture the aspects of memory usage due to the multiple layers of the architecture implementation, from the Java virtual machine environment down to the underlying system support layers

and tools, we then describe an experimental approach to measure the actual observed memory consumption by the Global method execution in a real environment.

9.1.1.1 Analytical Modeling of the Memory Space

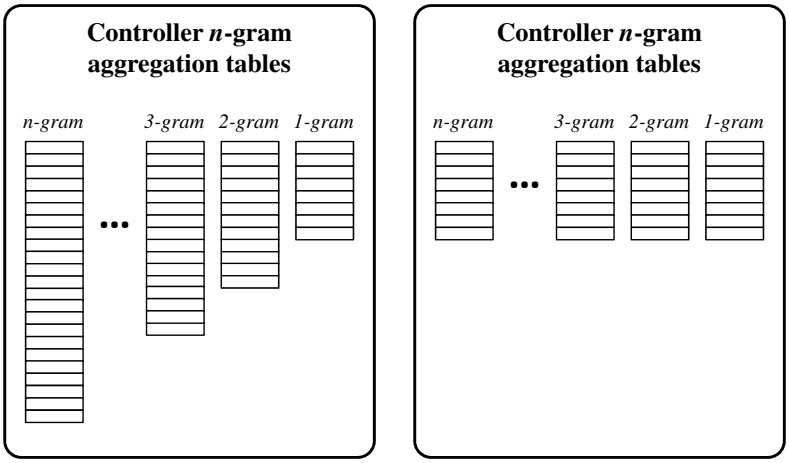
In this section we discuss the memory usage of the Global method as a function of the *corpus* size. For each phase the memory model considers the space occupied by the n-gram tables within each controller and the n-gram tables kept in the distributed in-memory store. We first assume that the space occupied by each table entry is a unit of memory cost and only consider the total number of entries per table. Then we estimate the memory cost in terms of bytes per n-gram entry.

Memory for the n-gram Distributed In-memory Store

For any of the phases the Global method requires the memory space to keep a set of n-gram tables in the KVS servers. Let us consider the total sum of the distinct n-grams in the *corpus*, for n-grams from size 1 until the maximum n-gram size considered (n). This would be the required memory for all the n-gram tables in the case of a single KVS server. In the case of multiple machines (K), the total size of the n-gram tables (measured in terms of number of entries) that are kept within each KVS server is equal to the above mentioned total sum divided by the number of KVS servers. So, the total distributed n-gram store memory requirements grow logarithmically with the *corpus* size until the *plateau* regions. For each fixed *corpus* size the memory per KVS server machine is proportional to $1/K$.

Memory for the n-gram Tables within the Controllers

We consider the memory for each phase separately. In the case of phase one, the memory space occupied concerning the n-gram controller tables, for each n-gram size i, is determined by the size (in terms of the number of the entries) of the table $D_{i_{in}LocalPartition}$ in case of full local aggregation or the table $D_{i_{in}LocalSubPartition}$ in the case of partial local aggregation. Figure 9.1-a) shows the full local aggregation case, where for each n-gram size there is a local table with a size equal to the corresponding number of distinct n-grams per controller corresponding to its local *corpus* partition. Thus, in this case the memory space required per controller grows logarithmically with the partition size (i.e., *corpus* size divided by the number of controllers) until the distinct n-grams *plateaux* are reached, beyond which the per controller memory required becomes constant with the partition size. The case of partial local aggregation is illustrated in Figure 9.1-b) where each controller has a table for each n-gram size but all the tables have a fixed and equal size, configured to fit within the local available per machine memory. Thus, in this case there is always a fixed local memory requirement to keep the local n-gram aggregation tables per controller.



a) *Full local aggregation* ***b)*** *Partial local aggregation*

Figure 9.1: Controller n-gram aggregation tables.

In the cases of phases two and three, there are fixed size components local to each controller such as the memory buffers for the IBS chunks and for the output buffers. Those components are not considered in this analytical model since they represent a constant parcel in the total memory usage. In this model we only consider the memory space required by the local n-gram cache system, whose size depends on the cache system organization as discussed in chapters 6, 7, and 8. In the case of an infinite cache system the size is determined by the population of distinct sub n-grams, while in a finite cache system its size is fixed by design configuration. In the following we only consider the cache memory used in phase two.

For a fixed *corpus* size, when increasing the number of machines K, the size of the *corpus* partitions which are processed in phase one is proportional to $1/K$ but the number of distinct n-grams in each local partition table in the case of full local aggregation follows a logarithmic evolution with the partition size ($|C|/K$). When using partial local aggregation, the local tables size remains constant independently of the partition size.

In the case of phases two and three the size of the distinct n-gram local partition tables is also proportional to $1/K$ (due to the hashing distribution) but the distinct sub n-grams within those tables exhibit a power law evolution with K, determining the memory behavior of the cache system.

Thus, in the case of phase one with full local aggregation and also in phase two, when using an infinite cache, the local memory tables are both determined by the populations of distinct n-grams, leading to similar memory peak usages. On the contrary in the case of phase one with partial local aggregation, because it is limited by the design configuration, the memory peak during phase one is much lower than during phase two with an infinite cache.

Analysis of the Memory Cost per n-gram Entry

From the experiments and empirical analysis conducted we concluded that in the

average the size of an unigram is 6 character symbols. Table 9.1 shows the size in bytes of the primitive types used to support the representation of an n-gram, in the case of the Java language.

Table 9.1: Size in bytes of the primitive types in Java

Type	Size
char	2
int	4
double	8

In the context of the memory usage estimation an n-gram is represented by a string object, and the representation is valid for: i) The in-memory n-gram KVS server tables; ii) The auxiliary tables used by the controllers in all the phases; iii) The cache implementations. The data associated with each n-gram entry is an integer for storing the n-gram frequency count in phase one and is a double to store the other data related to the LocalMaxs definition (the n-gram glue and relevance).

Table 9.2 shows the size, expressed in bytes, needed to represent an n-gram when considering each phase individually (columns "Phase 1" to "Phase 2"). It also shows the per n-gram byte cost in the in-memory tables (column "KVS").

Table 9.2: n-grams sizes, in bytes, for the phases one and two, and for the server tables.

n-gram size	Controllers		KVS
	Phase 1	Phase 2	
1	16	16	32
2	28	28	44
3	40	40	56
4	52	52	68
5	64	64	80
6	76	76	92

In the column named "Phase 1" the space occupied per n-gram in the (full or partial) aggregation tables is presented. It was calculated by summing the space occupied by the n-gram byte representation and the size of the integer for the frequency counter. The n-gram byte space is calculated by multiplying the number of characters per unigram (6), times the number of bytes per character, times the n-gram size (n).

The column named "Phase 2" represents the space occupied per n-gram in the cache. It was calculated by summing the n-gram byte space calculated as for the "Phase 1" column and the size of the integer for the frequency counter.

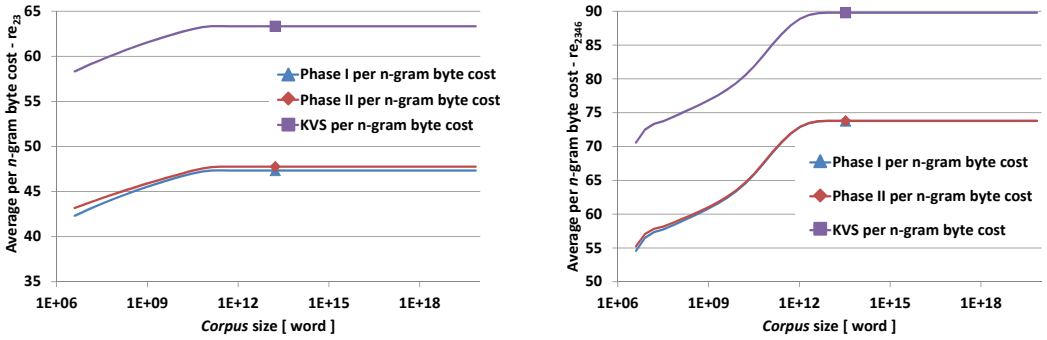
If a cache was used in phase three, each entry would occupy a number of bytes calculated as for phase two cache entries except that then there would be a double to represent the glue values instead of an integer (that is used for the frequency count).

When evaluating relevant expressions $re_{2...n}$ from bigrams up to a maximum n-gram

size ($n \geq 2$), the following n-gram sizes must be considered across the three phases: unigrams, bigrams, $\cdots$ up to n-grams of size $(n+1)$. As each n-gram size has a different byte cost, we consider an average per n-gram byte cost ($mSize_{n\text{–}gram}$, corresponding to the $EntrySize$ parameter on chapter 8) of an n-gram over the range of the considered n-gram sizes, defined as follows:

$$mSize_{n\text{–}gram} = \frac{\sum_{i=1}^{m} (sizeof\ (i\text{–}gram) \times |D_i|)}{\sum_{i=1}^{m} |D_i|} \tag{9.1}$$

where $sizeof\ (i\text{–}gram)$ represents the size in bytes of the representation of an n-gram of size i in each of the columns in the above table; $|D_i|$ represents the number of distinct n-grams of size i; and m is equal to $(n+1)$ for "Phase 1" and "KVS" columns, and is equal to n in the case of "Phase 2" column. Figure 9.2 shows the average per n-gram byte cost ($mSize_{n\text{–}gram}$) *versus* the *corpus* size for the columns presented in Table 9.2.



a) *Average per n-gram byte cost* re_{23}

b) *Average per n-gram byte cost* $re_{2...5}$

Figure 9.2: Average per n-gram byte cost *versus corpus* size. The Y axis represents the average per n-gram byte cost and the X axis represents the *corpus* size in words.

Figure 9.2-a) corresponds to the case of re_{23}, by using $n = 3$ in the definition of the value of m that is used in expression (9.1); and Figure 9.2-b) corresponds to the case $re_{2...5}$, by using $n = 5$ in the definition of the value of m that is used in expression (9.1).

Table 9.3 shows the values of the average per n-gram byte cost, for the curves presented in Figure 9.2, when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 9.3: Average per n-gram byte cost, in bytes, *versus corpus* size (in words).

$\|C\|$	re_{23}			$re_{2...5}$		
	Phase 1	**Phase 2**	**KVS**	**Phase 1**	**Phase 2**	**KVS**
$\mathbf{1.6 \times 10^7}$	43.24	43.92	59.24	57.35	57.84	73.35
$\mathbf{1.3 \times 10^8}$	44.46	44.95	60.46	58.94	59.26	74.94
$\mathbf{1.0 \times 10^9}$	45.55	45.91	61.55	60.84	61.05	76.84
$\mathbf{1.6 \times 10^{10}}$	46.77	47.07	62.77	64.53	64.65	80.53
$\mathbf{1.3 \times 10^{11}}$	47.34	47.71	63.34	69.11	69.19	85.11
$\mathbf{1.0 \times 10^{12}}$	47.33	47.77	63.33	72.87	72.91	88.87
$\mathbf{1.7 \times 10^{13}}$	47.33	47.77	63.33	73.78	73.80	89.78
$\mathbf{1.3 \times 10^{14}}$	47.33	47.77	63.33	73.78	73.80	89.78

The following conclusions can be taken:

- The average per n-gram byte costs increase with the *corpus* size and they stabilize when the *plateaux* are reached;
- They also increase with the maximum n-gram size (n) considered in the relevant expression evaluation.

As mentioned in chapter 1, this thesis did not focus on the optimizations of the memory representation of the n-gram data structures on each machine. Such optimizations can significantly contribute to reduce the above per n-gram byte cost.

Memory Space of the Controllers and KVS Servers

When evaluating relevant expressions from bigrams up to n-grams of size n, we can evaluate the memory space of the local controller tables in phase one or the cache system in phase two, by multiplying the corresponding total number of n-gram entries, times the values of the corresponding average per *n-gram* byte costs.

$$MemorySpaceController = \sum_{i=1}^{m} |D_i| \times mSize_{n\text{–}gram} \tag{9.2}$$

where $mSize_{n\text{–}gram}$ is obtained from column "Phase 1" or "Phase 2" in Table 9.3 and m is equal to $(n+1)$ for phase one and is equal to n in the case of phase two. In phase one D_i corresponds to the set of distinct n-grams of size i in the local partition, and in phase two D_i corresponds to the distinct sub n-grams of size i in the local cache system.

The total memory space for the in-memory n-gram tables is calculated as follows:

$$MemorySpaceKVS = \sum_{i=1}^{n+1} |D_i| \times mSize_{n\text{–}gram}(KVS) \tag{9.3}$$

where $mSize_{n\text{–}gram}$ is obtained from column "KVS" in Table 9.3 and $|D_i|$ is the total number of distinct n-grams of size i in the *corpus*. In a distributed in-memory store with K servers, the memory per server is $MemorySpaceKVS/K$.

By using the expressions (9.2) and (9.3) we can estimate the total memory space, for different problems, as a function of the *corpus* size, assuming a single machine case ($K = 1$). Table 9.4 shows the values of the memory usage estimates (expressed in Terabyte (10^{12}) (TB)), when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 9.4: Memory space estimates, in TB, *versus corpus* size (in words).

$\|C\|$	re_{23}			$re_{2...5}$		
	Phase 1	**Phase 2**	**KVS**	**Phase 1**	**Phase 2**	**KVS**
$\mathbf{1.6 \times 10^7}$	0.002	0.002	0.003	0.003	0.003	0.004
$\mathbf{1.3 \times 10^8}$	0.017	0.017	0.023	0.022	0.022	0.028
$\mathbf{1.0 \times 10^9}$	0.114	0.115	0.154	0.152	0.152	0.192
$\mathbf{1.6 \times 10^{10}}$	1.187	1.195	1.593	1.638	1.641	2.044
$\mathbf{1.3 \times 10^{11}}$	4.979	5.018	6.661	7.269	7.277	8.952
$\mathbf{1.0 \times 10^{12}}$	13.955	14.083	18.673	21.486	21.496	26.203
$\mathbf{1.7 \times 10^{13}}$	19.666	19.847	26.314	30.654	30.664	37.301
$\mathbf{1.3 \times 10^{14}}$	19.666	19.847	26.314	30.654	30.664	37.302

All the memory estimates presented above follow a logarithmic increase when the *corpus* size increases until reaching a *plateaux*, having the same behavior as the number of distinct n-grams (Figure 5.15 on page 105).

The following experimental results present a complementary view to the above analysis, because they allow to capture the memory space occupation of the runtime components as well as provide a global view of the actual memory space used in each phase of the Global method execution on real computing infrastructures.

9.1.1.2 Experimental Evaluation of the Memory Space

Experimental Evaluation Methodology

The simplified analysis of the memory space presented in the above section does not consider several constant scale factors which are determined by the implementation, e.g., the storage cost of the n-grams table and cache entries implemented as Java objects, and the memory consumption due to the dynamic instantiation of objects and threads by the Java runtime environment. Thus, an empirical approach was followed to characterize the memory behavior of the Global method, for different *corpus* sizes and numbers of machines, based on the memory profiling of the virtual machines used.

In this evaluation we do not consider the memory requirements of the underlying host system runtime environment that supports the execution of the logical virtual machines defined in section 4.2 (on page 63). We only evaluate the memory space occupied by the LocalMaxs Global method architecture components and by their associated runtime environment within each logical virtual machine, namely: including the guest operating system, the Java Virtual Machine environment and the workflow framework (denoted as memory M_{RT}); the controller process (denoted as memory $M_{Controller}$); and the KVS server process (denoted as memory M_{KVS}). In all the configurations considered each logical virtual machine includes an instance of one controller process and one KVS server process. Thus, the total memory required in each phase by each logical virtual machine is given by:

$$M_{Total} = M_{RT} + M_{Controller} + M_{KVS} \tag{9.4}$$

The performed measurements provide indications of the global memory occupation of the controller, or the KVS server process, or the runtime of the guest operating system, in a macroscopic view that does not provide detail on the internal behavior of those components.

Concerning the M_{RT} component, all experimentation was conducted using Linux as the guest operating system within each virtual machine, and the memory required by the Java Virtual Machine and the workflow runtime environment did not present significant changes across the considered experimental scenarios.

The $M_{Controller}$ component includes the memory of all the objects generated within the controller process during the execution of the LocalMaxs Global method phases, also including the cache component as this is also instantiated as an object inside the controller process.

The M_{KVS} component includes the memory of all the objects generated within the KVS server process. This includes the n-gram table partitions kept by the server located in each machine.

The memory usage of the components that support the execution of the Global method was obtained using profiling tools. Two additional task nodes were added to the workflow presented in Figure 5.3, on page 80. A first task, added in the beginning of the workflow, was used to start a background script on each machine, and a final task, added at the end of the workflow, was used to terminate the background scripts. During the execution of each phase of the LocalMaxs Global method the background script periodically samples the total memory used (parcel M_{Total} in expression (9.4)), using the "free" command available in the Linux operating system; and the memory used by each Java Virtual Machine, using the "jstat"[1] tool (measuring the parcels $M_{Controller}$ and M_{KVS}). The measurements are saved into a log file, one per machine, and are aggregated by the final task. The value of the time sample interval between the measurements is configurable (in the experiments reported here, it was set to 4 seconds).

Using this approach the parcel M_{RT} can be obtained by:

$$M_{RT} = M_{Total} - (M_{Controller} + M_{KVS}) \tag{9.5}$$

We used this methodology to observe the memory usage in the Global method, by evaluating its behavior along the three phases of the algorithm execution, using an infinite cache system without and with a Bloom filter, and also to understand the influence of the *corpus* size and the number of machines upon the memory behavior. The main goal of this evaluation was to understand the global behavior of the memory usage, namely, its

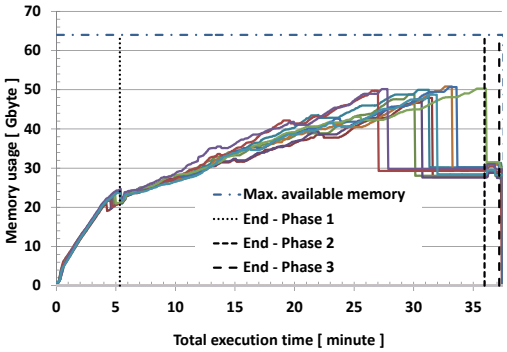
[1] jstat — Java Virtual Machine Statistics Monitoring Tool. This tool is bundled with the Oracle JDK and displays performance statistics for an instrumented HotSpot Java virtual machine (JVM)

peak instantaneous values per machine, and its evolution with the number of machines and the *corpus* size.

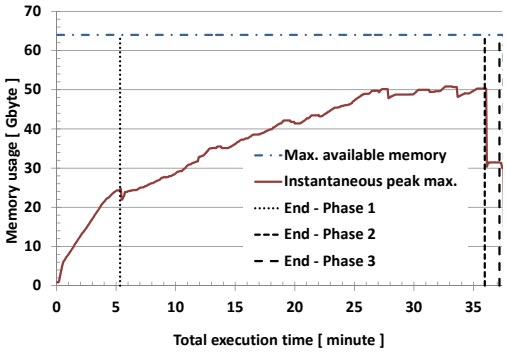
Experimental Results for a Fixed Size *Corpus* Using an Infinite Dynamic Cache

Applying the above methodology to a concrete case we present the experimental evaluation of the memory usage during the Global method execution, for the relevant expressions $re_{2...6}$ using a configuration (denoted as "Environment C") with an infinite dynamic cache system for a *corpus* of 64 Mw and 10 machines. This experiment was conducted in the Lunacloud public cloud and each machine was configured with 4 vCPU and 64 GB of RAM. Within each machine there was one controller for each phase and one KVS server.

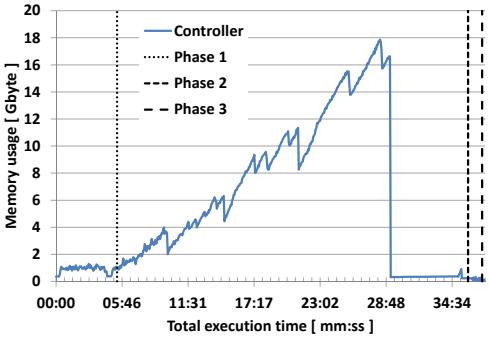
Figure 9.3 illustrates the results obtained. Figure 9.3-a) shows the instantaneous total memory on each of the 10 machines, where each curve represents a distinct machine, for the successive phases of the LocalMaxs Global method execution. The curve in Figure 9.3-b) represents the peak value of the instantaneous total memories among the 10 machines. The dotted vertical lines delimit the three phases of the LocalMaxs Global method. Figures 9.3-c) to 9.3-g) illustrate the evolution of the memory usage for one sample machine. Figure 9.3-c) represents the controller heap memory for each phase, Figure 9.3-d) represents the KVS server heap memory, and Figure 9.3-e) shows the total heap memory of all the machine components (the controller, which includes the cache system, and the KVS server), corresponding to an estimate of the sum $(M_{Controller} + M_{KVS})$. Figure 9.3-f) gathers the curves in Figure 9.3-b) and Figure 9.3e), and their difference is shown in Figure 9.3-g), which is an estimate of the memory of the runtime environment (M_{RT} as in expression (9.5)). Also indicated as a dotted line is the average memory of the runtime component over the entire duration of the algorithm.



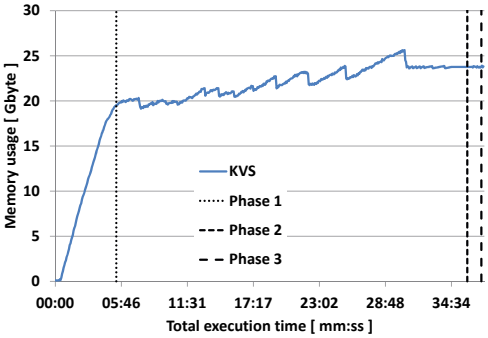
a) Instantaneous per machine total memory (M_{Total})



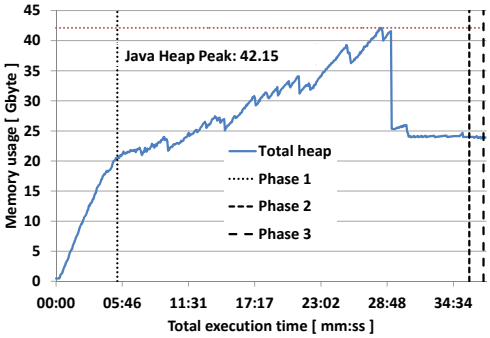
b) Instantaneous peak total memory among all the machines (M_{Total})



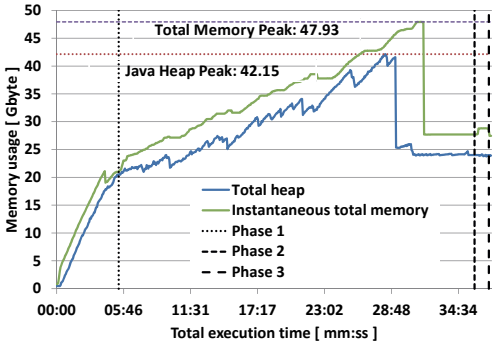
c) Controller plus cache heap memory of a sample machine ($M_{Controller}$)



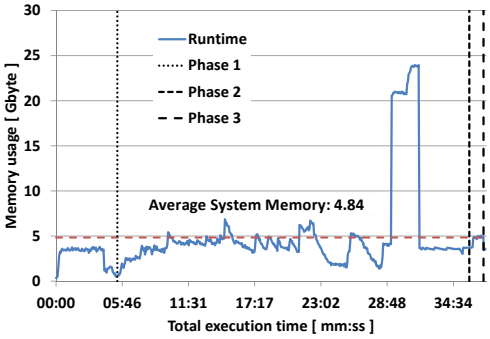
d) KVS heap memory of a sample machine (M_{KVS})



e) Heap total memory of a sample machine ($M_{Controller} + M_{KVS}$)



f) Instantaneous total memory (M_{Total}) and heap memory ($M_{Controller}$+M_{KVS}) of a sample machine

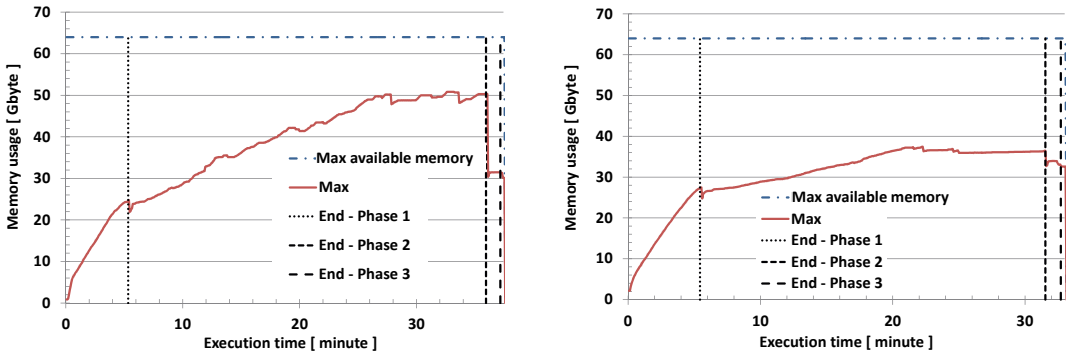


g) Memory of the runtime (M_{RT}) for a sample machine

Figure 9.3: Per machine memory usage for $re_{2...6}$, *corpus* size of 64 Mw and $K = 10$. The Y axis represents the memory in GB and the X axis represents the execution time in minutes.

Experimental Results for a Fixed Size *Corpus* Using an Infinite Dynamic Cache with a Bloom Filter

As discussed in chapter 8 the usage of Bloom filters allows to filter the singletons out of a dynamic cache system. The configuration used in "Environment C", above described, was extended with a Bloom filter and its impact upon the memory usage of the Global method execution was evaluated. Figure 9.4 shows the memory usage of the Global method when evaluating relevant expressions $re_{2...6}$ for a *corpus* of 64 Mw, 10 machines and a infinite dynamic cache, in two cases: without Bloom filter (Figure 9.4-a)); and with Bloom filter (Figure 9.4-b)). The figure shows the instantaneous peak total memory among all the machines in those two cases.



a) *Without Bloom filter* ***b)*** *With Bloom filter*

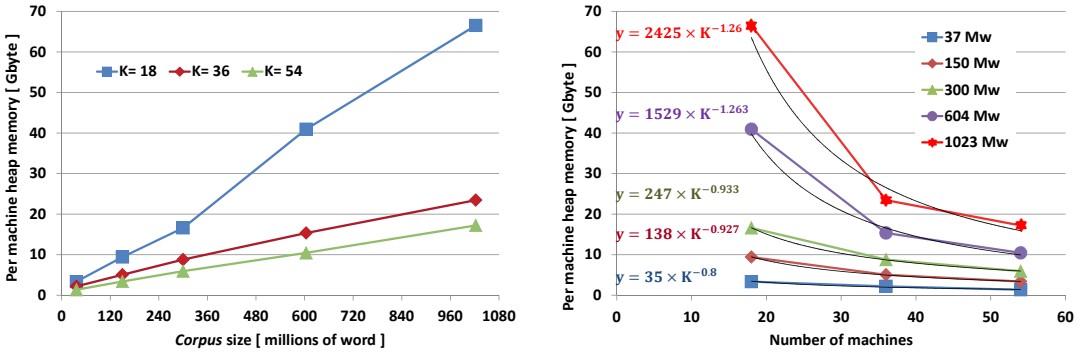
Figure 9.4: Per machine memory usage reduction due to Bloom filter for $re_{2...6}$ and a *corpus* size of 64 Mw and $K = 10$. The Y axis represents the memory usage in GB and the X axis represents the execution time in minutes.

The obtained results for this experiment show:

- Without Bloom filters the maximum per machine peak memory among all the machines is around 51 GB and with Bloom filter the maximum peak was around 37 GB;
- The usage of Bloom filters to exclude the singleton n-grams out of the dynamic cache, reduced the maximum per machine peak memory requirement from 51 to 37 GB, representing a reduction of almost 27% = 1 – 37/51.

Experimental Results for the Heap Memory Usage ($M_{Controller} + M_{KVS}$) when Varying the *Corpus* Size and Numbers of Machines

Figure 9.5 shows the evolution of the sum of total heap memory used by all the components (the controller and the KVS) per machine used in "Environment A" presented in section 6.5 (on page 160), i.e., evaluating re_{23} for different *corpus* sizes (37 up to 1023 Mw) and number of machines (18 virtual machines, each with 70 GB of RAM; 36 virtual machines, each with 32 GB of RAM; and 54 virtual machines, each with 20 GB of RAM). The total heap memory corresponds to the per machine memory due to the KVS servers and the controllers.



a) Corpus versus K

b) K versus corpus

Figure 9.5: Average total heap memory per machine for combined glue g_{234}, for different *corpus* sizes and numbers of machines. The Y axis represents the average total heap memory per machine in GB. In subfigure a) the X axis represents the *corpus* size in millions of words and in subfigure b) the X axis represents the number of machines.

Figure 9.5-a) shows the evolution of the total heap memory *versus* the *corpus* size for different numbers of machines ($K = 18$, $K = 36$ and $K = 54$). Figure 9.5-b) shows the evolution of the total heap memory for different *corpus* sizes (37 up to 1023 Mw) *versus* the number of machines. Table 9.5 shows the values corresponding to Figure 9.5.

Table 9.5: Average total heap memory per machine in GB for re_{23}, for different *corpus* sizes and numbers of machines.

K	**37 Mw**	**150 Mw**	**300 Mw**	**604 Mw**	**1023 Mw**
18	3	9	17	41	67
36	2	5	9	15	23
54	1	3	6	10	17

The following conclusions can be taken:

- From Figure 9.5 it is possible to determine the minimal number of machines necessary to ensure the heap memory requirements of the algorithm, for a given *corpus* size, and considering the available maximum system memory per machine supported by the environment;
- From Figure 9.5-b) one can conclude that in the considered ranges of *corpus* sizes and numbers of machines (K) the total heap memory consumption per machine follows a power law as a function of K, as illustrated by the fitting expressions of the empirically obtained curves. As the total heap memory values include both the memory space due to the controller and the server in each machine, the global behavior observed at this macroscopic level is the outcome of the combination of the memory behaviors of the controller and the server;
- These results illustrate how the maximum memory requirement of the Global method for a specific problem size (the *corpus* size $|C|$ and the value of n in $re_{2...n}$) can

be satisfied by adjusting the architecture configuration with an adequate number of machines (K).

9.1.2 Comparison of the Cache Approaches

The multiple alternatives for the configuration of the n-gram cache system are illustrated in Figure 9.6 where each node corresponds to a specific cache configuration and the arcs represent the possible transitions (numbered from 1 to 7 in the figure) between different configurations.

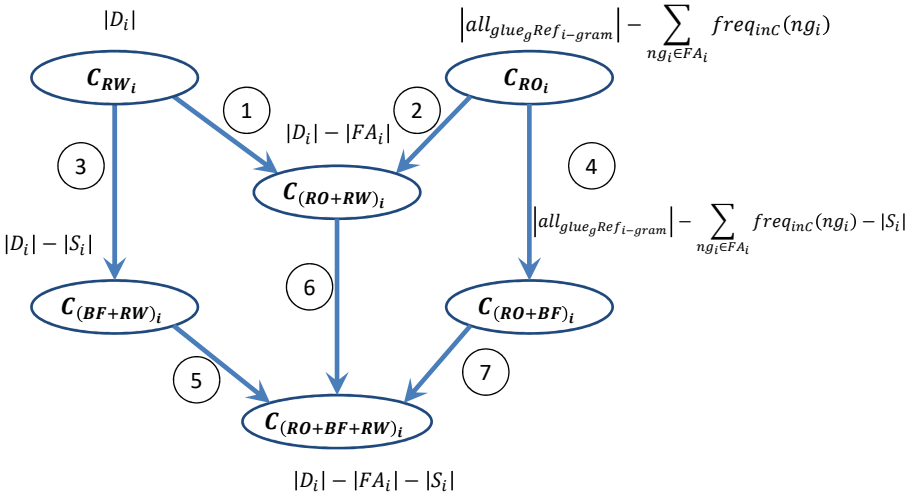


Figure 9.6: Cache configurations comparison.

The figure also shows the number of n-gram misses for each individual cache C_i configuration assuming an infinite dynamic cache, where $|D_i|$ and $|S_i|$ represent, respectively, the numbers of distinct and singleton n-grams of size i in the *corpus*, $\left|all_{glue_g Ref_{i-gram}}\right|$ is the total number of references in the cache input reference stream, and $|FA_i|$ is the size of the FAset for the n-grams of size i. The figure applies to the case of the individual caches for each n-gram of size i but can easily be adapted to the case of the entire cache system, by considering the global number of misses.

The transitions indicate the possible ways of designing a complete cache system configuration for the Global method, capable of achieving a desired target global miss ratio for a given *corpus* size and considering a glue calculation for a maximum n-gram size n.

Infinite Dynamic Cache. Although a dynamic cache ($C_{(RW)_i}$) with infinite capacity, able to contain all the distinct n-grams of size i, reduces the miss communication overhead compared to a system without cache, it still exhibits a significant miss ratio that is determined by the cold-start misses. On one hand, a subset of these misses is due to singleton n-grams which can be filtered by using a Bloom filter combined with the dynamic cache ($C_{(BF+RW)_i}$), allowing to achieve a reduction in the miss ratio although this is only effective in the *corpus* size range where singletons represent a significant percentage of the distinct

n-grams of that size. On the other hand, the time penalty of the cold-start misses can be reduced by performing a static prefetching of a subset of n-grams.

Infinite Static Cache. The latter improvement on the miss penalty can be achieved by using a single static cache ($C_{(RO)_i}$) with a FAset containing all the nonsingleton n-grams of size i, thus providing the best solution if there is enough memory capacity. In this case the global miss ratio of the cache system is virtually zero, assuming that the loading of the FAsets can be performed, for all the n-gram sizes required, with total time overlap with other useful computations.

Static Cache with Bloom Filter. However, if there is not enough memory to contain all the nonsingleton n-grams of size i, a static cache (C_{RO_i}) exhibits misses that include both nonsingleton and singleton n-grams. The singletons can be filtered out by using a Bloom filter combined with a static cache ($C_{(RO+BF)_i}$).

Static plus Dynamic Cache. Concerning the nonsingletons that were not included in the static cache FAset, their repetitions originate multiple misses out of the static cache, which can be filtered out by adding a dynamic cache following the static cache. This cache configuration ($C_{(RO+RW)_i}$) always exhibits a global miss ratio lower or equal than the miss ratios of a single dynamic cache or a single static cache. However, this latter configuration ($C_{(RO+RW)_i}$) requires an infinite memory capacity able to contain all the distinct n-grams of size i, i.e., the nonsingletons in the static cache FAset plus the remaining nonsingletons and the singletons, all of which must kept in the dynamic cache.

Static plus Dynamic Cache with Bloom Filter. Eventually we can reduce the infinite memory requirement of the above $C_{(RO+RW)_i}$ configuration by including a Bloom filter, which leads to a configuration of a static plus dynamic cache with Bloom filter ($C_{(RO+BF+RW)_i}$) whose required memory capacity is equal to the number of nonsingleton n-grams of size i. If there is not enough memory to contain all the nonsingleton n-grams for that size, the best possible cache configuration is the result of a trade-off involving: a static cache, a finite capacity dynamic cache and a Bloom filter. Assuming that we ignore the memory space due to the Bloom filter bitmap, this configuration allows to split the total available memory capacity into a part for the static cache FAset and the remaining for the finite dynamic cache. As a result of the finite cache capacity in this case, besides the cold-start misses, there are capacity misses out of the dynamic cache, which depend on the size of the dynamic cache and on the cache replacement strategy used.

Tables 9.6 and 9.7 summarize the multiple cache alternatives in terms of cache miss ratio and size for a single machine case considering $re_{2...5}$, infinite dynamic cache capacity, and three typical *corpus* sizes in the beginning, middle and end of the *corpus* sizes spectrum.

Table 9.6: Cache alternatives — Miss ratio and size.

Cache Type	Miss Ratio	Size
C_{RW}	$mr_{RW} = \frac{\lvert D_{All(1\cdots5)_{in}C}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$	$\lvert D_{All(1\cdots5)_{in}C}\rvert$
C_{BF+RW}	$\frac{\lvert D_{All(1\cdots5)_{in}C}\rvert - \lvert S_{All(1\cdots5)_{in}C}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$	$\lvert D_{All(1\cdots5)_{in}C}\rvert - \lvert S_{All(1\cdots5)_{in}C}\rvert$
C_{RO}	$\left(1 - h_{RO_{g_{2\cdots6}}}\right) = mr_{RO}$	$\lvert FA_{1\cdots5}\rvert$
C_{RO+BF}	$mr_{RO} - \frac{\lvert S_{All(1\cdots5)_{in}C}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$	$\lvert FA_{1\cdots5}\rvert - \lvert S_{All(1\cdots5)_{in}C}\rvert$
C_{RO+RW}	$mr_{RW} - \frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$	$\lvert D_{All(1\cdots5)_{in}C}\rvert$
$C_{RO+BF+RW}$	$mr_{RW} - \frac{\lvert FA_{1\cdots5}\rvert + \lvert S_{All(1\cdots5)_{in}C}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$	$\lvert D_{All(1\cdots5)_{in}C}\rvert - \lvert S_{All(1\cdots5)_{in}C}\rvert$

Table 9.7: Cache alternatives — Examples of miss ratio and cache size values.

Cache Type	Small *corpus* (10 Mw)	Medium *corpus* (1 Gw)	Large *corpus* (1 Tw)
C_{RW}	11.06% $size$=2.3×10^7	9.02% $size$=1.8×10^9	2.11% $size$=6.5×10^{10}
C_{BF+RW}	0.76% $size$=1.6×10^6	1.32% $size$=2.7×10^8	2.08% $size$=6.4×10^{10}
C_{RO}	43.34% $\rightarrow$ 0%* $size$=1.6×10^6	27.70% $\rightarrow$ 0%* $size$=2.7×10^8	0.04% $\rightarrow$ 0%* $size$=6.4×10^{10}
C_{RO+BF}	mr_{RO} − 10.30% $size$=$\lvert FA_{1\cdots5}\rvert - 2.1\times10^7$	mr_{RO} − 7.70% $size$=$\lvert FA_{1\cdots5}\rvert - 1.6\times10^9$	mr_{RO} − 0.02% $size$=$\lvert FA_{1\cdots5}\rvert - 6.9\times10^8$
C_{RO+RW}	11.06% − $\frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$ $size$=2.3×10^7	9.02% − $\frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$ $size$=1.8×10^9	2.11% − $\frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$ $size$=6.5×10^{10}
$C_{RO+BF+RW}$	0.76% − $\frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$ $size$=1.6×10^6	1.32% − $\frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$ $size$=2.7×10^8	2.08% − $\frac{\lvert FA_{1\cdots5}\rvert}{\lvert all_{glue_{g_{2\cdots6}Ref}}\rvert}$ $size$=6.4×10^{10}

$\star$Assuming total FAset loading time overlap and $FA_{1\cdots5} \equiv NS_{All(1\cdots5)_{in}C}$

The following observations can be made:

✦ For the C_{RW} cache we indicate its maximum size determined by the total number

of distinct n-grams and the corresponding global miss ratio of the cache $C_{1+\cdots+5}$ system; This corresponds to the global warm-start miss ratio as defined in chapter 6; The miss ratio goes from 11% in the 10 Mw *corpus* to 2% in the 1 Tw *corpus*;

- For the C_{BF+RW} cache its maximum size is determined by the total number of non-singleton n-grams and the corresponding global miss ratio goes from 0.8% in the 10 Mw *corpus* (due to the Bloom filter effect) to 2% in the 1 Tw *corpus* (where the singletons have practically disappeared);

- For the C_{RO} cache the columns in Table 9.6 show the generic expression for the global miss ratio and its corresponding FAset size; The columns in Table 9.7 correspond to the case where the FAset is equal to the set of all nonsingletons ($NS_{All(1\cdots5)_{in}C}$), which determines the cache size; The global miss ratio values of 43.34%, 27.70% and 0.04% indicated, respectively, for the 10 Mw, 1 Gw and 1 Tw *corpora*, result from expression (7.44) on page 210 and are exclusively due to the singleton n-grams which were not captured by the FAset; However, those singleton misses do not involve any remote fetching overhead; If, beyond that, we ensure that the FAset building and loading is performed with total time overlap with the glue calculations, then the miss ratio is virtually zero, as indicated by the arrow ($\rightarrow$ 0%);

- For the C_{RO+BF} cache both its size and its global miss ratio get reduced by the effect of filtering the singleton n-grams;

- For the C_{RO+RW} cache its size is the same as a single infinite dynamic cache and its global miss ratio, when compared to the single dynamic cache, gets reduced by the effect of the FAset elements;

- For the $C_{RO+BF+RW}$ cache both the filtering of singleton n-grams and FAset elements take effect.

9.1.3 Overall Evaluation of the Performance of the Global Method

In section 9.1.3.1 we present an analysis of the performance of the Global method over the entire *corpus* range by relying upon the estimated data from the theoretical model presented in chapter 3, and in section 9.1.3.2 we analyze the experimental performance results of the Global method for evaluating relevant expressions of bigrams and trigrams for *corpus* sizes up to 1 Gw.

9.1.3.1 Global Method Performance Prediction Over the Entire *Corpus* Range

Efficiency

For the Global method execution as a sequence of the three phases, the efficiency (E_{GM}) relative to an ideal sequential machine is given by:

$$E_{GM} = \frac{T_{0_{GM}}}{K \times T_{GM}(j)} \tag{9.6}$$

where $T_{0_{GM}}$ is the total computation time of the Global method in an ideal sequential machine without any overheads, being equal to the sum of the ideal computation times of the individual phases, i.e., T_{0_1} for phase one ($N_n(phase1) \times t_{op}$), T_{0_2} for phase two ($N_n(phase2) \times t_{g_n}$) and T_{0_3} for phase three ($N_n(phase3) \times t_{re}$); and $T_{GM}(j)$ is the total execution time of the Global method in a real machine (j) in a configuration with K machines $1 \leq j \leq K$, where only the communication overheads of each of the three phases are considered as discussed in chapter 5. From this we can obtain the following expression:

$$\frac{T_{0_{GM}}}{E_{GM}} = K \times T_{GM}(j) = K \times (T_1(j) + T_2(j) + T_3(j)) \tag{9.7}$$

where $T_1(j)$, $T_2(j)$, and $T_3(j)$ are the total execution times of each phase (1, 2, 3) of the Global method in a real machine j when using K machines. Expression (9.6) also applies to the efficiency of each individual phase (i, $1 \leq i \leq 3$):

$$E_{GM_i} = \frac{T_{0_i}}{K \times T_i(j)} \tag{9.8}$$

Thus, the Global method efficiency E_{GM} is related to individual phase efficiencies through the following expression:

$$\frac{T_{0_{GM}}}{E_{GM}} = \frac{T_{0_1}}{E_{GM_1}} + \frac{T_{0_2}}{E_{GM_2}} + \frac{T_{0_3}}{E_{GM_3}} = \frac{N_n(phase1) \times t_{op}}{E_{GM_1}} + \frac{N_n(phase2) \times t_{g_n}}{E_{GM_2}} + \frac{N_n(phase3) \times t_{re}}{E_{GM_3}} \tag{9.9}$$

Assuming a strict sequential execution of the three phases of the Global method then $T_{0_{GM}} = T_{0_1} + T_{0_2} + T_{0_3}$. Additionally assuming that $t_{op} \approx t_{g_n} \approx t_{re_n}$, then the efficiency of the Global method for evaluating relevant expressions $re_{n_1 \cdots n_2}$ is given by:

$$E_{GM} = \frac{Nn(phase1) + Nn(phase2) + Nn(phase3)}{\frac{Nn(phase1)}{E_{GM_1}} + \frac{Nn(phase2)}{E_{GM_2}} + \frac{Nn(phase3)}{E_{GM_3}}} \tag{9.10}$$

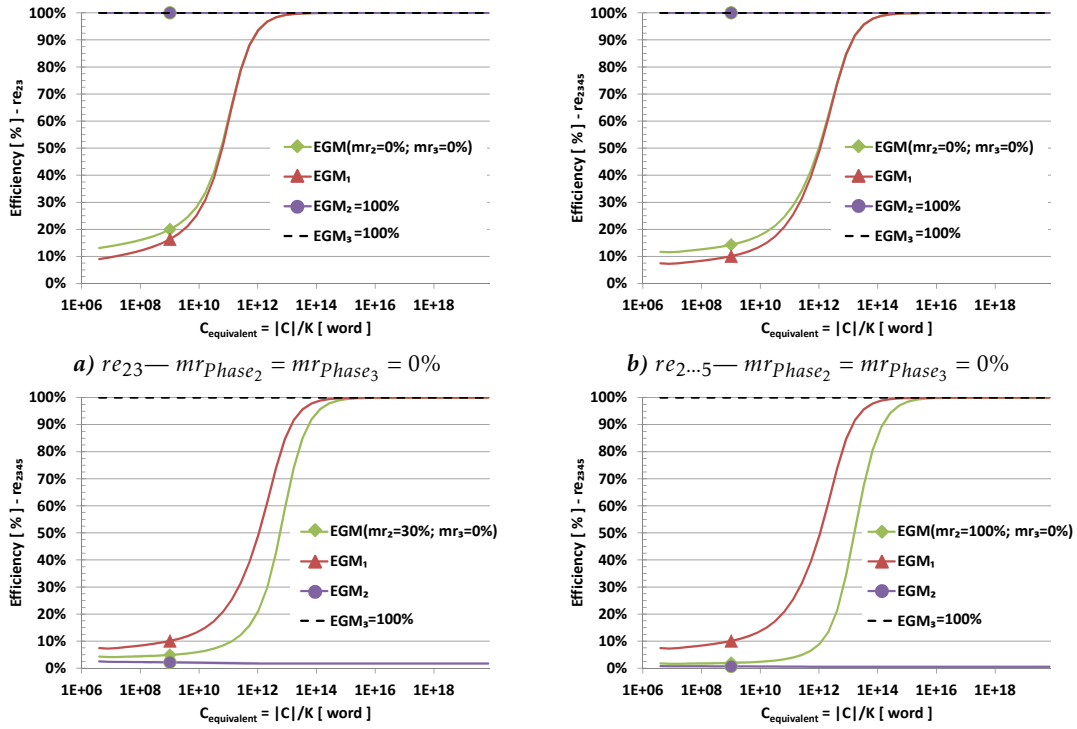
$$N_n(phase1) = |C| \times (n_2 + 1), \quad N_n(phase2) = \sum_{i=n_1}^{n_2+1} |D_i|, \quad N_n(phase3) = \sum_{i=n_1}^{n_2} |NS_i| \tag{9.11}$$

where $|D_i|$ is the total number of distinct n-grams of size i in the *corpus*, $|NS_i|$ is the total numbers of nonsingleton n-grams of size i in the *corpus*, $Nn(phase1)$, $Nn(phase2)$ and $Nn(phase3)$ are, respectively, the problem sizes for phase one, two and three; and E_{GM_1}, E_{GM_2} and E_{GM_3} are, respectively, the efficiencies in phase one, two and three with a configuration with K machines, as given by expression (9.8).

The influence of the efficiencies E_{GM_1} and E_{GM_2} upon the global efficiency E_{GM} is determined by the parcels $N_n(phase1)/E_{GM_1}$ and $N_n(phase2)/E_{GM_2}$ in expression (9.10). Although any increase in the efficiencies of either E_{GM_1} or E_{GM_2} contributes to increasing

E_{GM}, the contribution from E_{GM_1} becomes stronger and stronger as the *corpus* size increases, due to the linear growth of $N_n(phase1)$ with the *corpus* size compared to the slower logarithmic growth of $N_n(phase2)$.

Using expression (9.10) the Figure 9.7 was generated showing the efficiency of the Global method as a function of the *corpus* partition size. Figure 9.7-a) refers to re_{23} and Figures 9.7-c) to 9.7-d) refer to $re_{2...5}$.



a) re_{23}— $mr_{Phase_2} = mr_{Phase_3} = 0\%$

b) $re_{2...5}$— $mr_{Phase_2} = mr_{Phase_3} = 0\%$

c) $re_{2...5}$ — $mr_{Phase_2} = 30\%; mr_{Phase_3} = 0\%$

d) $re_{2...5}$ — $mr_{Phase_2} = 100\%; mr_{Phase_3} = 0\%$

Figure 9.7: Global method efficiency. The Y axis represents the efficiency and the X axis represents the partition size ($|C|/K$). Note that in subfigures a) and b) the curves of the efficiencies for phases two and three are overlapping ate the top 100% horizontal line.

In Figure 9.7, in all the phases we assume that $t_{op}/t_{n\text{–}gram} = t_{g_n}/t_{n\text{–}gram} = t_{re_n}/t_{n\text{–}gram} = 1/20$. In this comparison we assume the best possible case for the Global method, i.e., phase one with full local aggregation and phases two and three with miss ratios of zero. Thus, efficiencies E_{GM_2} and E_{GM_3} are assumed to be 100%. Even with this assumption we observe that the global efficiency is far below 100% except when the *corpus* size reaches the *plateaux*. In fact E_{GM} is mostly determined by the efficiency of phase one (E_{GM_1}) which increases progressively with the *corpus* size due to the corresponding increase of the granularity as discussed in chapter 5.

In fact, from expression (9.10) we obtain for the case of $E_{GM_2} = E_{GM_3} = 100\%$:

$$E_{GM} = \frac{N_n(phase1) \times \left(1 + \frac{N_n(phase2)+N_n(phase3)}{N_n(phase1)}\right)}{N_n(phase1) \times \left(\frac{1}{E_{GM_1}} + \frac{N_n(phase2)+N_n(phase3)}{N_n(phase1)}\right)} \approx E_{GM_1} \tag{9.12}$$

$$\frac{N_n(phase2)+N_n(phase3)}{N_n(phase1)} \ll 1 \Rightarrow \frac{N_n(phase2)+N_n(phase3)}{N_n(phase1)} \ll \frac{1}{E_{GM_1}} \tag{9.13}$$

Considering now the scenarios where E_{GM_2} or E_{GM_3} are below 100% (miss ratio greater than 0%) we observe a natural reduction in E_{GM} but the influence of E_{GM_1} remains dominant as shown in Figure 9.7-d) and also suggested by the following expression:

$$E_{GM} = \frac{1 + \frac{N_n(phase2)+N_n(phase3)}{N_n(phase1)}}{\frac{1}{E_{GM_1}} + \frac{N_n(phase2)/N_n(phase1)}{E_{GM_2}} + \frac{N_n(phase3)/N_n(phase1)}{E_{GM_3}}} \tag{9.14}$$

Using the data derived from the theoretical model presented in chapter 3, the ratios of the problem sizes between the three phases were obtained as illustrated in Figure 9.8.

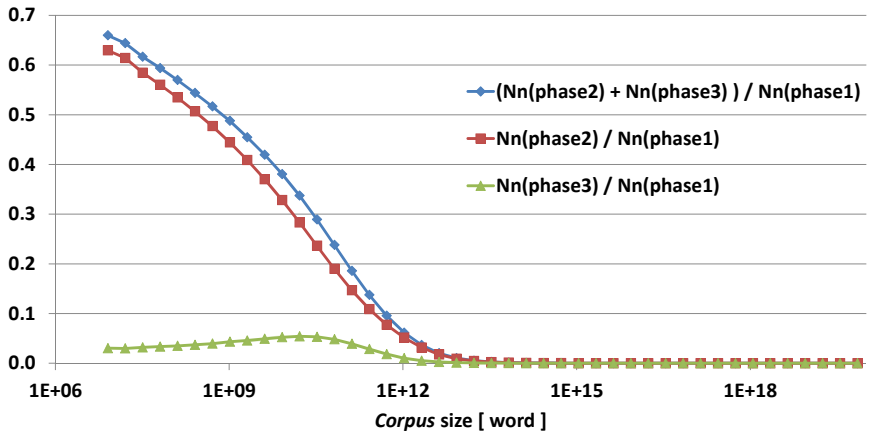


Figure 9.8: Problem size ratios of the three phases of the LocalMaxs Global method for the entire *corpus* size range. The Y axis represents the ratio of the problem sizes and the X axis represents the *corpus* size in words.

Influence of the Efficiency upon the Execution Times of Phases One and Two

As phase three has a very small fraction of the total execution time T_{GM}, we only compared with detail the ratio between the execution times of phases one and two.

Using the data from Figure 9.7 and Figure 9.8, the execution time ratios were calculated from the following expression:

$$T_1/T_2 = (N_n(phase1)/N_n(phase2)) \times \left(E_{GM_2}/E_{GM_1}\right) \tag{9.15}$$

Figure 9.9 shows the ratio between the execution time of phase one (T_1) over the execution time of phase two (T_2) in machine j in the case of the Global method when the *corpus* partition size ranges from 4 Mw up to 100 Tw.

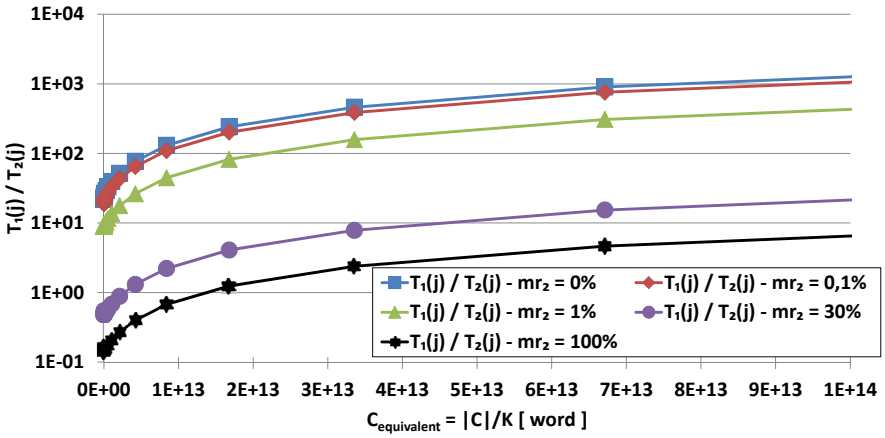


Figure 9.9: Per machine phase one execution time over phase two execution time ratio as a function of the *corpus* partition size, in Global method for $re_{2...5}$ and different phase two miss ratio values. The Y axis represents the ratio T_1/T_2, and the X axis represents the *corpus* partition size (in words).

In the case of phase one we assume full local aggregation. For phase two we assume five different scenarios with the following miss ratio values: 0%, 0.1%, 1%, 30% and 100%. Figure 9.10 is a detail of Figure 9.9 for the range of *corpus* partition size from 4 Mw up to 100 Gw.

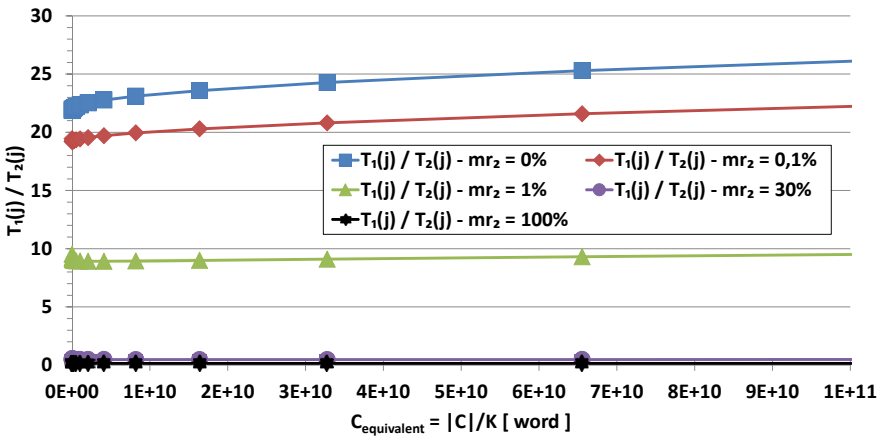


Figure 9.10: Detail of per machine phase one execution time over phase two execution time ratio as a function of the *corpus* partition size, in Global method for $re_{2...5}$ and different phase two miss ratio values. The Y axis represents the ratio T_1/T_2, and the X axis represents the *corpus* partition size (in words).

In expression 9.15, the factor $N_n(phase1)/N_n(phase2)$ is always greater than 1. While the numerator ($N_n(phase1)$) grows linearly with the *corpus* size, without any upper bound, the denominator ($N_n(phase2)$) only grows logarithmically with the *corpus* size and is upper bounded by the size of the *plateau* regions. Thus, for progressively larger *corpus* size, this first factor increases its contribution to the time ratio.

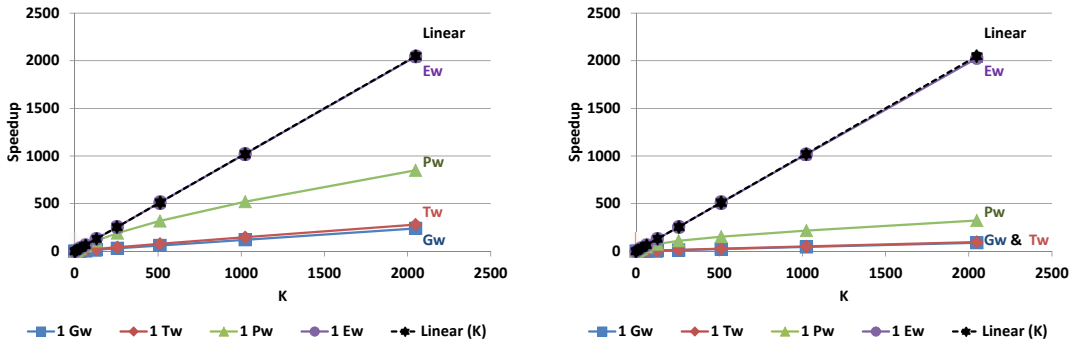
The second factor (E_{GM_2}/E_{GM_1}) is determined by the relative granularities of the two phases. The granularity (G_1) of phase one grows slowly from values below 1 for small size *corpus*, corresponding to low values of efficiency (E_{GM_1}), but around 10 Gw the granularity starts growing faster, driving the efficiency towards 100% in the *plateau* regions. The granularity of phase two (G_2) directly depends on the miss ratio of the cache system. In the ideal case of the miss ratio equal to zero, E_{GM_2}=100%, so the execution time T_1 is always greater than T_2 in the entire *corpus* range. In the scenario where the miss ratio is 30%, corresponding to a very low value of the efficiency E_{GM_2} the factor $\left(E_{GM_2}/E_{GM_1}\right)$ is much lower than 1: we observe $T_1 < T_2$ while the *corpus* size is not big enough to ensure that the factor $(N_n(phase1)/N_n(phase2))$ is sufficiently greater than 1, to compensate the low efficiency ratio $\left(E_{GM_2}/E_{GM_1}\right)$. This latter behavior only happens for larger *corpus* sizes.

Speedup

The (fixed *corpus* size) speedup (Sp_{GM}) of the Global method for K machines relative to an ideal sequential machine is given by:

$$Sp_{GM} = K \times E_{GM} \tag{9.16}$$

Using the data corresponding to Figure 9.7 and considering the values of the efficiency (E_{GM}) for each *corpus* size from 1 Gw to 1 Exaword (10^{18}) (Ew) and for different values of K, the obtained values for the speedup, for the evaluation of $re_{2...5}$, are shown in Figure 9.11 and Table 9.8.



a) $mr_{Phase_2} = 0\%$; $mr_{Phase_3} = 0\%$ ***b)*** $mr_{Phase_2} = 30\%$; $mr_{Phase_3} = 0\%$

Figure 9.11: Speedup of the Global method (relative to an ideal sequential machine) as a function of K, for $re_{2...5}$ and different *corpus* sizes. The Y axis represents the speedup and the X axis represents the number of machines.

Table 9.8: Efficiency and speedup of the Global method for $re_{2...5}$ (relative to an ideal sequential machine).

	K	1 Gw		1 Tw		1 Pw		1 Ew	
		E	*Sp*	*E*	*Sp*	*E*	*Sp*	*E*	*Sp*
	64	12%	7	19%	12	92%	59	100%	64
$mr_2 = 0\%$	**256**	12%	30	16%	41	74%	190	100%	256
&	**512**	12%	60	15%	77	62%	318	100%	512
$mr_3 = 0\%$	**1024**	12%	119	14%	146	51%	520	100%	1023
	2048	12%	239	14%	280	41%	850	100%	2042
	64	4%	3	6%	4	74%	47	100%	64
$mr_2 = 30\%$	**256**	4%	11	5%	14	43%	109	100%	256
&	**512**	4%	22	5%	26	30%	152	100%	511
$mr_3 = 0\%$	**1024**	4%	44	5%	50	21%	216	99%	1018
	2048	4%	88	5%	96	16%	322	99%	2026

The following conclusions can be taken:

- For each fixed value of K, the speedup (relative to an ideal sequential machine) increases with the *corpus* sizes following the corresponding increase of the efficiency; It approaches a linear speedup for larger *corpus* sizes; Before reaching the linear speedup regions, for each *corpus* size the efficiency decreases with K; However, provided that we stay above a certain level of efficiency we can still try to increase the number of machines in order to achieve a reduction in the execution time or to fit in the local memory required by each machine; For example, in the case of the *corpus* with 1 Tw and $mr_2 = 0\%$ the efficiency ranges from 16% with 256 machines to 14% with 2048 machines, while the speedup increases from 41 to 280, and the required per machine *corpus* partition goes from 1 Tw/256 $\approx$ 4 Gw to 1 Tw/2048 $\approx$ 512 Mw;
- The above speedup and efficiency metrics measure the performance of the Global method with K machines using the ideal sequential machine as the reference, so they provide an indication of the impact of the considered communication overheads of the parallel and distributed implementation; Furthermore, we can evaluate the relative efficiency, speedup and sizeup metrics, that indicate the performance when going from a configuration with K_1 machines to one with K_2 machines; From the above behavior data we can conclude that these relative speedup and sizeup performance metrics are close to linear, and relative efficiency is close to 1 (100%).

The latter aspect was also confirmed by the conducted experimentation in the *corpus* sizes range up to 1 Gw as presented in the following.

9.1.3.2 Analysis of the Experimental Results for *Corpus* up to 1 Gw

In this section we present the global performance results concerning the execution of the Global method in its three phases, for the evaluation of the relevant expressions of

bigrams and trigrams. The experimental results reported here were obtained in two different execution environments and system configurations. Although they correspond to different phases of the development of the work supporting this thesis, one related to the first complete implementation of the Global method, and the second corresponding to the implementation of an infinite dynamic cache for phase two, they are both presented here as they allow to illustrate the global behavior (in its speedup, sizeup and efficiency metrics) of the Global method approach as a parallel and distributed implementation of the LocalMaxs method.

The experimental results presented in previous chapters 6, 7 and 8, with a detailed analysis of the behavior of the dynamic, static and Bloom filter cache system configurations, can be seen as complementary to the global results presented here.

Execution Environment

We experimented with multiple runs of the Global method in the Lunacloud [Lun15] public cloud environment. We evaluated the extraction of bigram and trigram relevant expressions (re_{23}) from English *corpora* generated from Wikipedia [Wik16].

Each virtual machine used had similar characteristics: 4 CPU @ 1.5 GHz and a local disk storage of 40 GB. Each virtual machine included all the software components required: i) The Java classes for the Global method, the distributed in-memory KVS, and the AWARD workflow framework; ii) A JVM with version $\geq$ 1.7.X; iii) Linux Ubuntu with SSH access. The execution times shown are the average of 3 repeated runs for each experiment.

Here, we report on the results obtained in two experimental environments (A and B):

- Environment A was the first complete implementation of the Global method with a finite size cache system, and was used to process the following *corpus* sizes: 37, 75, 150, 225, 300, 450, 604, 900 and 1023 Mw. Three configurations were considered: 18 virtual machines, each with 70 GB of RAM; 36 virtual machines, each with 32 GB of RAM; and 54 virtual machines, each with 20 GB of RAM.

- Environment B was an improved implementation of the Global method with optimizations in the input processing and used an infinite on-demand dynamic cache system for all the n-gram sizes. This environment was used to process the following *corpus* sizes: 25, 227, 466, and 682 Mw, with distinct number of virtual machines (K): 1, 9, 16, 24, 32, 40, 48. For the considered configurations, for each value of K and *corpus* size, the virtual machines were configured with enough memory to ensure the behavior of an infinite cache system and the memory configuration (in GB) as presented in Table 9.9.

As the *corpus* sizes increase it becomes impossible to execute the Global method in a single machine due to insufficient local memory. Thus all the considered experimental performed metrics are defined as relative to a minimum base number of machines necessary

to run the algorithm for each *corpus* size. In environment B due to a higher number of machine configurations than in environment A, the memory configurations per machine were adjusted according to the memory requirements of each case (Table 9.9).

Table 9.9: Memory configurations in GB used per machine in environment B

K	*Corpus* size			
	25 Mw	227 Mw	466 Mw	682 Mw
1	90			
9	85	85		
16	48	48	70	
24	32	32	49	85
32	24	24	37	64
40	19	19	29	51
48	16	16	25	43

Global Method Execution Times

Tables 9.10 and 9.11 show the total execution times and the times of each individual phase of the Global method for "Environment A" and "Environment B".

Table 9.10: Global method execution time in environment A in minutes for re_{23} for different *corpus* sizes and numbers of machines.

	K	*Corpus* size									
		18 Mw	37 Mw	75 Mw	150 Mw	225 Mw	300 Mw	450 Mw	604 Mw	900 Mw	1023 Mw
Phase 1	**18**	1.12	1.38	2.36	9.10		16.33		33.71		56.67
	36		1.52		6.37		11.59		21.43		34.13
	54		1.55		5.22	7.19	9.61	13.61	17.73	26.63	29.74
Phase 2	**18**	7.75	9.68	15.79	30.48		59.13		154.41		252.46
	36		6.34		16.04		29.34		59.04		126.21
	54		5.38		11.92	16.97	22.21	30.89	39.98	59.13	65.86
Phase 3	**18**	1.63	2.37	4.07	7.42		24.30		27.54		51.42
	36		1.94		3.98		7.61		32.63		22.26
	54		1.48		3.26	4.30	5.31	7.69	9.98	14.55	16.18
Total 1,2,3	**18**	10.50	13.43	22.22	47.00		99.76		215.66		360.55
	36		9.80		26.39		48.55		113.10		182.60
	54		8.41		20.40	28.46	37.13	52.19	67.69	100.31	111.78

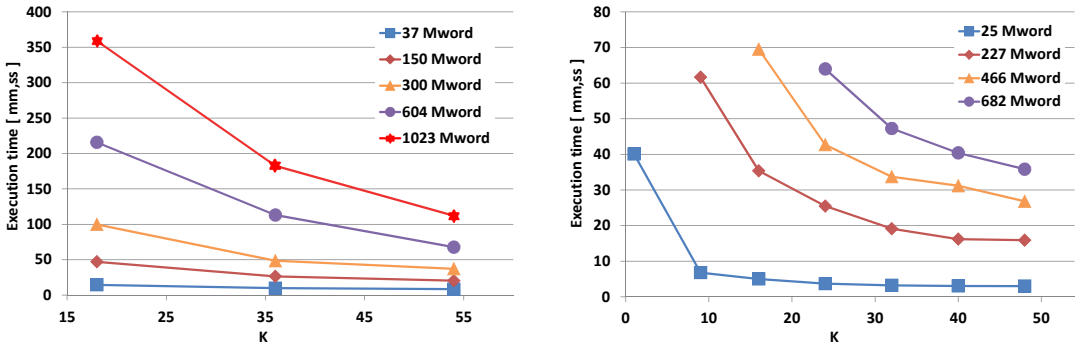
Table 9.11: Global method execution time in environment B in minutes for re_{23} for different *corpus* sizes and numbers of machines.

	K	*Corpus* size			
		25 Mw	227 Mw	466 Mw	682 Mw
	1	14.79			
	9	2.03	20.77		
	16	1.52	11.56	24.11	
Phase 1	**24**	1.15	8.70	15.45	23.04
	32	0.97	6.40	12.15	17.35
	40	0.98	5.76	12.06	16.31
	48	0.98	5.42	10.18	15.23
	1	19.15			
	9	3.66	32.20		
	16	2.64	18.81	36.60	
Phase 2	**24**	1.91	13.25	22.13	33.26
	32	1.72	10.06	17.27	24.13
	40	1.55	8.33	15.64	19.42
	48	1.52	8.29	14.10	16.49
	1	6.16			
	9	1.10	8.67		
	16	0.85	5.02	8.83	
Phase 3	**24**	0.64	3.52	5.11	7.67
	32	0.53	2.65	4.29	5.78
	40	0.52	2.10	3.47	4.66
	48	0.52	2.19	2.54	4.12
	1	40.10			
	9	6.78	61.64		
	16	5.01	35.38	69.53	
Total 1,2,3	**24**	3.70	25.48	42.69	63.97
	32	3.22	19.12	33.71	47.26
	40	3.05	16.19	31.17	40.39
	48	3.02	15.90	26.82	35.84

The behavior of the Global method execution time for different *corpora* and numbers of machines (K) is illustrated in Figure 9.12, which shows the total execution time *versus* the number of machines for both environments.

The following conclusions can be taken:

- Except for the smaller *corpus* the total execution time, for each fixed *corpus* size, decreases almost proportionally to $1/K$;
- This is the result of the accumulated effect of the three phases, and is consistent with the conclusions from the simplified analytical model of the time complexity of the Global method discussed in chapter 5.



a) *Total execution time — Environment A* ***b)*** *Total execution time — Environment B*

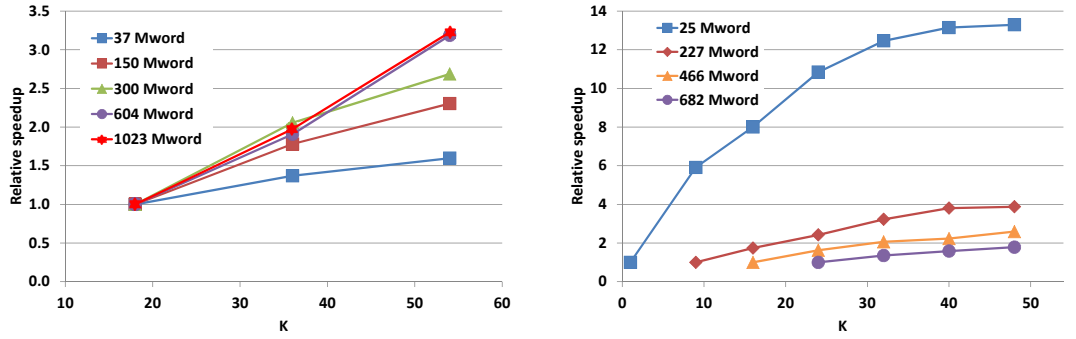
Figure 9.12: Global method total execution time for re_{23} as a function of K, for different *corpus* sizes, in environments A and B. The Y axis represents the execution time in minutes and the X axis represents the number of machines.

From this execution time data the following analysis regarding the relative performance metrics of the Global method can be derived.

Relative Speedup

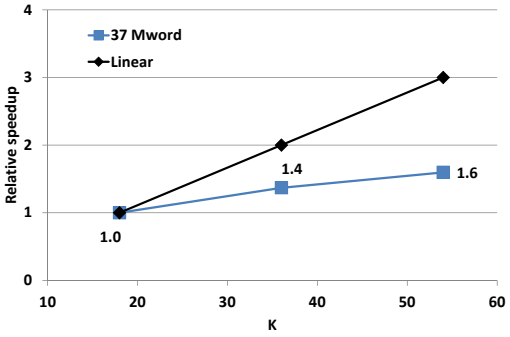
For each *corpus* size, we consider the value of K_1 corresponding to the minimal number of machines required to execute all phases of the Global method and this value is used as the reference in relation to which the relative speedup is evaluated: $Sp_{K_1 \to K}(K) = \frac{T(K_1)}{T(K)}$ and the corresponding values of this ratio are presented for all the K values in the experimented range of machine numbers.

Figure 9.13 shows the relative speedup (using Definition 2.5 on page 20) of the Global method *versus* the number of machines (K) size for both environments A and B. Figures 9.14 and 9.15 show with more detail, the data presented, respectively, in Figures 9.13-a) and 9.13-b). The lines with the filled lozenges (♦) and with the filled squares (■) represent, respectively, the linear (ideal) relative speedup and the obtained relative speedup.

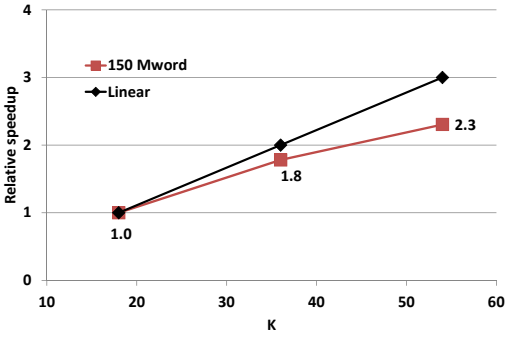


a) *Relative speedup — Environment A* ***b)*** *Relative speedup — Environment B*

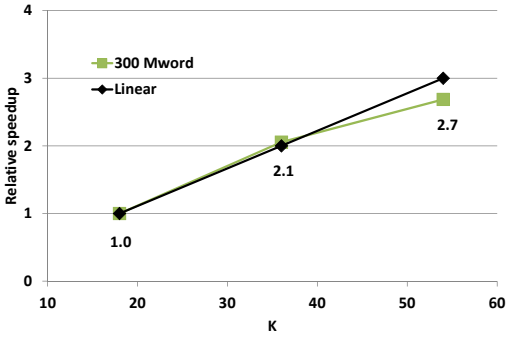
Figure 9.13: Global method relative speedup (strong scaling) for re_{23} as a function of K, for different *corpus* sizes, in environments A and B. The Y axis represents the relative speedup and the X axis represents the number of machines.



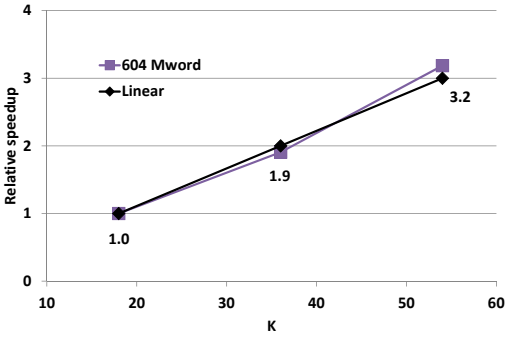
a) Corpus with 37 Mw



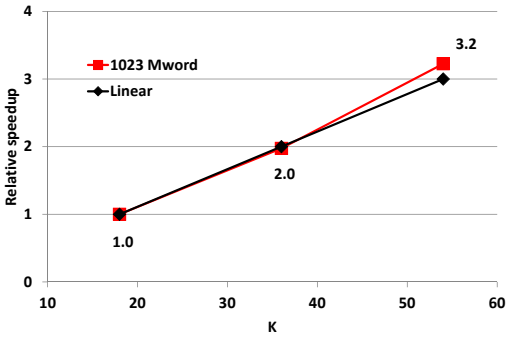
b) Corpus with 150 Mw



c) Corpus with 300 Mw

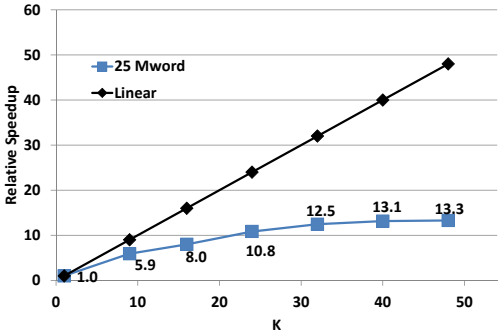


d) Corpus with 604 Mw

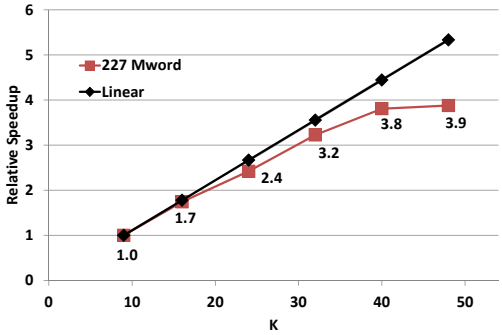


e) Corpus with 1023 Mw

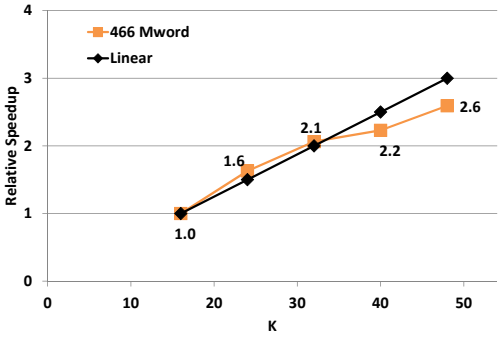
Figure 9.14: Detail of Figure 9.13-a) regarding the Global method relative speedup (strong scaling) for re_{23} as a function of K, for different *corpus* sizes, in environment A. The Y axis represents the relative speedup and the X axis represents the number of machines.



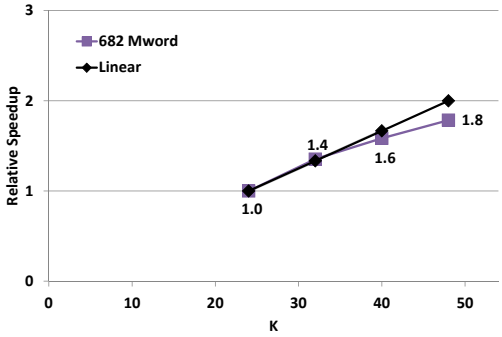
a) Corpus with 25 Mw



b) Corpus with 227 Mw



c) Corpus with 466 Mw



d) Corpus with 682 Mw

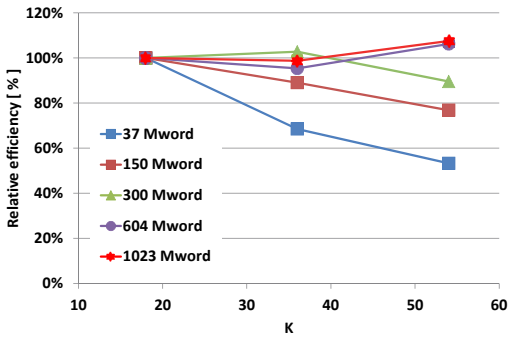
Figure 9.15: Detail of Figure 9.13-b) regarding the Global method relative speedup (strong scaling) for re_{23} as a function of K, for different *corpus* sizes, in environment B. The Y axis represents the relative speedup and the X axis represents the number of machines.

We observe that:

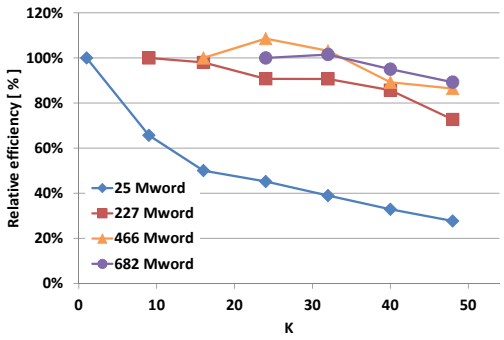
- The relative speedup is sublinear, which is related to the significant influence of the communication overheads as discussed in previous chapters;
- However, the relative speedup increases with the *corpus* size, and for the higher *corpus* sizes observed in both environments becomes close to linear.

Relative Efficiency

Figure 9.16 shows the relative efficiency (using Definition 2.6 on page 21) for both environments.



a) Relative efficiency — Environment A



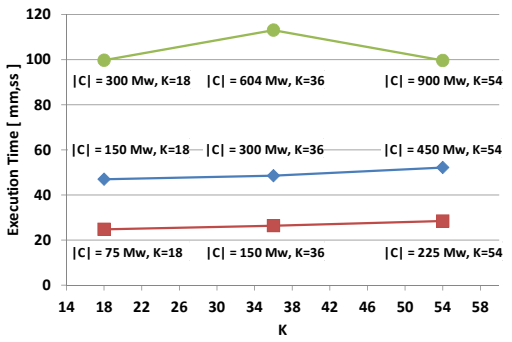
b) Relative efficiency — Environment B

Figure 9.16: Global method relative efficiency for re_{23} as a function of K, for different *corpus* sizes, in environments A and B. The Y axis represents the relative efficiency and the X axis represents the number of machines.

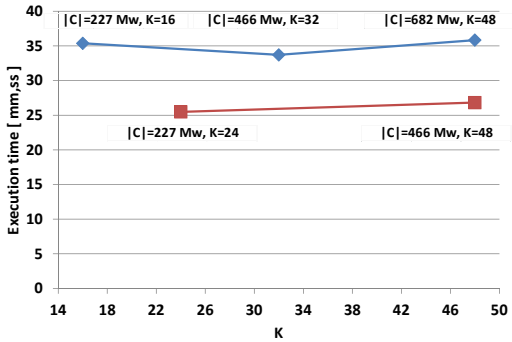
As a result of the above relative speedup behavior, the relative efficiency is around or above 80% and decreases only slightly with K (except for the smaller *corpus* size of 37 Mw in environment A, and for the *corpus* of 25 Mw in environment B).

Relative Sizeup

Figure 9.17 shows the weak scaling behavior corresponding to the relative sizeup (using Definition 2.7 on page 21) that was observed for the same environments.



c) Relative sizeup — Environment A



d) Relative sizeup — Environment B

Figure 9.17: Global method relative sizeup (weak scaling) for re_{23} as a function of K, when the *corpus* size scales with K, in environments A and B. The Y axis represents the relative sizeup and the X axis represents the number of machines.

The Global method shows a good relative weak scaling behavior because it keeps the execution time almost unchanged when the *corpus* size and the number of machines are scaled by the same factor.

Load Balancing

Overall, in the experiments conducted in both environments, the Global method exhibited a good load balancing behavior. This is related to the even distribution of work among the machines and to the n-gram table partitioning in any of the three phases.

For illustrative purposes, Figure 9.18 shows the per machine execution time ($T_2(j)$) for each of the three phases of the Global method for re_{23} and a *corpus* of size 466 Mw when using 24 virtual machines (environment B).

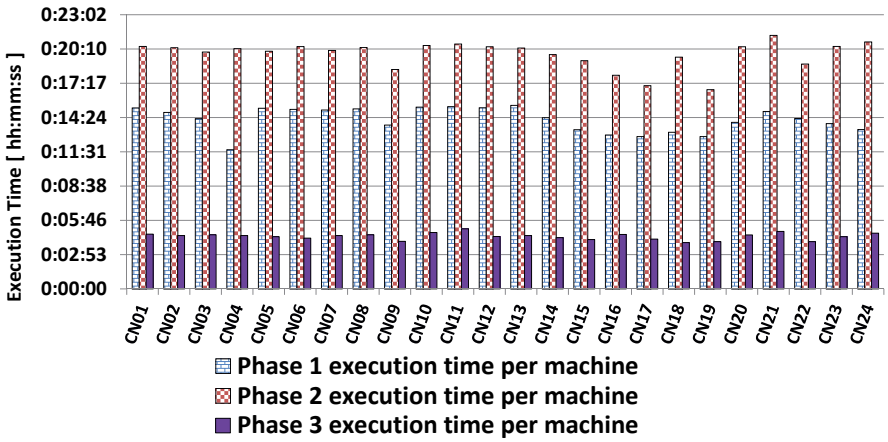


Figure 9.18: Three-phase per machine execution time — Environment B. The Y axis represents the execution time in minutes and the X axis represents the machines individually named from CN_{01} to CN_{24}.

The following conclusions can be made:

- All the machines, for all the three phases, exhibit similar completion times corresponding to well-balanced application load and n-gram table partitioning;
- Using the load balancing metric defined in expression (6.46), on page 178, the following values were obtained: 92% for phase one, 93% for phase two and 87% for phase three.

These results of the real execution were observed in public cloud environments, with consistent behaviors, in different times (2015 and 2016) [GSC15; GSC16].

9.2 Local Method

In this section we propose the Local method for the parallel implementation of LocalMaxs and evaluate it as an alternative to the Global method. Section 9.2.1 presents the rationale of the Local method. An implementation is described in section 9.2.2 and the time complexity of the implementation is analyzed in section 9.2.3. An analysis of the memory space of the Local method is presented in section 9.2.4.The evaluation of the Local method, with respect to its precision and recall is discussed in section 9.2.5.

9.2.1 Rationale of the Local Method

In the Local method, each machine is responsible for entirely processing a separate *corpus* partition, i.e., first counting the local occurrences of the n-grams in its partition, then

calculating the glue of the locally found distinct n-grams (using only their local counters), and finally evaluating the relevance of those distinct n-grams, which are then proposed as candidate relevant expressions, subject to being globally combined according to a global evaluation criterion. As such, this method requires a final global combination of the partially obtained sets of relevant expressions in order to identify a global set of relevant expressions. As a trade-off between the execution performance and the precision and recall, the Local method represents an alternative approach to the Global method, based on approximate and partial solutions identified in each machine.

9.2.2 Implementation of the Local Method

In this section we describe the logical workflow of the Local method and its implementation on the distributed architecture (chapter 4), followed by an analysis of its time complexity and its memory space, and the experimental results regarding the precision and recall metrics.

Logical Workflow. The Local method performs a parallel evaluation using multiple machines (Figure 9.19), each machine being responsible for the local application of a LocalMaxs instance to a separate *corpus* partition, i.e., counting the n-grams, calculating the glue, and identifying the relevance of the n-grams.

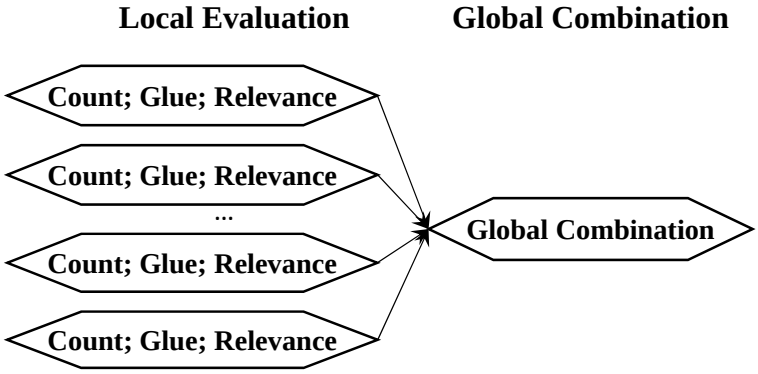


Figure 9.19: Local method logical workflow

The above parallel and local evaluation step is followed by a global combination of the separately obtained relevant expressions. Several combination methods can be considered. Currently we have studied the behavior of this approach using a combination based on the union of the partial relevant expression sets.

The local evaluation in each machine has three main components: the first one, consisting of the n-gram counting is proportional to the partition size ($|C|/K$); the other two components, related to the glue and relevance calculations, present a logarithmic-like dependency on the the *corpus* partition size. In a parallel implementation each machine outputs a reduced data volume (when compared to the Global method), corresponding to the locally identified relevant expressions. It only reports as a result the set of distinct

n-grams which are considered relevant expressions. The output volume per-partition has an upper bound equal to the number of distinct n-grams in each partition, excluding the singletons and the unigrams. The output communication, besides occurring in parallel among the multiple machines, is implemented asynchronously within each machine, thus exploiting overlapping with local computation. Note that the global combination, although represented logically by a single workflow node in Figure 9.19, does not have to be implemented in a centralized way. This combination can be implemented in a decentralized way using the KVS-based server architecture, where each n-gram is mapped into a specific KVS server by hashing.

The mapping of the Local method workflow into the distributed architecture (chapter 4) is straightforward. The critical issues are related to adjusting the granularity of the *corpus* partitions in order to satisfy the following requirements:

- An acceptable computation-to-communication granularity ratio (G);
- The locally generated n-grams tables fit into the available per machine local memory;
- The resulting precision and recall of the Local method are within acceptable thresholds.

9.2.3 Time Complexity of the Local Method Parallel Implementation

In this section we present an analytical model of the time complexity of the Local method and use this model to estimate its global performance metrics in the entire *corpus* range, by relying upon the estimated n-gram data from the theoretical model presented in chapter 3.

Given a *corpus* C and a configuration using K machines to produce the set of relevant expressions $re_{2\cdots n}$ this requires the evaluation of the relevance of the distinct nonsingleton n-grams from bigrams up to n-grams of size n. Each machine j must consider the n-grams occurring in its local partition $C(j)$ (of size $|C|/K$) in order to count the frequencies of their local occurrences for the distinct unigrams up to the n-grams of size $(n+1)$, followed by the calculation of the combined glues $g_{2\cdots(n+1)}$ of the distinct bigrams up to the n-grams of size $(n+1)$, which are then used to evaluate the corresponding relevance.

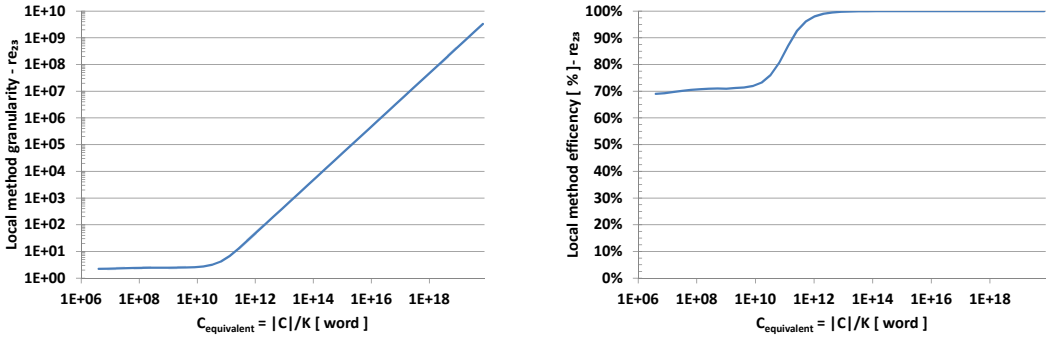
The computation-to-communication ratio in each machine ($1 \leq j \leq K$) is given by:

$$G(j) = \frac{\left(\frac{|C|}{K} \times (n+1) \times t_{op}\right) + \left(\left|D_{All_{2\cdots(n+1)}in(C(j))}\right| \times t_{g_n}\right) + \left(\left|NS_{All(2\cdots n)in(C(j))}\right| \times t_{re_n}\right)}{\left|NS_{All(2\cdots n)in(C(j))}\right| \times t_{n\text{–}gram}} \tag{9.17}$$

where t_{op}, t_{g_n}, and t_{re_n} are, respectively, the average times for the basic operations for counting the local frequencies, calculating the glues and finding relevant expressions of an n-gram; $t_{n\text{–}gram}$ is the average time for sending an n-gram to the corresponding KVS server; $D_{All_{(2\cdots(n+1))}in(C(j))}$ represents all the distinct n-grams of sizes 2 up to $(n+1)$ in a

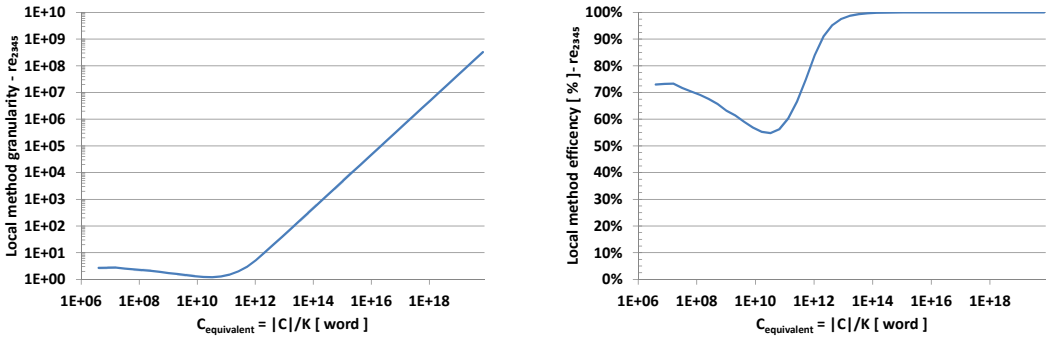
partition $C(j)$ of size $|C|/K$, and $NS_{All(2\cdots n)in(C(j))}$ represents the nonsingleton n-grams of sizes 2 up to n in a *corpus* partition.

Figures 9.20 and 9.21 show the granularity (G) and efficiency (E_{LM}) of the Local method *versus* the *corpus* partition size for two cases: finding relevant expressions of bigrams and trigrams (re_{23}) and from bigrams up to pentagrams ($re_{2\cdots5}$).



a) *Local method granularity* re_{23} ***b)*** *Local method efficiency* re_{23}

Figure 9.20: Local method granularity and efficiency for re_{23} *versus* the *corpus* partition size. The Y axis in subfigure a) represents the granularity and in subfigure b) it represents the efficiency relative to an ideal sequential machine. In both subfigures the X axis represents the *corpus* partition size in words.



a) *Local method granularity* $re_{2\cdots5}$ ***b)*** *Local method efficiency* $re_{2\cdots5}$

Figure 9.21: Local method granularity and efficiency for $re_{2\cdots5}$ *versus corpus* partition size. The Y axis in subfigure a) represents the granularity and in subfigure b) it represents the efficiency relative to an ideal sequential machine. In both subfigures the X axis represents the *corpus* partition size in words.

Both figures assume that $t_{op}/t_{n\text{–}gram} = t_{g_n}/t_{n\text{–}gram} = t_{re_n}/t_{n\text{–}gram} = 1/20$.

Table 9.12 shows the corresponding values to Figure 9.20 and Figure 9.21 when the *corpus* size ranges from 16 Mw up to 130 Tw.

Table 9.12: Local method granularity and efficiency for re_{23} and $re_{2...5}$ *versus corpus* size partition (in words).

$\|C\|/K$	re_{23}		$re_{2...5}$	
	G	E_{LM}	G	E_{LM}
$\mathbf{1.6 \times 10^7}$	2.30	70%	2.75	73%
$\mathbf{1.3 \times 10^8}$	2.42	71%	2.24	69%
$\mathbf{1.0 \times 10^9}$	2.44	71%	1.72	63%
$\mathbf{1.6 \times 10^{10}}$	2.74	73%	1.24	55%
$\mathbf{1.3 \times 10^{11}}$	6.71	87%	1.51	60%
$\mathbf{1.0 \times 10^{12}}$	49.6	98%	5.23	84%
$\mathbf{1.7 \times 10^{13}}$	790	100%	78.3	99%
$\mathbf{1.3 \times 10^{14}}$	6,320	100%	623	100%

We observe the following:

- The Local method efficiency (relative to an ideal sequential machine) reaches 100% when the *corpus* partition size ($|C|/K$) is in the range of the *plateau* regions; This is due to the fact that the number of nonsingleton n-grams (that are in the denominator of expression (9.17), which represents the communication volume) becomes constant with respect to the *corpus* partition size, while the computation time for counting and aggregating the n-gram frequencies (that is in the first parcel of the numerator of expression (9.17), corresponding to the phase one of LocalMaxs) continues to increase proportionally to the *corpus* partition size; Thus, the communication time becomes progressively smaller than the computation time, which is emphasized by the linear increase in the granularity (G) as illustrated in Figure 9.20-a) and Figure 9.21-a);

- For values of G over 10 the efficiency is above 91% and when G is greater than 100 the efficiency is above 99%;

- In all other regions of the *corpus* partition sizes spectrum, although the efficiency is lower than the one in the *plateau* regions, the value of granularity is always greater than 1; This is due to the fact that the numerator in expression (9.17) is always greater than the denominator, even considering a ratio of $t_{op}/t_{n\text{–}gram} = 1/20$; Thus, the efficiency of the Local method is always greater than 50%, in both of the considered cases of re_{23} and $re_{2...5}$ (Figure 9.20-b) and Figure 9.21-b));

- In the case of re_{23} the efficiency value stays around a constant value of 70% before the *plateaux*; In the case of $re_{2...5}$, the influence of a significant increase in the numbers of the distinct tetragrams, pentagrams and hexagrams leads to an increase in the communication volume (denominator of expression (9.17)), which overcomes the increase of the computation parcel in a region of *corpus* partition sizes around 10^8 to 10^{11} words for the English *corpus* analyzed; Due to this, the efficiency decreases in the above mentioned region and, after having reached a minimum for a

corpus partition size around 10^{11} words, the efficiency only recovers when the linear increase in the computation parcel overcomes the rate of growth of the numbers of nonsingletons.

Speedup

The speedup (Sp_{LM}) of the Local method for K machines and a fixed *corpus* size, relative to an ideal sequential machine, is given by:

$$Sp_{LM} = K \times E_{LM} \tag{9.18}$$

Using the data corresponding to Figure 9.21-b) and Table 9.12 when considering the values of the efficiency (E_{LM}) for each *corpus* size from 1 Gw to 1 Ew and for different values of K, the expression (9.18) was applied leading to Figure 9.22 and Table 9.13 for the case of $re_{2...5}$.

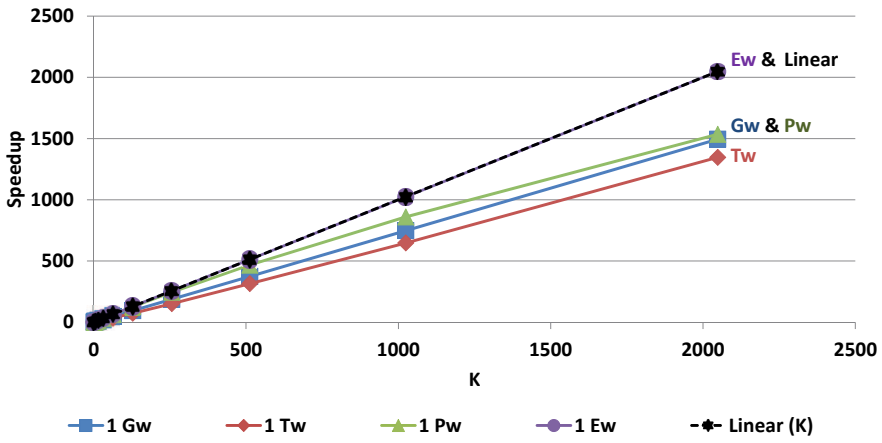


Figure 9.22: Speedup of the Local method relative to an ideal sequential machine, for $re_{2...5}$ as a function of K for different *corpus* sizes. The Y axis represents the speedup and the X axis represents the number of machines.

Table 9.13: Efficiency and speedup of the Local method relative to an ideal sequential machine, for $re_{2...5}$.

K	1 Gw		1 Tw		1 Pw		1 Ew	
	E_{LM}	Sp_{LM}	E_{LM}	Sp_{LM}	E_{LM}	Sp_{LM}	E_{LM}	Sp_{LM}
64	73%	47	55%	35	99%	63	100%	64
256	73%	187	59%	151	95%	244	100%	256
512	73%	374	61%	314	91%	466	100%	512
1024	73%	747	63%	647	84%	860	100%	1,024
2048	73%	1,494	66%	1,346	75%	1,534	100%	2,047

The following observations are in order:

- ✦ The speedup behavior (relative to an ideal sequential machine) tends to be linear as the size of the *corpus* increases, namely, being almost linear for very large *corpora* in the Petaword (10^{15}) (Pw) and Ew ranges;

- ✦ The behavior of the speedup with K is determined by the behavior of the efficiency curve as a function of $|C|/K$ (Figure 9.21) when we fix the *corpus* size ($|C|$).

9.2.4 Analysis of the Memory Space

The memory space occupied by each machine during the execution of the Local method for $re_{2\ldots n}$ is determined by the size of a local table containing all the distinct n-grams ($D_{All_{(1\ldots(n+1))}in(C(j))}$) from 1 to ($n$+1) within the local *corpus* partition ($C(j)$ of size $|C|/K$).

9.2.5 Precision and Recall

In order to evaluate the quality of the solutions of the Local method, we considered different *corpus* partition configurations by splitting an input *corpus* into different numbers of partitions. For each of these partition configurations we ran independent instances of the LocalMaxs method, and combined the corresponding partial results in order to generate a set of output relevant expressions. Then we compared the generated output set of each partition configuration with the set of relevant expressions extracted by the Global method for the same input *corpus*. The results of the experiments performed according to this methodology are presented and discussed in the following.

We use the traditional metrics of precision and recall to evaluate the Local method, but redefine these metrics with respect to the results of the Global method. Note that the Global method achieves the same precision and recall as the LocalMaxs original definition.

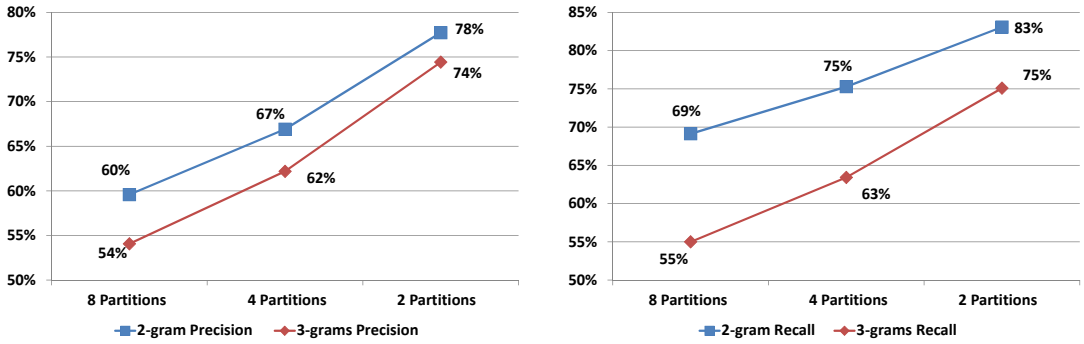
$$RelativePrecision(C,p) = \frac{Number\ Of\ True\ Local\ Positives}{|REsLocal(C,p)|} \tag{9.19}$$

$$RelativeRecall(C,p) = \frac{Number\ Of\ True\ Local\ Positives}{|REsGlobal(C)|} \tag{9.20}$$

where $REsGlobal(C)$ is the set of relevant expressions found by the Global method for a given *corpus* C; $REsLocal(C,p)$ is the set of relevant expressions found by the Local method for a given *corpus* C with p equal size partitions; $Number\ Of\ True\ Local\ Positives$ is the number of relevant expressions found by the Local method which are also in the $REsGlobal(C)$ set.

We wanted to evaluate the deviations of the Local method metrics with respect to the ideal case of 100 % where both Global and Local methods would be equivalent in terms of solutions quality. As this clearly depends on the *corpus* sizes and number of partitions, we conducted several experiments to investigate this behavior.

Experimental Results We have completed a preliminary evaluation of the solutions quality of the Local method for extracting bigram and trigram relevant expressions, from an English *corpus* size of 1 Gw and considering different numbers of equal size partitions: 2, 4 and 8. The global combination method considered in these experiments was based on the union of the locally obtained relevant expressions sets. The obtained precision and recall values are shown in Figure 9.23.



a) *Precision* ***b)*** *Recall*

Figure 9.23: Local method metrics versus the number of partitions for re_{23} and *corpus* size of 1 Gw, considering the union of partition results for the cases of $C(8)$=8 partitions, $C(4)$=4 partitions and $C(2)$=2 partitions. The Y axis represents the precision and recall, respectively in subfigure a) and subfigure b); and the X axis represents the number of partitions considered (8, 4 or 2).

We conclude the following:

- Both relative metrics (precision and recall) show to be increasing with the *corpus* partition size;
- This behavior is consistent with the fact that when the *corpus* size increases, the set of distinct n-grams in each partition tends to the set of distinct n-grams in the whole *corpus;*
- This puts the Local method as an alternative method that should be considered by taking into account, on one hand, the required precision and recall, and on the other hand the execution time.

9.3 Comparison of the Global and Local Methods

The main dimensions involved are illustrated in Figure 9.24.

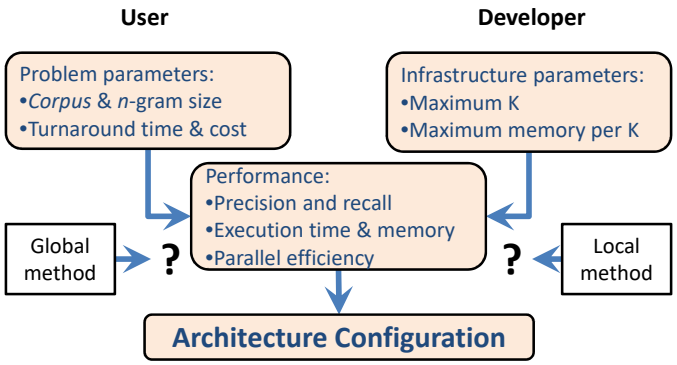


Figure 9.24: Dimensions involved.

9.3.1 Precision and Recall

The Local method achieves lower precision and recall values than the Global method, although these values gradually increase with the *corpus* partition size, becoming closer to the values obtained by the Global method.

9.3.2 Memory Space

In order to compare the memory space occupied by the Global and the Local methods, assuming equal *corpus* sizes and numbers of machines, we consider one controller and one KVS server in each local virtual machine for the Global method, and one controller in each local virtual machine for the Local method. Then we analyze the memory occupied by the local n-gram tables in each machine for both cases:

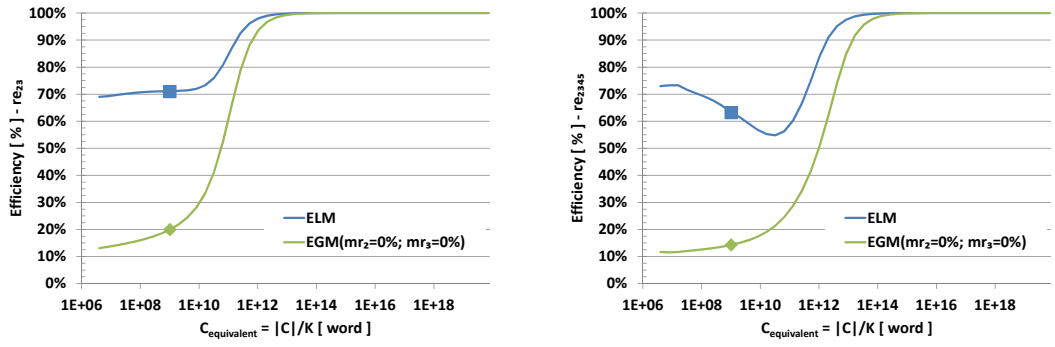
- **Global method:** The local n-gram table in the KVS server takes a number of entries equal to the size $\left|D_{All_{(1\cdots(n+1))}in(C)}\right|/K$; Thus, it grows logarithmically with the *corpus* size until reaching the *plateau* regions; On the other hand for each fixed *corpus* size each local KVS server table is proportional to $1/K$; Additionally the Global method requires memory space for the local cache in the controller in phase two, which takes a maximum number of entries (corresponding to the infinite cache case) equal to the size of the per machine set $D_{All_{(1\cdots n)}in\left(LocalTablePartition_{2\cdots(n+1)}\right)}(j)$ obtained as the average over the K machines. Each cache n-gram entry in phase two only contains an integer with the frequency count;

- **Local method:** The local n-gram table in the controller takes a number of entries equal to the size of the set $D_{All_{(1\cdots(n+1))}in(C(j))}$; Thus, it grows logarithmically with the *corpus* partition size ($|C|/K$) until reaching the *plateau* regions; Each n-gram table entry contains: an integer for the frequency and two doubles for the glue and relevance values.

However, the Global method provides the additional functionality of a global n-gram store that can be made persistent and kept in a repository with increased flexibility concerning the reutilization of the n-gram statistical data in different times and by different methods.

9.3.3 Estimated Performance

Efficiency. Figure 9.25 shows the comparison of the efficiency of the Local method (E_{LM}) against the efficiency of the Global method (E_{GM}), only taking into consideration the communication overheads. Both efficiencies are defined relatively to an ideal sequential machine without overheads. Figure 9.25-a) refers to re_{23} and 9.25-b) refers to $re_{2...5}$.



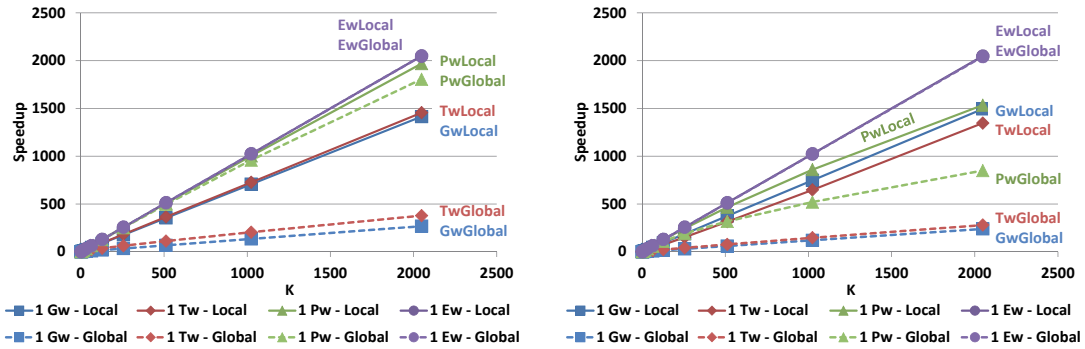
a) re_{23} ***b)*** $re_{2...5}$

Figure 9.25: Local method (E_{LM}) *versus* Global method (E_{GM}) efficiencies as a function of the *corpus* partition size for different relevant expression evaluations. The Y axis represents the efficiency and the X axis represents the *corpus* partition size ($|C|/K$).

In this comparison we assume the best possible case for the Global method, i.e., phase one with full local aggregation and considered that the cache miss ratios in phases two and three are zero. In both the methods we assume $t_{op}/t_{n\text{–}gram} = t_{g_n}/t_{n\text{–}gram} = t_{re_n}/t_{n\text{–}gram} = 1/20$.

The Global method efficiency is always lower than in the Local method efficiency except when the *plateaux* are reached where both efficiencies become 100%.

Speedup. Figure 9.26 shows a comparison of the speedup relative to an ideal sequential machine of the Local method *versus* the speedup of the Global method in two cases: re_{23}, presented in 9.26-a); and $re_{2...5}$ presented in 9.26-b).



a) re_{23} ***b)*** $re_{2...5}$

Figure 9.26: Local method *versus* Global method speedup as a function of K for different *corpus* sizes and relevant expression evaluations. The Y axis represents the speedup and the X axis represents the number of machines.

The following conclusions can be made:

- For the larger *corpus* sizes in the *plateau* regions, namely in the Ew range, both the Global and Local method exhibit linear speedup;
- Before the Ew range is reached, the Local method presents better performance speedup than the Global method.

9.4 Chapter Summary

While the Local method exhibits a better performance behavior concerning the parallel speedup and efficiency relative to an ideal sequential machine when compared to the Global method, it achieves lower precision and recall values. On the other hand, the Global method, besides ensuring the same precision and recall as LocalMaxs, also provides the additional functionality of generating the n-gram statistical data in the key-value-store, which can be kept in a persistent repository thus enabling a flexible reutilization of the n-gram tables by different algorithms and configurations of the workflow phases.

Overall, both methods exhibit almost linear relative speedup and sizeup metrics, thus favoring the scalability behavior of their implementations for large *corpus* sizes.

CHAPTER 10

Conclusions and Future Work

The results of the work achieved and future research directions are discussed in this chapter.

10.1 Overview of the Contributions

The main objective of this thesis was to contribute to the improvement of methods and tools enabling the extraction of relevant *n*-gram expressions from large *corpus* based on a statistical method, LocalMaxs, by exploiting parallel and distributed computing in order to achieve acceptable execution times and scalar behaviors in terms of *corpus* size and number of machines. As an outcome of this work the following main contributions are identified.

Statistical *n*-gram Distribution in Natural Language *Corpora*

The characteristics of the statistical distribution of *n*-grams in natural language *corpora* were studied, concerning the number of distinct *n*-gram and their frequencies of occurrences, namely, the repetitions of *n*-gram occurrences and how they vary with the *n*-gram size (from unigrams up to hexagrams) and the *corpus* size. This included the identification of the most frequently occurring *n*-grams and their accumulated sum of frequencies in well identified subsets of *n*-grams, and also the identification of the most rarely occurring *n*-grams in a *corpus*, namely, the singleton *n*-grams. The main characteristics identified by this study are the following:

- The number of distinct *n*-grams of each size grows logarithmically with the *corpus* size;
- For each *n*-gram size there is a well-identified *corpus* size threshold beyond which the number of distinct *n*-grams stabilizes in a *plateau* characterized by a constant

number of distinct n-grams, which is determined by the finiteness of each language vocabulary;

- The sizes of the populations of the distinct n-grams in a *corpus* exhibit great differences when going from unigrams to hexagrams, namely, $|D_1| \ll |D_2| \ll |D_3| \ll |D_4| \ll |D_5| \ll |D_6|$;
- The average number of repetitions of distinct n-grams in a *corpus* is greater for the smaller n-gram sizes and decreases as we go to the larger n-gram sizes; The number of repetitions of each distinct n-gram increases with the *corpus* size, leading to the disappearance of the singletons in the *plateau* regions;
- Given a *corpus* it is possible to determine the membership of the minimal set (named "Fixed Frequency Accumulation Set") of distinct n-grams for each n-gram size whose accumulated sum of frequencies represents a given percentage of the total *corpus* size.

The above characteristics were confirmed in the English and French languages, through empirical studies conducted over *corpora* with sizes in the range from 2 Mw to 1 Gw. The estimation of the behavior of the above characteristics, for the entire *corpus* range up to the *plateau* regions, was enabled through the proposal of theoretical model based on Zipf-Mandelbrot laws and the Poisson distribution, applied to the determination of distinct n-grams in a *corpus*.

The results of these studies were applied as input to feed a developed analytical model of the performance behavior of the two developed parallel and distributed implementations of LocalMaxs.

Distributed Architecture for *n*-gram Extraction

A distributed architecture was designed and implemented for the n-gram based extraction of relevant expressions, for multiphase methods relying on a distributed in-memory n-gram key-value-store. The architecture is organized in terms of logical virtual machines which encapsulate the execution, storage and communication mechanisms supporting the extraction algorithm functions, and enabling the deployment on top of cluster and cloud-based computing infrastructures. A working prototype was developed and runs on computer clusters and on different public cloud infrastructures such as LunaCloud and Amazon.

Parallel and Distributed LocalMaxs Implementations (Global and Local methods)

Two alternative methods for the parallel and distributed implementation of LocalMaxs were proposed and implemented on top of the distributed architecture. The Global method relies on a global n-gram store with the full n-gram frequencies counts, thus achieving the same precision and recall as LocalMaxs. On the other hand, the Local method is based on the combination of multiple independent and local evaluations executed by separate

machines, so although it exhibits better parallel execution performance than the Global method, it achieves lower precision and recall values than LocalMaxs.

While the Global method provides functionality for a flexible experimentation with alternative multiphase extraction methods, sharing the statistical n-gram data, the Local method is an optimized implementation but not so flexible.

An analytical performance model was developed for each of the methods allowing to estimate the behavior of the computation and communication components of the execution time. In the case of the Global method, the model estimates of its behavior were confirmed experimentally by the real execution results of the implemented prototype.

Both methods allow to achieve almost linear relative speedup and sizeup values. On one hand, this behavior is the result of the scalar design of the distributed n-gram architecture. On the other hand, it results from the characteristics of n-gram statistical distribution, namely, their logarithmic behavior with the *corpus* size and their evolution towards the *plateau* regions.

n-gram Cache System

An n-gram cache system was proposed and implemented to reduce the communication overheads occurring in the Global method due to the remote accesses to the key-value-store. An extensive study was conducted to analyze the behavior of several alternative cache configurations, concerning the miss ratio and miss time penalty metrics. First, the impact of cold-start n-gram misses in an infinite dynamic was evaluated. Additionally a cache warming-up strategy was proposed based on cooperative combined glue calculations to reduce the dynamic cache miss ratio and miss penalties. Then, a static prefetching cache was proposed based on the FAset concept leading to a significant reduction in the miss ratio, which becomes virtually zero in case of the implementation achieving total overlap of the FAset prefetching with other useful computations. The impact of integrating a Bloom filter into the cache system was also evaluated leading to significant reductions in the miss ratio and penalty and the cache size, in the *corpus* size regions where the percentages of singleton n-grams are significant.

The impact of the above dimensions was estimated by a proposed cache analytical model, and were also confirmed experimentally by measurements of real execution of the implemented Global method prototype. The effect of the multiple n-gram cache designs towards the performance of phase two of the Global method was also evaluated analytically and experimentally, leading to significant reductions in the phase two execution time.

10.2 Future Work

This work addressed multiple dimensions related to the objective of large scale n-gram extraction using a statistical method, ranging from a theoretical analysis of the n-gram

distribution, the parallel and distributed implementations issues, and performance evaluation. The results obtained showed the feasibility of the proposed solutions to ensure a scalable processing of large *corpora*. There is a real working prototype that runs on cluster and cloud infrastructures, although the experimentation conducted was limited regarding the maximum *corpus* sizes, numbers of machines and *n*-gram sizes considered.

Thus, several research directions remain open. Regarding the current architecture and its implementation, there are some aspects which can be improved and alternatives can be considered. Namely, the following aspects are worth of further investigation concerning the Global method:

- To analyze alternatives to improving the FAset prefetch overlap by anticipating the FAset construction during execution of phase one;
- To extend the analysis of the memory behavior of the Global method beyond the *corpus* ranges and machines considered in this thesis, in order to more precisely quantify the evolution of the memory requirements for larger scales;
- To extend the analysis of the distribution of sub *n*-grams in the local *n*-gram tables for larger numbers of machines;
- To better evaluate experimentally the trade offs involved in a combined cache system with finite capacity static and dynamic caches;
- To evaluate the design of a dynamic cache replacement management strategy based on the knowledge of the *n*-gram frequency counts.

Regarding the Local method, further experimentation with larger *corpus* size partitions is required in order to more precisely evaluate its precision and recall trends.

Orthogonally to both methods it will be interesting to consider the study of the integration of per machine memory optimizations of the *n*-gram data structures into the existing implementation.

Concerning the usability of the developed architecture the following aspects are open:

- To evaluate the proposed architecture to support other metrics and methods;
- To improve the post processing of the extracted list of relevant expressions by adding domain dependent semantic filters;
- To consider the use of the architecture to develop more advanced *n*-gram based applications, for example, related to the correlation of documents.

In the end of this work an open challenge still remains, related to further deepen the study of the behavior and the feasibility of the implementation for very large size *corpus* and numbers of machines, in the Tw range and beyond.

Bibliography

[Abr72] M. Abramowitz. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 10th ed. New York: Stegun, Irene A., eds., 1972. ISBN: 978-0-486-61272-0.

[Alc13] AlchemyAPI. *AlchemyAPI*. Alchemy API. 2013. URL: http://www.alchemyapi.com.

[Ama13a] Amazon AWS. *Amazon AWS - DynamoDB*. Amazon AWS. 2013. URL: http://aws.amazon.com/dynamodb/.

[Ama13b] Amazon AWS. *Amazon AWS - SimpleDB*. Amazon AWS. 2013. URL: http://aws.amazon.com/simpledb/.

[Ama16a] Amazon AWS. *Amazon AWS - Amazon Web Services*. Amazon AWS. 2016. URL: https://aws.amazon.com/.

[Ama16b] Amazon AWS. *Amazon AWS - Elastic Compute Cloud (EC2)*. Amazon AWS. 2016. URL: https://aws.amazon.com/ec2/.

[Ama16c] Amazon AWS. *Amazon AWS - Simple Storage Service (S3)*. Amazon AWS. 2016. URL: https://aws.amazon.com/s3/.

[Ama17] Amazon AWS. *Amazon AWS - Amazon EC2 Instance Types*. Amazon AWS. 2017. URL: https://aws.amazon.com/ec2/previous-generation/?nc1=h_ls.

[Anc13] Anchovy. *Anchovy*. Anchovy. 2013. URL: http://www.maxprograms.com/products/anchovy.html.

[Arr+14] D. Arroyuelo, C. Bonacic, V. G. Costa, M. Marín, and G. Navarro. "Distributed text search using suffix arrays." In: *Parallel Computing* 40.9 (2014), pp. 471–495. DOI: 10.1016/j.parco.2014.06.007.

[AGC12] L. Assunção, C. Gonçalves, and J. C. Cunha. "Autonomic Activities in the Execution of Scientific Workflows: Evaluation of the AWARD Framework." In: *Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on*. ATC '12. IEEE Computer Society, 2012, pp. 423 –430. DOI: 10.1109/UIC-ATC.2012.14.

[Att+10] M. Attia, A. Toral, L. Tounsi, P. Pecina, and J. Genabith. "Concordance for parallel texts." In: *Proceedings of the Workshop on multi-word Expressions: from Theory to Applications*. 2010, pp. 18–26.

[Baa01] R. H. Baayen. *Word Frequency Distributions (Text, Speech and Language Technology)*. 2001st ed. Springer, July 2001. ISBN: 9780792370178.

[BY+07] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. "The Impact of Caching on Search Engines." In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '07. Amsterdam, The Netherlands: ACM, 2007, pp. 183–190. ISBN: 978-1-59593-597-7. DOI: 10.1145/1277741.1277775.

[BYN00] R. A. Baeza-Yates and G. Navarro. "Block addressing indices for approximate text retrieval." In: *Journal of the American Society of Information Science* 51.1 (2000), pp. 69–82.

[BFR11] A. S. Balkir, I. Foster, and A. Rzhetsky. "A distributed look-up architecture for text mining applications using MapReduce." In: *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2011, pp. 1–11.

[BB13] K. Berberich and S. Bedathur. "Computing N-gram Statistics in MapReduce." In: *Proceedings of the 16th International Conference on Extending Database Technology*. EDBT '13. Genoa, Italy: ACM, 2013, pp. 101–112. ISBN: 978-1-4503-1597-5. DOI: 10.1145/2452376.2452389.

[Blo70] B. H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors." In: *Communications ACM* 13.7 (1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692.

[Boo67] A. D. Booth. "A "Law"of Occurrences for Words of Low Frequency." In: *Information and Control* 10.4 (1967), pp. 386–393. DOI: 10.1016/S0019-9958(67)90201-X. URL: http://dx.doi.org/10.1016/S0019-9958(67)90201-X.

[BJ99] D. Bourigault and C. Jacquemin. "Term Extraction+Term Clustering: an integrated platform for computer-aided terminology." In: *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*. Ed. by A. Lascarides, C. Gardent, and J. Nivre. EACL '99. Stroudsburg, PA, USA: The Association for Computer Linguistics, 1999, pp. 15–22. DOI: 10.3115/977035.977039.

[Bra+07] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. "Large language models in machine translation." In: *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2007, pp. 858–867.

[Bre+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. "Web caching and Zipf-like distributions: evidence and implications." In: *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 1. 1999, 126–134 vol.1. DOI: 10.1109/INFCOM.1999.749260.

[Car67] J. B. Carroll. "On sampling from a lognormal model of word frequency distribution." In: *Computational analysis of present-day American English* 16.3 (1967), pp. 406–424.

[Cha+06] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: A Distributed Storage System for Structured Data." In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*. OSDI '06. Seattle, WA: USENIX Association, 2006, pp. 15–15. URL: http://dl.acm.org/citation.cfm?id=1267308.1267323.

[CH90] K. W. Church and P. Hanks. "Word association norms, mutual information, and lexicography." In: *Comput. Linguist.* 16 (1990), pp. 22–29.

[Cop+02] A. Copestake, F. Lambeau, A. Villavicencio, F. Bond, T. Baldwin, I. A. Sag, and D. Flickinger. "Multiword Expressions: Linguistic Precision and Reusability." In: *3rd International Conference on Language Resources and Evaluation*. LREC '02. Las Palmas, Canary Islands, 2002, pp. 1941–1947.

[CA16] L. M. da Costa Assunção. "A Model for Scientific Workflows with Parallel and Distributed Computing." Doctoral dissertation. Faculdade de Ciências e Tecnologia / Universidade Nova de Lisboa, 2016.

[CO02] J. R. Curran and M. Osborne. "A Very Very Large Corpus Doesn't Always Yield Reliable Estimates." In: *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*. COLING-02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 1–6. DOI: 10.3115/1118853.1118861. URL: http://dx.doi.org/10.3115/1118853.1118861.

[Dai96] B. Daille. "Study and implementation of combined techniques for automatic extraction of terminology." In: *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*. Cambridge: MIT Press, 1996.

[DG04] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters." In: *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[DSL10] B. Debnath, S. Sengupta, and J. Li. "FlashStore: High Throughput Persistent Key-value Store." In: *Proc. VLDB Endow.* 3.1-2 (Sept. 2010), pp. 1414–1425. ISSN: 2150-8097. DOI: 10.14778/1920841.1921015.

[Dia03] G. Dias. "Multiword unit hybrid extraction." In: *Proceedings of the ACL 2003 workshop on Multiword expressions: analysis, acquisition and treatment*. Vol. 18. MWE '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 41–48. DOI: 10.3115/1119282.1119288.

[Dun93] T. Dunning. "Accurate methods for the statistics of surprise and coincidence." In: *Comput. Linguist.* 19 (1993), pp. 61–74.

[Dye+08] C. Dyer, A. Cordova, A. Mont, and J. Lin. "Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce." In: *Proceedings of the Third Workshop on Statistical Machine Translation*. StatMT '08. Columbus, Ohio: Association for Computational Linguistics, 2008, pp. 199–207. ISBN: 978-1-932432-09-1. URL: http://dl.acm.org/citation.cfm?id=1626394.1626427.

[Egg07] L. Egghe. "Untangling Herdan's law and Heaps' law: Mathematical and informetric arguments." In: *JASIST* 58.5 (2007), pp. 702–709. DOI: http://dx.doi.org/10.1002/asi.20524. URL: http://dblp.uni-trier.de/db/journals/jasis/jasis58.html#Egghe07a.

[Eka+10] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. "Twister: A Runtime for Iterative MapReduce." In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: ACM, 2010, pp. 810–818. ISBN: 978-1-60558-942-8. DOI: 10.1145/1851476.1851593.

[ELO08] T. Elsayed, J. Lin, and D. W. Oard. "Pairwise Document Similarity in Large Collections with MapReduce." In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*. HLT-Short '08. Columbus, Ohio: Association for Computational Linguistics, 2008, pp. 265–268. URL: http://dl.acm.org/citation.cfm?id=1557690.1557767.

[FAM00] K. T. Frantzi, S. Ananiadou, and H. Mima. "Automatic recognition of multi-word terms: the C-value/NC-value method." In: *International Journal on Digital Libraries* 3.2 (2000), pp. 115–130. DOI: 10.1007/s007999900023.

[Gab+04] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation." In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, 2004, pp. 97–104.

[GC91] W. Gale and K. Church. "Concordance for parallel texts." In: *of the 7th Annual Conference for the new OED and Text Research*. Oxford, 1991, pp. 40–62.

[Gao+] J. Gao, P. Nguyen, X. Li, C. Thrasher, M. Li, and K. Wang. *A Comparative Study of Bing Web N-gram Language Models for Web Search and Natural Language Processing*.

[GGL03] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google file system." In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43. DOI: 10.1145/945445.945450.

[GD03a] A. Gil and G. Dias. "Efficient mining of textual associations." In: *Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003 International Conference on*. 2003, pp. 549–554. DOI: 10.1109/NLPKE.2003.1275966.

[GD03b] A. Gil and G. Dias. "Using Masks, Suffix Array-based Data Structures and Multidimensional Arrays to Compute Positional Ngram Statistics from Corpora." In: *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18*. MWE '03. Sapporo, Japan: Association for Computational Linguistics, 2003, pp. 25–32. DOI: 10.3115/1119282.1119286.

[Glo13] Glossary Markup Language. *Glossary Markup Language*. 2013. URL: http://www.maxprograms.com/glossml/glossml.pdf.

[GAC12] C. Gonçalves, L. Assunção, and J. C. Cunha. "Data analytics in the cloud with flexible MapReduce workflows." In: *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. CloudCom '12. IEEE Computer Society, 2012, pp. 427–434. DOI: 10.1109/CloudCom.2012.6427527.

[GAC13] C. Gonçalves, L. Assunção, and J. C. Cunha. "Flexible MapReduce Workflows for Cloud Data Analytics." In: *International Journal of Grid and High-Performance Computing (IJGHPC), Special Issue on Cloud Computing - Algorithmic, Experimental, Prototyping and Implementation* 5.4 (2013), pp. 48–64. ISSN: 1938-0259. DOI: 10.4018/IJGHPC.

[GSC15] C. Gonçalves, J. F. da Silva, and J. C. Cunha. "A Parallel Algorithm for Statistical Multiword Term Extraction from Very Large Corpora." In: *High Performance Computing and Communications (HPCC), 2015 IEEE 17th International Conference on*. 2015, pp. 219–224. DOI: 10.1109/HPCC-CSS-ICESS.2015.72.

[GSC16] C. Gonçalves, J. F. Silva, and J. C. Cunha. "An *n*-gram cache for large-scale parallel extraction of multiword relevant expressions with LocalMaxs." In: *Accepted to 12th IEEE International Conference on eScience (eScience 2016)*. IEEE Computer Society, 2016.

[GAV14] A. González, C. Aliagas, and M. Valero. "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality." In: *ACM International Conference on Supercomputing 25th Anniversary Volume*. Munich, Germany: ACM, 2014, pp. 217–226. ISBN: 978-1-4503-2840-1. DOI: 10.1145/2591635.2667170.

[Goo16] Google. *Google Ngram Viewer*. Google. 2016. URL: https://books.google.com/ngrams.

[GG] Y. Gu and R. L. Grossman. "Sector and Sphere: The Design and Implementation of a High Performance Data Cloud." In: URL: http://www.ncbi.nlm.nih.gov/pubmed/19451100.

[H-S13] H-Store. *H-Store*. 2013. URL: http://hstore.cs.brown.edu/.

[Hai67] F. A. Haight. *Handbook of the Poisson Distribution*. New York: John Wiley & Sons, 1967.

[HZ03] S. Heinz and J. Zobel. "Efficient Single-pass Index Construction for Text Databases." In: *J. Am. Soc. Inf. Sci. Technol.* 54.8 (June 2003), pp. 713–729. ISSN: 1532-2882. DOI: 10.1002/asi.10268.

[HC04] Q. Huang and S. J. Cox. "Automatic Call Routing with Multiple Language Models." In: *HLT-NAACL 2004 Workshop: Spoken Language Understanding for Conversational Systems and Higher Level Linguistic Information for Speech Processing*. Ed. by S. Bangalore and H.-K. J. Kuo. Boston, Massachusetts, USA: Association for Computational Linguistics, 2004, pp. 25–30.

[HMC11] S. Huston, A. Moffat, and W. B. Croft. "Efficient Indexing of Repeated N-grams." In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. WSDM '11. Hong Kong, China: ACM, 2011, pp. 127–136. ISBN: 978-1-4503-0493-1. DOI: 10.1145/1935826.1935857.

[Hyp13] Hypertable. *Hypertable*. 2013. URL: http://hypertable.com/.

[IC97] A. Iyengar and J. Challenger. "Improving Web Server Performance by Caching Dynamic Data." In: *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*. USITS'97. Monterey, California: USENIX Association, 1997, pp. 5–5. URL: http://dl.acm.org/citation.cfm?id=1267279.1267284.

[JOV05] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. "BATON: a balanced tree structure for peer-to-peer networks." In: *Proceedings of the 31st international conference on Very large data bases*. VLDB '05. VLDB Endowment, 2005, pp. 661–672.

[JSc16] JSch - Java Secure Channel. *JSch - Java Secure Channel*. JCraft. 2016. URL: http://www.jcraft.com/jsch/.

[JSO13] S. JSON. *Simple, fast, extensible JSON encoder/decoder for Python*. 2013. URL: http://pypi.python.org/pypi/simplejson/.

[KK03] M. F. Kaashoek and D. R. Karger. "Koorde: A Simple Degree-Optimal Distributed Hash Table." In: *Peer-to-Peer Systems II - Second International Workshop, IPTPS*. Vol. 2735. Berkeley, CA, USA: Springer Berlin Heidelberg, 2003, pp. 98–107. DOI: 10.1007/978-3-540-45172-3_9.

[Kal+08] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. "H-Store: a High-Performance, Distributed Main Memory Transaction Processing System." In: *Proc. VLDB Endow.* 1.2 (2008), pp. 1496–1499. ISSN: 2150-8097. DOI: http://doi.acm.org/10.1145/1454159.1454211. URL: http://hstore.cs.brown.edu/papers/hstore-demo.pdf.

[Kim+05] M.-S. Kim, K.-Y. Whang, J.-G. Lee, and M.-J. Lee. "N-gram/2L: A Space and Time Efficient Two-level N-gram Inverted Index Structure." In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, pp. 325–336. ISBN: 1-59593-154-6.

[Kor99] A. Kornai. "Zipf's law outside the middle range." In: *Proceedings of the Sixth Meeting on Mathematics of Language (MOL), University of Central Florida*. 1999, pp. 347–356.

[Kra+11] M. Kratky, R. Baca, D. Bednar, J. Walder, J. Dvorsky, and P. Chovanec. "Index-based n-gram extraction from large document collections." In: *Digital Information Management (ICDIM), 2011 Sixth International Conference on*. 2011, pp. 73–78. DOI: 10.1109/ICDIM.2011.6093324.

[Kuh88] R. Kuhn. "Speech Recognition and the Frequency of Recently Used Words: A Modified Markov Model for Natural Language." In: *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*. COLING '88. Budapest, Hungry: ACM, 1988, pp. 348–350. ISBN: 963 8431 56 3. DOI: 10.3115/991635.991706.

[KM90] R. Kuhn and R. D. Mori. "A cache-based natural language model for speech reproduction." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1990, pp. 570–583.

[KM92] R. Kuhn and R. D. Mori. "Correction to: A cache-based naturalral language model for speech reproduction." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992, pp. 691–692.

[KF11] N. Kulkarni and M. A. Finlayson. "jMWE: a Java toolkit for detecting multi-word expressions." In: *Proceedings of the Workshop on Multiword Expressions: from Parsing and Generation to the Real World*. MWE '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 122–124.

[Lee+12] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon. "Parallel Data Processing with MapReduce: A Survey." In: *SIGMOD Rec.* 40.4 (2012).

[LW05] D. van Leijenhorst and T. van der Weide. "A formal derivation of Heaps' Law." In: *Information Sciences* 170.24 (2005), pp. 263 –272. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/j.ins.2004.03.006.

[LK06] D. Lemire and O. Kaser. "One-Pass, One-Hash n-Gram Statistics Estimation." In: *CoRR* abs/cs/0610010 (2006).

[LK10] D. Lemire and O. Kaser. "Recursive n-gram hashing is pairwise independent, at best." In: *Computer Speech & Language* 24.4 (2010), pp. 698–710. DOI: 10.1016/j.csl.2009.12.001.

[LB97] T. A. Letsche and M. W. Berry. "Large-scale information retrieval with latent semantic indexing." In: *Information Sciences* 100.1 (1997), pp. 105 –137. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/S0020-0255(97)00044-3.

[Lin+10] D. Lin, K. W. Church, H. Ji, S. Sekine, D. Yarowsky, S. Bergsma, K. Patil, E. Pitler, R. Lathbury, V. Rao, K. Dalwani, and S. Narsale. "New Tools for Web-Scale N-grams." In: *Proceedings of the International Conference on Language Resources and Evaluation*. 2010.

[LD10] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, 2010. ISBN: 1608453421, 9781608453429.

[LKM09] J. Lin, S. Konda, and S. Mahindrakar. *Low-Latency, High-Throughput Access to Static Global Resources within the Hadoop Framework*. 2009.

[LA09] Lüdeling and Anke. *Corpus Linguistics: An International Handbook (Handbucher Zur Sprach- Und Kommunikationswissenschaft/Handbo)*. 1st ed. Mouton de Gruyter, Mar. 2009. ISBN: 9783110207330.

[Lun15] LunaCloud. *LunaCloud*. LunaCloud. 2015. URL: www.lunacloud.com/.

[MM90] U. Manber and G. Myers. "Suffix Arrays: A New Method for On-line String Searches." In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '90. San Francisco, California, USA: Society for Industrial and Applied Mathematics, 1990, pp. 319–327. ISBN: 0-89871-251-3.

[Man53] B. Mandelbrot. "On the theory of word frequencies and on related Markovian models of discourse." In: *Structure of Language and its Mathematical Aspects*. Vol. 12. 1953, pp. 190–210.

[Man09] D. Y. Manin. "Mandelbrot's Model for Zipf's Law: Can Mandelbrot's Model Explain Zipf's Law for Language?" In: *Journal of Quantitative Linguistics* 16.3 (2009), pp. 274–285. DOI: 10.1080/09296170902850358.

[MN03] M. Marín and G. Navarro. "Distributed Query Processing Using Suffix Arrays." In: *String Processing and Information Retrieval: 10th International Symposium, SPIRE 2003, Manaus, Brazil, October 8-10, 2003. Proceedings*. Ed. by M. A. Nascimento, E. S. de Moura, and A. L. Oliveira. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 311–325. ISBN: 978-3-540-39984-1. DOI: 10.1007/978-3-540-39984-1_24.

[MV10] S. Martens and V. Vandeghinste. "An Efficient, Generic Approach to Extracting Multi-Word Expressions from Dependency Trees." In: *Proceedings of the Workshop on Multiword Expressions: from Theory to Applications*. MWE '10. Beijing, China: Association for Computational Linguistics, 2010, pp. 84–87.

[MM02] P. Maymounkov and D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 53–65. ISBN: 3-540-44179-4.

[McC76] E. M. McCreight. "A Space-Economical Suffix Tree Construction Algorithm." In: *J. ACM* 23.2 (Apr. 1976), pp. 262–272. ISSN: 0004-5411. DOI: 10.1145/321941.321946.

[Mel+01] S. Melink, S. Raghavan, B. Yang, and H. Garcia-Molina. "Building a Distributed Full-text Index for the Web." In: *ACM Trans. Inf. Syst.* 19.3 (July 2001), pp. 217–241. ISSN: 1046-8188. DOI: 10.1145/502115.502116.

[Mem16] Memcached. *A distributed memory objcect caching system*. Memcached. 2016. URL: http://memcached.org/.

[Mic13a] Microsoft Azure. *Windows Azure Blob Storage*. Microsoft Azure. 2013. URL: http://www.windowsazure.com/en-us/develop/net/how-to-guides/blob-storage/?fb=pt-br.

[Mic13b] Microsoft Azure. *Windows Azure Table Storage*. Microsoft Azure. 2013. URL: http://msdn.microsoft.com/en-us/library/windowsazure/jj553018.aspx.

[NV04] R. Navigli and P. Velardi. "Learning Domain Ontologies from Document Warehouses and Dedicated Web Sites." In: *Comput. Linguist.* 30.2 (June 2004), pp. 151–179. DOI: 10.1162/089120104323093276.

[Ngr13] Ngram Statistics Package. *Ngram Statistics Package*. 2013. URL: http://ngram.sourceforge.net/.

[Ope16] OpenMP. *OpenMP*. OpenMP. 2016. URL: http://openmp.org/wp/.

[Ous+10] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. "The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM." In: *SIGOPS Oper. Syst. Rev.* 43.4 (Jan. 2010), pp. 92–105. ISSN: 0163-5980. DOI: 10.1145/1713254.1713276.

[PBB02] Y. Park, R. J. Byrd, and B. K. Boguraev. "Automatic glossary extraction: beyond terminology identification." In: *Proceedings of the 19th international conference on Computational linguistics*. Vol. 1. COLING '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 1–7. DOI: 10.3115/1072228.1072370.

[Pea02] D. Pearce. "A Comparative Evaluation of Collocation Extraction Techniques." In: *Third International Conference on Language Resources and Evaluation*. Las Palmas, Spain, 2002.

[Pia14] S. T. Piantadosi. "Zipf's word frequency law in natural language: A critical review and future directions." In: *Psychonomic Bulletin & Review* 21 (5 2014), pp. 1112–1130. ISSN: 1069-9384. DOI: 10.3758/s13423-014-0585-6.

[Rat+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A scalable content-addressable network." In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 161–172.

[Ros00] R. Rosenfeld. "Two decades of statistical language modeling: where do we go from here?" In: *Proceedings of the IEEE* 88.8 (2000), pp. 1270–1278. ISSN: 0018-9219. DOI: 10.1109/5.880083.

[RD01] A. I. T. Rowstron and P. Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems." In: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. Middleware '01. London, UK: Springer-Verlag, 2001, pp. 329–350. ISBN: 3-540-42800-3.

[Sch10] N. Schmitt, ed. *An Introduction to Applied Linguistics*. 2nd ed. Routledge, Jan. 2010. ISBN: 9780340984475.

[Sec13] Sector/Sphere. *Sector/Sphere*. 2013. URL: http://sector.sourceforge.net/.

[Sem13] Sematext Key Phrase Extractor. *Sematext Key Phrase Extractor*. Sematext. 2013. URL: http://sematext.com/products/key-phrase-extractor/index.html.

[SGC16] J. F. Silva, C. Gonçalves, and J. C. Cunha. "A theoretical model for *n*-gram distribution in big data *corpora*." In: *2016 IEEE International Conference on Big Data (Big Data)*. 2016, pp. 134–141. DOI: 10.1109/BigData.2016.7840598.

[SL99] J. F. da Silva and G. P. Lopes. "A local Maxima Method and a Fair Dispersion Normalization for Extracting Multiword Units." In: *In Proceedings of the 6th Meeting on the Mathematics of Language*. Quinta Da Torre - Monte Da Caparica, 1999, pp. 369–381.

[Sil+99] J. F. da Silva, G. Dias, S. Guilloré, and J. G. P. Lopes. "Using LocalMaxs Algorithm for the Extraction of Contiguous and Non-contiguous Multiword Lexical Units." In: *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*. Vol. 1695. EPIA '99. Évora, Portugal: Springer Berlin Heidelberg, 1999, pp. 113–132. DOI: 10.1007/3-540-48159-1_9.

[Sin11] R. M. K. Sinha. "Stepwise mining of multi-word expressions in Hindi." In: *Proceedings of the Workshop on Multiword Expressions: from Parsing and Generation to the Real World*. MWE '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 110–115.

[Smi12] R. D. Smith. "Distinct word length frequencies: distributions and symbol entropies." In: *CoRR* abs/1207.2334 (2012).

[Sto+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for internet applications." In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160. DOI: 10.1145/383059.383071.

[Tay15] J. R. Taylor, ed. *The Oxford Handbook of the Word (Oxford Handbooks)*. 1st ed. Oxford University Press, Aug. 2015. ISBN: 9780199641604.

[Ter13a] Term Extraction. *Term Extraction*. fivefilters.org. 2013. URL: http://fivefilters.org/term-extraction/.

[Ter13] TerMine. *TerMine*. National Centre for Text Mining (NaCTeM). 2013. URL: http://www.nactem.ac.uk/software/termine/.

[Ter13b] Terminology Extraction. *Terminology Extraction*. Translated Labs. 2013. URL: http://labs.translated.net/terminology-extraction/.

[The12] The Apache Software Foundation. *Apache Hadoop Distributed File System*. The Apache Software Foundation. 2012. URL: http://hadoop.apache.org/.

[The13a] The Apache Software Foundation. *Apache Accumulo*. The Apache Software Foundation. 2013. URL: http://accumulo.apache.org/.

[The13b] The Apache Software Foundation. *Apache HBase*. The Apache Software Foundation. 2013. URL: http://hbase.apache.org/.

[The16a] The Apache Software Foundation. *Apache Cassandra*. The Apache Software Foundation. 2016. URL: http://cassandra.apache.org/.

[The16b] The Apache Software Foundation. *Apache Commons CLI*. The Apache Software Foundation. 2016. URL: http://commons.apache.org/.

[The16c] The Apache Software Foundation. *Apache Hadoop*. The Apache Software Foundation. 2016. URL: http://hadoop.apache.org/.

[The16d] The Apache Software Foundation. *Apache Logging Services*. The Apache Software Foundation. 2016. URL: http://logging.apache.org/log4j/2.x/.

[TZ92] J. A. Thom and J. Zobel. "A Model for Word Clustering." In: *JASIS* 43.9 (1992), pp. 616–627.

[Top13] Topia Termextract. *Topia Termextract*. PyPI - the Python Package Index. 2013. URL: http://pypi.python.org/pypi/topia.termextract/.

[VV13] P. Valiant and G. Valiant. "Estimating the Unseen: Improved Estimators for Entropy and other Properties." In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., 2013, pp. 2157–2165. URL: http://papers.nips.cc/paper/5170-estimating-the-unseen-improved-estimators-for-entropy-and-other-properties.pdf.

[Vei+99] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. "Adapting Cache Line Size to Application Behavior." In: *Proceedings of the 13th International Conference on Supercomputing*. ICS '99. Rhodes, Greece: ACM, 1999, pp. 145–154. ISBN: 1-58113-164-X. DOI: 10.1145/305138.305188.

[VND08] P. Velardi, R. Navigli, and P. D'Amadio. "Mining the Web to Create Specialized Glossaries." In: *Intelligent Systems, IEEE* 23.5 (2008), pp. 18 –25. DOI: 10.1109/MIS.2008.88.

[Vol13a] Voldemort. *Voldemort*. 2013. URL: http://www.project-voldemort.com/voldemort/.

[Vol13b] VoltDB. *VoltDB*. 2013. URL: http://voltdb.com/.

[Wan+10] K. Wang, C. Thrasher, E. Viegas, X. S.-L. Li, and B.-J. P. Hsu. "An Overview of Microsoft Web N-gram Corpus and Applications." In: 2010. URL: https://www.microsoft.com/en-us/research/publication/an-overview-of-microsoft-web-n-gram-corpus-and-applications/.

[WR91] Y. Wang and L. A. Rowe. "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture." In: *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*. SIGMOD '91. Denver, Colorado, USA: ACM, 1991, pp. 367–376. ISBN: 0-89791-425-2. DOI: 10.1145/115790.115855.

[WSN10] E. Wehrli, V. Seretan, and L. Nerima. "Sentence Analysis and Collocation Identification." In: *Proceedings of the Workshop on Multiword Expressions: from Theory to Applications*. MWE '10. Beijing, China: Association for Computational Linguistics, 2010, pp. 27–35.

[WH05] J. Wermter and U. Hahn. "Finding new terminology in very large corpora." In: *Proceedings of the 3rd international conference on Knowledge capture*. K-CAP '05. New York, NY, USA: ACM, 2005, pp. 137–144. DOI: 10.1145/1088622.1088648.

[Wik16] Wikipedia. *Wikimedia Downloads*. 2016. URL: http://dumps.wikimedia.org.

[WLB07] W. Wong, W. Liu, and M. Bennamoun. "Determining termhood for learning domain ontologies using domain prevalence and tendency." In: *Proceedings of the sixth Australasian conference on Data mining and analytics*. Vol. 70. AusDM '07. Darlinghurst, Australia: Australian Computer Society, Inc., 2007, pp. 47–54.

[Won07] Wong, Wilson and Liu, Wei and Bennamoun, Mohammed. "Determining termhood for learning domain ontologies in a probabilistic framework." In: *Proceedings of the sixth Australasian conference on Data mining and analytics*. Vol. 70. AusDM '07. Darlinghurst, Australia: Australian Computer Society, Inc., 2007, pp. 55–63.

[Yah13a] Yahoo. *Content Analysis Documentation for Yahoo*. Yahoo. 2013. URL: http://developer.yahoo.com/search/content/V2/contentAnalysis.html.

[Yah13b] Yahoo. *Yahoo! Quest*. Yahoo. 2013. URL: http://www.sandbox.yahoo.com/Quest.

[Yah13c] Yahoo. *Yahoo Term Extraction*. Yahoo. 2013. URL: http://developer.yahoo.com/search/content/V1/termExtraction.html.

[Zah+10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. "Spark: Cluster Computing with Working Sets." In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, 2010, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1863103.1863113.

[Zha+04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. "Tapestry: A resilient global-scale overlay for service deployment." In: *IEEE Journal on Selected Areas in Communications* 22.1 (2004), pp. 41–53. DOI: 10.1109/JSAC.2003.818784.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Tech. rep. UCB/CSD-01-1141. EECS Department, University of California, Berkeley, 2001.

[Zip35] G. K. Zipf. "The Psychobiology of Language: An Introduction to Dynamic Philology." In: *Cambridge M.I.T. Press*. Mass, 1935.

[Zip49] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Cambridge, Mass.: Addison-Wesley, 1949.

Appendix A

Controller Configuration File

Listing A.1 shows an example of the configuration file for the LocalMaxs Global method.

Listing A.1: Structure of ParLocalMaxs configuration file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TextMiningConfig>
    <Locale>
        <language>PT</language>
        <country>pt</country>
    </Locale>
    <LocalMaxs>
        <space>32</space>
        <sentenceSeparator>46</sentenceSeparator>
        <specials>!?:;,()</specials>
        <specialsASCII></specialsASCII>
        <specialsAsTokens>true</specialsAsTokens>
        <newLineReplacementSymbol>@#@#</newLineReplacementSymbol>
        <inputBatchSizeForPhase1>256</inputBatchSizeForPhase1>
        <inputBatchSizeForPhase2>256</inputBatchSizeForPhase2>
        <inputBatchSizeForPhase3>256</inputBatchSizeForPhase3>
        <outputBatchSizeForPhase1>4096</outputBatchSizeForPhase1>
        <outputBatchSizeForPhase2>4096</outputBatchSizeForPhase2>
        <outputBatchSizeForPhase3>4096</outputBatchSizeForPhase3>
        <orderOfGrama>1</orderOfGrama>
        <minOrderOfGrama>1</minOrderOfGrama>
        <maxOrderOfGrama>6</maxOrderOfGrama>
        <executeLocalMaxsInDASS>false</executeLocalMaxsInDASS>
    </LocalMaxs>
    <CloudServices>
        <S3BucketName>tm-b20f8af2-964e-49c2-838f-1ffdbf703c95</S3BucketName>
    </CloudServices>
    <Queues>
```

```
29          <sizeInputQueue>16</sizeInputQueue>
30          <sizeOutputQueue>64</sizeOutputQueue>
31      </Queues>
32      <ThreadPools>
33          <ConnectionsPool>
34              <poolSize>1</poolSize>
35              <numberOfWorkers>0</numberOfWorkers>
36          </ConnectionsPool>
37          <WorkersPool>
38              <poolSize>1</poolSize>
39              <numberOfWorkers>0</numberOfWorkers>
40          </WorkersPool>
41      </ThreadPools>
42      <WorkersTypes>
43          <Workers>
44              <worker>tm.workers.CountGramasWorker</worker>
45              <merger>tm.workers.CountGramasMerger</merger>
46              <saver>tm.workers.CountGramasSaver</saver>
47          </Workers>
48          <Workers>
49              <worker>tm.workers.CalcGluesWorker</worker>
50              <merger></merger>
51              <saver>tm.workers.CalcGluesSaver</saver>
52          </Workers>
53          <Workers>
54              <worker>tm.workers.FindRelevancyWorker</worker>
55              <merger></merger>
56              <saver>tm.workers.FindRelevancySaver</saver>
57          </Workers>
58      </WorkersTypes>
59      <BurstSizes>
60          <readBurstSize>512</readBurstSize>
61          <writeBurstSize>2048</writeBurstSize>
62      </BurstSizes>
63      <Cache>
64          <indexDefaultCacheManager>0</indexDefaultCacheManager>
65          <PreFetchSrvServiceName>ZipfPreCacheService</PreFetchSrvServiceName>
66          <PreFetchSrvServiceHostName>localhost</PreFetchSrvServiceHostName>
67          <PreFetchSrvServicePortNumber>40082</PreFetchSrvServicePortNumber>
68          <CacheManager>
69              <CorpusLanguage>English</CorpusLanguage>
70              <ClassName>tm.cache.ZipfReadWrite</ClassName>
71              <CacheEntry>
72                  <size>16384</size>
73                  <frequencyThresholdFactor>10.0</frequencyThresholdFactor>
74                  <intendedFrequencyCoverage>-1.0</intendedFrequencyCoverage>
75                  <dataFileName></dataFileName>
76                  <localFAset>false</localFAset>
77              </CacheEntry>
78          </CacheManager>
```

```
    </Cache>
    <Datasets>
        <Dataset>
            <Input>
                <Type>0</Type>
                <NameExtractor>tm.dass.SentenceExtractor</NameExtractor>
                <NameDASS>tm.dass.LocalInputFileDASS</NameDASS>
            </Input>
            <Output>
                <Type>1</Type>
                <NameDASS>tm.dass.KeyValueTableDASS</NameDASS>
                <DatasetLocation>
                    <DatasetLocationEntry>
                        <serviceName>ngrams01</serviceName>
                        <hostName>localhost01</hostName>
                        <portNumber>40013</portNumber>
                        <table></table>
                        <userName></userName>
                        <password></password>
                    </DatasetLocationEntry>
                </DatasetLocation>
            </Output>
        </Dataset>
        <Dataset>
            <Input>
                <Type>1</Type>
                <NameDASS>tm.dass.KeyValueTableDASS</NameDASS>
                <DatasetLocation>
                    <DatasetLocationEntry>
                        <serviceName>ngrams01</serviceName>
                        <hostName>localhost01</hostName>
                        <portNumber>40013</portNumber>
                        <table></table>
                        <userName></userName>
                        <password></password>
                    </DatasetLocationEntry>
                </DatasetLocation>
            </Input>
            <Output>
                <Type>1</Type>
                <NameDASS>tm.dass.KeyValueTableDASS</NameDASS>
                <DatasetLocation>
                    <DatasetLocationEntry>
                        <serviceName>glues01</serviceName>
                        <hostName>localhost01</hostName>
                        <portNumber>40013</portNumber>
                        <table></table>
                        <userName></userName>
                        <password></password>
                    </DatasetLocationEntry>
```

```
                </DatasetLocation>
            </Output>
        </Dataset>
        <Dataset>
            <Input>
                <Type>1</Type>
                <NameDASS>tm.dass.KeyValueTableDASS</NameDASS>
                <DatasetLocation>
                    <DatasetLocationEntry>
                        <serviceName>glues01</serviceName>
                        <hostName>localhost01</hostName>
                        <portNumber>40013</portNumber>
                        <table></table>
                        <userName></userName>
                        <password></password>
                    </DatasetLocationEntry>
                </DatasetLocation>
            </Input>
            <Output>
                <Type>1</Type>
                <NameDASS>tm.dass.KeyValueTableDASS</NameDASS>
                <DatasetLocation>
                    <DatasetLocationEntry>
                        <serviceName>expressions01</serviceName>
                        <hostName>localhost01</hostName>
                        <portNumber>40013</portNumber>
                        <table></table>
                        <userName></userName>
                        <password></password>
                    </DatasetLocationEntry>
                </DatasetLocation>
            </Output>
        </Dataset>
    </Datasets>
</TextMiningConfig>
```

Appendix B

Tools Configuration File

Listing B.1 shows the main structure of the configuration file used by the tools.

Listing B.1: Structure of tools configuration file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AppConfig>
    <CloudProviders>
        <ProviderAmazon>
            <EnableConnectToSelected>true</EnableConnectToSelected>
            <NumberOfExecutors>5</NumberOfExecutors>
            <AccountsAmazon>
                <AccountAmazon>
                    <FriendlyName>AWS Account</FriendlyName>
                    <CredsFileName>AwsCredentials.properties</CredsFileName>
                    <Storage>233f3ef8-8f01-430f-91de-6b5846a294c6</Storage>
                </AccountAmazon>
            </AccountsAmazon>
            <AMIOwners>
                <AMIOwner>
                    <Owner>999870900606</Owner>
                    <FriendlyName>User</FriendlyName>
                </AMIOwner>
            </AMIOwners>
            <InstancesTypes>
                <InstanceType>
                    <Name>m1.small</Name>
                    <Description>Micro</Description>
                </InstanceType>
            </InstancesTypes>
            <InstancesTags>
                <InstanceTag>
                    <InstanceTag>awards</InstanceTag>
```

```
29                </InstanceTag>
30            </InstancesTags>
31        </ProviderAmazon>
32        <ProviderAzure>
33            <EnableConnectToSelected>true</EnableConnectToSelected>
34            <NumberOfExecutors>5</NumberOfExecutors>
35            <AccountsAzure>
36                <AccountAzure>
37                    <FriendlyName>Azure Accunt</FriendlyName>
38                    <CredsFileName>AzureCredentials.xml</CredsFileName>
39                    <Storage>tm-31a10979-d39f-4225-98b5-36db7a6e7d94</Storage>
40                </AccountAzure>
41            </AccountsAzure>
42        </ProviderAzure>
43        <ProviderLuna>
44            <EnableConnectToSelected>true</EnableConnectToSelected>
45            <NumberOfExecutors>5</NumberOfExecutors>
46            <AccountsLunaCloud>
47                <AccountLunaCloud>
48                    <FriendlyName>Luna Account</FriendlyName>
49                    <CredsFileName>LunaCredentials.xml</CredsFileName>
50                </AccountLunaCloud>
51            </AccountsLunaCloud>
52        </ProviderLuna>
53    </CloudProviders>
54    <WorflowSettings>
55        <Name>workflow.xml</Name>
56        <AditionalLibs>tools/lib/*</AditionalLibs>
57        <InputFileName>Input.txt</InputFileName>
58        <BaseFileOptionsJVM>optionsJVM.txt</BaseFileOptionsJVM>
59        <OptionsJVM>
60            <OptionJVM>
61                <TaskName>Task1</TaskName>
62                <JVMOptionsFileName>OptionsTask1.txt</JVMOptionsFileName>
63                <JVMOptionsLocalDirectory>.\</JVMOptionsLocalDirectory>
64                <JVMOptionsRemoteDirectory>tools/</JVMOptionsRemoteDirectory>
65            </OptionJVM>
66        </OptionsJVM>
67    </WorflowSettings>
68    <WorkflowTasksNames>
69        <TasksMapReduce>
70            <RecordExtractor>tasks.mr.RecordExtractor</RecordExtractor>
71            <Mapper>tasks.mr.Mapper</Mapper>
72            <Reducer>tasks.mr.Reducer</Reducer>
73            <MapperCollector>tasks.mr.MapCollector</MapperCollector>
74            <ReducerCollector>tasks.mr.RedCollector</ReducerCollector>
75            <TaskSpliter>tasks.mr.TaskSpliter</TaskSpliter>
76            <TaskMapper>tasks.mr.TaskMapper</TaskMapper>
77            <TaskMerger>tasks.mr.TaskMerger</TaskMerger>
78            <TaskReducer>tasks.mr.TaskReducer</TaskReducer>
```

```
        </TasksMapReduce>
        <TasksTextMining>
            <StartUp>tasks.tm.TaskStartUp</StartUp>
            <CleanUp>tasks.tm.TaskCleanUp</CleanUp>
            <Statistics>tasks.tm.TaskStatistics</Statistics>
            <TextMining>tasks.tm.TaskTextMiningV1</TextMining>
            <Dummy>tasks.tm.TaskDummy</Dummy>
        </TasksTextMining>
    </WorkflowTasksNames>
    <RemoteHostsSettings>
        <DefaultConection>0</DefaultConection>
        <OptionsSSH>
            <OptionSSH>
                <ConnectionName>Localhost</ConnectionName>
                <UserName>user</UserName>
                <Password>password</Password>
                <PortNumber>22</PortNumber>
                <PrivateKeyFileName>auth.pem</PrivateKeyFileName>
                <RemoteDirectory>tools</RemoteDirectory>
            </OptionSSH>
        </OptionsSSH>
    </RemoteHostsSettings>
    <FileSets>
        <FileSet>
            <FileSetName>File Set</FileSetName>
            <FileSetItem>file1</FileSetItem>
            <FileSetItem>file2</FileSetItem>
        </FileSet>
    </FileSets>
    <AWARDConfigurationFile>
        <JavaBinary>java</JavaBinary>
        <MainSpaceName>MySpace</MainSpaceName>
        <AwaExecBaseDir>tools/AwaExecutor/</AwaExecBaseDir>
        <AwaExec>AwaExecutor.jar</AwaExec>
        <WorkflowDirectory>tools/wkfs/</WorkflowDirectory>
        <ConfigFileNamePublic>awardConfig-Public.xml</ConfigFileNamePublic>
        <ConfigFileNamePrivate>awardConfig.xml</ConfigFileNamePrivate>
    </AWARDConfigurationFile>
    <AWARDTools>
        <DirectoryAWAWaiting>tools/LaunchAWAwaiting/</DirectoryAWAWaiting>
        <JarLaunchAWAWaiting>LaunchAWAwaiting.jar</JarLaunchAWAWaiting>
        <CmdLaunchAWAWaiting>java -jar %s %s %s %s</CmdLaunchAWAWaiting>
        <DirectoryWorkflowStart>tools/LaunchWkfstart/</DirectoryWorkflowStart>
        <JarLaunchWorkflowStart>LaunchWkfstart.jar</JarLaunchWorkflowStart>
        <CmdLaunchWorkflowStart>java -jar %s %s %s</CmdLaunchWorkflowStart>
        <DumpTimes>tools/DumpTimes.jar</DumpTimes>
        <DumpTimesTextMining>tools/DumpTimesTM.jar</DumpTimesTextMining>
    </AWARDTools>
    <TextMiningSettings>
        <ConfigFile>config-TextMining.xml</ConfigFile>
```

```
129         <CorpuraDirectory>.</CorpuraDirectory>
130         <TargetCorpuraDirectory>files/</TargetCorpuraDirectory>
131         <TargetProfilingDirectory>/tmp/</TargetProfilingDirectory>
132         <WorkflowGenerators>
133             <Generator>tm.wks.gen.SinglePartition</Generator>
134             <Generator>tm.wks.gen.MultiplePartitionsMultipleTasks</Generator>
135             <Generator>tm.wks.gen.MultiplePartitionsSingleTask</Generator>
136         </WorkflowGenerators>
137         <KeySizeStatisticsFieldsRequired>
138             <Mininum>false</Mininum>
139             <Maximum>false</Maximum>
140             <Average>false</Average>
141             <StandardDeviation>false</StandardDeviation>
142             <CoefficientOfVariation>false</CoefficientOfVariation>
143             <NumberOfPrimaryKeys>true</NumberOfPrimaryKeys>
144             <NumberOfAllKeys>true</NumberOfAllKeys>
145             <NumberOfValues>true</NumberOfValues>
146             <NumberOfSingletons>true</NumberOfSingletons>
147         </KeySizeStatisticsFieldsRequired>
148         <ConfigMemoryUsageFieldsRequired>
149             <IsDeepMemoryUsage>false</IsDeepMemoryUsage>
150             <HeapMax>false</HeapMax>
151             <HeapCommitted>false</HeapCommitted>
152             <HeapUsed>true</HeapUsed>
153             <NonHeapMax>false</NonHeapMax>
154             <NonHeapCommitted>false</NonHeapCommitted>
155             <NonHeapUsed>false</NonHeapUsed>
156         </ConfigMemoryUsageFieldsRequired>
157         <ConfigSaveOptionsFieldsRequired>
158             <UseFormatXML>false</UseFormatXML>
159             <UseHeaders>false</UseHeaders>
160             <NegativePhase1>true</NegativePhase1>
161             <DumpPhase1>true</DumpPhase1>
162             <NegativePhase2>false</NegativePhase2>
163             <DumpPhase2>false</DumpPhase2>
164             <NegativePhase3>false</NegativePhase3>
165             <DumpPhase3>false</DumpPhase3>
166             <Phase3Threshold>0</Phase3Threshold>
167         </ConfigSaveOptionsFieldsRequired>
168         <OptionsMetrics>
169             <CacheMetricsFileName>CacheMetrics</CacheMetricsFileName>
170             <CacheMetricsOnLogger>false</CacheMetricsOnLogger>
171             <SaveCacheMetricsAfterPhase2>true</SaveCacheMetricsAfterPhase2>
172             <ResetCacheAfterGlueCalc>false</ResetCacheAfterGlueCalc>
173             <CacheContentsFileName>CacheContents</CacheContentsFileName>
174             <SaveCacheAfterGlueCalc>false</SaveCacheAfterGlueCalc>
175             <CacheSaveIncludeNGramsValues>false</CacheSaveIncludeNGramsValues>
176             <BloomMetricsFileName>BloomMetrics</BloomMetricsFileName>
177             <SaveBloomMetricsAfterPhase2>false</SaveBloomMetricsAfterPhase2>
178             <ShowBloomMetricsOnLogger>false</ShowBloomMetricsOnLogger>
```

```
            <TestBloomFilterOnKVS>false</TestBloomFilterOnKVS>
            <ScriptsDirectory>/home/textm/phd-launch</ScriptsDirectory>
            <LaunchScripts>false</LaunchScripts>
            <StartUpScriptName>startUp.sh</StartUpScriptName>
            <CleanUpScriptName>cleanUp.sh</CleanUpScriptName>
            <StartUpScriptArguments>Args</StartUpScriptArguments>
            <CleanUpScriptArguments>Args</CleanUpScriptArguments>
        </OptionsMetrics>
    </TextMiningSettings>
    <ApplicationsSettings>
        <CommandsFile>config-Commands.txt</CommandsFile>
        <TagsFile>config-Tags.txt</TagsFile>
        <Apps>Apps</Apps>
        <AppLibs>tools/lib/apps/</AppLibs>
        <AWARDMapReduceLibs>tools/lib/mr/</AWARDMapReduceLibs>
        <TexMiningLibs>tools/lib/tm/</TexMiningLibs>
        <TargetFilePathSeparator>;</TargetFilePathSeparator>
        <TargetFileSeparator>\</TargetFileSeparator>
        <WorkingDirectory>C:\Users\cajo\AppData\Local\Temp\</WorkingDirectory>
        <WebTools>
            <BaseAddress>http://cjsg.dynip.sapo.pt/~cajo/phd/</BaseAddress>
            <DownloadHosts>download.php</DownloadHosts>
        </WebTools>
    </ApplicationsSettings>
    <ApplicationsPorts>
        <ApplicationPort>
            <AppName>AwardSpace</AppName>
            <AppHostName>localhost</AppHostName>
            <AppPortNumber>40001</AppPortNumber>
            <AppLookupPortNumber>-1</AppLookupPortNumber>
            <AppLookupName></AppLookupName>
            <AppHttpPortNumber>40002</AppHttpPortNumber>
        </ApplicationPort>
    </ApplicationsPorts>
    <Log4JavaFileSettings>
        <Unbounded>log4j-unbounded.properties</Unbounded>
        <Bounded>log4j.properties</Bounded>
        <TasksUnbounded>log4j-unbounded-Tasks.properties</TasksUnbounded>
        <TasksBounded>log4j-Tasks.properties</TasksBounded>
    </Log4JavaFileSettings>
</AppConfig>
```

Appendix C

Complementary Data for the Static plus Dynamic Cache System

Tables C.1 to C.3 show the obtained FAset sizes for *corpus* with sizes 64, 128 and 256 Mw when varying the number of machines (K) from 6 up to 18.

Table C.1: Per machine FAset sizes (in number of n-grams) *versus* K for a *corpus* of 128 Mw

K	$\left\|\left(FA_1, g_2, h_{RO_{C_1}}\right)\right\|$	$\left\|\left(FA_2, g_3, h_{RO_{C_2}}\right)\right\|$	$\left\|\left(FA_3, g_4, h_{RO_{C_3}}\right)\right\|$
6	91 705	869 764	898 035
9	81 144	729 928	700 050
12	76 632	626 081	595 508
18	67 062	487 520	461 299

Table C.2: Per machine FAset sizes (in number of n-grams) *versus* K for a *corpus* of 256 Mw

K	$\left\|\left(FA_1, g_2, h_{RO_{C_1}}\right)\right\|$	$\left\|\left(FA_2, g_3, h_{RO_{C_2}}\right)\right\|$	$\left\|\left(FA_3, g_4, h_{RO_{C_3}}\right)\right\|$
12	112 761	884 693	851 694
18	99 698	714 976	645 794

Table C.3: Per machine FAset sizes (in number of n-grams) *versus* K for a *corpus* of 511 Mw

K	$\left\|\left(FA_1, g_2, h_{RO_{C_1}}\right)\right\|$	$\left\|\left(FA_2, g_3, h_{RO_{C_2}}\right)\right\|$	$\left\|\left(FA_3, g_4, h_{RO_{C_3}}\right)\right\|$
18	154 939	1 003 684	929 668

Tables C.4 to C.14 show the measured values regarding the analysis of the behavior of a cache system composed by a static cache followed by a dynamic one, that can be used in an analysis similar to the one presented in section 7.2.3.3 of chapter 7 on page 224, for different corpus sizes, namely 64, 128 and 256 Mw when varying the number of machines (K) from 3 up to 18.

Table C.4: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 64 Mw and 3 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	88.06	91.42
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	63.69
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	8.16	10.24	8.16
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		8.88	19.21
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			10.57
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	11.84	1.70	0.42
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		31.12	17.10
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			59.43

Table C.5: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 64 Mw and 6 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	86.17	89.92
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	57.53
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	5.75	10.55	8.80
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		4.79	23.48
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			6.87
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	14.25	3.28	1.28
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		35.21	18.99
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			63.13

Table C.6: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 64 Mw and 9 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	84.72	88.75
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	53.83
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	4.78	10.79	9.26
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		2.81	24.16
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			5.05
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	15.22	4.49	1.99
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		37.19	22.01
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			64.95

Table C.7: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 64 Mw and 12 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	83.62	87.96
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	50.98
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	3.98	11.09	9.75
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		1.74	23.82
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			4.03
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	16.02	5.29	2.29
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		38.26	25.20
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			65.97

Table C.8: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 64 Mw and 18 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	82.00	86.51
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	47.46
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	3.19	11.76	10.37
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		0.19	22.94
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			2.39
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	16.81	6.23	3.12
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		39.81	29.60
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			67.61

Table C.9: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 128 Mw and 6 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	86.93	90.83
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	59.90
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	6.01	10.07	8.05
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		5.72	19.52
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			7.56
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	13.99	3.00	1.12
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		34.28	20.58
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			62.44

Table C.10: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 128 Mw and 9 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	85.64	89.99
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	56.40
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	5.09	10.26	8.27
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		3.95	21.35
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			5.78
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	14.91	4.10	1.74
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		36.05	22.25
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			64.22

Table C.11: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 128 Mw and 12 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	84.58	89.10
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	53.70
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	4.34	10.60	8.92
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		2.94	22.57
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			4.99
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	15.66	4.82	1.98
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		37.06	23.72
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			65.01

Table C.12: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 128 Mw and 18 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \left\lvert all_{glue_g Ref_{1\text{–}gram}} \right\rvert$	80.00	83.19	88.03
$hits_{RO_{C_2}} / \left\lvert all_{glue_g Ref_{2\text{–}gram}} \right\rvert$		60.00	50.22
$hits_{RO_{C_3}} / \left\lvert all_{glue_g Ref_{3\text{–}gram}} \right\rvert$			30.00
$hits_{RW_{C_1}} / \left\lvert all_{glue_g Ref_{1\text{–}gram}} \right\rvert$	3.59	11.17	9.26
$hits_{RW_{C_2}} / \left\lvert all_{glue_g Ref_{2\text{–}gram}} \right\rvert$		1.45	23.21
$hits_{RW_{C_3}} / \left\lvert all_{glue_g Ref_{3\text{–}gram}} \right\rvert$			3.73
$misses_{(RO+RW)_{C_1}} / \left\lvert all_{glue_g Ref_{1\text{–}gram}} \right\rvert$	16.41	5.64	2.71
$misses_{(RO+RW)_{C_2}} / \left\lvert all_{glue_g Ref_{2\text{–}gram}} \right\rvert$		38.55	26.57
$misses_{(RO+RW)_{C_3}} / \left\lvert all_{glue_g Ref_{3\text{–}gram}} \right\rvert$			66.27

Table C.13: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 256 Mw and 12 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \left\lvert all_{glue_g Ref_{1\text{–}gram}} \right\rvert$	80.00	85.32	90.04
$hits_{RO_{C_2}} / \left\lvert all_{glue_g Ref_{2\text{–}gram}} \right\rvert$		60.00	56.18
$hits_{RO_{C_3}} / \left\lvert all_{glue_g Ref_{3\text{–}gram}} \right\rvert$			30.00
$hits_{RW_{C_1}} / \left\lvert all_{glue_g Ref_{1\text{–}gram}} \right\rvert$	4.63	10.25	8.24
$hits_{RW_{C_2}} / \left\lvert all_{glue_g Ref_{2\text{–}gram}} \right\rvert$		3.94	20.78
$hits_{RW_{C_3}} / \left\lvert all_{glue_g Ref_{3\text{–}gram}} \right\rvert$			5.98
$misses_{(RO+RW)_{C_1}} / \left\lvert all_{glue_g Ref_{1\text{–}gram}} \right\rvert$	15.37	4.44	1.72
$misses_{(RO+RW)_{C_2}} / \left\lvert all_{glue_g Ref_{2\text{–}gram}} \right\rvert$		36.06	23.04
$misses_{(RO+RW)_{C_3}} / \left\lvert all_{glue_g Ref_{3\text{–}gram}} \right\rvert$			64.02

Table C.14: Average number of hit and miss values in percentage (%) for isolated glues and a *corpus* of 256 Mw and 18 machines.

Hit and miss percentages	g_2	g_3	g_4
$hits_{RO_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	80.00	84.11	89.15
$hits_{RO_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		60.00	52.76
$hits_{RO_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			30.00
$hits_{RW_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	3.93	10.72	8.47
$hits_{RW_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		2.59	22.10
$hits_{RW_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			4.78
$misses_{(RO+RW)_{C_1}} / \lvert all_{glue_g Ref_{1\text{–}gram}} \rvert$	16.07	5.17	2.38
$misses_{(RO+RW)_{C_2}} / \lvert all_{glue_g Ref_{2\text{–}gram}} \rvert$		37.41	25.14
$misses_{(RO+RW)_{C_3}} / \lvert all_{glue_g Ref_{3\text{–}gram}} \rvert$			65.22


UNIVERSIDADE NOVA DE LISBOA
OMNIS CIVITAS CONTRA SE DIVISA NON STABIT

Carlos Jorge de Sousa Gonçalves

Mestre em Engenharia Informática

Parallel and Distributed Statistical-based Extraction of Relevant Multiwords from Large *Corpora*

Dissertação para obtenção do Grau de Doutor em

Informática

Dezembro, 2017

Carlos Jorge de Sousa Gonçalves

Mestre em Engenharia Informática

Parallel and Distributed Statistical-based Extraction of Relevant Multiwords from Large *Corpora*

Dissertação para obtenção do Grau de Doutor em

Informática

Dezembro, 2017

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Carlos Gonçalves 2017

Parallel and Distributed Statistical-based Extraction of Relevant Multiwords from Large Co 2017