José Henrique Martins Cabrita Rio

Licenciatura em Engenharia Informática

Persistência de Dados na Cloud recorrendo a uma Estrutura de Dados Métrica

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Prof. Doutor Sérgio Marco Duarte, Prof. Auxiliar,
Universidade Nova de Lisboa

Júri:

Presidente: Prof. Doutor António Maria Lobo César Alarcão Ravara

Arguente: Prof. Doutor Luís Filipe Fernandes Silva Marcelino

Vogal: Prof. Doutor Sérgio Marco Duarte

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2011

Persistência de Dados na Cloud recorrendo a uma Estrutura de Dados Métrica

Copyright © José Henrique Martins Cabrita Rio, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objectivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Para os meus pais.

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao Professor Sérgio Duarte, pela motivação, crítica e apoio prestado. Sem a sua colaboração e o seu conhecimento não teria sido possível realizar esta dissertação.

Ao corpo docente do Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa pelos ensinamentos prestados, ao longo dos últimos anos, que em muito contribuíram para a evolução dos meus conhecimentos. Uma nota de especial apreço para os Professores Fernanda Barbosa, Henrique Domingos e Margarida Mamede pelas ideias e apoio prestado ao longo desta dissertação.

Aos meus colegas que me acompanharam neste percurso e sem os quais não seria possível chegar até aqui.

Aos meus pais, José Manuel e Henriqueta, por todo o carinho, dedicação, educação e por terem feito de mim o que sou hoje.

À minha namorada, Rita, pelo amor, paciência e por estar ao meu lado nos momentos bons e menos bons, ao longo dos últimos anos.

Ao resto da família e amigos pelo apoio, motivação, amizade e por nunca me deixarem desistir dos meus objectivos.

A todos os que não foram mencionados e que, de alguma forma, contribuíram para a minha formação enquanto Ser Humano, o meu mais sincero agradecimento.

Resumo

A rápida evolução dos dispositivos móveis, nomeadamente através da integração de sensores, tem permitido o aparecimento de novos paradigmas de aplicações, como o sensoriamento participado. Explorando a mobilidade, captura e partilha de dados por vários utilizadores tem-se por objectivo a criação de novas interpretações do mundo físico. Contudo, a validade e interesse desses dados não se esgota momentaneamente, havendo grande interesse na preservação destes. A sua persistência possibilita, por exemplo, criar aplicações onde se explora uma dimensão de evolução temporal ou histórica.

O armazenamento de grandes quantidades de dados, como os provenientes do sensoriamento participado, levanta diversos problemas. Além de exigir uma capacidade crescente adequada e uma ampla disponibilidade dos dados, há também a necessidade de os organizar de modo a assegurar um acesso eficiente.

Nesta dissertação é proposta uma solução para a persistência de dados de sensoriamento participado baseada no armazenamento em *cloud* e no recurso a uma estrutura de dados métrica, com vista à sua indexação. Por um lado, procura-se beneficiar da capacidade quase infinita da *cloud*. Por outro lado, organizando os dados com base numa função de distância métrica, pretende-se explorar os padrões de forte relação geográfica e temporal, tipicamente associados aos dados de sensoriamento participado. A avaliação experimental do protótipo realizado na plataforma Google App Engine mostra que esta abordagem atinge os objectivos principais, apresentando tempos de acesso adequados.

Palavras-chave: Sensoriamento Participado, persistência na *cloud*, pesquisa por proximidade

Abstract

The rapid evolution of mobile devices, through the integration of sensors, has allowed the emergence of new application paradigms such as participatory sensing. Exploiting the mobility, capture and data sharing by multiple users it has been designed to create new interpretations of the physical world. However, the validity and the interest of these data does not end in that moment and there is a great interest in their preservation. Their persistence enables, for example, the deployment of applications where we can explore a dimension of temporal or historical evolution.

Storing large amounts of data, such as those from participatory sensing, raises several problems. Besides requiring an adequate growing capacity and a wide availability, there is also a need of organizing them in a way to ensure an efficient access.

In this work we propose a solution to persist data from participatory sensing, based on the cloud storage and the use of a metric data structure in order to their indexing. On one hand, we seek to benefit from the almost infinite capacity of the cloud. On the other hand, organizing the data based on a metric distance function, we intend to explore the patterns of high spacial and geographical relationship, typically associated with the participatory sensing data. The experimental evaluation of the prototype developed on the Google App Engine platform shows that this approach achieves the main objectives, with appropriate access times.

Keywords: Participatory Sensing, persistence in cloud, range search

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

Nas últimas décadas a evolução nos campos da ciência e da tecnologia tem permitido ao ser humano criar aparelhos que melhoram, de forma mais ou menos significativa, a sua vida. Esta evolução tem passado pela procura de novas funcionalidades, dimensões mais reduzidas e, em particular, mobilidade.

Todo este desenvolvimento levou ao aparecimento de telefones móveis, que, gradualmente, foram adquirindo novas capacidades. Hoje, os telemóveis possuem significativos poderes computacionais e de armazenamento, têm ligação à Internet e, muitos deles, já apresentam sensores (de velocidade, posição, temperatura, etc.).

A vulgarização dos dispositivos móveis com sensores e da Internet, bem como a integração destes, levou ao surgimento de um novo paradigma aplicacional denominado sensoriamento participado [3, 4, 5]. Com ele, a partilha de informações relativas a localizações no espaço e no tempo tornou-se possível, permitindo a um grande número de pessoas usufruir destes dados para melhorar o seu dia-a-dia, através da recolha feita nos dispositivos dos vários utilizadores.

Neste paradigma emergente, vários dispositivos móveis, de forma colaborativa, formam uma vasta rede sensorial, cobrindo uma zona ampla e permitindo novas interpretações do mundo, com maior dinamismo e sem os custos associados à criação de uma infra-estrutura física dedicada.

Existem actualmente algumas aplicações que ilustram o potencial desta área, como são os casos do Pothole Patrol [6] e do CarTel [7]. O primeiro caso é especialmente vocacionado para a inspecção do estado de conservação das estradas, fazendo uma recolha dos locais onde potencialmente haverá irregularidades no piso. Já o segundo, tem como objectivo a identificação de padrões de trânsito: a utilização dos dados capturados por sensores de posição presentes nos dispositivos dos utilizadores e a combinação com as

respectivas estampilhas temporais, permite criar um mapa de tráfego no espaço e no tempo.

A construção de uma boa representação da realidade é facilitada pela existência de um número elevado de amostras. As amostras apresentam características espaciais e temporais, devendo cada registo guardar os dados das coordenadas GPS, uma estampilha temporal e também os outros atributos que se pretendem partilhar, sobre os quais recairá o factor dominante do tamanho deste, podendo ser reduzido se se tratarem de dados sensoriais ou bastante grande se forem dados *media*.

É neste âmbito que surge o tema da dissertação: a persistência de dados provenientes de sensoriamento participado.

1.1 Motivação

Sem a persistência dos dados adquiridos todas as vistas criadas apresentam essencialmente características espaciais, podendo responder a perguntas simples, relacionadas com o que é verificado correntemente num determinado local. O sensoriamento participado pode, no entanto, comportar a temporalidade de todos os dados permitindo interpretações mais complexas, como por exemplo, a averiguação de tendências num determinado local, sem excluir outras mais simples como interrogações acerca de determinado espaço e tempo. A conjugação de ambas as vertentes permite ainda determinar se o que ocorre correntemente é comum face ao passado registado.

Estas questões só serão passíveis de resposta se os vários dados forem armazenados para consulta posterior. Uma utilidade interessante que poderá resultar deste armazenamento é, por exemplo, a escolha de rotas por parte de um sistema de navegação por GPS. Estes dados permitem fazer o cálculo não só através da distância ou do tempo médio que é levado a percorrer cada trajecto, mas também com base na estatística dos dados guardados (relativos a velocidades, por exemplo), podendo o percurso ideal variar consoante o dia, ou o momento deste.

Além dos valores que se pretendem partilhar, estes terão associados outros, de dois âmbitos diferentes, relacionados com a localidade e temporalidade dos factos. Assim sendo, qualquer valor guardado terá sempre associado um par de coordenadas geográficas (obtidas com o auxílio de GPS) bem como uma estampilha temporal que o permita localizar no tempo.

O armazenamento dos dados é uma tarefa que apresenta várias vertentes. Para além do tamanho variável destes, quanto maior a rede, maior será o número de amostras produzidas e, por sua vez, maior a quantidade de informação a armazenar. Outra das preocupações reside ao nível dos padrões de consumo dos dados. Na perspectiva em que muitas das aplicações de sensoriamento participado pretendem ser um auxílio em questões mais práticas, relacionadas com o dia-a-dia dos seus utilizadores e o ambiente que os rodeia, poderemos, então, tirar a seguinte ilação: o consumo dos dados tenderá a ser próximo do seu local de produção. Será, portanto, importante prever uma arquitectura

que beneficie a distribuição dos dados de acordo com proximidade a estes locais.

A *cloud* materializa um novo paradigma de alojamento e processamento de dados. Este teve origem no aproveitamento dos excedentes de recursos das grandes companhias, sendo actualmente um negócio com expressão própria, denotada através do surgimento de um grande número de provedores independentes. A *cloud*, como paradigma, caracteriza-se por uma elevada capacidade de armazenamento, elevada disponibilidade e um veloz acesso aos dados. A capacidade de armazenamento além de elevada, é igualmente elástica, podendo aumentar ou diminuir de acordo com as necessidades do sistema, o que viabiliza a sua livre expansão. Fiabilidade e suporte para a implementação de políticas de segurança são outras das características interessantes que a *cloud* oferece.

Para resolver o problema da persistência dos dados parece ser bastante promissora a colocação destes dados na *cloud* [8]. Esta disponibiliza uma grande capacidade de computação e armazenamento para as massas, sendo uma hipótese bastante viável para certo tipo de aplicações/serviços.

Embora dispense os custos de montagem e manutenção de uma infra-estrutura própria, implica o pagamento de taxas mensais (de acordo com a capacidade de armazenamento utilizada, número de acessos realizados ou tempo de funcionamento do CPU, por exemplo), consoante o(s) tipo(s) de serviço(s) requerido(s). Existe, assim, interesse em desenvolver uma solução que racionalize os custos, dispersando os dados de acordo com a qualidade de serviço desejada. Uma das soluções poderá ser baseada, por exemplo, na criação de metadados de indexação em *clouds* com maior capacidade de computação (tipicamente mais dispendiosas), efectuando a persistência dos dados noutras mais simples e económicas.

1.2 Principais requisitos

No sistema que se pretende desenvolver, existem três requisitos fundamentais: armazenamento, indexação e diversidade. A sua relevância e as diferentes abordagens de resolução serão alvo de discussão individual nas subsecções seguintes.

1.2.1 Armazenamento

O armazenamento de informação é uma das vertentes de grande parte das aplicações informáticas. O suporte desta poderá ser problemático, devido à necessidade de uma estrutura que o capacite. A complexidade da infra-estrutura é variável, de acordo com o número de dados produzidos pelo sistema, mas que em grande escala poderá dar origem a elevados custos de investimento e manutenção.

No contexto do sensoriamento participado poderão ser enunciadas as seguintes propriedades acerca dos dados a armazenar:

- A quantidade de armazenamento é elevada, devido ao grande número de amostras produzidas (essencial para a credibilidade do sistema)

- A dimensão de cada registo é dependente da aplicação em causa, podendo variar entre poucos *bytes* (que contêm apenas alguma informação sensorial) até muitos *kilobytes* ou até mesmo *megabytes* (onde o objectivo é a partilha de mensagens ou dados *media*, por exemplo). O CarTel [7] e o Microblogging [9] produzem dados divergentes em dimensão.
- O número de dados presentes no sistema aumenta com o tempo. A constante submissão de informação implica que a capacidade de armazenamento seja crescente.
- Os dados apresentam uma forte significância quando conhecidos o momento e o local de produção. Estes deverão ser estampilhados temporal e geograficamente (através de coordenadas de GPS, por exemplo).

Uma das formas de minimizar o custo do armazenamento seria através da escolha de um modelo complemente distribuído, onde todos os dados estariam guardados nos aparelhos dos utilizadores. Eles próprios possuem alguma capacidade de armazenamento, enfatizando o ideal de comunidade e partilha de informação e recursos, enunciado como uma das bases do paradigma do sensoriamento participado. Neste modelo, cada um dos dispositivos limitar-se-ia a arquivar os dados produzidos por si próprio, cabendo ao utilizador ser responsável pela quantidade de dados que produz, dado que também o é pela sua manutenção. Porém, a nível prático, isto poderá ser prejudicial para a solução desejada. Aquando de uma pesquisa, por exemplo, para uma resposta completa, a pergunta teria de percorrer todos os nós na rede, o que implicaria um atraso na obtenção da mesma. No entanto, esta resposta dependeria sempre dos nós que naquele momento estivessem conectados e, por conseguinte, parte da informação (ou até mesmo toda) poderia não se encontrar disponível, ficando inacessível para o utilizador que efectuou a pesquisa. O sistema de comunicação correria, ainda, um sério risco de estrangular. Se a resposta for enviada directamente para o utilizador final e não houver agregação intermédia, o volume total poderá ser excessivo, sobrecarregando a aplicação.

Em contraste com a solução totalmente distribuída, é possível ter uma arquitectura completamente centralizada. Uma das principais fragilidades, que já foi enunciada na fase introdutória desta secção, é o custo da infra-estrutura, havendo poucos interessados em realizar tal investimento inicial.

Além do custo, há outras questões que minimizam as grandes vantagens de uma arquitectura deste estilo. O servidor é um ponto central de falha e de estrangulamento. Se neste ocorrer algum erro, a aplicação poderá deixar de funcionar correctamente, o que não é de todo desejável. Para além disso, com um número excessivamente elevado de pedidos, o servidor poderá não dar resposta a todos de forma eficiente, dado que os terá de processar em simultâneo. Com estas desvantagens, os pontos positivos (simplicidade e completude) acabam por ser insuficientes para a escolha inequívoca desta abordagem.

A solução baseada em *cloud* combina as principais características de uma solução centralizada, ao mesmo tempo que preserva os grandes benefícios da distribuição. Simplicidade de acesso (para o utilizador, do ponto de vista deste o modelo é centralizado) e

fiabilidade mantêm-se, ao mesmo tempo que são mitigados os problemas de disponibilidade e durabilidade.

A durabilidade dos dados é um dos principais motivos que torna a *cloud* numa solução tão interessante. Sem a necessidade de aquisição de infra-estruturas, é possível ter os dados replicados em vários servidores, o que aumenta igualmente a disponibilidade do serviço, através de um eficiente balanceamento de carga. Alguns serviços possibilitam ainda uma replicação *offsite*, ou seja, em locais distintos do planeta. No contexto do sensoriamento participado, isto poderá traduzir-se num acréscimo de desempenho, desde que os dados estejam armazenados em servidores "próximos" do local da captura das amostras. É igualmente nesta zona em que existe maior probabilidade do consumo dos mesmos. O armazenamento, porém, tem um custo elevado, pelo que é imperativo fazer uma gestão dos dados presentes no sistema. O factor temporal apresenta-se, no caso geral, como o selector principal na política de remoção da informação, tornando prioritários (para manter no sistema) os dados mais recentes. Outra alternativa para a redução de custo seria deixar os dados na periferia, nos dispositivos dos utilizadores. No entanto, a falta de conectividade dos mesmos poderia resultar na inoperância do sistema.

Hoje em dia, podemos já encontrar os seguintes tipos de serviços de armazenamento na *cloud*: armazenamento em bruto (guarda ficheiros de dimensões e tipos variados como sequências de *bytes*), armazenamento em base de dados (relacional ou não-relacional, que consiste em guardar registos numa ou mais tabelas) e alojamento de aplicações (disponibiliza aplicações *online*, executando-as nestes servidores). Estes serviços poderão ser utilizados individualmente ou de forma combinada, permitindo soluções mais estruturadas e eficientes.

1.2.2 Indexação

Os dados provenientes de sensoriamento participado, estando estreitamente relacionados com as coordenadas geográficas aos quais dizem respeito e ao instante em que foram capturados, apresentam um forte sentido de localidade e temporalidade.

A maioria das aplicações existentes na área apresentam somente a faceta da localidade, sendo um factor-chave para a criação de vistas. Racionalmente, seria complicado criar uma vista de todo o globo em simultâneo, sendo necessária restringi-la a uma determinada zona. Esta zona pode ser definida por dois parâmetros: um centro e um raio. O centro indicará o ponto central da vista (normalmente a posição do utilizador no momento) e o raio a sua extensão. Esta configuração exprime a existência de um intervalo de valores de interesse e, assim, a necessidade de redução do âmbito na pesquisa dos dados (só interessam os pontos mais "próximos" do centro). Esta é a base da pesquisa por proximidade.

Em bases de dados de grande dimensão, é incomportável percorrer todos os registos, quando estamos interessados apenas numa pequena parte. Assim, torna-se indispensável a inserção dos dados de forma organizada, de modo a facilitar as operações futuras a

efectuar sobre estes. Definindo uma função, com base nas características dos dados, que permita a diferenciação (métrica) entre dois registos, poderá ser vantajoso indexar todos os dados de acordo com as distâncias entre si. Este tipo de organização permite reduzir substancialmente o número de cálculos necessários, tornando-se mais simples e rápido seleccionar os dados de interesse.

O índice deverá ser dinâmico e genérico. Por um lado, o dinamismo do índice permite a inserção e remoção de dados ao longo do tempo. Por outro, a sua faceta genérica além de permitir que vários tipos de dados possam ser inseridos, possibilita ainda a criação de funções de distância alternativas. Com a persistência dos dados, as características temporais passam a ter maior relevância na determinação dos valores pertencentes ao intervalo de interesse. Além destas, outras características (cor de um veículo, por exemplo) são passíveis de influenciar o âmbito das pesquisas, pelo que é necessário que a função de distância as comporte, atribuindo um maior ou menor peso consoante a sua importância.

Actualmente, a *cloud* não apresenta qualquer serviço que resolva este problema de forma simples e directa, pelo que será necessário recorrer a uma estrutura de indexação que seja capaz de o solucionar. Mais adiante, voltar-se-á a abordar este aspecto, apresentando uma proposta para a sua concretização.

1.2.3 Diversidade

Apesar do armazenamento em *cloud* comportar replicação como forma de tolerar falhas e oferecer uma elevada disponibilidade, é desejável que exista uma diversidade de provedores no armazenamento dos dados. Esta diversidade implica um custo total superior (no caso de existência de replicação), mas torna a solução mais flexível e contribui para evitar alguns problemas futuros.

A diversidade de provedores oferece maior independência, ao mesmo tempo que permite uma melhor gestão de custos. Se um determinado provedor aumentar o custo do serviço, este poderá ser abandonado facilmente, sem necessidade de migrar os dados aí contidos que, para além de dispendioso, poderá ser de difícil execução, consoante os contratos estabelecidos. Em casos extremos de falência do provedor, se não houver replicação dos dados noutras *clouds*, o risco é muito elevado, podendo originar o colapso do sistema.

A inclusão desta característica traz dificuldades acrescidas à implementação do sistema. De um modo geral, existem dois tipos de dados que exigem persistência: os dados em partilha e os dados relativos ao índice. Esta divergência verifica-se igualmente nos métodos utilizados para efectuar a sua persistência. Por um lado, temos os dados em partilha que consistem em registos que contêm os valores que o utilizador pretende partilhar. Estes não apresentam dependências, pelo que o processo de armazenamento se torna simples, sendo apenas necessária a atribuição de uma chave. Por outro lado, existem os dados pertencentes ao índice. Estes dados são manipulados constantemente, sempre que se dá a inserção ou remoção de elementos. Como apresentam um grande

número de dependências entre eles, é necessário adoptar políticas de forte consistência no armazenamento, de modo a garantir o bom funcionamento do índice. As políticas de consistência são relativamente simples de implementar quando a coordenação se efectua dentro da mesma *cloud*. Porém, e embora existam diversos protocolos *standard* que simplificam a comunicação entre serviços de diferentes provedores, efectuar tal coordenação com outras *clouds* é difícil de resolver, havendo o risco elevado de tornar o índice inconsistente.

1.3 Objectivos

Os objectivos da dissertação passam por estudar e avaliar a *cloud* como suporte para a persistência de dados adequado a aplicações de sensoriamento participado. Nesse sentido, este trabalho envolve a concepção, prototipagem e avaliação de uma solução de armazenamento de dados por recurso a uma estrutura de dados métrica, através da sua adaptação para operar sobre as facilidades de alojamento e processamento oferecidas pelos ambientes em *cloud*.

As características principais da solução a propor são:

- Pesquisa por proximidade;
- Modelo de distribuição heterogéneo que permita que os dados estejam replicados ou particionados em mais do que uma *cloud*.

Através da utilização da *cloud*, características complementares poderão ser obtidas, desde que esta se dê de forma adequada. Entre elas são de destacar:

- Grande capacidade de armazenamento de dados;
- Elevada disponibilidade;
- Fiabilidade.

A heterogeneidade ao nível do armazenamento, umas das grandes motivações para o trabalho, pode ser obtida através da reserva de áreas de armazenamento em *clouds* diferentes, disponibilizadas por diferentes provedores, replicando total ou parcialmente os dados por estas.

A eficiência do sistema não dependerá exclusivamente do uso da *cloud*, pelo que a escolha da arquitectura e das políticas de acesso e distribuição dos dados serão determinantes.

1.4 Contribuições previstas

Nesta dissertação estão previstas as seguintes contribuições:

- Arquitectura e modelo de persistência na *cloud* para dados com características métricas, como os provenientes de sensoriamento participado;
- Adaptação de um índice métrico sobre memória distribuída para tipos genéricos;
- Implementação de um protótipo, respectiva validação experimental e a sua disponibilização para futuros desenvolvimentos.

A arquitectura será definida pela escolha dos vários componentes do sistema e pelo modo como estes comunicam. O maior peso da arquitectura está presente na selecção dos tipos de serviço existentes na *cloud* a utilizar.

O desenvolvimento do índice métrico consistirá na adaptação para memória distribuída de uma estrutura de dados métrica, existente para funcionar em memória principal/secundária. Esta deverá ser genérica e dinâmica.

Já o protótipo a implementar deverá reflectir as assumpções feitas, de modo a que estas sejam testadas na prática, através do funcionamento de uma pequena aplicação estudo-de-caso.

1.5 Estrutura do documento

O documento é composto por um total de 6 capítulos. Neste primeiro capítulo é dada uma ideia geral do problema e da linhagem da solução que se pretende desenvolver. Aqui são discutidas as motivações para o tema da dissertação bem como a evolução histórica desta área. São igualmente abordadas as questões centrais do problema (armazenamento, pesquisa por proximidade e heterogeneidade), bem como estabelecidos objectivos e contribuições para o trabalho.

O capítulo 2 apresenta o trabalho relacionado. Este oferece de forma mais completa informações acerca do sensoriamento participado e exemplos de aplicações, de tipos e serviços de armazenamento na *cloud* e de estruturas de dados métricas (nomeadamente da RLC). É igualmente feito um ponto de análise ao estado da arte.

A parte directamente relacionada com a solução começa a ser descrita no capítulo 3 (Desenho e especificação). Neste capítulo são descritos a modelação dos dados e os métodos de armazenamento utilizados. Melhoramentos de *caching* e redundância, entre outros, são igualmente abordados em profundidade.

O capítulo 4 apresenta o protótipo desenvolvido nesta dissertação. Neste capítulo pretende-se demonstrar o que foi implementado na aplicação estudo-de-caso, assim como dar alguns exemplos da sua utilização. Limitações do protótipo e/ou das tecnologias utilizadas serão também discutidas, ao mesmo tempo que as suas respectivas consequências no trabalho final.

A validação experimental é feita no capítulo 5 . É apresentado um estudo comparativo entre várias versões do índice métrico, todas elas submetidas à execução de um *benchmark*

criado. A análise incide essencialmente sobre as métricas temporais de execução das operações, havendo ainda lugar a uma avaliação da ocupação espacial dos dados.

O 6º e último capítulo é dedicado a analisar as conclusões tiradas acerca do trabalho realizado. Algumas guias para trabalho futuro são igualmente apresentadas.

2

Trabalho relacionado

2.1 Sensoriamento Participado

O sensoriamento participado [3, 4, 5] é um paradigma emergente para aplicações móveis. Através deste paradigma, pretende-se que comuns dispositivos móveis colaborem de forma a criar uma grande rede sensorial, realizando tarefas de sensoriamento numa zona ampla, que surge como efeito lateral da rotina dos seres humanos e dos seus movimentos. A grande disponibilidade, que actualmente se verifica, de *smartphones*, *iPods*, etc., permite a criação de uma rede abrangente sobre a qual é possível fazer aplicações relevantes para a vida diária das pessoas, gerando vistas e interpretações detalhadas do mundo físico [10], sem custos associados, no que diz respeito ao desenvolvimento de uma densa e abrangente área de infra-estruturas. Assim, estas aplicações têm responsabilidades elevadas ao nível do sensoriamento, aprendizagem, partilha e visualização da informação.

A construção de uma boa representação exige um elevado o número de dados necessários, de preferência recolhidos por aparelhos diversos. Para além disto, a representação poderá ser somente momentânea, bem como geograficamente limitada, implicando que todos os dados recolhidos sejam marcados temporal e geograficamente.

Uma das grandes problemáticas existentes nesta área, tal como na generalidade dos sistemas distribuídos, é a escolha da melhor arquitectura. Se a opção recair por uma arquitectura centralizada, a gestão dos dados torna-se mais simples e confiável, não havendo necessidade de recorrer a *standards* ou outras entidades externas. Porém, esta traz diversos problemas: o repositório é único, exigindo encargos financeiros que desencorajam as iniciativas de uma comunidade. De forma oposta, uma solução distribuída permite uma maior escalabilidade, através das dispersões de carga e de armazenamento.

Nesta, não há necessidade da existência de um servidor de grande capacidade, mas exige-se que o serviço seja mais *community-based* e que os recursos sejam disponibilizados pela comunidade. Já de forma mais prática e justa, existem alguns sistemas baseados em redes *ad-hoc*. Porém, esta é uma opção com falhas graves, devido aos recursos limitados dos dispositivos móveis. Esta arquitectura exige um esforço muito elevado dos recursos participantes na rede, tanto ao nível da computação como da comunicação. Embora haja uma grande perda na descoberta de serviços e na disseminação de conteúdos, apresenta, no entanto, uma modularidade e independência, em relação a instituições, muito elevadas.

Alguns cuidados são igualmente necessários ao nível dos dados. Estes, para que a aplicação tenha uma boa credibilidade, deverão ser em número bastante elevado, todos eles estampilhados temporal e geograficamente, dadas as suas características. Em relação às estampilhas, a preocupação não é muito elevada, dado que os desacertos não serão demasiado grandes e não provocarão o mau funcionamento da aplicação. Porém, existem algumas preocupações noutros aspectos: o mau processamento dos dados e o efeito do observador. Ao nível do processamento, a captura de dados por sensores mal calibrados pode introduzir informação errónea no sistema. Aliás, o facto de toda uma comunidade poder produzi-los livremente faz com que a ocorrência destes se multiplique. Por outro lado, se os dados não se puderem tornar anónimos [4], de forma simples, muitos utilizadores serão desencorajados a participar na sua reunião. Este é também um dos desafios existentes, podendo a rede desenvolver serviços que ajudem os indivíduos a negociar as relações de privacidade.

2.1.1 Exemplos de aplicações

2.1.1.1 Pothole Patrol

O Pothole Patrol [6] é uma aplicação de sensoriamento participado que tem como objectivo detectar e reportar as condições do piso das estradas. Esta aplicação é baseada na existência de um conjunto de veículos com sensores que, aproveitando de forma oportunista a mobilidade inerente a estes, recolhem dados acerca da vibração e da posição, processando-os para inferir acerca das condições da via.

Esta aplicação é centralizada, porém igualmente colaborativa, utilizando sensores de aceleração em 3 eixos e dispositivos GPS embebidos num computador instalado nos veículos. Estes sensores deverão estar instalados em pontos estratégicos de modo a oferecerem resultados tão precisos quanto possível. Para o processamento desta aplicação, foi utilizada uma aproximação por *machine-learning*. Assim, foi feita uma recolha manual de dados para um *training set*, atribuindo as localizações a várias classes de anomalias na estrada. Para efeitos de computação, o *software* recolhe os valores das acelerações verticais (eixo-Z) e laterais (eixo-X), bem como a velocidade do veículo. Antes de enviar resultados para a central (através de ligações oportunas de Wi-Fi ou por rede móvel) são efectuados alguns cálculos para compactação e consolidação dos resultados. Para tal, são efectuados

agrupamentos e selecções, de modo a eliminar o maior número de erros, aumentando o grau de confiança de vibrações detectadas repetidamente, o que minimiza o número de falsos positivos. Neste campo, a aprendizagem da máquina torna-se indispensável, dado que a experiência passada ajuda a inferir se as acelerações verificadas se devem realmente a um buraco na estrada ou a um carril transposto, por exemplo.

A arquitectura presente nesta aplicação enfatiza em grande parte a mobilidade e a comunicação intermitente como sendo os problemas chave, promovendo um esforço local extra, para que a comunicação remota seja bastante mais reduzida. Esta é também a grande motivação deste artigo no âmbito do trabalho em curso, dado que apresenta um modelo simples, mas realista e eficaz, para o problema em questão.

2.1.1.2 CarTel

O CarTel [7] é um sistema de computação móvel designado para recolher, processar, enviar e visualizar dados provenientes de sensores localizados em unidades móveis, como veículos automóveis. Este sistema é essencialmente uma *framework* de programação e poderá ter aplicações múltiplas, nomeadamente a monitorização do tráfego rodoviário.

Este sistema está assente em três componentes básicos: o portal (localização central que hospeda os dados e as aplicações (*web-based*) e que funciona como ponto central de configuração); o ICEDB (processador de *queries* tolerante a atrasos) e a CafNet (pilha de rede tolerante a atrasos).

O funcionamento deste sistema não é trivial e implica alguns protocolos que o tornam bastante eficiente. Baseado no processamento de *queries*, apresenta duas vertentes: pedido de recolha de dados pela central e pedido de resultados pelo nó remoto. Como intermediário surge o ICEDB, que dá suporte a esta base de dados distribuída em que os nós, no terreno, colaboram para aumentar a quantidade de informação na rede. Ao contrário de outros sistemas, a entrega dos resultados não é feita automaticamente à aplicação receptora, mas sim à CafNet. Este componente tem a responsabilidade de monitorizar as condições de rede, informando ambos os lados dos momentos em que a comunicação é possível, fazendo uma bufferização dos dados quando esta não se pode realizar.

De modo geral, este modelo é bastante genérico, permitindo uma inúmera quantidade e variedade de aplicações sobrejacentes. Outro ponto forte está relacionado com a capacidade de tratamento de dados heterogéneos, através da instalação (de forma simples) de adaptadores nos nós remotos que permitem transformar os dados nos formatos *standard*, antes de os enviar para o portal.

Tendo em conta a dissertação, este sistema apresenta um valor acrescido na apreensão de ideias acerca da manutenção dos dados e no armazenamento destes numa entidade superior, com capacidade própria de computação e envio de resultados.

2.2 Armazenamento na *cloud*

O denominado *cloud storage* é um modelo de armazenamento na rede em que os dados são guardados em múltiplos servidores reais, normalmente propriedade de terceiros, em detrimento dos servidores dedicados. As companhias detentoras destes servidores operam em centros de dados de grande dimensão, podendo os utilizadores comprar ou alugar-lhes capacidade de armazenamento, para lá colocar a sua informação, de acordo com as necessidades. Podendo estar fisicamente em servidores distintos, os operadores de centros de dados virtualizam os recursos de acordo com os requisitos do utilizador, mostrando-os como servidores virtuais, podendo ser completamente geridos por este.

2.2.1 Armazenamento

Os serviços de armazenamento na *cloud* não funcionam todos do mesmo modo [8], sendo bastante heterogéneos no que diz respeito ao sistema que os sustenta bem como na sua própria abordagem. Existem três tipos de armazenamento na *cloud*:

- **Armazenamento em bruto:** guarda ficheiros como uma sequência de *bytes*, indexados por chave;
- **Armazenamento em bases de dados:** dados guardados sob a forma de registos em bases de dados relacionais ou não-relacionais;
- **Alojamento de aplicações:** permite colocar *online* aplicações completas.

Os grandes dominantes são os serviços de armazenamento de dados em bruto [11, 12, 13, 14, 15, 16]. Este tipo serviço é bastante comum e é colocado à disposição do utilizador por diversas companhias. Os ficheiros são guardados como longas sequências de *bytes* indexadas por chave numa tabela de dispersão. Alguns assemelham-se a um sistema de ficheiros convencional, tal como os disponíveis nos computadores pessoais, mas em que os mesmos são colocados em servidores remotos, mais propriamente nas *clouds* que o suportam. Estes serviços apresentam uma capacidade quase ilimitada, sendo somente restritivos no que diz respeito ao tamanho de cada ficheiro, suportando um máximo de 5 GB. A teoria em que se baseia este tipo de armazenamento é a mesma para todos os serviços que o suportam: é feita a transferência do ficheiro desejado para o servidor remoto e este é indexado por uma chave única, para que se conheça, de forma unívoca, de que ficheiro se trata.

Com uma incidência no mercado substancialmente menor encontramos o armazenamento em bases de dados. Este armazenamento pode ser baseado em dois modelos distintos: relacional [12, 17] e não-relacional [18].

As bases de dados relacionais são bastante comuns em qualquer sistema informático actualmente, sendo bons exemplos o MySQL, o DB2 da IBM e o SQL Server da Microsoft. Estas bases de dados permitem agrupar dados em tabelas que se relacionam entre si, e sobre as quais é possível fazer diversas operações intra e inter tabelas.

As bases de dados não-relacionais consistem num conjunto de simples tabelas que, ao invés do que sucede nas bases de dados relacionais, não podem ser ligadas entre si. Estas tabelas são denominadas por domínios e estes não são mais do que colecções de itens, descritos como pares atributo-valor. Sobre estes domínios, o número de operações é bastante mais reduzido do que o existente nas bases de dados relacionais, não havendo as de inter-domínio e sendo as de intra-domínio limitadas.

Existe ainda um tipo de armazenamento bem mais modular e compacto. Com uma abordagem substancialmente diferente, temos os serviços de alojamento de aplicações [19]. Este tipo de serviços permite colocar na *cloud* uma aplicação de ambiente *web-based*, bem como todos os dados envolvidos nesta.

Em [20] é apresentada uma listagem com alguns dos serviços actualmente disponíveis na *cloud*. Além da enumeração, estes são descritos com algum detalhe e são apresentados os custos inerentes à sua utilização.

2.2.2 Modelo de acessos

O tipo de comunicação mais presente na *web* actualmente é baseado em *standards*, tal como o XML, que permite de forma rápida e prática a interacção de diferentes aplicações. Assim, a grande maioria destes serviços apresenta sistemas de comunicação baseados em REST e SOAP, que funcionam sobre HTTP, aumentando a sua interoperabilidade com outras aplicações e serviços.

2.2.3 Segurança

A segurança é uma das grandes preocupações neste tipo de sistemas. O facto de se trabalhar muitas vezes com dados críticos e sigilosos implica a implementação de fortes políticas de acesso de forma proteger essa mesma informação. Algumas destas políticas incluem: geração de certificados SSL para transferência de dados, criação de ACLs (*Access Control Lists*) e autenticação de *queries*.

2.2.4 Disponibilidade e durabilidade

Recorrendo a um serviço desta natureza, não há necessidade de deter uma grande infraestrutura de servidores, nem tão pouco do espaço onde a alojar, porque os dados estão todos guardados em servidores remotos de outras companhias. As empresas que os fornecem são normalmente bastante credenciadas, tendo acesso a grandes canais de comunicação, o que diminui a probabilidade de eventuais quebras de acesso ao serviço. Segundo a maioria das empresas provedoras dos serviços na *cloud*, a disponibilidade garantida é sempre acima dos 99%.

A durabilidade dos dados está praticamente assegurada, através da replicação feita internamente. Consoante o serviço, assim será efectuada a cópia dos dados. Alguns serviços possibilitam a chamada replicação *offsite*, ou seja, a replicação num local geográfico distinto, evitando a perda de dados em caso de catástrofe, por exemplo. No entanto, este

será um serviço mais dispendioso, não fazendo normalmente parte das ofertas base, nas quais os preços são bastante competitivos.

No caso dos serviços de bases de dados é também possível criar *backups* automáticos e os denominados *database snapshots*, à semelhança do que acontece nos serviços de bases de dados locais. Algumas companhias, como por exemplo a Amazon [17, 18], permitem ainda a criação de réplicas de leitura. Estas são bastante úteis, nomeadamente quando o número de leituras na base de dados é bastante superior ao número de escritas, diminuindo a sobrecarga de cada servidor individualmente, sem grande custo no desempenho geral.

2.2.5 Custos

Nesta secção serão apresentados os custos para os vários tipos de serviço, de forma a obter uma noção dos valores envolvidos na utilização destes.

Comum à generalidade do serviços, é exigido o pagamento do armazenamento, do tráfego e do número de operações (este último não se verifica nos serviços de alojamento de aplicações). Já o pagamento do tempo de utilização do CPU não é comum a todos eles. Esta tarifa é aplicada às bases de dados e aos serviços de alojamento de aplicações mas não é cobrada no armazenamento em bruto.

As várias vertentes de cobrança estão dispostas, de forma resumida, na Tabela 2.1.

Tabela 2.1: Custos gerais do armazenamento na *cloud*

	Armazenamento	Tráfego	Pedidos	Tempo CPU	Email
Armazenamento em bruto	✔	✔	✔	✘	✘
Bases de dados	✔	✔	✔	✔	✘
Alojamento de aplicações	✔	✔	✘	✔	✔

2.2.5.1 Armazenamento em bruto

Para o armazenamento em bruto, os preços apresentados em seguida representam valores médios, dado o elevado número de alternativas existentes. Ainda assim, como poderá ser visível através da consulta das tarifas de cada serviço [20], estas não variam muito das médias apresentadas, devido à necessidade de praticar valores semelhantes para manter a competitividade.

- Armazenamento (mensal): \$0.15/GB
- Transferência de dados (*upload* e *download*): \$0.15/GB
- Transacções: \$0.01/10000 pedidos de GET e \$0.01/1000 pedidos de PUT

2.2.5.2 Base de dados relacionais

As bases de dados relacionais na *cloud* apresentam tarifas bastante distintas nos diferentes provedores. Muitos apresentam um custo mensal fixo, através da reserva de uma determinada capacidade e tempo de funcionamento, ao passo que outros são pagos de acordo com a utilização efectuada. Porém, para exemplo, será utilizada, aqui e na secção 2.2.5.3 a tarifa que a Amazon aplica para o RDS [17]. Embora possua também a possibilidade de assinatura de custo fixo, esta modalidade irá apresentar-nos os valores por excesso, permitindo-nos, no entanto, uma maior liberdade a nível de serviço.

- Armazenamento (mensal): $0.11/GB
- Transferência de dados (*upload* e *download*): $0.15/GB
- Transacções: $0.11/milhão de pedidos
- Tempo de funcionamento da base de dados: Tabela 2.2

Tabela 2.2: Custo do tempo de CPU

Capacidades	Funcionamento (por hora)
***Small* DB**	$0.12
***Large* DB**	$0.48
***Extra Large* DB**	$0.97

De referir que estas classes não estão directamente ligadas ao tamanho da base de dados mas, para uma base de dados de tamanho elevado, o poder computacional da máquina terá de ser muito superior do que para uma base de dados de tamanho reduzido.

2.2.5.3 Base de dados não-relacionais

Neste tipo de serviço é utilizado como referência o que é oferecido pela Amazon, no caso, o SimpleDB [18].

- Armazenamento (mensal): $0.275/GB
- Transferência de dados (*upload* e *download*): $0.15/GB
- Transacções: $0.11/milhão de pedidos
- Tempo de CPU: $0.154/hora

2.2.5.4 Alojamento de aplicações

Este serviço permite o alojamento de aplicações na *cloud*. Como referência será utilizada a tarifa da Google para o App Engine [19], visível na Tabela 2.3.

Tabela 2.3: Custo do serviço de alojamento de aplicações

Recurso	Valor
Download	$0.12/GB
Upload	$0.10/GB
Tempo de CPU	$0.10/hora
Armazenamento	$0.15/GB mês
Email	$0.0001/destinatário

2.2.6 Discussão

A recolha de informação efectuada acerca dos vários serviços existentes na *cloud*, e a respectiva criação de cenários de utilização [20], permitiram tirar algumas conclusões.

Qualquer solução que envolva o armazenamento na *cloud* é bastante dispendiosa. Para além dos custos de armazenamento, há que contabilizar o número de pedidos, a quantidade de transferências e, em alguns casos, o tempo de funcionamento do CPU. Aliás, segundo os cenários criados, com o crescimento da utilização do sistema e respectivo aumento do número de operações de pesquisa efectuadas sobre este, o valor a pagar pelo número de transferências apresenta-se como factor dominante, passando o custo de armazenamento a representar uma pequena fracção. Igualmente para o cenário referido, as transferências são sobretudo de descarga, sendo preferível optar por serviços em que o custo destas seja mais reduzido.

No caso das bases de dados há outros valores envolvidos: os referentes ao tempo de funcionamento do CPU. Se a base de dados estiver em funcionamento permanentemente, o custo é de $86.40, mensalmente, segundo as tarifas praticadas pela Amazon [17, 18]. Este valor, porém, pressupõe o funcionamento do serviço num servidor de baixa capacidade de processamento. Para a aumentar, o custo terá igualmente um valor superior, podendo traduzir-se numa elevada percentagem do pagamento total mensal a fazer pelo sistema.

De referir ainda a existência de um serviço de alojamento de aplicações. Este apresenta uma modularidade bastante elevada, permitindo oferecer serviços na *cloud* que esta ainda não possua. É dotado de suporte a várias bibliotecas, o que aumenta o seu âmbito de utilização e o torna numa solução bastante interessante para certas aplicações, compensando os vários custos adjacentes envolvidos (funcionamento, transferência de dados, etc.).

2.2.7 Sistemas de armazenamento

A escolha de uma arquitectura apropriada é indispensável para a implementação de um sistema informático, sendo fundamental para a definição de conceitos como a disponibilidade, a fiabilidade e a escalabilidade, entre outros. É em torno de alguns destes conceitos que será feito um estudo comparativo de vários sistemas que suportam os serviços da

cloud, descritos anteriormente.

Habitualmente, podemos dividir as arquitecturas em dois grandes grupos: centralizadas e não-centralizadas (ou distribuídas). Cada um destes grupos possui características muito particulares e nenhum deles apresenta a solução óptima. Consoante o âmbito da aplicação, assim será melhor optar por uma solução completamente centralizada ou por algo totalmente distribuído. No entanto, soluções híbridas não são de excluir, sendo que a *cloud* é um exemplo disso.

Existem diversos tipos de armazenamento na *cloud*: armazenamento em bruto, em base de dados não-relacional, etc. Estes serviços de armazenamento são suportados por um sistema de ficheiros local aos servidores, que organiza os dados, e incorpora um sistema de gestão que conhece o modo como estes estão dispostos. Porém, também o modo como são implementados permite torná-los mais ou menos interessantes. Do estudo efectuado, foram identificados dois grandes tipos de sistemas de armazenamento: sistemas de ficheiros hierárquicos e não-hierarquicos (vulgo, bases de dados não-relacionais, ainda que possam ser usadas para armazenamento em bruto). No campo dos sistemas de ficheiros hierárquicos podemos encontrar o GFS (Google File System [21]) que permite a utilização de *paths* tal como nos sistemas operativos locais, oferecendo uma interface bastante familiar aos utilizadores. Já as bases de dados não-relacionais permitem a criação de tabelas, com múltiplas colunas e linhas, sendo que cada linha é identificada por uma chave, que a torna única no sistema. Alguns destes sistemas utilizam tais tabelas como índice para o armazenamento de ficheiros binários, permitindo, por vezes, a ligação entre vários sistemas. Este é o caso do Bigtable [22]. Apresentando-se como uma base de dados não-relacional, permite, no entanto, que os ficheiros binários inseridos sejam guardados de forma hierárquica, utilizando o GFS para os armazenar. Já sistemas como o Dynamo [23] ou o Cassandra [24] organizam-nos de modo semelhante aos registos em tabelas, oferecendo menor versatilidade.

Guardar dados na *cloud* é dispendioso, exigindo bastante ponderação antes de se optar por este tipo de armazenamento. As características oferecidas por esta permitem um nível de desempenho quase único, que a torna bastante atractiva. Um desses aspectos é a disponibilidade, que pode ser obtida através do balanceamento de carga, por exemplo. Qualquer serviço na *cloud* coloca foco nesta questão e é também por causa dela que se pode considerar que estes sistemas possuem uma arquitectura híbrida. A forma de acesso a eles é feita com um estilo centralizado. Porém, estes portos virtualizam uma densa rede de servidores individuais, que estão ligados e partilham informação, guardando internamente várias réplicas dos mesmos dados. Logicamente, a distribuição dos servidores poderá estar mais ou menos organizada. No caso dos sistemas Cassandra e Dynamo, internamente, os nós estão ligados em anel, à imagem das redes sobrepostas estruturadas. Cada nó tem um identificador único e apresenta ligações com alguns dos seus vizinhos, uns mais próximos, outros mais distantes. Já o aspecto físico é bem menos claro, podendo os vários servidores estarem colocados em diversas regiões do globo, o que aumenta a durabilidade dos dados.

A organização em anel é, em muitos aspectos, a forma mais simples de gerir a colocação das réplicas. O conjunto de chaves dos dados é dividido em vários intervalos. Cada um destes servidores é responsável por armazenar um dado intervalo, estando o intervalo de valores deste intimamente relacionado com o identificador do nó. A escalabilidade dinâmica é também bastante simples com este modelo. Um nó, ao entrar no anel, ficará responsável por armazenar parte dos dados que os vizinhos contêm, libertando a carga dos mesmos.

Já o Bigtable e o GFS têm uma organização completamente diferente. Existe um servidor central (*master*) que gere a indexação dos dados nos vários servidores secundários. No entanto, o sistema minimiza alguns dos potenciais problemas de ter um ponto central de falha ao permitir que, após a descoberta, o utilizador comunique directamente com o servidor que detém realmente os dados. O *master* não precisa de armazenar o conhecimento que tem sobre os outros servidores. A informação acerca do estado dos *slaves* é passada através das respostas às mensagens de *heartbeat* que são enviadas periodicamente pelo *master*.

Outro dos factores interessantes e diferenciadores é o modo como os dados são armazenados fisicamente. O caso do Cassandra é o mais simples. Neste sistema, os ficheiros (com os registos da base de dados) são armazenados em disco de modo sequencial. Esta tarefa é sobejamente simplificada devido à assunção feita pelo sistema, de que os dados podem ser adicionados e removidos, mas nunca modificados. No entanto, esta simplicidade é inversamente proporcional à extensibilidade do sistema. Tendo sido desenvolvido para suportar os dados da rede social Facebook, vê-se limitado a aplicações menos mutáveis, sendo porém altamente eficiente para estas, dado que raramente necessita de *locks* e não lida com problemas de concorrência.

Os sistemas que permitem alterações de dados são muito mais complexos, nomeadamente para resolver a questão da consistência das réplicas. Muitos sistemas optam pela solução mais simples: submetida uma escrita ao sistema, esta é feita de forma síncrona nas várias réplicas, sendo a operação rejeitada caso não possa ser efectuada de forma completa. Porém, para além do número elevado de rejeições de operações, provoca um baixo desempenho ao nível do processamento de pedidos, dado que as várias réplicas permanecem bloqueadas por longos períodos. Já o Dynamo opta por uma solução totalmente assíncrona e que permite a existência de múltiplas versões dos mesmos dados. Esta divergência de versões ocorre quando há dificuldades na obtenção de *quórum*, cabendo à aplicação definir a política para a escolha da versão que permanece. Esta solução apresenta-se benéfica para ambos os lados, dado que permite ao sistema adiar e relegar para terceiros as decisões de resolução de conflitos, ao mesmo tempo que oferece às aplicações superiores a possibilidade de preferência sobre que dados manter. Já o GFS resolve por si mesmo os conflitos, quando vários utilizadores vêem versões diferentes ao mesmo tempo, recorrendo a *checksums* e números de versão, por exemplo.

2.2.8 Exemplos de repositórios alojados na *cloud*

2.2.8.1 Panoramio

O Panoramio [25] é um serviço fornecido pela Google que permite o armazenamento de fotografias de forma geo-referenciada. Este serviço inclui os dados da empresa que são utilizados nas suas aplicações, como é o caso do Google Earth. Do ponto de vista desta dissertação, seria interessante poder fazer uso destes mesmos dados e incluí-los na aplicação a desenvolver.

O Panoramio possui uma API pública que permite a utilização dos seus dados por terceiros, nomeadamente a sua visualização em *websites* externos. Esta API é baseada em *widgets*, o que, neste âmbito, não parece ser tão vantajoso. Para o utilizador vulgar, esta solução é bastante conveniente, permitindo de forma simples e agradável a visualização das imagens desejadas. No entanto aqui, o objectivo seria ter controlo total na obtenção destas, de forma a dar-lhes utilização de acordo com o desejado.

Os pedidos ao Panoramio são bastante configuráveis, aceitando diversos parâmetros no URL, para que o objecto de retorno tenha as características desejadas. Esta API é uma biblioteca de JavaScript que fornece elementos gráficos e capacidades de pesquisa, permitindo a visualização dos *widgets* num *website* externo. Para além desta hipótese, é também possível obter a resposta sob a forma de *templates* HTML, que dispensam qualquer programação em JavaScript.

2.2.8.2 Flickr

O Flickr [26] é uma comunidade de partilha de fotografias que possibilita, de uma forma fácil, colocar e partilhá-las *online*, permitindo ainda adicionar-lhes metadados e comentários. Tal como no Panoramio, é possível a geo-referenciação destas.

O Flickr possui uma interface de programação aberta aos seus utilizadores. Qualquer pessoa poderá criar a sua própria aplicação para oferecer, ao público, dados do Flickr, do modo mais conveniente. Para criar um aplicativo, o programador terá de ter uma chave de acesso ao sistema. Os pedidos poderão ser realizados de uma das três seguintes formas: REST, XML-RPC ou SOAP. As respostas, para além destes protocolos, poderão ser obtidas por JSON e PHP.

A API apresenta-se bastante detalhada e bem organizada, permitindo uma fácil aprendizagem das várias possibilidades que esta oferece. Cada chamada, apresenta pelo menos um exemplo de resposta e os vários erros possíveis de ocorrer, bem como a designação de cada um deles. Ao contrário do Panoramio, a interface de programação do Flickr é bastante mais flexível, podendo ser alvo de interesse para a dissertação em curso.

2.3 Estruturas de dados métricas

As estruturas de dados métricas têm como um dos principais objectivos oferecer pesquisas eficientes. De entre estas destacam-se a pesquisa por proximidade, a pesquisa exacta

e a pesquisa dos k vizinhos mais próximos. Aquando do momento de inserção dos dados, estes são dispersos por várias zonas, de acordo com o grau de semelhança que têm com determinados elementos-chave. Estes elementos-chave, consoante a estrutura em causa, denominam-se centros de agrupamento ou *pivots*. De entre as estruturas de dados métricas baseadas em *pivots* temos a VPTree [27] (Vantage Point Tree) e a LAESA [28] (Linear Approximating Eliminating Search Algorithm). Já exemplos de estruturas baseadas em agrupamentos são a RLC [29] (Recursive Lists of Clusters) e a GNAT [30] (Geometric Near-neighbor Access Tree)

A semelhança entre dois elementos é dada através de uma função de distância, que é baseada nas suas propriedades. A função de distância (ou métrica) é o ponto central para a organização destas estruturas e terá de apresentar quatro características fundamentais: (Representando $d(x, y)$ a função de distância entre os objectos x e y)

- Não-negatividade: $d(x, y) \geq 0$ (a distância entre dois objectos tem sempre valor maior ou igual a 0);
- Identidade: $d(x, y) = 0 \iff x = y$ (se a distância entre dois objectos for igual a 0, então estes são iguais);
- Simetria: $d(x, y) = d(y, x)$ (a distância entre os mesmos dois objectos é igual, independentemente da ordem em que a função é aplicada sobre estes);
- Desigualdade triangular: $d(x, y) \leq d(x, z) + d(z, y)$ (a soma das distâncias de dois objectos a um terceiro é maior ou igual que a distância entre eles).

Este tipo de estruturas tem como objectivo minimizar o número de distâncias calculadas entre os vários elementos da base de dados, nas pesquisas por proximidade ou semelhança. As pesquisas por proximidade consistem em encontrar todos os elementos cuja distância a um elemento *X* não seja superior a um raio *r* (*X* e *r* são parâmetros da pesquisa), ou seja, os mais semelhantes a si mesmo, sendo o nível de semelhança definido pelo raio. Com a distribuição por semelhança, não há necessidade de expandir a pesquisa a todas as zonas, dado que é garantido que somente em algumas poderão ser encontrados dados de interesse.

2.3.1 RLC

A RLC [29] (Recursive Lists of Clusters) é uma estrutura de dados métrica, genérica e dinâmica. Permite armazenar qualquer tipo de objecto e a inserção e remoção dos dados pode ocorrer quando desejável, mesmo após o carregamento inicial. Nestes aspectos, a RLC apresenta vantagens em relação a outras estruturas de dados métricas que são mais limitativas (estrutura fixa e tipo restrito de dados armazenados).

A RLC é organizada em listas de agrupamentos. Cada agrupamento poderá conter outras listas de agrupamentos, apresentando uma estrutura hierárquica. Isto determina

que cada um deles possa pertencer a vários agrupamentos (de profundidade sucessivamente mais reduzida). Cada agrupamento (em inglês, *cluster*) possui as seguintes propriedades:

- Centro (c);
- Raio (r);
- Interior (I) (conjunto de objectos da base de dados contidos em]0,r]).

Todos os agrupamentos possuem o mesmo raio, tal como é visível na Figura 2.1, retirada de [1].

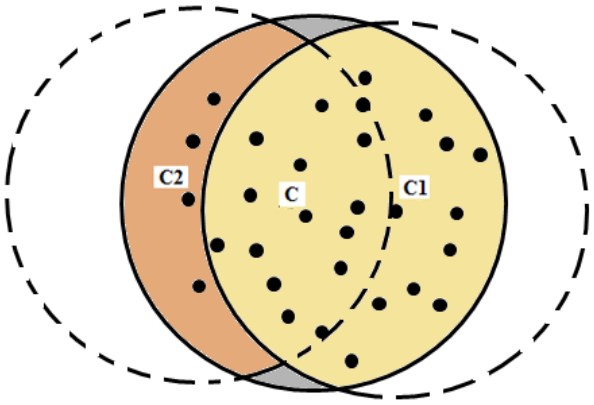


Figura 2.1: Interior de um agrupamento de centro c com raio constante [1].

Esta organização é obtida tendo por base a distância entre os vários objectos ou, mais concretamente, entre os objectos a inserir e os objectos de referência dos vários agrupamentos (centros), sendo para isso utilizada uma função de distância que tem em conta as características de cada um. De forma a facilitar a organização da RLC e as suas operações, para o mesmo nível (de profundidade), um objecto só poderá estar contido dentro de um agrupamento, sendo inserido no primeiro da lista que o pode conter. Deste modo, se dois agrupamentos ao mesmo nível se intersectarem, os objectos pertencentes a essa área encontrar-se-ão no primeiro agrupamento destes a ser seleccionado na iteração da lista de agrupamentos.

O interior de um agrupamento pode ser de dois tipos: uma lista de agrupamentos ou uma folha. Uma lista de agrupamentos consiste num nó interior, que acrescenta novos níveis de indirecção e dispersão dos dados. Já uma folha é um nó terminal, onde são armazenados os vários objectos pertencentes ao interior desse mesmo agrupamento (não é considerado que o centro está contido no interior do agrupamento, ficando indexado no próprio agrupamento e não na folha). As folhas são vectores com uma determinada capacidade-limite, após a qual se efectua uma redistribuição dos dados, dando origem a uma nova RLC. Neste caso, os elementos da antiga folha são reinseridos na nova RLC,

dando lugar a uma nova distribuição por agrupamentos e respectivas folhas, a serem criadas. Estes vectores têm a particular característica de nunca terem posições vazias entre os elementos. O raio de todos os agrupamentos e o número máximo de objectos nas folhas são os dois parâmetros de construção desta estrutura.

Baseado neste esquema, cada objecto apresenta uma lista de distâncias aos centros dos vários agrupamentos em que está contido. Esta lista é acedida em primeiro lugar sempre que se pretende inquirir se um dado objecto pertencerá ao conjunto-resposta, evitando um número excessivo de cálculos de distâncias.

Os tópicos seguintes apresentam as três operações possíveis sobre a RLC: inserção, remoção e pesquisa.

- **Inserção:** Para inserir é necessário iterar a sequência de agrupamentos até encontrar um em que o objecto possa ser colocado.
  - Se não for encontrado nenhum agrupamento é criado um novo, que é adicionado à cauda da RLC;
  - Caso seja encontrado, ocorrerá um dos dois seguintes cenários: se o agrupamento for uma lista de agrupamentos, insere-se recursivamente neste; em situação contrária poderá ser inserido nesse vector, se houver espaço. Se não houver espaço, uma nova RLC é criada e tanto o novo objecto como os elementos dessa folha são inseridos neste novo nó.
- **Remoção:** A remoção funciona de forma similar: é procurada na árvore de agrupamentos o objecto que se pretende remover. Caso o objecto seja centro de um agrupamento, este é eliminado e o seu interior é reinserido na RLC que o continha. Se o objecto estiver contido numa folha, este é removido, ocorrendo ainda a compactação da RLC, que contém o agrupamento da primeira, numa nova folha, caso o número total de elementos seja igual à capacidade máxima das folhas.
- **Pesquisa:** A pesquisa por uma zona de proximidade decorre da seguinte forma:
  - Se a região em interrogação estiver contida no interior do agrupamento, a pesquisa continua no interior deste, parando a iteração;
  - Se a região em interrogação engloba a do agrupamento, os objectos do interior são adicionados ao conjunto-resposta, continuando a iteração;
  - Se as regiões se intersectam (e nenhuma está contida noutra) a pesquisa continua pelo interior do agrupamento, retomando mais tarde a iteração;
  - Se as regiões são disjuntas, o interior do agrupamento é ignorado e a iteração continua.

A análise das operações da RLC com n elementos, para $n \geq 0$, permitiu concluir que, em média, a operação de inserção efectua $O(\log n)$ computações de distância, a operação

de remoção $O(\log^2 n)$ e as operações de pesquisa necessitam de $O(n^\gamma)$ para qualquer $\gamma \in [0, 1]$ [31].

Para esta dissertação, a RLC é particularmente útil para resolver o problema da pesquisa por proximidade, permitindo suportar diferentes tipos de objectos como texto e documentos de som [32, 33], dependendo do âmbito da aplicação. Num contexto de sensoriamento participado, as pesquisas são baseadas essencialmente nas coordenadas geográficas e estas obrigam a uma certa margem de erro, de modo a obter resultados das áreas circundantes mais próximas.

2.3.1.1 RLC em memória secundária

Além da implementação em memória principal, a RLC possui igualmente algumas versões em memória secundária [34, 35]. Essencial é o seu caracter dinâmico, factor que torna interessante a sua persistência, dado ser possível adicionar novos elementos quando necessário.

A versão implementada em [34] apresenta a informação guardada em páginas de tamanho fixo, podendo estas ser de 3 diferentes tipos:

- Cabeçalho: contém a informação global da estrutura e do tipo de objectos;
- Páginas de agrupamentos: contêm as listas de agrupamentos sob a forma de listas ligadas de vectores de agrupamentos. Para cada agrupamento são guardados: centro, lista de distâncias do centro aos centros dos agrupamentos em que está contido, número de elementos no interior e apontador para a (primeira) página com o interior do agrupamento;
- Páginas de folhas: contêm as folhas sob a forma de listas ligadas de vectores de pares <elemento, lista de distâncias>.

Ao contrário de outras estruturas, a RLC apresenta-se igualmente bastante eficiente em memória secundária, sendo eficaz em pesquisas por proximidade e competitiva nas inserções. Em [34], para a obtenção de um melhor desempenho, o raio dos agrupamentos diminui com a profundidade, visível na Figura 2.2, retirada de [1], ao contrário de outras em que este é constante. Esta alteração prende-se com o facto de, aquando da transformação de uma folha numa RLC, o número de agrupamentos com necessidade de serem criados ser reduzido e, assim, haver uma diminuta dispersão dos dados, rapidamente originando novas expansões. Deste modo, a nova RLC torna-se mais larga e menos profunda, o que reduz o nível de recursividade e implica uma menor leitura de novas páginas.

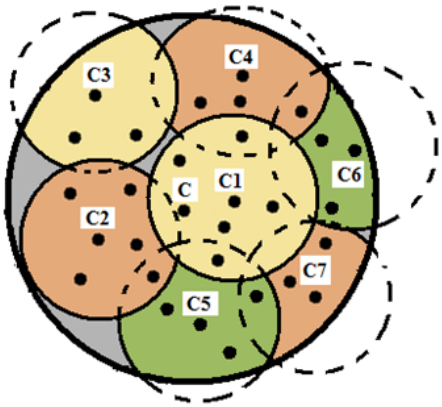


Figura 2.2: Interior de um agrupamento de centro *c* em que o raio diminui com a profundidade [1].

2.4 Conclusão

A pesquisa acerca de alguns dos trabalhos na área do sensoriamento participado permitiu concluir que existe uma elevada necessidade de persistência dos dados. Uma vez recolhidos, o valor da sua informação não se esgota nesse exacto instante, pelo que a sua consulta posterior se torna interessante e, por vezes, essencial para um elevado número de aplicações [6, 7]. A persistência destes dados é complexa, especialmente devido aos seus padrões de acesso muito particulares. Tipicamente, não estamos interessados num valor concreto, mas sim numa gama de valores presentes numa determinada área, em redor de um ponto de referência. Este ponto está frequentemente associado ao local em que utilizador se encontra nesse momento, pelo que é possível inferir que o consumo dos dados está relacionado com o seu local de captura. Exemplo disso é o Pothole Patrol [6], em que os utilizadores interessados em conhecer as condições do piso das estradas são aqueles que as frequentam ou irão frequentar, estando muito provavelmente próximos das mesmas. Deste modo, seria ideal guardar os dados perto dos locais de captura, diminuindo a latência das operações a efectuar sobre o sistema.

A lógica do armazenamento dos dados poderá ter um elevado impacto na capacidade de resposta do sistema. A indexação dos dados numa estrutura de dados métrica, seria igualmente benéfica para um sistema com estas características. Este tipo de estruturas tem como objectivo diminuir o número de cálculos nas operações de pesquisa, o que diminui o tempo de resposta para a obtenção de resultados. Os dados encontram-se "separados" por zonas de proximidade, pelo que somente em algumas (as únicas exploradas) se poderão encontrar registos que possam fazer parte do conjunto-resposta.

Uma solução interessante para o armazenamento dos dados parece ser a utilização da *cloud*. Actualmente existem vários serviços, de diversos provedores e tipos. De entre os principais, poderemos enunciar os serviços de armazenamento de dados em bruto [11, 13, 14], armazenamento em base de dados [17, 18] e alojamento de aplicações [19].

Consoante o tipo de serviço requerido, assim serão os custos envolvidos.

Menos dispendiosos são os serviços de armazenamento de dados em bruto. Estes apresentam, habitualmente, um armazenamento do tipo chave-valor (à imagem das tabelas de dispersão), que é bastante útil, mas, simultaneamente, muito simplista e limitado. Por não lidar com blocos e apontadores de ficheiro, não permite alterações, somente inserções e remoções. A sua implementação é simples, assim como o custo reduzido, na medida em que este resulta essencialmente das taxas de capacidade ocupada e da largura de banda utilizada. Semânticas de escrita utilizando conjuntos de chaves pode oferecer-lhes consistência, o que é essencial para muitos tipos de aplicações.

Outro dos tipos de serviço oferecido é o armazenamento em base de dados. Esta pode ser relacional ou não relacional, sendo que ambas são bastante distintas. Os serviços de base de dados relacional apresentam características semelhantes às que existem fora da *cloud*. Permitem realizar operações intra e intertabelas e possuem um elevado número de funcionalidades e configurações. Por outro lado temos as bases de dados não-relacionais. Estas apresentam-se bastante mais simples que as anteriores. Os registos estão guardados em domínios e só é possível realizar operações internamente. Em ambos os casos, o tempo de CPU despendido é taxado, assim como a capacidade e largura de banda utilizadas, mensalmente.

Por último temos os serviços de alojamento de aplicações. Estes permitem colocar aplicações *online*, tendo normalmente associada uma base de dados não-relacional, para registar dados de suporte. Este tipo de serviço permite desenhar soluções complexas, apresentando, no entanto, custos elevados, em tudo semelhantes aos do caso anterior.

Os serviços existentes, além de permitirem a sua utilização individual, permitem uma utilização conjunta, de forma coordenada. O modo como os serviços são combinados influencia de forma determinante o custo final do sistema. Uma estratégia hábil poderá passar pelo armazenamento dos dados de maiores dimensões e menos mutáveis em serviços de armazenamento em bruto, por ser mais económico. Registos mais complexos e de suporte ao sistema deverão ficar nos serviços de bases de dados. Caso seja utilizado um serviço de alojamento de aplicações, a base de dados não-relacional associada será a preferencialmente utilizada, por se encontrar próxima do mesmo, onde o seu acesso é mais rápido e não influencia negativamente a latência das operações.

Aplicações como o Flickr [26], são exemplos de repositórios de enormes dimensões, cujo conteúdo é contribuído por uma grande comunidade de utilizadores espalhados pelo planeta. Estes poderão ser explorados, utilizando algumas das técnicas de *crowdsourcing*, na base de aplicações de sensoriamento participado. Nesse sentido, a indexação e utilização externa destes dados poderá alimentar novas aplicações.

3

Desenho e especificação

No capítulo anterior foram discutidos alguns temas de particular relevância para esta dissertação. Neste, ir-se-á descrever o modelo desenhado para o sistema a implementar, bem como a sua especificação e modo de funcionamento, tendo por base os estudos realizados. Assim, e tal como visto anteriormente, a RLC desempenhará um papel fundamental na solução a apresentar, sendo a estrutura responsável por indexar os dados e, consequentemente, organizá-los de modo eficiente. As operações a efectuar no contexto do sistema incidirão em operações atómicas sobre a RLC (de inserção e pesquisa, por exemplo), bem como outras operações de suporte, descritas mais adiante.

3.1 Arquitectura do sistema

O sistema a desenvolver nesta dissertação tem como motivação o armazenamento de dados provenientes de sensoriamento participado ou, mais genericamente, de grandes conjuntos de dados passíveis de serem indexados metricamente. Como foi descrito anteriormente, a arquitectura a propor é baseada na *cloud*, que apresenta características híbridas, entre as arquitecturas centralizada e distribuída.

A Figura 3.1 esquematiza a solução proposta para o problema. Como é observável, podem ser definidas três grandes áreas. Mais abaixo na figura encontramos os utilizadores do sistema. Ao centro estão a interface e lógica do sistema, representadas pelo *front-end*, e o índice, ambos presentes na *cloud* (no provedor Z). No topo, são observáveis as *clouds* dos provedores A, B e C que armazenam os dados e oferecem a diversidade e independência desejadas.

Os utilizadores, através dos seus dispositivos móveis que se encontram ligados à rede (por ligações oportunistas de Wi-Fi, por exemplo), comunicam com o sistema, efectuando

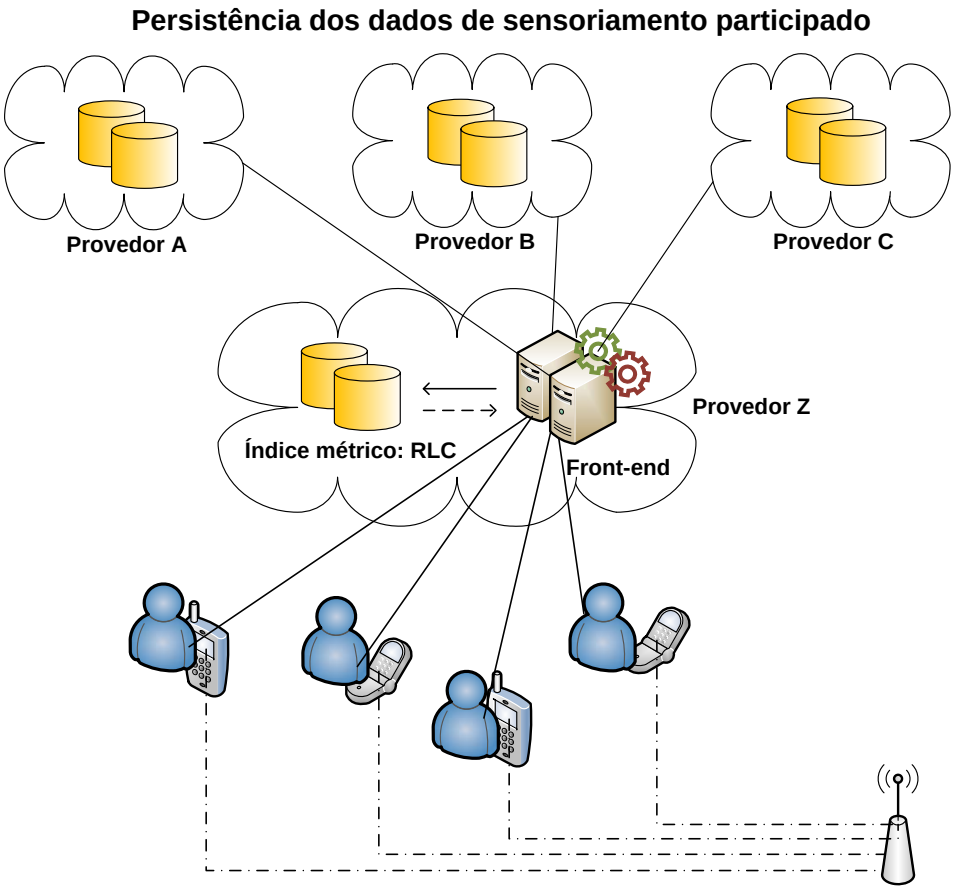


Figura 3.1: Modelo da arquitectura do sistema orientado para sensoriamento participado

pedidos de inserção de amostras e pesquisas, por exemplo. Os pedidos são direccionados à *cloud* do provedor Z, onde se encontra o *front-end* do sistema. O *front-end* é a parte responsável pela comunicação com os outros componentes (índice e provedores externos) e que implementa a interface com o utilizador. Esta interface permite que o utilizador observe o sistema como um serviço centralizado, permanecendo transparente a sua natureza distribuída.

O *front-end*, além de responsável por toda a comunicação realizada, é ainda o componente que efectua a lógica do sistema. Para tal, na *cloud* do provedor Z, deverá estar disponível um serviço de alojamento de aplicações. Este tipo de serviço permite a execução de uma aplicação que, neste caso, será responsável pelo processamento de pedidos e coordenação das operações de leitura e escrita, por exemplo. As últimas operações poderão ter origem interna ou externa, consoante os dados em causa.

Internamente, os serviços de alojamento de aplicações poderão estar acompanhados por alguns serviços de armazenamento (com tabelas chave-valor). Neste caso, é pretendido o armazenamento num estilo de base de dados não-relacional, como é exemplo o Bigtable [22], para armazenar os dados relacionados com o índice métrico. Este serviço permite guardar de modo compacto toda a informação relativa ao índice e à sua gestão,

com uma latência reduzida, o que permite respostas mais rápidas.

A diversidade é aplicada ao nível dos dados em partilha. Através de interfaces REST, por exemplo, o *front-end* comunica com as *clouds* de outros provedores, replicando as amostras inseridas pelo utilizador. Com este modelo, os dados encontram-se deslocalizados do índice, o que oferece um nível superior de flexibilidade. Assim, os mesmos dados poderão ser utilizados em diferentes contextos, com diversos tipos de índices (neste caso foi estudada a RLC) e parametrizações. Através desta separação, é igualmente possível escolher o tipo de serviço consoante os requisitos de armazenamento dos dados. As amostras, como são imutáveis, poderão ficar em serviços de armazenamento de dados em bruto, por exemplo, permitindo alguma economia dos custos. Além disso, o número de provedores nesta área é bastante mais vasto, actuando em diversas áreas do globo, o que permitirá a colocação dos dados em locais mais próximos da sua captura.

Ao nível da interacção, a arquitectura presente pode ser descrita de várias formas. Do ponto de vista do utilizador esta é feita com um estilo cliente-servidor. Embora o serviço de alojamento de aplicações (onde se encontra o *front-end*) seja constituído por uma vasta rede de nós, que mitigam o problema do ponto central de falha, o acesso é virtualizado num único, sendo a própria rede responsável pelo balanceamento da carga. Já a comunicação entre o *front-end* e as outras *clouds* é feita de forma diferente. Embora o estilo cliente-servidor seja mantido, a interacção é efectuada pelo *front-end* com múltiplos servidores, dos quais terá de ter conhecimento prévio, efectuando cada comunicação de forma explicita. Internamente, cada um dos serviços remotos é distribuído, sendo o acesso, no entanto, igualmente virtualizado num único ponto.

3.1.1 Modelo de interacção

Este sistema a desenvolver pretende implementar, essencialmente, dois tipos de operações, inserção e pesquisa de dados.

Na inserção, o utilizador deverá submeter amostras ao sistema, com o intuito de as partilhar com a comunidade. Os dados poderão ser transmitidos de forma simples, por via dos seus valores reais (sem necessidade de um encapsulamento de acordo com o índice do sistema), através de interfaces *standard* (como REST, por exemplo). Ao nível da comunicação, este pedido deverá ser não bloqueante, já que não existem resultados de interesse para comunicar ao utilizador (num contexto de sensoriamento participado).

As operações de pesquisa apresentam algumas diferenças. Nestas, o utilizador interroga o sistema com o objectivo de recolher informações referentes às suas necessidades. Assim, deverá incluir, além do centro de pesquisa (igualmente sem encapsulamento especial), o intervalo de interesse da mesma (um valor real, por exemplo). Ao contrário da operação anterior, a comunicação deverá ser bloqueante, aguardando o utilizador pela recolha e transferência dos resultados.

Além destas, também a operação de remoção poderá ser interessante em alguns âmbitos de aplicação, ainda que não seja prioritária. Do ponto vista do utilizador, esta

processar-se-á de modo semelhante à inserção, igualmente com um estilo de comunicação não bloqueante.

3.1.2 Índice métrico

A RLC é uma estrutura de dados métrica, genérica e dinâmica, tal como descrito na secção 2.3.1. Esta pode conter informação de qualquer tipo (desde que passível de diferenciação entre si através de uma função métrica de distância) e a mesma ser inserida ou removida em qualquer altura da vida da estrutura. A organização dos dados é feita através da sua distribuição por vários agrupamentos, o que permite uma elevada eficiência nas pesquisas por proximidade. A Figura 3.2 (baseada em [29]) apresenta o esquema da RLC (com redução do raio de acordo com a profundidade) em memória principal. Cada agrupamento contém informação do centro (c), raio (r), nível de profundidade (l), número de elementos (s) e interior (I).

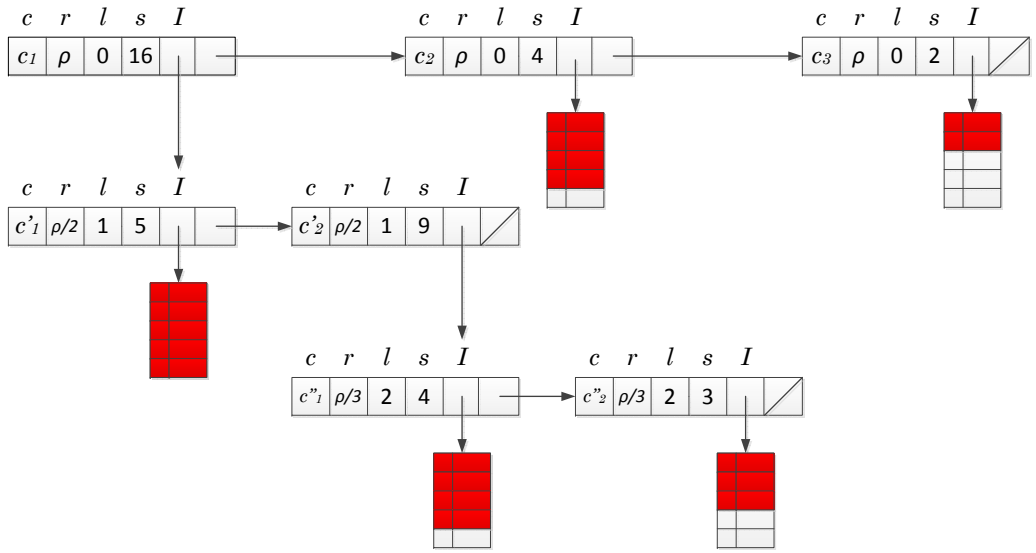


Figura 3.2: Exemplo da RLC em memória principal com raio inicial ρ e folhas de capacidade 5

No contexto do sensoriamento participado, os seguintes factos apresentam-se favoráveis à utilização da estrutura:

- Genérica: Permite a utilização da RLC independentemente do âmbito da aplicação, oferecendo esta mesma característica ao sistema.

- Dinâmica: O sensoriamento participado, por exemplo, implica uma recolha progressiva dos dados, ao longo do tempo, permitindo a evolução das vistas a criar. A estrutura de indexação destes terá de ser dinâmica, para que os mesmos possam ser inseridos em qualquer momento.

- Desempenho: Com o grande número de dados que é característico dos sistemas baseados neste paradigma, um bom desempenho será preponderante, especialmente ao nível das pesquisas por proximidade que, como se espera, serão as operações dominantes.

Como já foi referido anteriormente, para correr na *cloud*, a RLC deverá ser adaptada para executar num serviço de alojamento de aplicações. Estes serviços apresentam suporte para executar programas escritos numa determinada linguagem, permitindo um modelo de interacção com o cliente do tipo pedido/resposta, através de um protocolo conhecido.

O sistema de suporte de dados históricos provenientes de um contexto de sensoriamento participado pressupõe uma longa duração de vida, bem como um elevado número de dados envolvidos. Com o aumento da cardinalidade torna-se impraticável manter todos os dados em memória principal. O sistema deverá colocá-los em memória secundária, através do seu armazenamento noutros serviços que o possibilitem. Esta transferência dos dados para memória secundária é igualmente necessária para garantir a manutenção do sistema num tempo de vida a longo-prazo, tornando-o resiliente a falhas do servidor, desde que a persistência dos dados seja efectuada de modo adequado, como será explicado nas secções seguintes.

A versão original da RLC, com um raio fixo para todos os agrupamentos, não apresenta as melhores características para persistência em memória secundária. Deste modo, a versão descrita na secção 2.3.1.1, apresenta a RLC com um tamanho variável no raio dos agrupamentos, diminuindo de acordo com o aumento da profundidade destes. Esta alteração ofereceu um melhor desempenho à RLC em memória secundária, tornando-a mais larga e menos profunda. É a partir desta versão da RLC (mas em memória principal), que será feita a adaptação para memória distribuída, utilizada nesta dissertação.

Outro dos âmbitos deste sistema é garantir a diversidade de cópias. Esta propriedade permitirá ao sistema não ficar refém de um único provedor de serviço, quer em termos de custos, quer ao nível da durabilidade dos dados. Como tal, a informação relevante inserida no sistema, deverá ser replicada em várias *clouds*. O modo como será efectuada esta multiplicidade de cópias será alvo de estudo na secção 3.2.3.

3.2 RLC em Memória Secundária Distribuída

3.2.1 Modelação da estrutura

A utilização da estrutura no contexto da *cloud* implica a modelação do seu armazenamento em memória distribuída. Pretende-se que este armazenamento seja efectuado com base em pares chave-valor, cuja chave determina univocamente qual o objecto correspondente. Para tal, algumas referências em memória da estrutura deverão ser transformadas em chaves em memória distribuída na *cloud*. Esta modelação deverá ter em conta diversos factores:

- Concorrência: Objectos de baixa granularidade entrarão mais facilmente em conflito aquando de operações de modificação da estrutura concorrentes;
- Gestão de memória: A informação contida nos objectos deverá ser a estritamente necessária para realizar as operações imediatas. O excesso de informação no objecto provoca um desperdício de memória utilizada aquando da sua leitura;
- Operações de I/O: Estas operações referem-se a leituras e escritas em memória distribuída, dos objectos da estrutura. Tal como nos sistemas locais (sistemas de operação, por exemplo) este tipo de operações é um dos grandes factores de contenção. A modelação dos dados da RLC deverá permitir uma minimização destas operações, de modo a diminuir o tempo total de execução;
- Diversidade: A modelação da estrutura deve permitir que os dados possam ser replicados noutras *clouds*, de modo totalmente independente.

Tendo em conta as questões acima descritas, foi escolhida uma abordagem que divide a estrutura por nós, ou seja, nós interiores (RLCs) e nós terminais (folhas). Cada nó tem associada uma chave, o que permite fazer uso dos serviços que armazenam dados baseados no par chave-valor (o valor é o próprio nó), estando esta referenciada no nó com profundidade inferior que lhe dá origem, de modo a que os nós fiquem completamente encadeados. Nas folhas, é necessário efectuar outra alteração. Por motivos que serão explicados adiante, os objectos que contêm a informação passarão a ficar desligados das folhas. Os objectos, tal como os nós, são identificados por uma chave, sendo esta suficientemente genérica, de modo a não estar dependente da *cloud* de um provedor em específico. No local das folhas em que anteriormente estavam os dados, passa a estar a guardada a chave respectiva, para que estes possam ser acedidos sempre que necessário. O modelo descrito está representado na Figura 3.3.

3.2.1.1 Discussão

Esta solução satisfaz, razoavelmente, os objectivos pretendidos. Com uma granularidade ao nível do nó, numa RLC com um número elevado de dados, a probabilidade da inserção de dois objectos colidir nos mesmos nós é bastante reduzida. Esta reside essencialmente em casos de uma grande semelhança entre os objectos (de modo a que a inserção se dê na mesma folha) ou outro estado pontual em que haja necessidade de adicionar um novo agrupamento num nó que já tem uma modificação em curso, por exemplo. Por outro lado, não há um elevado desperdício de memória, dado que a informação contida no nó apresenta uma elevada probabilidade de ser consumida quase na totalidade (consoante a operação em causa), durante esse passo. A informação essencial de cada agrupamento (centro e raio) encontra-se presente no respectivo nó que o contém, não sendo necessárias leituras adicionais para verificar quais os agrupamentos em que a operação deverá prosseguir. Um nó mais profundo só necessita de ser trazido para memória

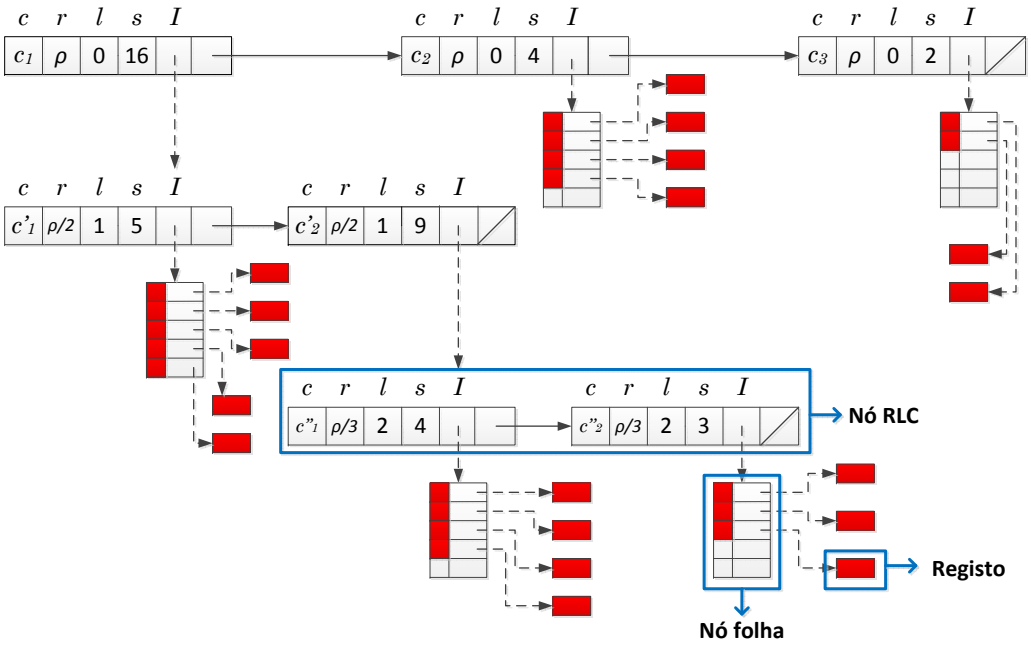


Figura 3.3: Exemplo da RLC desenvolvida para memória distribuída com raio inicial ρ e folhas de capacidade 5

principal se, efectivamente, a pesquisa (ainda que para inserir um novo objecto) tiver de decorrer dentro deste, minorando o número de leituras de nós.

Uma solução alternativa, poderia sugerir a divisão por agrupamentos. Porém, esta apresenta-se desaconselhada, tendo em conta o elevado número de operações de leitura envolvidas para obter os agrupamentos pertencentes a cada nó. Utilizando como exemplo uma operação de pesquisa em que todos os 100 agrupamentos de um nó são testados, nesta solução seriam necessárias efectuar 100 operações de leitura, o que não compensa o carregamento de um objecto de maiores dimensões.

A solução desenhada, devido a algumas limitações já referidas no capítulo 1, não prevê a diversidade de cópias do índice. A dificuldade de coordenação de várias *clouds*, em que se dão modificações concorrentes, impediu que este tipo de replicação fosse efectuada nesta dissertação. Porém, será facilmente comportável um estilo de replicação mais simples, como o primário/secundário. Em *background* os vários nós poderão ser persistidos noutras *clouds*, reduzindo a dependência de um único provedor.

A diversidade de cópias passa, igualmente, pela replicação dos dados. Embora seja possível replicar todos os nós do índice, tal solução é redutora do ponto de vista de utilização. Para outra aplicação utilizar estes dados, teria de ter conhecimento acerca da estrutura e de como as operações se processam nesta. Além disto, outros índices não poderiam ser criados sobre os mesmos dados. Isto impediria a sua utilização noutros contextos ao abrigo do espírito de partilha de informação existente no sensoriamento

participado, em que os mesmos dados poderão ser utilizados pela comunidade para âmbitos completamente diferentes. Assim, todos os registos (centros de agrupamento e dados pertencentes às folhas) deverão ser replicados, oferecendo a desejada diversidade de cópias. Além de dispensar a presença dos dados no mesmo serviço da RLC, a replicação poderá passar pela colocação destes em serviços de armazenamento em bruto, que são mais económicos e apresentam as características desejadas para o tipo de persistência em causa.

A outra alteração de especial importância é a deslocalização dos pontos (registos) contidos das folhas. Nesta solução, as folhas apenas contêm as listas de distâncias e as chaves, de cada ponto. A chave determina univocamente o ponto correspondente àquela posição, permitindo a sua leitura. Esta aproximação tem como custo algumas operações de I/O (quando é preciso aceder aos dados reais indexados nas folhas), mas oferece a versatilidade desejada. Por outro lado, reduz o tamanho das folhas, oferecendo uma maior eficiência na gestão da memória. O tamanho da chave é fixo, ao passo que os registos apresentam tamanhos variáveis. Assim, é possível indexar registos de qualquer dimensão (dados *media*, por exemplo), sem que isto afecte a estrutura das folhas. Nas RLCs o mesmo não é verificável, dado que o ponto não é deslocalizado, pelo que soluções alternativas poderão ter de ser estudadas caso não seja encontrada uma forma económica de representar a informação.

As alterações produzidas na estrutura não mudaram a topologia da mesma, pelo que a complexidade das operações será equiparável. Em memória secundária, a latência elevada das operações de I/O sugere que a complexidade seja medida em função destas. Porém, não foram elaborados estudos formais nesse sentido.

3.2.2 Representação e Consistência

O desenho do modelo de persistência dos dados na *cloud* de uma RLC para tipos genéricos, deu origem à definição de duas classes de objectos, nós e dados, sendo estes últimos designados por pontos no contexto da estrutura. Ambos são armazenados através de pares chave-valor (em que o valor representa o próprio objecto), sendo que no segundo caso a chave deverá ser suficientemente genérica para que haja compatibilidade nas várias *clouds*.

Na *cloud*, os tipos de serviço são vastos, permitindo diversas formas de armazenamento de dados. Porém, uma solução demasiado sofisticada e específica poderia reduzir de sobremaneira o número de serviços a que poderíamos recorrer para oferecer diversidade. Este tipo de solução obrigaria ainda a que o serviço a utilizar fosse, provavelmente, bastante mais dispendioso, algo que não seria desejável especialmente quando se pretende replicar os dados. Assim, preferencialmente, dever-se-á escolher uma solução que possa recorrer a um serviço de armazenamento de dados em bruto.

Esta solução apresenta-se simples e eficaz, sendo somente necessário que os objectos em questão (nós e pontos) possam ser serializados. A serialização deverá ser feita

pela aplicação presente no serviço de alojamento de aplicações (*front-end*) aquando da necessidade de efectuar uma escrita. A sequência de *bytes* resultante é associada a uma chave única, podendo ser posteriormente restaurada aquando de um pedido de leitura, cabendo à mesma aplicação a interpretação dos *bytes* de modo a reconstruir o objecto.

O processo que leva à persistência dos dados requer algum cuidado. Para o descrever, será utilizado, como exemplo, um pedido de inserção na RLC.

Quando se trabalha com estruturas de dados é necessário garantir que estas se mantêm coerentes, após efectuar cada uma das operações desejadas. Uma referência não existente ou desactualizada, por exemplo, poderá implicar um mau funcionamento do sistema, levando à sua ruptura. Assim, há que garantir que os vários objectos são guardados em consonância e que, no fim das alterações, nenhuma parte da estrutura fica inconsistente.

Os registos são os mais simples de persistir. Sendo totalmente independentes (não contêm apontadores, neste caso chaves, para outros objectos), poderão ser guardados tão breve quanto possível e desejado. Este tipo de objecto, poderá ser guardado em qualquer altura, desde que esta seja anterior à dos próprios nós da estrutura, que terão de ter conhecimento da chave bem como a garantia de que a informação foi realmente persistida.

A persistência dos nós da RLC é um pouco mais complexa. Consoante o tipo de alteração a ser efectuada acontecerá uma das seguintes situações:

- Inserção de um ponto numa folha: após o ponto ter sido guardado e haver informação da respectiva chave, esta é colocada na folha e a mesma terá de ser persistida.
- Ponto inserido não está contido na área de nenhum agrupamento e é criado um novo: um novo agrupamento com centro no ponto a ser inserido é criado no nó interior (RLC) em que é detectada esta condição. Uma nova folha, ainda vazia, é criada. Nó interior e folha são persistidos.
- Folha onde o ponto (um registo no contexto da RLC) deve ser inserido está cheia e tem de ser transformada numa nova RLC: os elementos presentes na folha são reinseridos e toda a nova subárvore tem de ser persistida, tal como a RLC mais profunda que a contém.

A primeira das situações possíveis é a de mais simples resolução, dado que a alteração na estrutura é feita num único nó, não havendo dependências em risco. Porém, nos outros casos, e em especial no terceiro, se uma das operações de persistência não se realizar, a consistência da estrutura é posta em causa. É necessário garantir que o conjunto de todas as operações de escrita é feito de forma atómica. Os vários nós alterados são colocados num *log* local e, quando a inserção na RLC estiver completa, são feitas todas as escritas, atomicamente[1]. Com este modelo de escrita retardada e de forma transaccional, é garantida a consistência da estrutura, mantendo o sistema resistente a falhas.

[1] As transacções possuem características ACID (em inglês, *Atomicity, Consistency, Isolation, Durability*; ou seja, Atomicidade, Consistência, Isolamento, Durabilidade), garantido que o conjunto de operações nunca é feito de forma parcial: escritas completadas com êxito ou falha na totalidade.

O sistema desenvolvido nesta dissertação tem como pressuposto a existência de um único processo escritor. Embora seja interessante a inclusão de escritas concorrentes, essa questão não foi endereçada neste trabalho.

3.2.3 Diversidade

A existência de uma diversidade de cópias tem custos acrescidos para a solução, mas com o benefício de lhe oferecer maior extensibilidade e independência. No modelo descrito, a redundância é feita ao nível dos dados e não da estrutura do índice. Através deste, aplicações com outros âmbitos e arquitecturas poderão utilizar os mesmos dados, sem necessitar de ter conhecimento acerca da RLC, a estrutura de dados métrica que os indexa neste sistema.

Para gerir este sistema de cópias há apenas a necessidade de estender o modelo de armazenamento chave-valor a várias *clouds*, como demonstra a Figura 3.4. A lógica deste sistema tem como uma das principais tarefas a atribuição de uma chave a cada elemento. Esta chave deverá ser suficientemente genérica, de modo a possibilitar a sua utilização em qualquer um dos serviços de armazenamento a utilizar. Deste modo, o índice não necessita de conhecer aspectos de diversidade, sendo totalmente transparente a localização dos objectos. Se tal não ocorrer, há a necessidade de criar e indexar várias chaves (de acordo com as limitações dos serviços) para cada objecto, o que constitui um elevado desperdício de memória e condiciona o modelo de execução das operações do índice. No caso da replicação não ser completa, a chave representada no índice poderá indicar as localizações em que é possível encontrar o registo, sendo que a sua chave real, em cada um dos serviços, será representada apenas por uma parte desta, interpretada no componente de resolução de chaves.

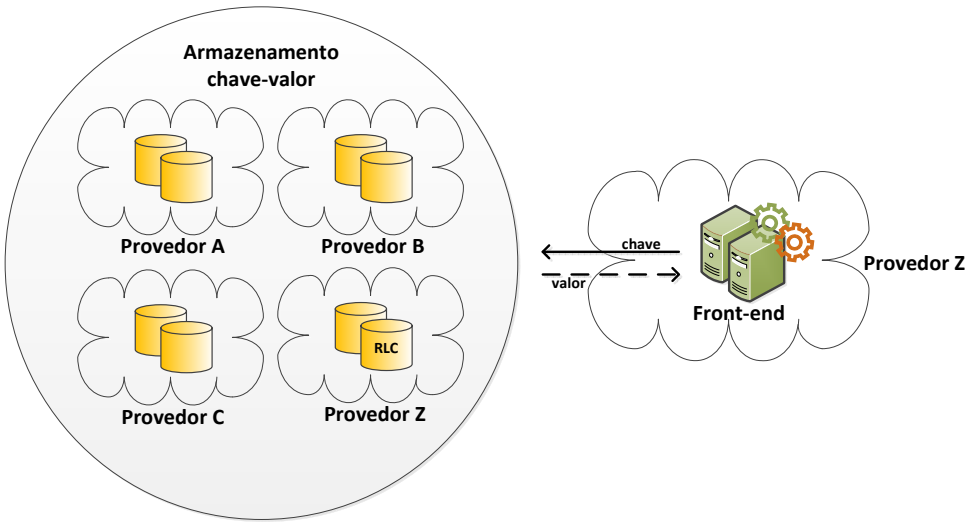


Figura 3.4: Modelo de armazenamento chave-valor com diferentes *clouds*

Durante o processo de inserção, e depois de atribuída a chave, o registo tem ser persistido nos serviços disponibilizados por diferentes provedores. Esta persistência deverá ser efectuada de forma atómica, de modo a que os dados estejam replicados. Deste modo, evita-se que o abandono de um serviço implique a perda parcial de informação, o que poderia trazer consequências adversas para o funcionamento do sistema. A atomicidade não é, porém, fácil de garantir. Tratando-se de serviços diferentes, fornecidos por provedores concorrentes, a coordenação e cooperação entre estes será de difícil execução, pelo que o mecanismo terá de ser efectuado totalmente do lado do *front-end*. A operação é considerada bem sucedida quando a persistência individual em cada uma das *clouds* é completada com êxito.

Nas operações de leitura, o sistema terá de fornecer a chave ao componente de armazenamento. Este deverá proceder à resolução da chave, e efectuar a leitura do(s) local/-locais que considerar mais apropriado. Por fim, o objecto deverá ser retornado à camada superior, sem que esta tenha conhecimento da real localização do mesmo.

3.3 Melhoramentos

3.3.1 *Caching*

O acto de fazer *caching* consiste em trazer para uma memória mais rápida dados que possuem um tempo de acesso lento ou, guardar temporariamente resultados de computações. A colocação de ambos numa *cache* só terá interesse se a informação vier a ser utilizada mais vezes, num futuro próximo.

No contexto deste problema, o *caching* a realizar consiste em manter em memória principal (numa *cache*), temporariamente, alguns dos objectos que foram tornados persistentes, nomeadamente nós da RLC e pontos (registos). Este mecanismo será um auxilio importante na eficiência do sistema, permitindo poupar várias operações de I/O, que são uma grande fonte de contenção.

A existência de uma *cache* levanta, porém, duas questões. Por um lado, há que garantir a consistência e validade dos valores presentes. Leituras desactualizadas poderão levar à instabilidade do índice, o que não é de todo desejado. Por outro, é necessário efectuar uma gestão do espaço e, como tal, priorizar os elementos de modo a excluir os de menor importância quando for necessário.

A manutenção da consistência é, provavelmente, a mais crítica, já que poderá influir no funcionamento do sistema médio/longo prazo. A dificuldade de implementação desta está sobretudo dependente do nível de concorrência do sistema. No caso de existir somente um processo escritor, não é necessário lidar com alterações concorrentes, simplificando o sistema de *cache*. Como a escrita dos nós é retardada, o mesmo deverá acontecer com as alterações destes na *cache*. Assim, os nós deverão ser colocados numa estrutura temporária (sempre com prioridade de leitura mais elevada), ao mesmo tempo que os correspondentes mais antigos são invalidados. Aquando da persistência com sucesso

dos nós, os valores na estrutura temporária devem ser colocados na *cache* principal. Os registos, por terem uma persistência imediata, poderão ser colocados logo de seguida na *cache*. Caso seja necessário garantir a consistência com vários escritores, medidas adicionais terão de ser implementadas. Transacções baseadas em *locks* ou estampilhas temporais poderão ser utilizadas.

A existência de um tamanho limite, que será sempre dependente do serviço a utilizar, para a *cache* implica que haja uma política de expiração associada a esta. Esta política terá de ser elaborada após efectuado um estudo aprofundado acerca dos objectos lidos. No contexto de utilização da RLC, por exemplo, é possível julgar de imediato que nós menos profundos, terão maior incidência de leituras, dado que qualquer operação tem início na raíz e prossegue nos filhos subsequentes (se justificável). Deste modo, uma boa política de *caching* deverá dar especial importância aos nós menos profundos, mantendo-os em memória durante mais tempo (desde que não hajam alterações que os invalidem). Uma política baseada em LRU (*least recently used*) poderá não ser adequada, já que quando o *working-set* for de grande dimensão, os nós mais perto da raiz tendem a ser excluídos da *cache*.

No caso dos pontos, igualmente para a RLC, não há, no entanto, uma localização preferencial, estando todos ao mesmo nível: nas folhas (os centros de agrupamento estão imediatamente em memória principal aquando da leitura do nó, pelo que não é necessário fazer *caching* deles). O algoritmo da *cache* para este tipo de objectos poderá ser, consoante o âmbito, baseado em LRU, isto é, mantendo em *cache* os objectos vistos mais recentemente. Seja no mesmo pedido ou nos pedidos seguintes, os últimos pontos serão mais provavelmente revisitados do que qualquer um dos outros, por questões de proximidade de interesse. Ainda assim, estudos futuros acerca da utilização do sistema poderão transmitir informação mais detalhada que possa justificar uma maior sofisticação da priorização dos registos.

A *cache* poderá ser simplesmente implementada através de uma tabela de dispersão associada a uma fila com prioridade, para cada tipo de objecto que poderá conter. A tabela de dispersão terá indexados os objectos através de um modelo chave-valor, em que as chave corresponde à chave real do objecto e o valor ao próprio objecto. A fila será o suporte à política de gestão do espaço, priorizando as chaves dos objectos a eliminar.

A utilização da *cache* é baseada no sistema de armazenamento chave-valor. Quando o sistema requer a leitura de um objecto com uma determinada chave, esta deve ser interpretada no componente de resolução de chaves. Após a sua interpretação, caso o objecto correspondente à chave real esteja em cache (e este esteja válido) é imediatamente devolvido à camada superior, caso contrário, procede-se à respectiva operação de I/O. O índice métrico deverá desconhecer a origem do objecto. Ao nível das escritas o processo é semelhante, devendo a camada intermédia colocar os valores associados à sua chave real na *cache*, quando for o momento adequado.

3.3.2 *Shadow Buckets*

Ao longo deste capítulo têm sido descritas um conjunto de modificações efectuadas na RLC com vista à sua persistência na *cloud*. Entre as mais importantes, podemos referir a transformação das referências em memória principal dos nós para referências de objectos persistidos em memória distribuída. Esta alteração era necessária, de modo a não ter a RLC completa em memória em cada um dos pedidos. Outra alteração foi efectuada ao nível das folhas, com a deslocalização dos registos, num processo semelhante ao anterior. Neste caso, a solução prende-se com uma opção de desenho, que é negativa para dados de pequena dimensão mas preferível se trabalharmos com dados de tamanho elevado (*media*, por exemplo). Em qualquer dos casos há um impacto negativo das operações de pesquisa, devido ao custo da leitura dos pontos.

O modelo de armazenamento proposto é genérico (no tipo e na dimensão dos dados), porém, as opções tomadas provocam um pior desempenho do que o desejado, em casos mais simples. De forma a mitigar estas desvantagens, alguns melhoramentos poderão ser adicionados. É neste contexto que surge um novo tipo de nó: o *shadow bucket*.

Este novo objecto consiste na cópia de um nó folha (*bucket*), mas em que os dados não se encontram deslocalizados, ou seja, os dados reais estão contidos no próprio objecto, tal como na versão original da RLC. Isto resulta numa forma de *caching* e *prefetching* dos registos. A chave de um *shadow bucket* deverá estar fortemente relacionada com a da folha correspondente, não havendo necessidade de fazer referência a ambas.

A alteração apresentada é um melhoramento da solução inicial, reduzindo o número de leituras, que não altera a estrutura, ou seja folhas e *shadow buckets* poderão coexistir no sistema, como é visível na Figura 3.5. Porém, a existência dos *shadow buckets* não deverá ser totalmente transparente para a RLC. Nas operações de pesquisa, quando é exigida a leitura de folhas, devem tentar ser obtidas primeiramente as cópias e somente em caso de inexistência utilizarão o objecto original.

A criação dos *shadow buckets*, no entanto, não apresenta um modelo fixo, variando consoante os requisitos do sistema. Este tipo de nó poderá estar localizado em memória distribuída ou em cache; pode ser criado de modo probabilístico, por exemplo; e a sua criação poderá decorrer aquando da inserção dos dados ou na primeira pesquisa.

3.3.2.1 Operações

Ao nível das operações, as inserções e remoções na RLC deverão ser sempre feitas nas folhas originais, por pertencerem garantidamente à estrutura e apresentarem consistência de chaves dentro desta. Os *shadow buckets* (se persistidos em memória estável), sendo cópias, podem apresentar-se desactualizados ou inexistentes. No caso do *shadow bucket*, correspondente à folha em que o elemento foi inserido, existir *a priori*, este deverá ser invalidado e removido, no imediato ou ao fim de algum tempo, consoante o nível de consistência de resultados pretendida. Nas remoções, o processo deverá decorrer de modo semelhante, invalidando e removendo a cópia desactualizada. Já as operações de

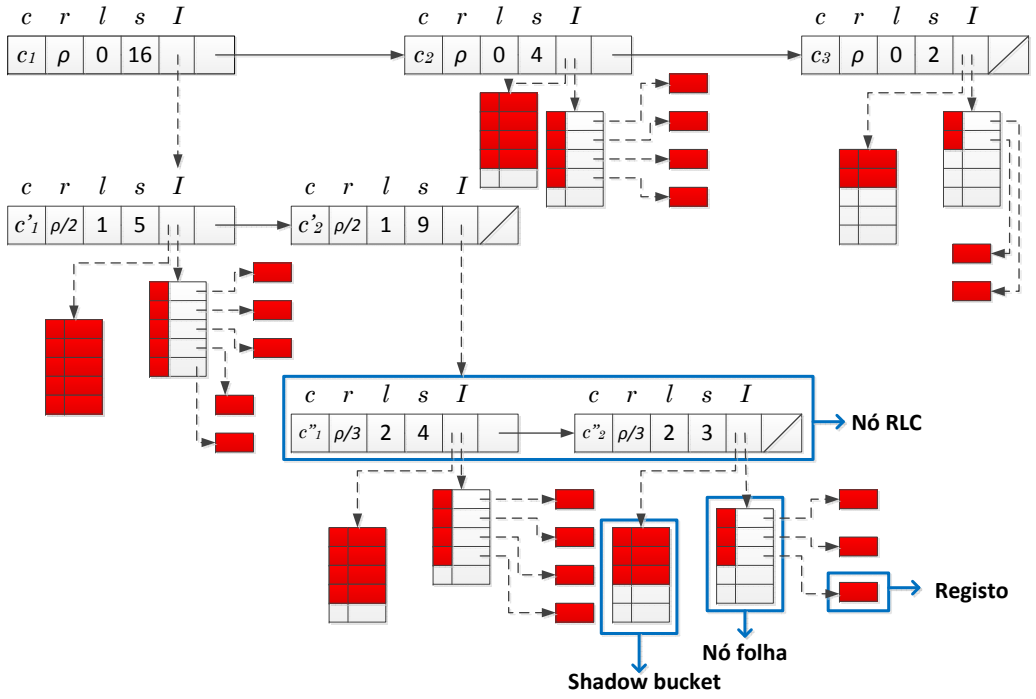


Figura 3.5: Exemplo da RLC desenvolvida para memória distribuída, com recurso a *shadow buckets*, de raio inicial ρ e folhas de capacidade 5

pesquisa deverão correr preferencialmente sobre estes nós, de modo a agilizar este processo.

3.3.2.2 Criação

A criação dos *shadow buckets* poderá ser efectuada em momentos distintos, consoante os requisitos do sistema. Um dos momentos aponta para o da inserção de dados numa folha, ao passo que a alternativa poderá ser o da primeira pesquisa registada sobre esta. Em qualquer um dos casos, esta operação deve ser sempre realizada em *background*, de modo a não prejudicar o pedido em curso.

A criação do *shadow bucket* logo após o momento da inserção, garante uma eficiência superior das pesquisas, já que quando estas entram na zona do agrupamento mais profundo que o contém, não será necessário efectuar operações de I/O extra para a leitura dos registos. Por outro lado, criar o *shadow bucket* aquando do momento da primeira pesquisa que percorre a folha correspondente, reduz o espaço total ocupado por estes objectos. Neste caso, não serão criados *shadow buckets* de folhas que nunca foram pesquisadas.

Além do momento da criação, também a frequência com que esta é efectuada pode variar. Em geral, a existência de *shadow buckets* melhorará o desempenho das pesquisas. Esta melhoria é gradual e aumenta de acordo com o número de cópias existentes. Em

certas aplicações, a redução parcial do número de leituras de registos poderá ser suficientemente aceitável, não havendo necessidade de gastar maior capacidade com *shadow buckets* de outras folhas.

3.3.2.3 Localização

Os *shadow buckets* poderão estar armazenados fisicamente na *cloud* (à semelhança dos outros nós) ou ser colocados em *cache*. A primeira hipótese garante, no caso de uma criação total, que a pesquisa poderá ser sempre feita utilizando os *shadow buckets*. Já a segunda não apresenta custos adicionais de alojamento, que têm um acréscimo elevado no caso da existência de muitas folhas.

Se o processo de criação e gestão dos *shadow buckets* é simples quando estes se encontram armazenados, o mesmo não sucede quando são mantidos em *cache*. Uma das dificuldades prende-se com o tamanho limitado da mesma. Quando o espaço reservado a *shadow buckets* estiver completo, alguns elementos aí presentes poderão ter de ser removidos. O estabelecimento de prioridades poderá passar, à semelhança do que acontece com os registos, por uma política LRU. A consequência natural da escolha desta localização é o desaparecimento de *shadow buckets* que já tinham sido criados anteriormente, desperdiçando o trabalho realizado.

3.3.2.4 Discussão

O melhoramento através da existência de *shadow buckets* pode estar presente de diversos modos, sendo que as suas características dependem do sistema em causa.

Num sistema em que um elevado desempenho é crucial, a persistência dos *shadow buckets* poderá ser a melhor solução, associada à sua criação logo após o momento de modificação da estrutura. Como o espaço é ilimitado, os *shadow buckets* não são removidos após a sua criação, desde que não se verifiquem alterações na folha correspondente. A percentagem de objectos a criar poderá ser variável, de acordo com a relação custo-desempenho desejada.

Por outro lado, se o cenário for um sistema económico, a localização dos *shadow buckets* na *cache* é o mais indicado. Esta vertente, poderá ser adicionada a qualquer sistema que possua um mecanismo de *caching*, sem um acréscimo de custo para o mesmo. Neste caso, não será justificável a criação imediata das cópias (sob via de nunca serem utilizadas), pelo que esta deve ser guiada pelas pesquisas efectuadas. A criação parcial dos *shadow buckets*, segundo um padrão de interesse (nível de preenchimento da folha, por exemplo), poderá ser vantajosa, na medida que permite rentabilizar o espaço limitado disponível com objectos que produzam melhorias significativas no desempenho.

4

Protótipo

Neste capítulo será apresentado o protótipo realizado para esta dissertação. O protótipo foi desenvolvido na linguagem Java e teve como âmbito a pesquisa de imagens de rosto, num portal alojado na *cloud*.

Este protótipo tem como objectivo demonstrar, na prática, a validade do desenho proposto no capítulo anterior, bem como avaliar (empiricamente) que nível de desempenho se pode esperar. A maioria dos pressupostos descritos foram implementados, salvo algumas excepções, por manifestas limitações temporais ou restrições do sistema em que foi alojada a aplicação, o Google App Engine [19]. Nas secções seguintes, o protótipo desenvolvido será descrito detalhadamente, acompanhado por uma descrição da plataforma utilizada e das limitações de ambos. A avaliação culminará no capítulo seguinte em que será feita a validação experimental do protótipo, com o intuito de analisar a viabilidade da solução estudada.

4.1 Âmbito da aplicação estudo de caso

A aplicação estudo de caso elaborada nesta dissertação teve como tema a pesquisa por alcance de imagens de rosto, surgindo no âmbito do reconhecimento facial.

O reconhecimento facial é um campo que tem apresentado bastante desenvolvimento na última década. Envolvendo investigadores nas áreas do processamento de imagem, reconhecimento de padrões e redes neuronais, por exemplo, tem recebido especial atenção devido ao elevado número de aplicações práticas possíveis, acompanhadas da existência de tecnologias fiáveis que as sustentem. Estas aplicações apresentam, no entanto,

ainda alguns problemas, havendo particulares dificuldades em gerir as variações de luminosidade (especialmente em ambientes exteriores) ou de postura dos participantes envolvidos [36].

As oportunidades comerciais existentes neste campo, especialmente para os videojogos, são um factor extra de incentivo à inovação. Para além destes, *smart cards*, com o objectivo de identificar um indivíduo, são objecto de desenvolvimento, aparecendo como alternativa a *scanners* biométricos, por exemplo, que não dispensam a cooperação dos alvos a identificar.

O processo de funcionamento de um sistema de reconhecimento facial genérico envolve as seguintes fases:

- Detecção facial - descobrir presença de um rosto, estimação da postura, normalização (preparação da informação para as fases seguintes), etc.;
- Extracção das características - redução dos dados através da extracção das características essenciais;
- Reconhecimento facial - identificação/verificação.

Um método de agilizar este processo de modo eficiente passa pelo recurso a *frameworks* desenvolvidas nesta área. Neste trabalho, à semelhança do que acontece em [37], foi utilizado o método Eigenfaces.

O método Eigenfaces, proposto por Turk e Pentland em 1991 [38], é fundamentado numa *framework* que fornece a habilidade de aprender a reconhecer novos rostos de modo não-supervisionado. Para tal, usa uma aproximação da teoria da informação, retendo os dados mais relevantes (espaço de rostos ou *feature space*) de um grupo de caras (conjunto de treino) que melhor as distingue entre si, com recurso a um sistema baseado em *machine learning*. O espaço de rostos é definido pelos denominados *eigenfaces*, que consistem nos vectores próprios ou *eigenvectors* (não necessariamente de partes específicas do rosto como olhos ou nariz) do conjunto de imagens de treino.

O reconhecimento de um novo rosto passa por projectar a imagem no espaço de rostos e classificá-la através da comparação das suas posições com os vectores próprios do conjunto de treino, dando origem a um vector de pesos que a representa. O vector de pesos obtido é passível de ser utilizado no cálculo de uma função métrica, o que se apresenta como um método eficaz e de elevado desempenho, dispensando aceder à imagem real. Assim, a RLC presente no protótipo utiliza uma função métrica para calcular a distância euclidiana entre dois vectores de pesos distintos, de modo a definir o grau de semelhança entre duas imagens de rosto. Outras funções alternativas poderão ser implementadas, desde que possuam características métricas e possam ter como base de cálculo vectores de valores reais, de tamanho igual mas desconhecido *a priori*.

4.2 Ambiente do protótipo

A aplicação estudo de caso elaborada foi desenvolvida totalmente no ambiente da *cloud*. Além do armazenamento, também a lógica da aplicação (onde se encontra, por exemplo, a parte funcional da RLC) foi colocada a executar nesta, num serviço de alojamento de aplicações.

O serviço de alojamento de aplicações utilizado na realização desta dissertação é o disponibilizado pela Google, o Google App Engine [19], daqui em diante designado por GAE. Este serviço permite a execução de programas desenvolvidos numa de três linguagens: Java, Python e Go. A linguagem Java, em que foi desenvolvido o protótipo, é suportada quase na sua totalidade, à excepção de algumas bibliotecas de domínio específico, como as relacionadas com a manipulação de imagens, e as chamadas ao sistema, de modo a prevenir uma utilização incorrecta e/ou abusiva por parte das aplicações aí alojadas. Por outro lado, algumas bibliotecas adicionais poderão ser carregadas, desde que não utilizem os métodos interditos, desempenhando um papel importante na compatibilidade com aplicações já existentes e até mesmo no auxílio ao desenvolvimento de outras novas.

As diferenças para as aplicações locais surgem sobretudo ao nível da comunicação. Enquanto que localmente são passados parâmetros aquando da chamada do programa através da linha de comandos, por exemplo, nas aplicações alojadas no GAE a comunicação é feita através de *servlets*. Os *servlets* são instanciações de interfaces com o utilizador, através dos quais este comunica com o servidor, fornecendo os dados necessários para a execução do seu pedido. Assim, "à frente" da funcionalidade da aplicação está um *servlet*, responsável por recebê-lo, tratá-lo e devolver os resultados finais ao utilizador.

O serviço do GAE, à semelhança de outros na *cloud*, baseia-se na virtualização de um servidor de domínio público, possuindo um URL, único, e através do qual é possível aceder aos dados pretendidos. Sob o protocolo HTTP, podem ser efectuados pedidos de GET e POST em *browsers* ou outras aplicações desenvolvidas para o efeito.

Além de alojar aplicações, o GAE tem ainda associado um centro de dados, com capacidade para a criação de bases de dados não-relacionais. Neste, é simples de abstrair o armazenamento em bruto, bastando para isso associar cada objecto serializável (*BLOB*) a uma chave única.

4.3 Arquitectura

No capítulo 3 foi introduzida uma arquitectura proposta para o sistema que se pretende implementar. Esta apresenta uma divisão em três áreas distintas: utilizadores; *front-end* e índice; e armazenamento. As questões relativas ao processamento e ao armazenamento devem ser resolvidas na *cloud*, por esta apresentar características que vão de encontro às necessidades do sistema.

Esta arquitectura foi implementada parcialmente no protótipo. O *front-end* é suportado pelo servidor de aplicações do GAE, ao passo que os dados da RLC são armazenados através do serviço de bases de dados não-relacionais que o acompanha, na Bigtable [22]. Porém, devido a limitações temporais, o protótipo não endereçou a questão da diversidade, pelo que todo o armazenamento chave-valor é efectuado na Bigtable.

4.3.1 Comunicação

Postos os vários componentes envolvidos no sistema, é possível resumir o padrão de utilização do mesmo:

- O utilizador contacta o servidor através de um determinado URL, associando-lhe os parâmetros necessários para a operação;
- Do lado do GAE, o *servlet* é colocado em memória (se não tiver ocorrido nenhum pedido recente para o mesmo) e recebe a informação introduzida;
- Ao processar os parâmetros introduzidos, recolhe do centro de dados os registos desejados, interpretando-os no contexto da aplicação e realizando as operações necessárias até obter o resultado final;
- O *servlet* escreve no fluxo de saída a página HTML, por exemplo, correspondente ao resultado computado, que é posteriormente processada pela aplicação do utilizador que efectuou o pedido.

4.4 Interface e funcionalidades

O protótipo implementado possui duas funcionalidades principais: inserção e pesquisa por proximidade de imagens de rosto. Estas operações são efectuadas na aplicação em execução no GAE, através de pedidos HTTP.

O portal desenvolvido apresenta uma página inicial com um conjunto de *radio boxes* para a selecção da operação desejada. Na mesma página, estão presentes duas caixas para escolha de ficheiro: uma para carregar a imagem a inserir e outra para o ficheiro com o objecto que possui as propriedades resultantes do método Eigenfaces. Este último é necessário devido às limitações do serviço, dado que estes dados não podem ser computados *online*. Sendo esta a única informação necessária para efectuar pesquisas, a imagem de rosto não necessita de ser carregada para efectuar a operação. Além dos componentes anteriores, existe também uma caixa de texto, para a inserção do valor do raio nas operações de pesquisa.

Ao nível das respostas, ambas as operações apresentam resultados diferentes. No caso das inserções, a resposta é simples, estando apenas relacionada com a realização bem sucedida, ou não, do pedido. Já no caso de se desejar efectuar uma pesquisa, o resultado é bastante diferente. Para este, é efectuada uma listagem das várias figuras

incluídas no conjunto-resposta, devendo para isso ser efectuado um novo pedido HTTP para carregar cada uma delas (a realizar pela aplicação do utilizador).

Outras funcionalidades estão disponíveis no sistema, porém, a sua utilização deverá ser reduzida ao suporte de gestão do mesmo. Entre elas encontramos a remoção dos nós marcados e a eliminação dos dados do sistema.

4.5 Implementação

A implementação do protótipo foi realizada na linguagem Java e endereça, na maioria dos aspectos, as soluções apresentadas anteriormente.

Um dos principais componentes do sistema, o módulo de armazenamento chave-valor, foi implementado na classe Storage. Cada pedido, no início da execução, inicializa uma instância desta classe, que o acompanha na maioria das operações. Além de endereçar a atribuição e resolução de chaves, oferece o suporte necessário para efectuar um *caching* local. Este, ao contrário do proposto no desenho, é bastante limitado, já que não é partilhado pelos vários pedidos. Assim, apenas regista os pontos obtidos do *datastore* (os nós nunca são lidos mais que uma vez), poupando algumas leituras extra. Quando é feito um pedido de leitura para um ponto, este módulo verifica se o mesmo já se encontra presente em memória, efectuando a operação de leitura no *datastore* somente em caso negativo.

Além de lidar com o problema das leituras, as instâncias da classe Storage são também responsáveis por resolver o problema das escritas. Com o modelo retardado idealizado, é mantida em memória uma lista de nós a escrever, até as operações de modificação da estrutura estarem concluídas. Nesta altura, o *log* é descarregado, persistindo os novos valores em memória distribuída. Para tal, são utilizadas transacções, oferecidas para um modelo de persistência de objectos baseado em JDO (Java Data Objects). Porém, a sua implementação no GAE não é trivial. Durante uma transacção, apenas objectos cujas chaves estejam relacionadas podem ser persistidos. A relação das chaves é simples de induzir, mas pode ter consequências indesejadas no armazenamento, já que todos os objectos são guardados no mesmo nó físico. Assim, apenas as folhas apresentam relações de chave, estando ligadas ao nó acima. Deste modo, é satisfeito o requisito de que nas expansões das folhas, todos os nós resultantes são guardados de forma atómica, ignorando os casos restantes em que esta não é primordial. É ainda de referir que este módulo é responsável por registar os nós a remover do sistema. Quando uma folha é expandida, deverá ser removida somente após algum tempo, de modo a não provocar a falha de algum processo corrente. Assim, é guardada uma lista de nós em memória secundária, para que, posteriormente, uma função de administração os possa remover.

Os *shadow buckets* foram implementados neste protótipo através do seu armazenamento no *datastore*. Aquando de uma inserção numa folha, é criada uma nova Task, que é colocada numa TaskQueue (serviço disponibilizado pelo GAE que permite a execução

de trabalhos em *background*). Com base na geração de valores aleatórios e numa probabilidade de criação associada, o tipo da tarefa resultante poderá indicar a criação ou remoção do *shadow bucket*. Nas operações de pesquisa, quando esta necessita de obter um nó terminal, prioriza a leitura do *shadow bucket*, efectuando a leitura da folha no caso de a anterior não se poder concretizar.

4.6 Conclusão e limitações

Neste capítulo foi efectuada uma descrição alargada do protótipo implementado, que permite realizar pesquisas de imagens de rosto num portal alojado na *cloud*.

No portal desenvolvido, é possível efectuar operações de inserção de imagens de rosto e pesquisas por proximidade das mesmas, fazendo uso de uma interface simples. Aquando de uma pesquisa por proximidade, a listagem das imagens pertencentes é devolvida, dispensando qualquer acção extra do utilizador para as visualizar.

Ao nível do sistema de suporte, alguns componentes não se apresentam implementados na totalidade. O sistema de *caching*, tal como foi sugerido no desenho, não foi implementado. Algumas simplificações foram incluídas, nomeadamente para evitar leituras repetidas dentro do mesmo pedido, porém, entre pedidos diferentes, estes valores não são partilhados. Esta ausência de *cache* traz algumas limitações aos melhoramentos propostos. Os *shadow buckets* são criados de modo percentual e em memória secundária, ou seja, uma percentagem das folhas tem persistida a respectiva cópia. Parte da validação experimental passa pela variação deste valor, como se poderá verificar no capítulo seguinte. Igualmente para agilizar este processo, a criação dos *shadow buckets* é feita durante a inserção, e não durante a pesquisa, sendo colocada a executar uma nova tarefa em *background*, através do uso de Tasks, colocadas numa TaskQueue.

A capacidade de diversidade não se encontra igualmente presente. Devido às restrições temporais não foi possível incluir esta faceta, pelo que é sugerida para trabalho futuro. No entanto, a implementação reflecte os pressupostos enunciados por esta, pelo que as alterações a efectuar no protótipo serão mínimas. Todos os dados foram armazenados num sistema Bigtable, que acompanha o GAE.

Ao nível da própria RLC, há duas alterações fundamentais a notar. Em primeiro lugar, não foi implementada a operação de remoção. Esta operação, no contexto de um sistema com muitos dados armazenados como os provenientes de sensoriamento participado, não é fundamental. O objectivo passa pelas inserções e consultas, pelo que o peso acrescentado pela remoção individual de um único valor não traz benefícios particulares. A remoção de dados poderia ser efectuada de forma conjunta, com uma filtragem destes aquando da recomputação da RLC (com o objectivo de ajustar parâmetros). Igualmente, o número de nós presentes no interior de cada agrupamento não é mantido. Embora isto seja de fácil execução em memória principal, não o é em memória distribuída. O número de operações de escrita extra pode ser muito elevado, pelo que o peso não o justifica.

5

Validação experimental

Neste capítulo, pretende-se demonstrar e interpretar um conjunto de resultados obtidos através de testes efectuados ao protótipo implementado.

Como foi referido anteriormente, nesta dissertação foi criado um portal que tem como objectivo a pesquisa de imagens de rosto, por proximidade, utilizando para isso uma estrutura de dados métrica. As operações a efectuar centram-se na inserção e pesquisa de imagens, sendo o desempenho da última crítico na generalidade dos casos. Assim, serão apresentados alguns valores relativos a métricas do tempo de execução destas operações, bem como outras medições de operações subatómicas das mesmas. Valores relacionados com a dimensão dos dados serão igualmente apresentados.

5.1 Introdução

Nesta secção é feita uma breve introdução acerca da validação experimental efectuada. Além das apresentações da base de dados e do *benchmark* utilizados, é feita uma descrição das limitações técnicas do GAE [19] bem como do modo em que estas influenciaram os testes realizados. Parametrizações da RLC e raios de pesquisa utilizados são igualmente alvos de análise.

5.1.1 Base de dados

Na validação experimental efectuada ao protótipo implementado foi utilizada somente uma base de dados de imagens de rosto: a *Faces94*.

A *Faces94*, disponível em [2], é uma base de dados de imagens de rosto composta por 3040 fotografias de um total de 153 indivíduos. Cada indivíduo está representado em

aproximadamente 20 imagens, obtidas a uma distância fixa da câmara e sem variações de luminosidade, enquanto o sujeito profere algumas palavras (por forma a obter diversas expressões faciais de modo natural).

Os sujeitos representados na base de dados apresentam maioritariamente uma idade compreendida entre 18 e 20 anos (não excluindo a existência de indivíduos mais velhos). São de vários géneros e raças e apresentam outras características faciais distintas, como por exemplo a existência de barba/bigode e óculos.

As imagens apresentam-se em formato JPEG de 24 bits e com uma resolução de 180x200 *pixels*, permitindo um rápido processamento por parte do sistema *Eigenfaces* [38] e ocupando individualmente um espaço reduzido em memória (principal/secundária).

Alguns exemplos das imagens de rosto presentes no pacote *Faces94* podem ser visualizados na Figura 5.1.

Figura 5.1: Exemplos de imagens de rosto presentes na base de dados *Faces94* [2]

5.1.2 *Benchmark*

Com o objectivo de obter resultados consentâneos e de possível comparação, foi criada uma bateria de testes sob a base de dados *Faces94*, a submeter às várias versões da estrutura de dados, alvos de avaliação. Deste modo, os valores obtidos referem-se à mesma sequência de operações, com os mesmos dados, sobre estruturas com estados de vida iguais. Em todos os testes foi utilizada a distância euclidiana para diferenciar os dados. Os cálculos efectuados apenas fazem uso dos vectores de pesos (resultantes da aplicação do método Eigenfaces [38]), não necessitando de recursos de computação muito grandes. Como estão contidos nos pontos, é dispensada a leitura das imagens de rosto correspondentes.

As operações sujeitas a avaliação foram as de inserção e pesquisa. Embora o tamanho da base de dados esteja longe de ser comparável ao número de dados envolvidos num contexto de sensoriamento participado, por exemplo, a inserção de todas as imagens de rosto contribuiu para a minimização da perda do cenário, permitindo a obtenção de valores relevantes para a avaliação da solução. Sendo a RLC uma estrutura de dados dinâmica, em que as inserções de dados podem ocorrer em qualquer altura da vida desta,

a ordem pela qual são inseridos os dados influenciará a própria organização da estrutura. Como tal, é necessário garantir que nas várias execuções da bateria de testes, a ordem de inserção das imagens de rosto se mantém, de modo a que a RLC resultante apresente sempre o mesmo aspecto, para estruturas de configuração igual. Caso este facto não se verifique, os tempos de execução das operações serão impossíveis de comparar, por se referirem a estruturas em estados de vida distintos. Deste modo, foi definida uma ordem arbitrária das imagens pela qual é sempre feita a inserção, em qualquer uma das estruturas e configurações. O número total de operações de inserção é de 3040, não havendo repetição de objectos.

No caso das operações de pesquisa a problemática é ligeiramente diferente. Este tipo de operações não provoca qualquer alteração na estrutura, pelo que a ordem não é primordial. É sobretudo necessário garantir que as imagens dos centros de pesquisa são mantidas e que estas são efectuadas na mesma altura da vida da estrutura. Assim, foram seleccionadas 1500 imagens (de um universo de 3040) para serem pesquisadas. Esta selecção foi feita de modo aleatório e sem exclusão, ou seja, o conjunto final poderá conter repetições. Para que os resultados díspares sejam minimizados e os valores relevantes no universo, todas as pesquisas foram efectuadas após o conjunto de inserções estar completo, ou seja, quando as 3040 imagens de rosto já se encontravam na RLC.

5.1.3 Limitações externas

Como já foi descrito anteriormente, o serviço do Google App Engine [19] utilizado apresenta algumas limitações, quer inerentes ao próprio sistema, quer por ter sido utilizada a versão gratuita do mesmo. Estas limitações influenciaram negativamente o protótipo e a validação efectuada do mesmo. Abaixo, é apresentada uma listagem dos principais problemas encontrados, acompanhada da respectiva influência na validação e soluções alternativas adoptadas.

- O Google App Engine não suporta todas as bibliotecas Java, nomeadamente algumas relacionadas com a manipulação de imagens. Por não ser possível fazer a computação dos vectores de pesos *online*, houve necessidade de efectuar alterações ao protótipo. Os resultados ficam igualmente menos concordantes com o que seria espectável, dado que o tempo de computação destes não pode ser contabilizado. De modo a minimizar os danos, em nenhum dos testes efectuados, em todas as estruturas, este tempo é contabilizado, permitindo a comparação dos valores com pesos de operação semelhantes.

- Na versão gratuita, o tempo máximo de execução de um pedido é de 30 segundos. Se após os 30 segundos a execução do mesmo não estiver completa, a operação é abortada. Em situações normais, não será espectável que qualquer um dos pedidos exceda o tempo máximo permitido. Porém, o serviço apresenta alguns picos de utilização (que poderão ser visíveis em medições apresentadas nesta secção) e em

algumas operações ("aleatórias") a excepção é levantada. Também nesta versão, existem limitações nos tempos de CPU e no número de operações, entre outros. Por exemplo, a quota diária do número de leituras e escritas possível no *datastore* é bastante reduzida, impossibilitando a validação com uma base de dados de maior dimensão, em tempo útil.

- O *servlet*, ao fim de poucos minutos sem realizar trabalho, é removido de memória. Quando é efectuada uma nova chamada sobre este, é necessário proceder à sua recolocação, sofrendo o pedido algum atraso na execução, bem como os pedidos imediatamente subsequentes, ainda que com menor impacto. Deste modo, o primeiro pedido de cada conjunto de inserções e de pesquisas é ignorado, não sendo contabilizados os seus resultados na estatística final.

5.1.4 Parametrizações

O protótipo apresentado visa ser uma simplificação do objectivo real do sistema: a sua aplicação para suporte de dados provenientes de sensoriamento participado. Com uma amostra de dados substancialmente inferior, as parametrizações utilizadas apresentam maior relevância num contexto comparativo entre si, do que os seus valores efectivos. A pesquisa de valores óptimos não foi, assim, alvo prioritário, pelo que na validação deste protótipo os valores atribuídos aos vários parâmetros (raio do agrupamento, capacidade máxima das folhas e raios de pesquisa) fazem referência aos idealizados em [37]. Nesse trabalho, foi utilizada também a base de dados *Faces94* [2], entre outras, em conjugação com a RLC, embora seja a versão em que o raio dos agrupamentos se mantém constante com o aumentar da profundidade.

Assim, para a criação da RLC, foram utilizados três raios de agrupamento alternativos: 1388, 2776, 5552. Já no tamanho máximo de cada folha, os valores utilizados são diferentes dos apresentados em [37]. Esta diferença deve-se às alterações existentes no método de criação das próprias folhas, em que estas, por economia de espaço, crescem dinamicamente, segundo potências de base 2. De forma a ter uma gama de valores suficientemente distinta que permitisse a existência de diferenças significativas, os valores utilizados são: 4, 16 e 64.

Com os valores de referência acima enunciados, cada versão do sistema testada apresenta nove RLCs diferentes, para as quais será efectuado um estudo comparativo envolvendo diversos aspectos (tempo de execução de inserções, número de leituras e escritas, etc.).

Ao nível das operações de pesquisa, os raios utilizados estão de acordo com os obtidos em [37]. A escolha do raio preferencial respeita as condições impostas de proximidade dos pontos óptimos das taxas de verdadeiros positivos (1) e falsos positivos (0) e de a média do número de resultados da pesquisa ser não superior a 130% da média do número de imagens por indivíduo. O raio seleccionado foi o de 3338, sendo igualmente realizadas pesquisas para raios de 50% (curtas) e 120% (longas), ou seja, 1669 e 4006,

respectivamente.

5.1.5 Metodologia

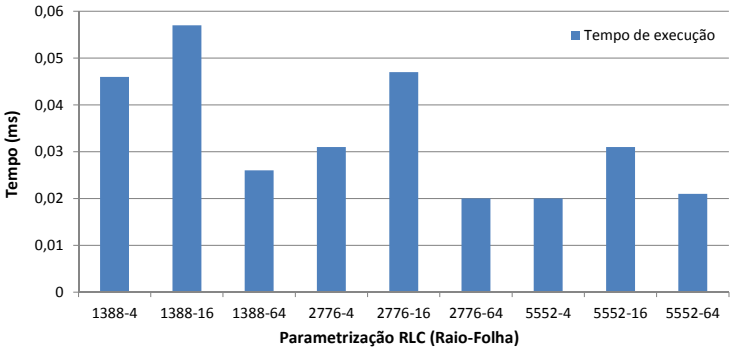
A validação experimental apresentada tem como objectivo analisar os resultados obtidos nos testes efectuados sobre o protótipo implementado. Além deste sistema, constituído pela solução proposta com os respectivos melhoramentos, foi analisada uma versão alternativa, de modo a estabelecer pontos de comparação. O mesmo *benchmark* foi então submetido em duas versões de índice: RLC em memória principal e RLC em memória distribuída.

As métricas obtidas através da submissão das baterias testes são sobretudo de índole espacial e temporal. Ao nível espacial, pretende-se estabelecer uma comparação índice-dados, de modo a inferir acerca do peso da indexação na base de dados utilizada. Além disso, o estudo das dimensões médias de cada objecto será um auxílio na elaboração de estimativas em criações de cenários de utilização do sistema. Ao nível temporal, o estudo centra-se na medição dos tempos médios de execução das operações de inserção e pesquisa. Estas métricas permitem tirar ilações acerca da viabilidade da solução, ao passo que os resultados acerca de algumas operações subatómicas serão úteis na compreensão dos valores e na sugestão de alguns melhoramentos.

5.2 Métrica temporal da operação de inserção

5.2.1 RLC em memória principal

Em memória principal, os nós e pontos da RLC estão ligados entre si por meio de referências em memória, de acordo com o desenho da estrutura. Deste modo, não há contenção em operações de I/O e o processamento define a totalidade do tempo de execução. A Figura 5.2 apresenta o gráfico e a respectiva tabela com os valores da métrica dos tempos médios de execução da operação de inserção, para as várias parametrizações escolhidas.



Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	0,046	0,031	0,020
	16	0,057	0,047	0,031
	64	0,026	0,020	0,021

Figura 5.2: Tabela e gráfico com os resultados dos tempos médios de inserção na RLC em memória principal (sem armazenamento de imagem de rosto)

Como seria espectável, os tempos totais são bastante reduzidos, sendo que a média para cada uma das parametrizações se situa na casa das centésimas do milissegundo.

Porém, estes valores são bastante irregulares, variando entre os 0,020 ms e os 0,057 ms. A elevada diferença entre os valores mínimo e máximo (aproximadamente o triplo) demonstra a elevada influência da parametrização no desempenho na RLC em memória principal.

Nesta estrutura, o maior factor de contenção na operação de inserção centra-se no processamento dentro dos vários nós, nos quais os agrupamentos de cada um são testados sequencialmente. O raio inicial da RLC, apresenta, neste caso, uma influência bastante acentuada. Como é visível na figura, o tempo médio de cada inserção tende a diminuir com o aumento do raio. Este parâmetro tem uma influência especial na raiz (nó RLC inicial). Com raios maiores, existe uma menor dispersão inicial dos dados, contribuindo para raízes com um menor número de agrupamentos e estruturas com maiores níveis de profundidade (para o mesmo tamanho de folha). Como a raiz é o nó inicial, é nesta que ocorre a primeira selecção dos elementos, havendo uma elevada probabilidade de este ser o nó interior com o maior número de agrupamentos. No caso geral, é na raiz que ocorrem grande parte das computações (da medida de distância), daí que seja natural (para a base de dados e parâmetros em estudo) haver a manifestação desta tendência.

A capacidade das folhas (o outro parâmetro da RLC), também se manifesta nos resultados. Para este caso, os melhores resultados (para RLCs com o mesmo raio) tendem a acontecer quando a capacidade é maior. Quando a capacidade é excedida, a folha transforma-se numa RLC, dando origem a novas folhas, mais profundas. Capacidades menores, dão origem a frequentes transbordos, pelo que estrutura fica com maior profundidade. Assim, nas RLCs com grandes capacidades de folha, a profundidade é menor, existindo menos nós a percorrer e distâncias a calcular.

A parametrização desta estrutura deve ser cuidada, mas levar as tendências manifestadas na operação de inserção ao limite, poderá não ser ideal. Um raio demasiado grande provoca uma fraca distribuição inicial, algo que só será mitigado com o aprofundar dos nós da RLC, dando origem a recursividade desnecessária. Por outro lado, capacidades de folha muito elevadas diminuem a acção a RLC. A RLC tem pouca profundidade e os dados tendem a acumular-se nas mesmas folhas, aumentando a tendência de esta se assemelhar a um vector ordenado.

5.2.2 RLC em memória distribuída

A versão testada nesta secção é a solução final do sistema implementado. Tanto nós como registos encontram-se "desligados", estando armazenados em memória distribuída na *cloud*. Ao contrário do que é sugerido no capítulo 3, nós e pontos foram limitados ao armazenamento na mesma *cloud*, o que levou à diminuição da latência da leitura de pontos. Num sistema real, a leitura destes seria bastante mais cara, devido à sua localização remota.

Genericamente, poder-se-à dividir o tempo de execução em três áreas distintas: comunicação, processamento efectivo e I/O. Como é observável no gráfico da Figura 5.3,

para a operação de inserção, o tempo gasto na comunicação é superior ao despendido nas operações de leitura e escrita. O tempo de processamento, embora representado em cada uma das colunas, não é visível, devido ao seu valor muito reduzido. O tempo médio de processamento das nove parametrizações é de 0,358 ms, sendo que comunicação e I/O consomem 322,534 ms e 197,644 ms, respectivamente. No total, os tempos de execução nas nove parametrizações variam entre os 458,442 ms e os 584,137 ms (visível com maior detalhe no Apêndice 7.1).

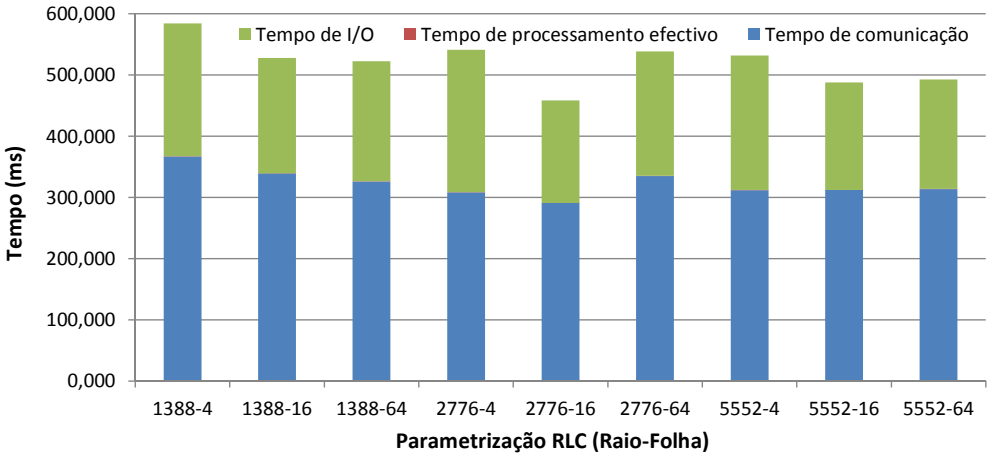
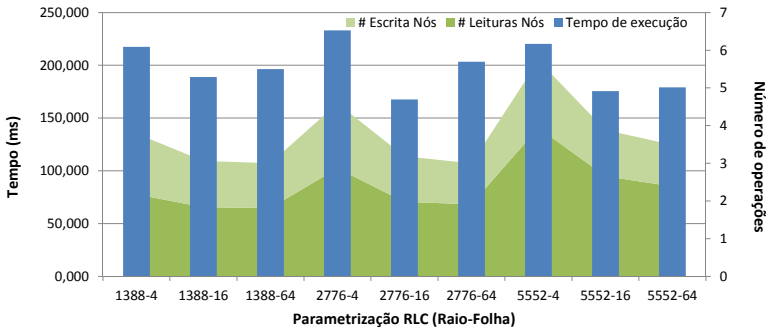


Figura 5.3: Gráfico com os resultados dos tempos médios de inserção, parcelados, na RLC em memória distribuída

Os tempos médios de execução no GAE [19] (Figura 5.4), embora não sejam tão discrepantes proporcionalmente como ocorria na RLC em memória principal, permitem verificar que, para cada raio, os tempos mais elevados verificam-se quando a capacidade das folhas é mais reduzida, ao mesmo tempo que os melhores ocorrem com folhas de capacidade 16.



Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	217,402	233,171	220,333
	16	188,850	167,617	175,618
	64	196,478	203,438	179,111

Figura 5.4: Tabela e gráfico com os resultados dos tempos médios de execução no GAE durante uma inserção na RLC em memória distribuída (tempo de total exceptuando o tempo de comunicação)

As folhas que só têm capacidade para 4 elementos, tendem a encher com relativa frequência, originando sucessivas transformações em novas RLCs. Além do número de

escritas ser superior, este facto contribui ainda para o aumento da profundidade. RLCs mais profundas, exigem ainda, em média, um maior número de leituras de nós para cada operação. Embora o número de leituras e escritas seja menor para RLCs com folhas de capacidade 64, as diferenças para as RLC r-16 são reduzidas, não compensando o tempo perdido na leitura de nós de maior dimensão.

No que diz respeito à variação do raio, não é possível observar um padrão claro. Se, por um lado, o número de leituras aumenta de acordo com o raio, por outro, o número de escritas (que consome mais tempo) não demonstra nenhuma tendência, acabando por esconder a primeira. De referir que, ao contrário do que acontece na versão em memória principal, o número de elementos nos nós interiores não é actualizado, pelo que não é necessário escrever toda a hierarquia a cada inserção. Assim, somente os nós criados ou cuja estrutura foi alterada são persistidos.

As operações sobre os nós (leitura e escrita) não as únicas operações de I/O efectuadas durante uma inserção na RLC. O tempo despendido nestas representa uma significativa, mas minoritária, percentagem do total, cerca de 25%. O restante não é muito variável, em condições normais, para qualquer operação de inserção. Entre elas poderemos destacar: as escritas da imagem de rosto e do ponto que contem os dados, além das leituras e escritas de variáveis de controlo. A conexão ao *datastore* está também incluída nesta estatística, no entanto, em média, esta rondará 1 milissegundo e pode manter-se aberta durante todo o pedido.

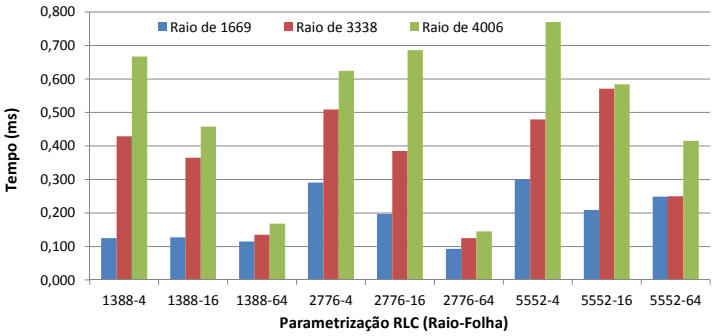
Estabelecendo uma comparação com os resultados obtidos em [34], é possível concluir que existem algumas semelhanças. O número de leituras de nós, tende a ser crescente com o aumento do raio inicial da RLC, ainda que, no trabalho anterior, para raios demasiados curtos, a tendência se inverta. Do mesmo modo, o número de escritas é superior para raios de maior dimensão. Importa, no entanto, referir que em [34] é guardado o número de elementos em cada nó RLC, pelo que, a cada inserção, toda a hierarquia é rescrita. Nesta dissertação somente os nós cuja estrutura é modificada são persistidos, minorando as diferenças. Já no caso da capacidade das folhas, as conclusões são mais difíceis de obter. As dimensões utilizadas apresentam variações nas ordens de grandeza, pelo que somente alguns resultados podem ser comparados. Para estes, em ambos os trabalhos, as diferenças são bastante reduzidas, não sendo possível marcar claramente uma tendência.

5.3 Métrica temporal da operação de pesquisa

5.3.1 RLC em memória principal

A eficiência das pesquisas por proximidade é um dos grandes objectivos da RLC. Como tal, os valores das métricas dos tempos médios de pesquisa são bastante reduzidos, situando-se abaixo dos milissegundos. Os tempos obtidos nas operações de pesquisa

para três raios diferentes, nas várias parametrizações da RLC, estão representados na Figura 5.5. Neste teste, além do tempo de pesquisa, é contabilizado o tempo de criação da página *web* com os resultados, à semelhança do que sucede na versão distribuída na *cloud*.



Tempo (ms)		Folha	Raio de pesquisa		
			1669	3338	4006
Raio	1388	4	0,125	0,429	0,667
		16	0,127	0,365	0,458
		64	0,115	0,135	0,168
	2776	4	0,291	0,509	0,624
		16	0,197	0,385	0,686
		64	0,093	0,125	0,145
	5552	4	0,301	0,479	0,770
		16	0,209	0,571	0,584
		64	0,249	0,250	0,415

Figura 5.5: Tabela e gráfico com os resultados dos tempos médios de pesquisa na RLC em memória principal

À semelhança do que sucede na operação de inserção, a parametrização da RLC tem um impacto muito significativo no tempo de execução das operações de pesquisa. Um exemplo bem ilustrativo é a execução da operação com o maior raio de pesquisa (4006) em que os valores variam entre 0,145 ms e 0,770 ms (o valor máximo representa mais do quíntuplo do valor mínimo).

De um modo geral, RLCs com raios iniciais mais reduzidos apresentam melhores resultados, ao contrário do que sucedia na operação de inserção. Embora a raiz tenha um número muito elevado de agrupamentos, estes têm uma área mais reduzida, permitindo excluir um grande número de dados que se encontram em agrupamentos mais distantes. O elevado número de exclusões de agrupamentos feito na raiz diminui significativamente o número de chamadas recursivas e, igualmente, o cálculo de distâncias.

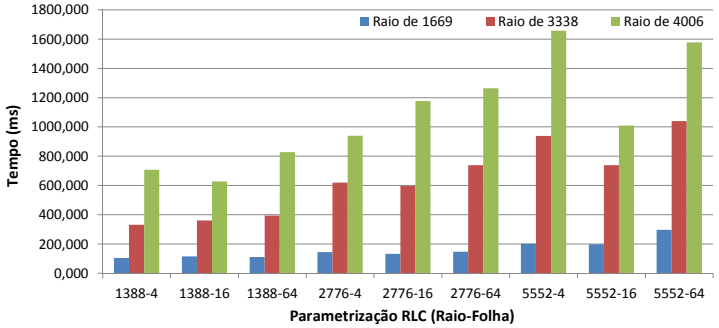
Para cada raio, os melhores resultados ocorrem nas RLCs com capacidade de folha mais elevada. A profundidade não é tão acentuada, pelo que o número de nós internos é reduzido, não havendo necessidade de testar tantos agrupamentos, com todos os cálculos de distâncias que os mesmos implicam.

Ainda que os melhores resultados tenham sido obtidos na RLC 2776-64 (mais equilibrada), a 1388-64 apresenta valores semelhantes, o que comprova a tendência verificada.

5.3.2 RLC em memória distribuída

Na RLC em memória distribuída, os tempos médios de cada operação de pesquisa tendem a aumentar com o crescimento do raio inicial dos agrupamentos. Os tempos mais reduzidos ocorrem nas RLCs de raio 1388, enquanto os mais elevados estão presentes nas de raio 5552, como é visível na Figura 5.6. No entanto, existem algumas excepções, das quais a mais notória ocorre na RLC 5552-4 com uma pesquisa de raio 4006, devido a picos do sistema que provocaram um tempo médio excessivo em cada leitura (cerca de 1

milissegundo acima no caso dos nós).



Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	104,634	331,163	708,123
		16	115,590	361,182	627,804
		64	111,760	393,974	828,454
	2776	4	144,360	619,554	940,535
		16	133,458	597,311	1177,736
		64	147,899	738,142	1264,538
	5552	4	202,218	939,073	1657,392
		16	197,759	738,352	1009,118
		64	296,524	1039,572	1577,083

Figura 5.6: Tabela e gráfico com os resultados dos tempos médios de execução no GAE durante uma pesquisa na RLC em memória distribuída

Igualmente verificável é o facto de, na maioria das vezes, para cada raio, os melhores tempos ocorrerem nas estruturas mais equilibradas, com um número máximo de 16 elementos por folha, ao contrário das estruturas com parametrização r-64, que apresentam normalmente os piores.

Estas tendências são derivadas do número de operações I/O necessárias, como demonstra o gráfico da Figura 5.7.

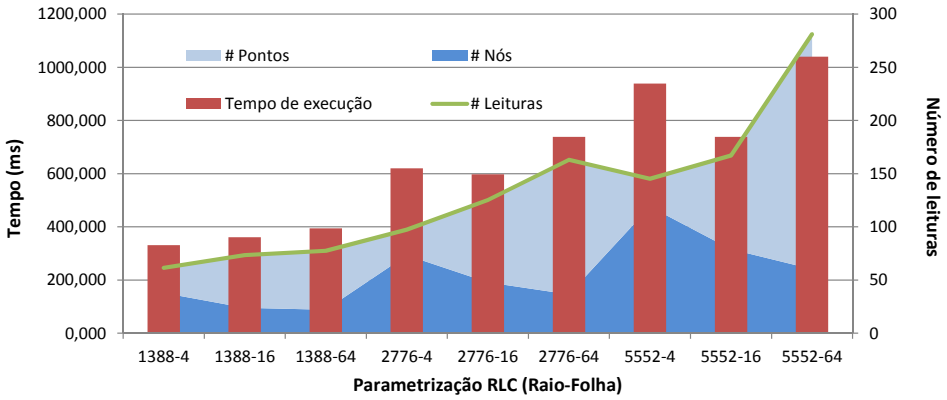


Figura 5.7: Gráfico com os resultados dos tempos médios de execução no GAE (no eixo principal) e do número de leituras (no eixo secundário), durante uma pesquisa com raio 3338 na RLC em memória distribuída

O aumento do raio provoca um maior do número de nós e pontos lidos. O primeiro facto deriva da maior distribuição de valores na raiz para raios reduzidos, o que diminui a profundidade da estrutura. Pelo mesmo motivo são lidos menos pontos, já que os presentes em cada agrupamento são somente os mais próximos e não há tantas leituras desperdiçadas. Em relação à variação da capacidade das folhas, o comportamento é divergente. O número de nós diminui com o crescimento da capacidade (folhas maiores originam menos expansões e profundidade) ao passo que o número de pontos lidos aumenta (com o aumento da profundidade a distribuição dos dados é maior). Sendo o

número de pontos lidos tipicamente superior ao número de nós lidos, a tendência do número total de leituras segue aproximadamente a do primeiro, como pode ser visível em maior detalhe no Apêndice (na secção 7.1.2.1).

Em relação ao tempo despendido em cada acção, é possível concluir que com o aumento do raio de pesquisa, o tempo de comunicação (em média, 274,433 ms) vai perdendo peso no total. Com um raio de pesquisa reduzido, o peso da comunicação é bastante elevado (cerca de 63%, em média), ao passo que com um raio de 4006, o peso desce para 22%, passando a grande parcela a ser ocupada pelo tempo de I/O, visível nas secções verdes de cada coluna (Figuras 5.8 e 5.9). À semelhança do que foi verificado anteriormente, o tempo de computação efectiva não é visível devido à sua insignificância no tempo total. Para um raio curto a média é de 0,352 ms e para um raio longo é de 1,337 ms, o que ajuda a demonstrar o elevado poder de computação da *cloud*.

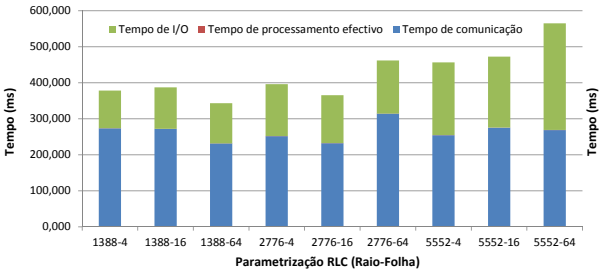


Figura 5.8: Gráfico com os resultados dos tempos médios de uma pesquisa, parcelados, com raio 1669 na RLC em memória distribuída

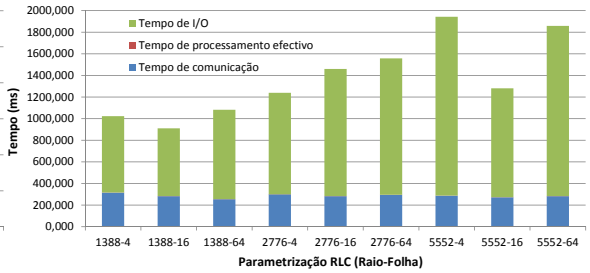


Figura 5.9: Gráfico com os resultados dos tempos médios de uma pesquisa, parcelados, com raio 4006 na RLC em memória distribuída

Em comparação com [34], o padrão é mantido no que diz respeito à dimensão do raio. Porém, a variação na capacidade das folhas (nas dimensões semelhantes) traz efeitos contrários, devido à deslocalização dos pontos imposta nesta dissertação.

5.3.3 RLC em memória distribuída (*shadow buckets*)

Os resultados apresentados nesta secção têm como objectivo validar experimentalmente a versão melhorada da distribuição da RLC, através da criação de *shadow buckets*. Nesta solução, os *shadow buckets* encontram-se armazenados na *cloud* e são criados de forma percentual. Não sendo ainda a solução ideal para a maior parte das aplicações, permite fazer uma análise das vantagens/desvantagens da mesma, bem como sugerir algumas direcções para o futuro.

Na validação experimental seguinte foram efectuados testes com 3 probabilidades de criação diferentes: 33%, 66% e 100%. A versão completa da análise pode ser consultada no Apêndice, nas secções 7.1.2.2, 7.1.2.3 e 7.1.2.4.

5.3.3.1 33% *shadow buckets*

No gráfico da Figura 5.10 está presente a comparação dos tempos médios de execução da operação de pesquisa na RLC contendo, aproximadamente, 33% de *shadow buckets* com a versão padrão. À semelhança do que sucede com a versão padrão, o tempo de execução aumenta de acordo com o raio inicial da estrutura e a capacidade de folha crescentes.

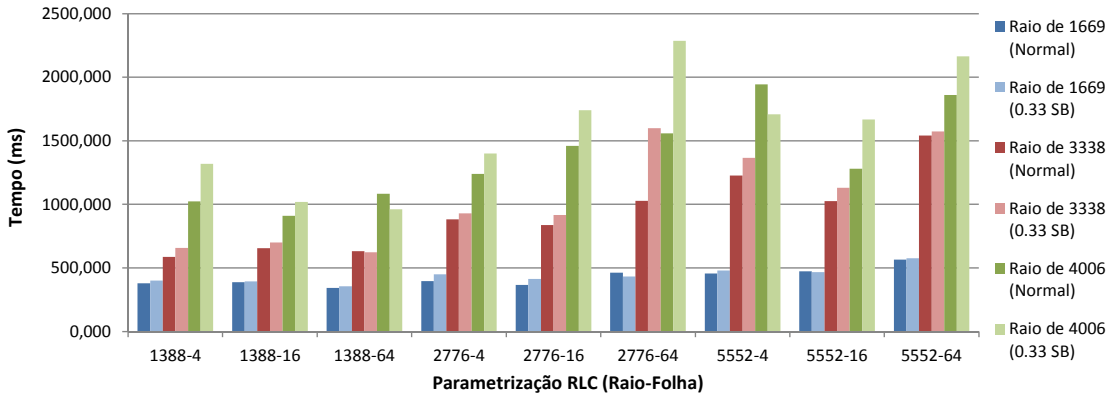


Figura 5.10: Gráfico com os tempos médios de execução de pesquisas na RLC em memória distribuída com e sem 33% de *shadow buckets*

No geral, há uma subida dos tempos de execução, sendo esta ligeiramente mais acentuada quando a capacidade da folha é a menor. Devido à fraca coerência de desempenho verificada nos tempos de leitura dos objectos (aos quais a aplicação é alheia), padrões mais claros não são possíveis de verificar.

As diferenças observadas em relação à versão padrão são justificadas pela alteração do número de operações I/O requeridas. A Tabela 5.1 apresenta o número médio de leituras para cada operação de pesquisa, com os três raios em teste e nas nove parametrizações escolhidas para a RLC. Na célula à direita de cada valor está presente a diferença percentual de leituras em relação à versão padrão da RLC em memória distribuída.

Tabela 5.1: Tabela com os resultados e análise do número de leituras efectuadas durante uma pesquisa na RLC em memória distribuída com 33% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	18,817	▲6%	68,628	▲12%	150,593	▲15%
		16	17,133	▼7%	75,659	▲3%	158,194	▲3%
		64	15,863	▼14%	73,029	▼6%	151,134	▼6%
	2776	4	23,631	▲7%	111,231	▲14%	212,292	▲15%
		16	22,145	▼9%	130,128	▲4%	235,111	▲3%
		64	24,686	▼18%	137,779	▼15%	237,410	▼17%
	5552	4	37,298	▲12%	178,070	▲23%	305,802	▲22%
		16	38,282	▼1%	171,238	▲3%	276,386	▼2%
		64	59,522	▼20%	242,461	▼14%	365,582	▼15%

Embora haja uma acentuada descida do número de pontos lidos (devido aos *shadow*

buckets utilizados), na maioria dos casos, sobretudo para capacidades de folha mais reduzidas, o número total de leituras é superior (no pior dos casos em 23%). Isto deve-se ao facto de a redução nas leituras de pontos não compensar o aumento verificado nas operações com nós. De facto, o número de leituras efectivas mantém-se, o problema surge no número de leituras falhadas, também elas contabilizadas. Quando a computação chega ao último nó interior (em profundidade) e é necessário fazer uma leitura descendente, este tenta sempre ler o *shadow bucket*, por não possuir qualquer informação acerca da sua (in)existência. Em caso de falha, procede à leitura da folha ligada à estrutura, que possui os dados deslocalizados.

Assim, o decréscimo no número total de leituras só é especialmente verificável para os casos em que a folha da RLC tem capacidade máxima (em teste), em que as leituras dos pontos assumem um papel preponderante nos números finais.

5.3.3.2 66% *shadow buckets*

No gráfico da Figura 5.11 está presente a comparação dos tempos médios de execução da operação de pesquisa na RLC contendo, aproximadamente 66% *shadow buckets* com a versão padrão. Os tempos de execução com a presença dos *shadow buckets* são mais baixos quase na totalidade dos testes, em comparação com os correspondentes na versão padrão.

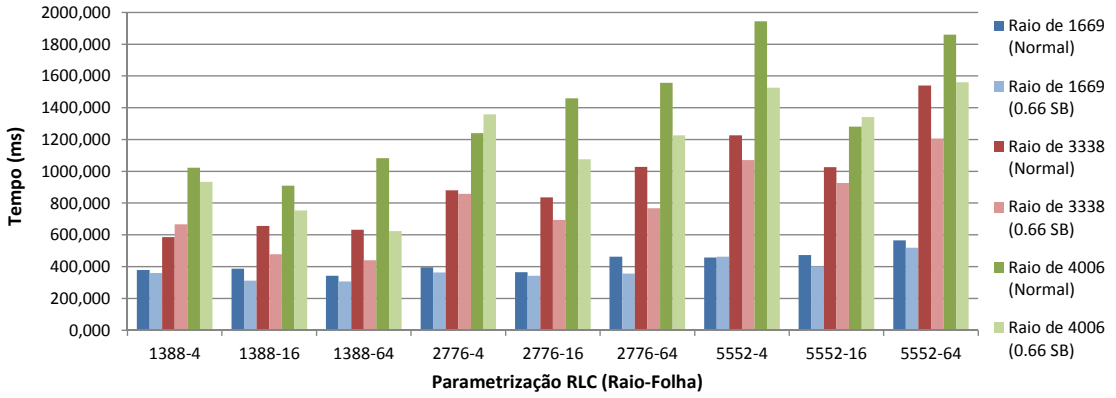


Figura 5.11: Gráfico com os tempos médios de execução de pesquisas na RLC em memória distribuída com e sem 66% de *shadow buckets*

É possível observar algumas modificações nas tendências dos resultados. Embora os tempos de execução continuem a aumentar de acordo com o raio (não são expectáveis alterações nesta dado que os *shadow buckets* não influenciam a profundidade), deu-se uma aproximação dos valores quando este se mantém, para capacidades de folha diferentes. Esta aproximação é consequência da redução do número de leituras de pontos, proporcional à capacidade das folhas. Nas RLCs com raio igual a 1388 as diminuições em termos absolutos foram suficientes para se verificar uma inversão do padrão.

Na Tabela 5.2 podemos comprovar que todos os valores desceram em relação à versão

testada anteriormente. Por um lado, houve uma quebra natural na leitura de pontos, por outro, a percentagem superior de *shadow buckets* impediu que houvessem tantas leituras falhadas, contribuindo para a redução do número de nós lidos.

Tabela 5.2: Tabela com os resultados e análise do número de leituras efectuadas durante uma pesquisa na RLC em memória distribuída com 66% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	15,984	▼10%	61,451	▼0,1%	133,164	▲1%
		16	11,389	▼38%	48,574	▼34%	101,990	▼34%
		64	10,285	▼45%	49,071	▼37%	100,681	▼38%
	2776	4	19,872	▼10%	97,287	▲0,1%	182,587	▼1%
		16	15,154	▼38%	86,771	▼31%	152,165	▼34%
		64	17,441	▼42%	95,241	▼42%	157,688	▼45%
	5552	4	32,432	▼2%	157,172	▲8%	264,831	▲5%
		16	30,580	▼21%	132,274	▼21%	208,256	▼26%
		64	51,354	▼31%	189,012	▼33%	276,947	▼36%

Porém, em alguns casos, ainda não é vantajosa a existência deste melhoramento. Ainda assim, as diferenças não são significativas, sendo que todos os valores são inferiores a 10%, acima do número de leituras na versão padrão. Todos eles se verificam em RLCs cuja parametrização inclui uma capacidade de folha reduzida, onde a leitura total de pontos é mais diminuta.

5.3.3.3 100% *shadow buckets*

O gráfico da Figura 5.12 apresenta a comparação dos tempos de execução das várias operações de pesquisa na versão padrão e na versão com 100% de *shadow buckets* criados. Na versão testada nesta subsecção os tempos são sobejamente inferiores, sendo esta diferença especialmente visível quando as folhas têm a maior capacidade.

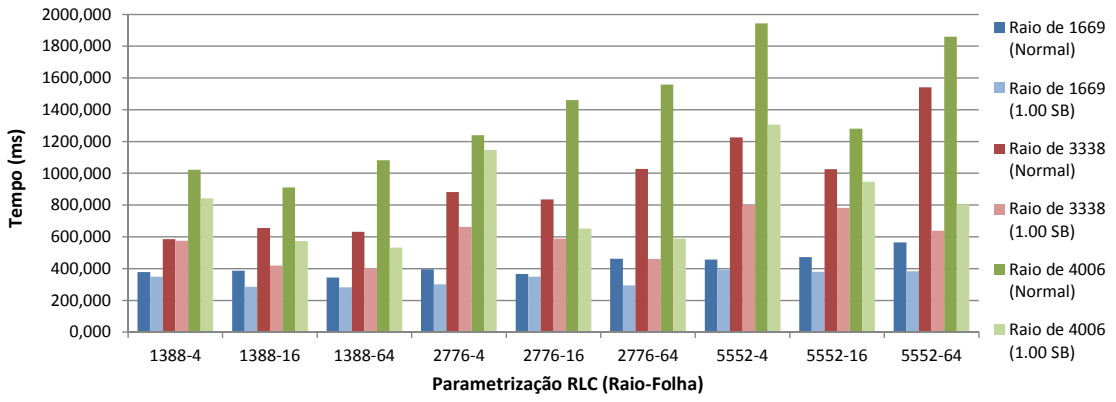


Figura 5.12: Gráfico com os tempos médios de execução de pesquisas na RLC em memória distribuída com e sem 100% de *shadow buckets*

Neste caso, é possível verificar uma clara alteração de tendência. Os tempos de execução, para cada raio inicial da RLC, são decrescentes com o aumento da capacidade das folhas, sendo que em 8 das 9 ocasiões (pares <raio RLC, raio pesquisa>) o menor valor se encontra quando a folha pode conter até 64 elementos. Esta alteração é devida à anulação do número de leituras de pontos, bem como à redução do número de leituras de nós. Embora os *shadow buckets* existam na totalidade, estes só são criados quando um novo elemento é inserido na folha, o que provoca leituras falhadas de folhas vazias.

Tabela 5.3: Tabela com os resultados e análise do número de leituras efectuadas durante uma pesquisa na RLC em memória distribuída com 100% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	11,860	▼33%	45,911	▼25%	98,600	▼25%
		16	5,650	▼69%	28,841	▼61%	58,402	▼62%
		64	4,879	▼74%	26,670	▼66%	53,616	▼67%
	2776	4	16,113	▼27%	82,691	▼15%	151,175	▼18%
		16	8,671	▼64%	50,943	▼59%	83,170	▼64%
		64	6,514	▼78%	38,759	▼76%	60,444	▼79%
	5552	4	28,222	▼15%	136,648	▼6%	225,240	▼10%
		16	20,019	▼48%	89,668	▼46%	134,221	▼52%
		64	16,719	▼77%	68,115	▼76%	97,972	▼77%

No total, o número de leituras é claramente inferior ao do teste anterior, como demonstra a Tabela 5.3. Como é possível observar, o menor decréscimo da percentagem de leituras (em relação à versão padrão) ocorre na RLC 5552-4, para um raio de pesquisa de 3338 mas, em alguns casos, a redução de leituras ronda os 80%. Esta redução (em percentagem) aumenta com o crescimento do tamanho das folhas, apresentando os melhores valores quando esta tem capacidade para 64 elementos. Em todas as situações o número de leituras é menor nesta versão com totalidade de *shadow buckets* do que na versão padrão, sendo bastante compensatória a criação de todos os *shadow buckets*.

Esta última versão é bastante semelhante, estruturalmente, à produzida em [34]. Ao contrário do que sucedia na versão padrão, neste caso, as tendências vão de encontro às verificadas no trabalho anterior. Assim, há um aumento do número de leituras com uma dimensão de raio crescente, ao passo que as mesmas diminuem com a variação, no mesmo sentido, da capacidade das folhas.

5.4 Métrica espacial da RLC em memória distribuída

Nesta secção é feita uma análise espacial do sistema, na sua forma original, sem *shadow buckets*. São analisadas as diversas entidades (objectos) guardadas na *cloud*, bem como a sua cardinalidade e o espaço que ocupam, nas várias parametrizações em teste.

O gráfico circular da Figura 5.13 apresenta as proporções médias de armazenamento de entidades nas diversas parametrizações em estudo. Este está dividido em 4 partes:

Imagens, Pontos, Nós e Outros. Cada parte representa uma ou mais entidades diferentes:

- Imagens: imagens de rosto inseridas no sistema e armazenadas na *cloud*;
- Pontos: objectos com os dados pertencentes às imagens inseridas no sistema, como os *eigenvalues* e os vectores de pesos;
- Nós: objectos com os nós da RLC, tanto interiores (RLCs) como folhas;
- Outros: informação de controlo da estrutura, como sequenciadores e lista de nós a remover, por exemplo.

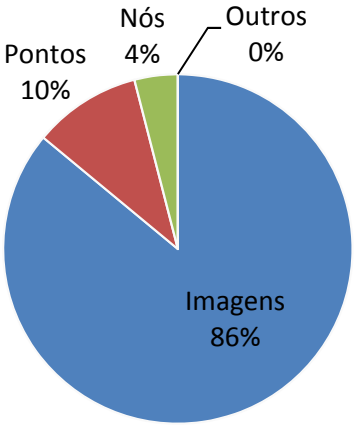


Figura 5.13: Gráfico circular com a proporção média do espaço ocupado pelas entidades

A observação cuidada do gráfico permite concluir que, no âmbito do protótipo implementado, a grande maioria (86%) da informação armazenada é referente aos dados submetidos ao mesmo: as imagens. Estes dados não trazem um peso muito acrescido às operações do sistema, nomeadamente às pesquisas, em que as mesmas nunca são lidas, servindo essencialmente para funções de suporte, como a visualização das imagens pertencentes ao conjunto-resultado. Um dos grandes factores de contenção da RLC em memória distribuída é o acesso aos pontos, que contêm a informação compactada da imagem correspondente. A pertença de 10% do espaço total ocupado dá conta do custo elevado desta entidade. Porém, se as imagens tiverem um tamanho regular, a proporção entre imagens e pontos nunca se irá diferenciar, dado que possuem uma relação de um para um entre si.

Os nós da estrutura apresentam um valor mais variável, como pode ser verificado no estudo efectuado na secção 7.2 do Apêndice. O espaço ocupado pelo conjunto destes está intimamente relacionado com a capacidade das folhas da RLC, mas igualmente com o raio inicial dos agrupamentos da mesma. Com o crescimento da capacidade das folhas, o espaço ocupado pelos nós diminui, comprovando que o aumento do tamanho das folhas é compensado, espacialmente, pela redução da cardinalidade dos nós. A diferença ocorrida entre extremos é definida pelo tamanho do raio inicial. Com um raio

inicial menor, a maior diferença apresenta o valor mais baixo, dado que a elevada distribuição inicial impede uma grande profundidade, implicando um menor número total de nós. Pelo mesmo motivo, as folhas de maior dimensão não verão a sua capacidade preenchida, sendo desperdiçado algum espaço pelas posições vazias nestas. Já RLCs com parametrizações de raio mais largo possuem uma fraca distribuição inicial que, associada a capacidades de folha reduzidas, obriga à criação recursiva de nós. Porém, quando a folha pode atingir maiores dimensões, a ocupação total de cada é bastante mais elevada, apresentando maior economia de espaço.

A bateria de testes efectuada, para as várias parametrizações da RLC utilizadas, deu origem, em média, ao armazenamento de um total de 21 MBytes de dados. Este total está dividido na proporção capturada anteriormente e as entidades mais significativas aparecem detalhadas na Tabela 5.4. Esta confirma algumas observações verificadas anteriormente. Por um lado, é comprovada a relação de um para um entre folhas e pontos, ao mesmo tempo que se dá conta de que cada imagem possui um tamanho cerca de 9 vezes superior ao de um ponto. Por outro, conclui-se que o número de nós é necessariamente mais reduzido que os outros dois (844 contra 3040 de cada) que, auxiliado pelo seu baixo tamanho médio (distante das imagens e não tanto dos pontos), contribui para uma pequena percentagem do espaço total ocupado pela própria estrutura.

Tabela 5.4: Tabela com a cardinalidade e dimensão média das entidades presentes no sistema

Entidades (objectos)	Cardinalidade média	Tamanho médio (por objecto)
Imagens	3040	6 kBytes
Pontos	3040	668 Bytes
Nós	844	1 kByte

5.5 Reflexão

Perante os resultados obtidos nos testes demonstrados neste capítulo, é possível inferir algumas conclusões acerca de cada uma das estruturas.

Na RLC em memória principal, as parametrizações são determinantes para o desempenho das operações, na medida em que influenciam de forma notória os tempos de computação. Porém, inserções e pesquisas apresentam tendências diferentes, pelo que não existe uma concordância de valores. Tendo em conta o âmbito de utilização da estrutura, será preferível optimizar os parâmetros para as operações de pesquisa, priorizando o desempenho destas. O seu objectivo é oferecer uma consulta eficiente a dados passíveis de diferenciar através de uma função de distância métrica, pelo que será esta a operação predominante.

A RLC em memória secundária distribuída apresenta outro tipo de preocupações.

Com tempos de computação igualmente baixos, o desempenho das operações vai ser determinado sobretudo pelas operações de leitura/escrita. Também neste caso, a escolha dos parâmetros da estrutura é determinante no nível de eficiência da mesma, ainda que as diferenças sejam mais notórias na operações de pesquisa.

Os melhores resultados tendem a ocorrer quando o raio inicial da RLC é reduzido e as folhas apresentam capacidades limitadas. Raios mais pequenos descriminam melhor os dados, proporcionando a sua melhor distribuição. Do mesmo modo, folhas de capacidade reduzida aumentam o número de nós, mas produzem menos pontos candidatos, resultando em menos leituras desperdiçadas. Dado o custo de acesso à *cloud* local tipicamente ser mais baixo que o acesso remoto aos pontos (não implementado no protótipo), existem fortes argumentos para usar este tipo de parametrização. Além disso, como há menos nós que pontos no sistema, é de esperar que um política de *caching* orientada aos nós seja mais eficaz para diminuir o número de leituras. Esta última é ainda adjuvada pelo padrão existente num conjunto de operações sobre a RLC, em que nós menos profundos são lidos recorrentemente.

Num cenário de sensoriamento participado, com um grande número de dados, a cardinalidade dos nós do índice é substancialmente mais elevada. Assim, a probabilidade de utilização de um nó em *cache* diminui, diminuindo igualmente o *hit ratio*. Através da redução do número de nós, que pode ser obtida com aumento da capacidade das folhas, este problema é minorado. Porém, provoca uma grande leitura de pontos, que não farão parte do resultado. Assim, esta alteração deve ser acompanhada pela inclusão dos dados nas folhas, de modo a condensar o número de leituras. Este novo tipo de objectos, os *shadow buckets*, podem revelar-se um grande auxílio para o bom desempenho da RLC, porém, a implementação actual, desenvolvida nesta dissertação, não é totalmente adequada. A estrutura não tem conhecimento acerca de quais os *shadow buckets* existentes, traduzindo-se em penalizações de tempo quando não é possível efectuar a leitura de um determinado objecto. De forma a mitigar o problema, futuras implementações poderiam incluir uma estrutura auxiliar (um *bitmap*, por exemplo), que indique quais os *shadow buckets* existentes. Este oráculo deverá ser passível de colocar em *cache*, com o intuito de permitir consultas rápidas.

Em suma, os resultados obtidos demonstram a viabilidade da solução desenvolvida. A RLC em memória distribuída na *cloud* permite a execução de pesquisas por proximidade eficientes, desde que parametrizada com os pressupostos descritos. O estudo de políticas de *caching* e da criação de *shadow buckets* poderá oferecer informação que permita melhorar o funcionamento da estrutura neste ambiente.

6

Conclusões

6.1 Conclusão

Motivado pelo surgimento de alguns paradigmas aplicacionais que produzem um grande número de dados, como o sensoriamento participado, este trabalho teve como objectivo tornar persistentes esses mesmos conteúdos em memória distribuída. Para tal, foi escolhida uma abordagem que faz uso das potencialidades da *cloud*, uma solução que surge como alternativa aos modelos de armazenamento centralizados e distribuídos, promovendo uma alternativa híbrida que oferece elevada capacidade, disponibilidade e fiabilidade. Devido à necessidade de execução de pesquisas com características especiais, neste tipo de paradigmas, foi criado um índice para pesquisas de dados métricos, efectuado sobre uma estrutura do mesmo tipo, a Recursive Lists of Clusters [1, 34] (RLC).

Para a criação deste sistema, foram efectuadas as seguintes tarefas:

- Pesquisa e descrição dos diversos tipos de serviços existentes na *cloud*, com especial foco nas características de funcionamento, modelos de armazenamento e custos;
- Definição da arquitectura do sistema, com a respectiva separação dos 3 componentes lógicos: utilizador, índice (RLC) a executar em serviço de alojamento de aplicações e persistência de dados em serviço de armazenamento de dados em bruto;
- Modelação da RLC em memória distribuída. Baseada na RLC (com diminuição de raio com o aumento da profundidade) em memória principal, foi criada uma versão da mesma para ser suportada em memória distribuída e, assim, executar na *cloud*. Esta solução teve como uma das grandes motivações a possibilidade de utilização de repositórios heterogéneos, de modo a evitar a dependência de um

único provedor;

- Desenvolvimento de um protótipo assente no serviço Google App Engine [19] (GAE) e que teve como âmbito a criação de um portal para inserção e pesquisa métrica de imagens de rosto.

O protótipo realizado focou-se na adaptação da estrutura de dados métrica RLC para operar sobre memória secundária distribuída, modelada para um armazenamento baseado em chave-valor. Devido a limitações de tempo, é utilizada uma única *cloud*, ficando o índice e os dados alojados conjuntamente. Apesar desta limitação, a experiência obtida com a concretização bem sucedida deste protótipo oferece a confiança de que o modelo de arquitectura proposto é realizável e adequado, face aos objectivos de persistência de dados provenientes de sensoriamento participado. A separação em termos lógicos do índice e dos dados facilita ainda o armazenamento de grandes quantidades de dados de uma forma compatível com uma gestão de custos racional e sem ficar dependente de um provedor em particular.

Com o objectivo de avaliar o desempenho do sistema, foram realizadas as seguintes actividades:

- Utilização da RLC em memória principal para a elaboração de estudos comparativos;

- Escolha da base de dados a utilizar. Foi escolhida a base de dados *Faces94* [2] por conter um número razoável de dados e existirem alguns trabalhos na área que a utilizam [37];

- Definição de um *benchmark* que contivesse inserções e pesquisas, a submeter às várias estruturas em teste;

- Escolha das parametrizações da RLC. Com recurso à pesquisa bibliográfica, foram definidos 3 raios iniciais e 3 capacidades de folha, permitindo o teste com um total de 9 parametrizações diferentes;

- Validação experimental do sistema implementado. Foram efectuados testes com operações de inserção e pesquisa (3 raios diferentes) sobre as 9 parametrizações de várias versões da RLC. As versões da RLC testadas foram: RLC em memória principal, RLC em memória distribuída e RLC em memória distribuída com recurso a *shadow buckets*.

A análise de resultados permitiu concluir que a RLC em memória distribuída possui um desempenho satisfatório, sendo uma solução promissora para certo tipo de aplicações. O tempo de execução de cada operação depende sobretudo das operações I/O envolvidas, sendo necessário encontrar uma parametrização adequada, para a RLC, que

as minore. Tal como a função de distância, a parametrização deve ser definida pelo administrador e deverá apresentar, preferencialmente, um raio inicial reduzido e uma capacidade limitada para as folhas. Estes produzem uma distribuição mais eficiente dos dados, sem exigir um grande peso à leitura de pontos nas operações de pesquisa (muito lenta em *clouds* remotas).

O melhoramento da eficiência do índice poderá ser obtido através da implementação de um sistema de *caching*. Uma política orientada aos nós é mais eficaz na redução do número de leituras, ao mesmo tempo que apresenta padrões claros de acesso (nós menos profundos são lidos frequentemente). Porém, num contexto de sensoriamento participado, o aumento no número de nós provoca quebras no *hit ratio*, havendo necessidade de aumentar a capacidade das folhas para diminuir a profundidade. Neste cenário, a criação de *shadow buckets*, como forma de condensar as leituras dos pontos, apresenta-se bastante favorável, ainda que a implementação actual apresente algumas limitações.

6.2 Contribuições

Ao nível das contribuições deste trabalho, são de destacar:

- Pesquisa bibliográfica acerca dos serviços existentes actualmente na *cloud* e a sua classificação em termos de serviços e custos;
- Criação de uma arquitectura para suporte de aplicações que envolvam a persistência de um grande número de dados com padrões de pesquisa por semelhança ou proximidade;
- Desenvolvimento de uma aplicação estudo de caso envolvendo a criação de um portal *web* para inserção e pesquisa de imagens de rosto;
- Validação métrica temporal e espacial que visa uma percepção detalhada de todos os custos envolvidos nas operações permitidas sobre a RLC.

Em relação à RLC original (em memória principal), a grande diferença ao nível de funcionalidade é a ausência da operação de remoção. O objectivo da solução implementada é o armazenamento de um grande volume de dados, provenientes de sensores, por exemplo, pelo que a manutenção de valores históricos contribui para a obtenção de resultados mais fiéis, e a sua remoção individual não se apresenta essencial.

Finalmente, é possível concluir que o trabalho efectuado apresenta uma boa solução de futuro, sendo uma área que ainda tem muito por explorar. O tempo disponível para a realização da dissertação não permitiu um estudo mais aprofundado, pelo que trabalhos posteriores poderão apresentar resultados mais elaborados e soluções alternativas que resolvam lacunas identificadas.

6.3 Trabalho futuro

O tempo disponível para o desenvolvimento do trabalho não foi suficiente para a implementação de algumas funcionalidades e características desejadas, nem tão pouco para a elaboração de todos os estudos de potencial interesse acerca do mesmo.

No campo do desenho da solução, seria interessante desenvolver os seguintes estudos:

- Conceber o índice métrico suportando escritores concorrentes, em uma ou mais *clouds* em simultâneo;
- Suporte para diferentes (em tipo ou parametrização) índices métricos em simultâneo;
- Implementação de políticas de segurança para garantir a privacidade dos dados e aferir o seu impacto no modelo desenhado;
- Desenvolvimento de um sistema de *caching* orientado à distribuição, com vista a ser partilhado por pedidos concorrentes;
- Criação de uma função de gestão da estrutura. A recomputação da RLC em *background* permitiria a adaptação dos parâmetros da mesma para a obtenção de um melhor desempenho bem como a filtragem de dados não desejados, sem um custo muito acrescido;
- Desenvolvimento do processo de criação de *shadow buckets* em memória principal.

Já relativamente à área de implementação e validação do protótipo podem ser apontadas as seguintes direcções:

- Implementação da operação de remoção no índice métrico;
- Validação experimental da solução com uma base de dados de maior dimensão, à semelhança das existentes em aplicações de sensoriamento participado;
- Análise da frequência de leituras de nós e pontos na RLC para a criação de políticas de *cache*.

Bibliografia

[1] A. Sarmento and M. Mamede, "Uma Estrutura de Dados Métrica Genérica, Dinâmica, em Memória Secundária," *inforum.org.pt*, pp. 79–90, 2010.

[2] "Face Recognition Data. http://cswww.essex.ac.uk/mv/allfaces/index.html, May 2011."

[3] A. T. Campbell, "The Rise of People-Centric Sensing," *IEEE Internet Computing*, vol. 12, pp. 12–21, July 2008.

[4] D. Cuff and M. Hansen, "Urban sensing: out of the woods," *Communications of the ACM*, vol. 51, no. 3, 2008.

[5] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. Srivastava, "Participatory sensing," in *World Sensor Web Workshop*, pp. 1–5, Citeseer, 2006.

[6] J. Eriksson, L. Girod, B. Hull, and R. Newton, "The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring," *conference on Mobile*, pp. 29–39, 2008.

[7] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Cartel: a distributed mobile sensor computing system," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, pp. 125–138, ACM, 2006.

[8] J. Varia, "Cloud architectures," *White Paper of Amazon*, 2008.

[9] D. Zhao and M. Rosson, "How and why people twitter: the role that micro-blogging plays in informal communication at work," in *Proceedings of the ACM 2009 international conference on Supporting group work*, pp. 243–252, ACM, 2009.

[10] H. Ferreira, S. Duarte, and N. Preguiça, "4sensing–decentralized processing for participatory sensing data," in *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pp. 306–313, IEEE, 2010.

[11] "Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/, December 2010."

[12] "Windows Azure | Microsoft Platform Hosting | Online Application | Application Hosting. http://www.microsoft.com/windowsazure/windowsazure/, November 2010."

[13] "Enterprise Cloud Storage - Nirvanix Storage Delivery Network. http://www.nirvanix.com/, November 2010."

[14] "Cloud Files - Unlimited Online Storage & Data Storage Services. http://www.rackspacecloud.com/cloud_hosting_products/files/, November 2010."

[15] "Google Storage for Developers - Google Code. http://code.google.com/intl/pt-PT/apis/storage/, November 2010."

[16] "EMC Atmos Family - Cloud Services, Public Cloud. http://www.emc.com/products/family/atmos.htm, November 2010."

[17] "Amazon Relational Database Service (Amazon RDS). http://aws.amazon.com/rds/, January 2011."

[18] "Amazon SimpleDB. http://aws.amazon.com/simpledb/, January 2011."

[19] "Google App Engine - Google Code. http://code.google.com/intl/pt-PT/appengine/, December 2010."

[20] "Armazenamento na cloud. http://goo.gl/KbtSV, September 2011.".

[21] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29–43, ACM, 2003.

[22] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[23] D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," in *In Proc. SOSP*, pp. 205–220, Citeseer, 2007.

[24] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[25] "Panoramio - Fotografias do Mundo. http://www.panoramio.com/, October 2010."

[26] "Welcome to Flickr - Photo Sharing. http://www.flickr.com/, October 2010."

[27] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pp. 311–321, Society for Industrial and Applied Mathematics, 1993.

[28] M. Micó, J. Oncina, and E. Vidal, "A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements," *Pattern Recognition Letters*, vol. 15, no. 1, pp. 9–17, 1994.

[29] M. Mamede, "Recursive lists of clusters: A dynamic data structure for range queries in metric spaces," *Computer and Information Sciences-ISCIS 2005*, pp. 843–853, 2005.

[30] S. Brin, "Near neighbor search in large metric spaces," in *21th International Conference on Very Large Data Bases (VLDB 1995)*, 1995.

[31] M. Mamede, "A dynamic data structure for range queries in high dimensional metric spaces," tech. rep., Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2007.

[32] F. Barbosa, "Similarity-based retrieval in high dimensional data with Recursive Lists of Clusters: a study case with Natural Language Dictionaries," in *Information Management and Engineering, 2009. ICIME'09. International Conference on*, pp. 432–436, IEEE, 2009.

[33] F. Costa and F. Barbosa, "Timbre Similarity Search with Metric Data Structures," in *13th European Conference on Research and Advanced Technologies on Digital Libraries (ECDL 2009)*, 2009.

[34] A. Sarmento, "Estruturas de dados métricas genéricas em memória secundária," Master's thesis, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, Monte de Caparica, 2010.

[35] C. Rodrigues, "Implementação de sistemas de indexação para espaços métricos," *Relatório do Projecto Final de Curso, Departamento de Informática, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Quinta da Torre*, pp. 2829–516, 2006.

[36] W. Zhao, R. Chellappa, P. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.

[37] P. Silva, "Pesquisa de imagens de rosto," Master's thesis, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, Monte de Caparica, 2009.

[38] M. Turk and A. Pentland, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pp. 586–591, IEEE, 1991.

7

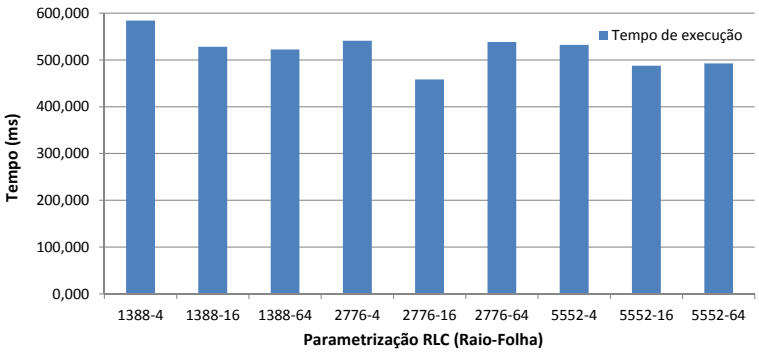
Apêndice

7.1 Estudos da análise métrica temporal

7.1.1 Análise da operação de inserção

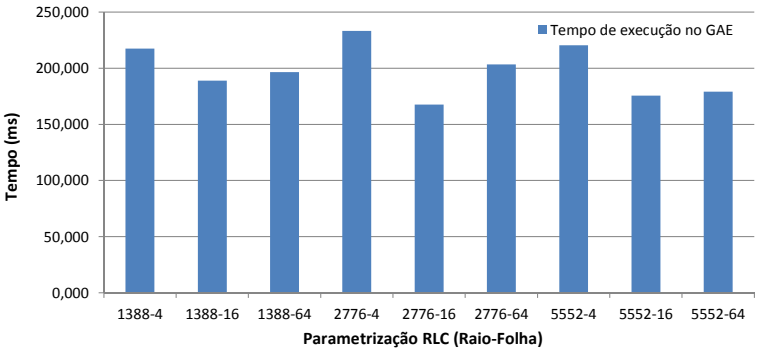
7.1.1.1 RLC em memória distribuída

Nesta subsecção é apresentado o estudo completo resultante dos testes efectuados à operação de inserção na RLC em memória distribuída.



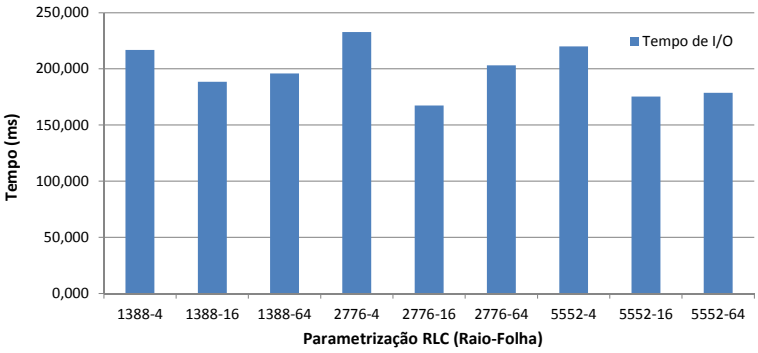
Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	584,137	541,204	532,016
	16	527,990	458,442	487,720
	64	522,385	538,332	492,600

Figura 7.1: Tabela e gráfico com os resultados dos tempos médios de inserção na RLC em memória distribuída



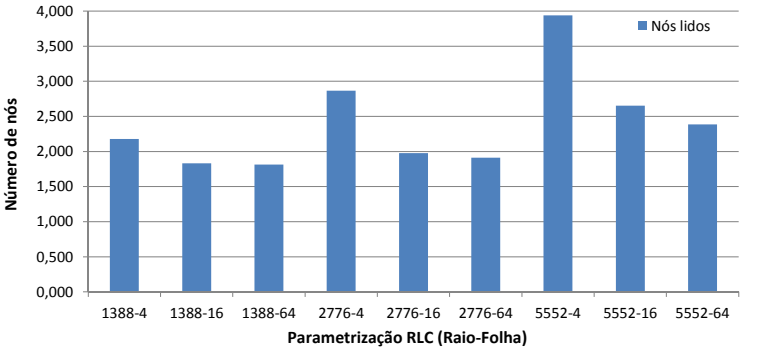
Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	217,402	233,171	220,333
	16	188,850	167,617	175,618
	64	196,478	203,438	179,111

Figura 7.2: Tabela e gráfico com os resultados dos tempos médios de execução no GAE durante uma inserção na RLC em memória distribuída



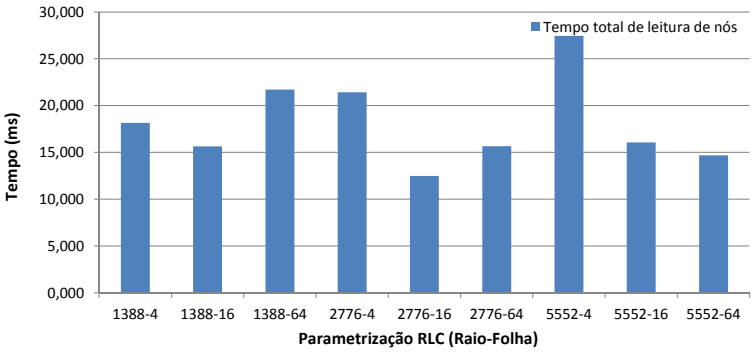
Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	216,831	232,788	220,045
	16	188,489	167,449	175,338
	64	195,914	203,190	178,751

Figura 7.3: Tabela e gráfico com os resultados dos tempos médios de acesso ao *datastore* durante uma inserção na RLC em memória distribuída



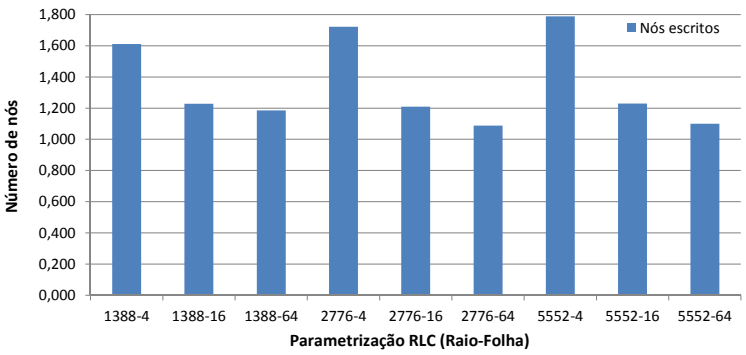
		Raio 1388	Raio 2776	Raio 5552
Folha	4	2,178	2,867	3,940
	16	1,831	1,977	2,654
	64	1,814	1,911	2,386

Figura 7.4: Tabela e gráfico com os resultados da média do número de nós lidos durante uma inserção na RLC em memória distribuída



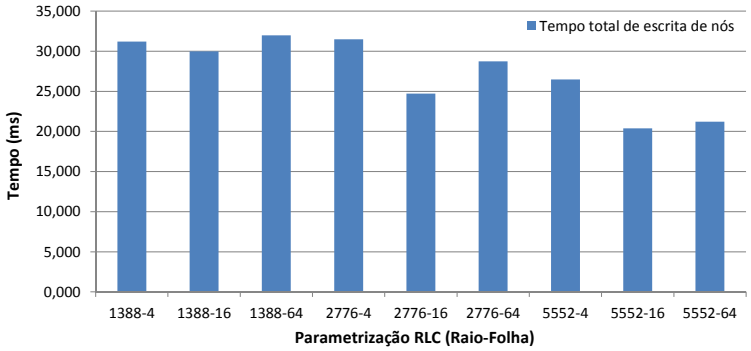
Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	18,155	21,431	27,447
	16	15,643	12,487	16,066
	64	21,715	15,652	14,684

Figura 7.5: Tabela e gráfico com os resultados do tempo médio gasto na leitura de nós durante uma inserção na RLC em memória distribuída



		Raio 1388	Raio 2776	Raio 5552
Folha	4	1,611	1,722	1,788
	16	1,228	1,210	1,229
	64	1,186	1,088	1,100

Figura 7.6: Tabela e gráfico com os resultados da média do número de nós escritos durante uma inserção na RLC em memória distribuída



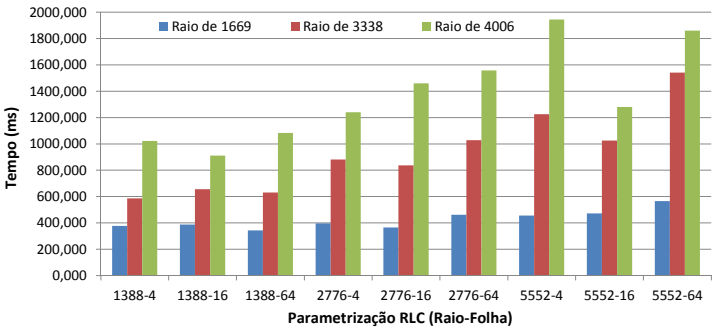
Tempo (ms)		Raio 1388	Raio 2776	Raio 5552
Folha	4	31,208	31,474	26,472
	16	29,984	24,728	20,390
	64	31,989	28,750	21,204

Figura 7.7: Tabela e gráfico com os resultados do tempo médio gasto na escrita de nós durante uma inserção na RLC em memória distribuída

7.1.2 Análise da operação de pesquisa

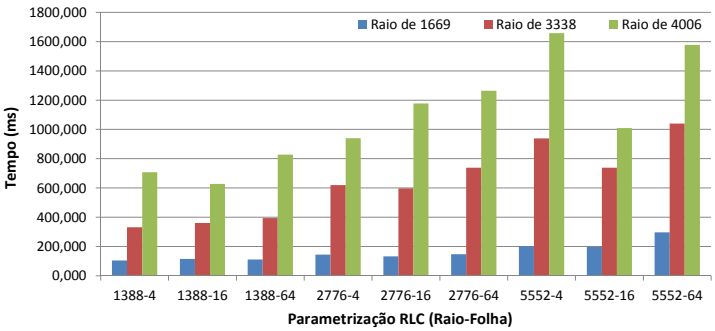
7.1.2.1 RLC em memória distribuída

Nesta subsecção é apresentado o estudo completo resultante dos testes efectuados à operação de pesquisa, para três raios diferentes, na RLC em memória distribuída.



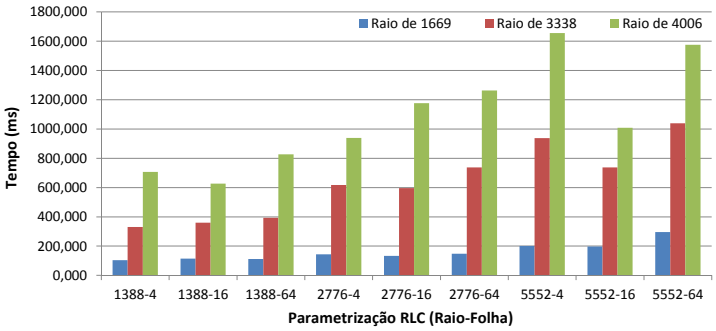
Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	378,230	586,056	1022,387
		16	387,416	655,980	910,377
		64	343,252	631,482	1082,883
	2776	4	396,049	881,099	1239,873
		16	365,685	836,362	1459,998
		64	462,195	1027,799	1557,704
	5552	4	456,654	1225,903	1943,801
		16	472,903	1025,913	1280,813
		64	565,200	1540,559	1859,353

Figura 7.8: Tabela e gráfico com os resultados dos tempos médios de pesquisa na RLC em memória distribuída



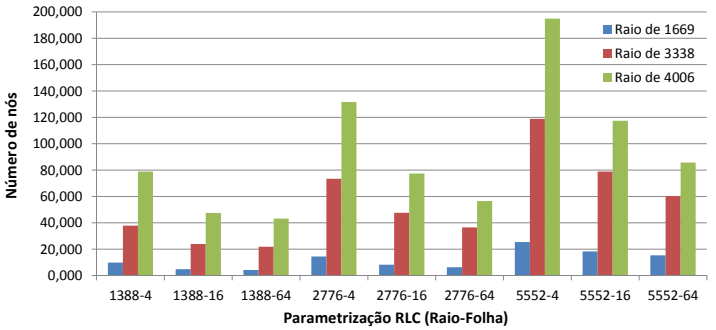
Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	104,634	331,163	708,123
		16	115,590	361,182	627,804
		64	111,760	393,974	828,454
	2776	4	144,360	619,554	940,535
		16	133,458	597,311	1177,736
		64	147,899	738,142	1264,538
	5552	4	202,218	939,073	1657,392
		16	197,759	738,352	1009,118
		64	296,524	1039,572	1577,083

Figura 7.9: Tabela e gráfico com os resultados dos tempos médios de execução no GAE durante uma pesquisa na RLC em memória distribuída



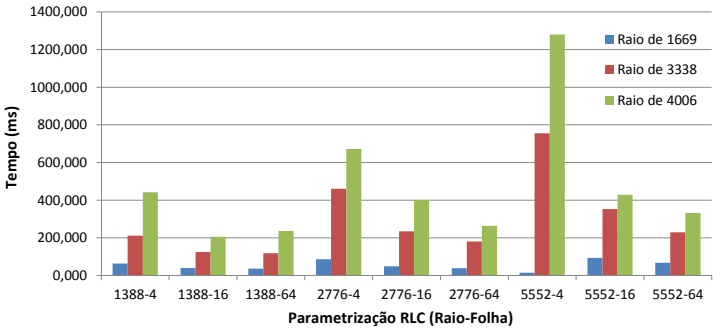
Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	104,193	330,758	706,721
		16	115,237	360,426	626,853
		64	111,400	393,486	827,015
	2776	4	143,996	617,802	939,416
		16	133,127	596,578	1176,188
		64	147,601	737,127	1262,977
	5552	4	201,921	938,017	1655,844
		16	197,367	737,543	1008,189
		64	296,193	1038,441	1575,550

Figura 7.10: Tabela e gráfico com os resultados dos tempos médios de acesso ao *datastore* durante uma pesquisa na RLC em memória distribuída



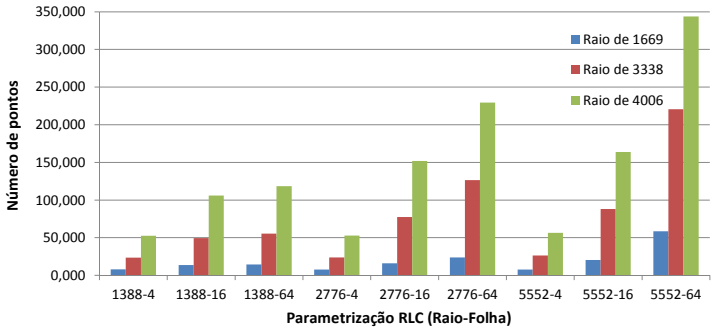
		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	9,920	37,892	78,966
		16	4,896	23,946	47,500
		64	4,215	21,956	43,170
	2776	4	14,474	73,380	131,657
		16	8,252	47,732	77,420
		64	6,312	36,603	56,588
	5552	4	25,458	118,852	194,853
		16	18,301	78,999	117,406
		64	15,443	60,277	85,737

Figura 7.11: Tabela e gráfico com os resultados da média do número de nós lidos durante uma pesquisa, para os 3 raios em teste, na RLC em memória distribuída



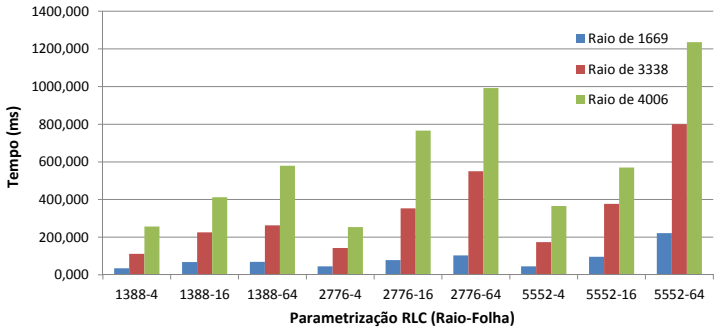
Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	63,111	211,559	441,887
		16	39,792	124,957	205,201
		64	35,953	118,346	236,513
	2776	4	86,296	460,223	672,302
		16	48,873	234,017	400,207
		64	38,633	180,400	263,927
	5552	4	15,017	754,954	1280,526
		16	93,524	352,642	428,414
		64	67,397	229,755	332,105

Figura 7.12: Tabela e gráfico com os resultados do tempo médio gasto na leitura de nós durante uma pesquisa, para os 3 raios em teste, na RLC em memória distribuída



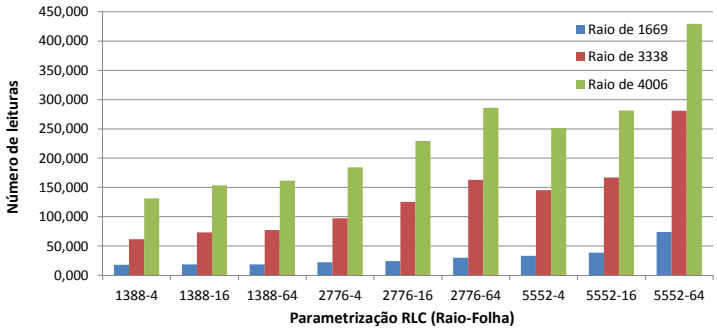
		Folha	Raio de pesquisa		
			1669	3338	4006
Raio	1388	4	7,886	23,616	52,441
		16	13,598	49,354	106,092
		64	14,332	55,448	118,409
	2776	4	7,606	23,834	52,785
		16	16,013	77,442	151,829
		64	23,760	126,341	229,560
	5552	4	7,792	26,288	56,378
		16	20,252	87,969	163,930
		64	58,591	220,709	343,804

Figura 7.13: Tabela e gráfico com os resultados da média do número de pontos lidos durante uma pesquisa, para os 3 raios em teste, na RLC em memória distribuída



Tempo (ms)		Folha	Raio de pesquisa		
			1669	3338	4006
Raio	1388	4	34,671	111,634	256,647
		16	67,871	225,377	411,858
		64	68,844	263,247	579,121
	2776	4	44,513	142,431	253,284
		16	77,682	352,744	765,804
		64	103,113	550,188	992,141
	5552	4	44,458	173,935	365,636
		16	95,855	376,825	570,033
		64	221,265	800,634	1235,574

Figura 7.14: Tabela e gráfico com os resultados do tempo médio gasto na leitura de pontos durante uma pesquisa, para os 3 raios em teste, na RLC em memória distribuída

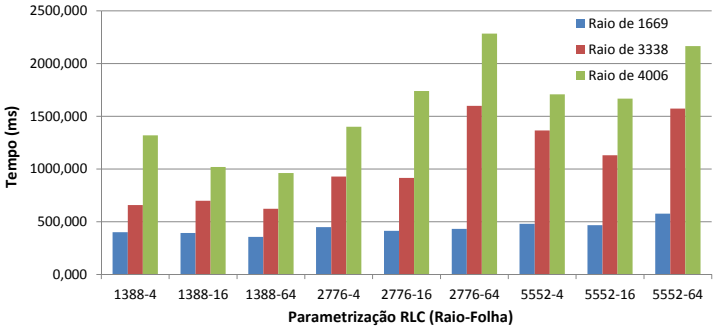


		Folha	Raio de pesquisa		
			1669	3338	4006
Raio	1388	4	17,806	61,508	131,407
		16	18,494	73,300	153,592
		64	18,547	77,404	161,579
	2776	4	22,080	97,214	184,442
		16	24,265	125,174	229,249
		64	30,072	162,944	286,148
	5552	4	33,250	145,140	251,231
		16	38,553	166,968	281,336
		64	74,034	280,986	429,541

Figura 7.15: Tabela e gráfico com os resultados da média do número leituras efectuadas durante uma pesquisa, para os 3 raios em teste, na RLC em memória distribuída

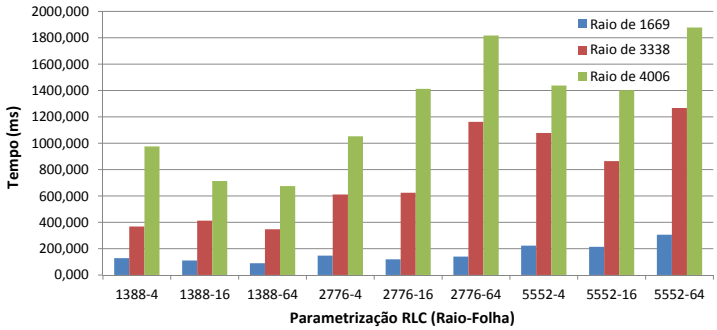
7.1.2.2 RLC em memória distribuída (33% *shadow buckets*)

Nesta subsecção é apresentado o estudo completo resultante dos testes efectuados à operação de pesquisa, para três raios diferentes, na RLC em memória distribuída com 33% de *shadow buckets*.



Tempo (ms)	Folha	Raio de pesquisa 1669	3338	4006
Raio 1388	4	400,430	657,573	1318,999
	16	393,968	699,150	1019,469
	64	355,513	623,625	961,013
Raio 2776	4	449,025	929,229	1400,448
	16	414,076	916,479	1739,577
	64	431,572	1599,811	2285,268
Raio 5552	4	479,484	1365,906	1707,635
	16	466,859	1129,669	1666,983
	64	576,698	1572,674	2165,108

Figura 7.16: Tabela e gráfico com os resultados dos tempos médios de pesquisa na RLC em memória distribuída com 33% de *shadow buckets*



Tempo (ms)	Folha	Raio de pesquisa 1669	3338	4006
Raio 1388	4	127,118	368,379	975,385
	16	109,652	411,733	712,852
	64	89,881	346,905	674,872
Raio 2776	4	146,709	610,942	1053,152
	16	119,577	623,636	1413,437
	64	139,254	1162,481	1818,194
Raio 5552	4	222,474	1078,593	1437,831
	16	213,448	864,988	1401,937
	64	306,273	1267,101	1878,981

Figura 7.17: Tabela e gráfico com os resultados dos tempos médios de acesso ao *datastore* durante uma pesquisa na RLC em memória distribuída com 33% de *shadow buckets*

Tabela 7.1: Tabela com os resultados e análise do número de nós lidos durante uma pesquisa na RLC em memória distribuída com 33% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	13,712	▲38%	54,219	▲43%	117,907	▲49%
		16	7,604	▲55%	40,290	▲68%	81,540	▲72%
		64	6,636	▲57%	37,692	▲72%	75,134	▲74%
	2776	4	18,442	▲27%	95,877	▲31%	177,630	▲35%
		16	11,871	▲44%	72,822	▲53%	119,749	▲55%
		64	9,761	▲55%	60,071	▲64%	93,583	▲65%
	5552	4	31,883	▲25%	158,811	▲34%	264,377	▲36%
		16	25,027	▲37%	116,193	▲47%	175,062	▲49%
		64	23,043	▲49%	96,892	▲61%	139,522	▲63%

Tabela 7.2: Tabela com os resultados e análise do número de pontos lidos durante uma pesquisa na RLC em memória distribuída com 33% de *shadow buckets*

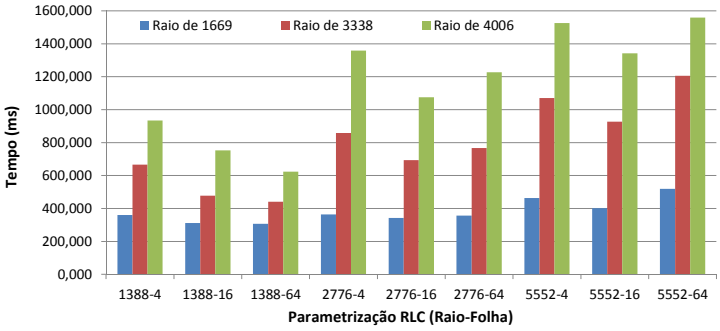
		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	5,105	▼35%	14,409	▼39%	32,686	▼38%
		16	9,529	▼30%	35,369	▼28%	76,654	▼28%
		64	9,227	▼56%	35,337	▼36%	76,000	▼36%
	2776	4	5,189	▼32%	15,354	▼36%	34,662	▼34%
		16	10,274	▼36%	57,306	▼26%	115,362	▼24%
		64	14,925	▼37%	77,708	▼38%	143,827	▼37%
	5552	4	5,415	▼31%	19,259	▼27%	41,425	▼27%
		16	13,255	▼35%	55,045	▼37%	101,324	▼38%
		64	36,479	▼38%	145,569	▼34%	226,060	▼34%

Tabela 7.3: Tabela com os resultados e análise do número de leituras efectuadas durante uma pesquisa na RLC em memória distribuída com 33% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	18,817	▲6%	68,628	▲12%	150,593	▲15%
		16	17,133	▼7%	75,659	▲3%	158,194	▲3%
		64	15,863	▼14%	73,029	▼6%	151,134	▼6%
	2776	4	23,631	▲7%	111,231	▲14%	212,292	▲15%
		16	22,145	▼9%	130,128	▲4%	235,111	▲3%
		64	24,686	▼18%	137,779	▼15%	237,410	▼17%
	5552	4	37,298	▲12%	178,070	▲23%	305,802	▲22%
		16	38,282	▼1%	171,238	▲3%	276,386	▼2%
		64	59,522	▼20%	242,461	▼14%	365,582	▼15%

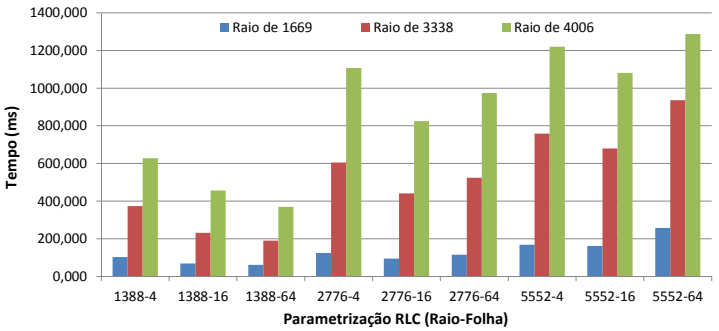
7.1.2.3 RLC em memória distribuída (66% *shadow buckets*)

Nesta subsecção é apresentado o estudo completo resultante dos testes efectuados à operação de pesquisa, para três raios diferentes, na RLC em memória distribuída com 66% de *shadow buckets*.



Tempo (ms)	Folha	Raio de pesquisa 1669	3338	4006
Raio 1388	4	359,670	666,138	933,722
	16	312,135	477,526	753,214
	64	306,515	440,507	623,080
Raio 2776	4	363,482	858,765	1358,997
	16	342,360	693,954	1075,172
	64	357,126	767,524	1226,755
Raio 5552	4	462,817	1070,626	1525,396
	16	401,587	926,468	1341,846
	64	519,500	1205,850	1559,488

Figura 7.18: Tabela e gráfico com os resultados dos tempos médios de pesquisa na RLC em memória distribuída com 66% de *shadow buckets*



Tempo (ms)	Folha	Raio de pesquisa 1669	3338	4006
Raio 1388	4	102,943	373,360	627,864
	16	68,907	231,645	456,264
	64	60,763	189,803	369,486
Raio 2776	4	124,891	604,699	1106,839
	16	94,165	441,466	824,507
	64	115,485	524,386	974,155
Raio 5552	4	167,653	758,444	1220,274
	16	162,187	679,608	1081,136
	64	257,039	936,191	1287,853

Figura 7.19: Tabela e gráfico com os resultados dos tempos médios de acesso ao *datastore* durante uma pesquisa na RLC em memória distribuída com 66% de *shadow buckets*

Tabela 7.4: Tabela com os resultados e análise do número de nós lidos durante uma pesquisa na RLC em memória distribuída com 66% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	13,038	▲31%	51,468	▲36%	111,191	▲41%
		16	6,613	▲35%	33,726	▲41%	68,557	▲44%
		64	5,714	▲36%	31,943	▲46%	64,005	▲48%
	2776	4	17,206	▲19%	89,011	▲21%	164,215	▲25%
		16	10,118	▲23%	60,897	▲28%	100,132	▲29%
		64	8,163	▲29%	49,322	▲35%	76,855	▲36%
	5552	4	29,736	▲17%	146,774	▲23%	243,349	▲25%
		16	22,641	▲24%	102,771	▲30%	154,524	▲32%
		64	21,149	▲37%	86,907	▲44%	124,853	▲46%

Tabela 7.5: Tabela com os resultados e análise do número de pontos lidos durante uma pesquisa na RLC em memória distribuída com 66% de *shadow buckets*

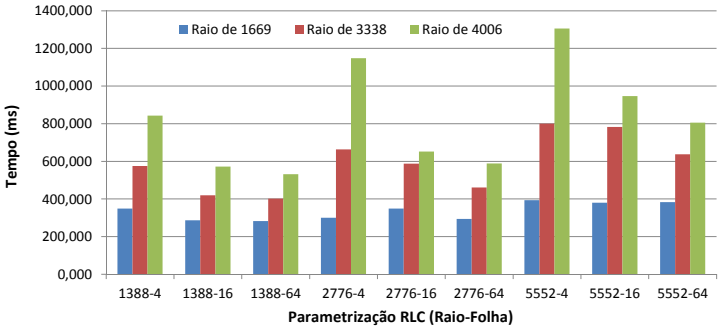
		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	2,946	▼63%	9,983	▼58%	21,973	▼58%
		16	4,776	▼65%	14,848	▼70%	33,433	▼68%
		64	4,571	▼68%	17,128	▼69%	36,676	▼69%
	2776	4	2,666	▼65%	8,276	▼65%	18,372	▼65%
		16	5,036	▼69%	25,874	▼67%	52,033	▼66%
		64	9,278	▼61%	45,919	▼64%	80,833	▼65%
	5552	4	2,696	▼65%	10,398	▼60%	21,482	▼62%
		16	7,939	▼61%	29,503	▼66%	53,732	▼67%
		64	30,205	▼48%	102,105	▼54%	152,094	▼56%

Tabela 7.6: Tabela com os resultados e análise do número de leituras efectuadas durante uma pesquisa na RLC em memória distribuída com 66% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	15,984	▼10%	61,451	▼0,1%	133,164	▲1%
		16	11,389	▼38%	48,574	▼34%	101,990	▼34%
		64	10,285	▼45%	49,071	▼37%	100,681	▼38%
	2776	4	19,872	▼10%	97,287	▲0,1%	182,587	▼1%
		16	15,154	▼38%	86,771	▼31%	152,165	▼34%
		64	17,441	▼42%	95,241	▼42%	157,688	▼45%
	5552	4	32,432	▼2%	157,172	▲8%	264,831	▲5%
		16	30,580	▼21%	132,274	▼21%	208,256	▼26%
		64	51,354	▼31%	189,012	▼33%	276,947	▼36%

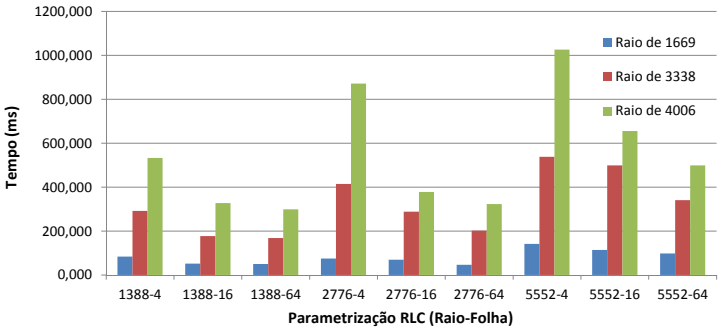
7.1.2.4 RLC em memória distribuída (100% *shadow buckets*)

Nesta subsecção é apresentado o estudo completo resultante dos testes efectuados à operação de pesquisa, para três raios diferentes, na RLC em memória distribuída com 100% de *shadow buckets*.



Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	348,854	575,575	843,128
		16	286,567	419,945	572,642
		64	282,761	402,398	532,011
	2776	4	300,564	663,056	1147,997
		16	348,734	588,059	652,305
		64	294,161	461,065	588,294
	5552	4	394,019	800,417	1305,923
		16	379,751	782,384	946,525
		64	383,364	637,835	805,241

Figura 7.20: Tabela e gráfico com os resultados dos tempos médios de pesquisa na RLC em memória distribuída com 100% de *shadow buckets*



Tempo (ms)		Folha	Raio de pesquisa 1669	3338	4006
Raio	1388	4	83,742	292,041	532,752
		16	51,544	177,125	327,210
		64	49,672	168,216	298,710
	2776	4	75,041	414,839	871,347
		16	69,270	288,152	378,400
		64	46,798	202,448	323,046
	5552	4	141,648	538,608	1026,199
		16	113,722	499,080	656,073
		64	98,457	340,452	499,272

Figura 7.21: Tabela e gráfico com os resultados dos tempos médios de acesso ao *datastore* durante uma pesquisa na RLC em memória distribuída com 100% de *shadow buckets*

Tabela 7.7: Tabela com os resultados e análise do número de nós lidos durante uma pesquisa na RLC em memória distribuída com 100% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	11,860	▲20%	45,911	▲21%	98,600	▲25%
		16	5,650	▲15%	28,841	▲20%	58,402	▲23%
		64	4,879	▲16%	26,670	▲21%	53,616	▲24%
	2776	4	16,113	▲11%	82,691	▲13%	151,175	▲15%
		16	8,671	▲5%	50,943	▲7%	83,170	▲7%
		64	6,514	▲3%	38,759	▲6%	60,444	▲7%
	5552	4	28,222	▲11%	136,648	▲15%	225,240	▲16%
		16	20,019	▲9%	89,668	▲14%	134,221	▲14%
		64	16,719	▲8%	68,115	▲13%	97,972	▲14%

Tabela 7.8: Tabela com os resultados e análise do número de leituras efectuadas durante uma pesquisa na RLC em memória distribuída com 100% de *shadow buckets*

		Folha	Raio de pesquisa					
			1669		3338		4006	
Raio	1388	4	11,860	▼33%	45,911	▼25%	98,600	▼25%
		16	5,650	▼69%	28,841	▼61%	58,402	▼62%
		64	4,879	▼74%	26,670	▼66%	53,616	▼67%
	2776	4	16,113	▼27%	82,691	▼15%	151,175	▼18%
		16	8,671	▼64%	50,943	▼59%	83,170	▼64%
		64	6,514	▼78%	38,759	▼76%	60,444	▼79%
	5552	4	28,222	▼15%	136,648	▼6%	225,240	▼10%
		16	20,019	▼48%	89,668	▼46%	134,221	▼52%
		64	16,719	▼77%	68,115	▼76%	97,972	▼77%

7.2 Estudos da análise métrica espacial

7.2.1 Análise espacial da RLC em memória distribuída

Os gráficos das Figuras 7.22, 7.23 e 7.24 são representativos das proporções de ocupação espacial dos vários objectos guardados no sistema, para a sua versão sem melhoramentos. As figuras estão organizadas de acordo com o raio inicial da RLC, correspondendo respectivamente aos raios 1388, 2776 e 5552. Em cada um dos gráficos circulares é possível observar 4 variáveis: Imagens, Pontos, Nós e Outros. Estes dizem respeito aos vários tipos de objectos guardados. A variável "Nós"inclui tanto nós interiores como folhas ao passo que "Outros"contém informação de gestão do sistema, como sequenciadores, lista de nós a remover, etc.

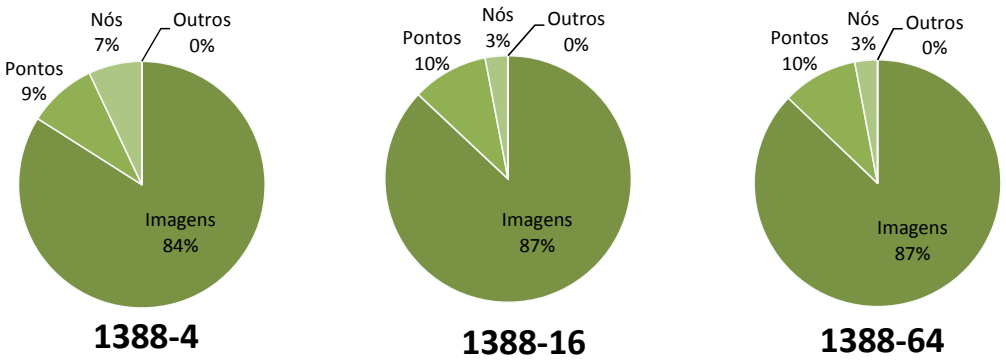


Figura 7.22: Gráficos representativos do espaço ocupado pelos objectos nas RLCs de raio inicial 1388 em memória distribuída

Através da análise dos gráficos da Figura 7.22, referentes às parametrizações 1388-4, 1388-16 e 1388-64, é possível observar um crescimento das percentagens de imagens e pontos à medida que a capacidade das folhas aumenta. O crescimento não é devido ao aumento da cardinalidade destes, mas sim à redução número de nós, fruto de uma menor quantidade de expansões em profundidade por parte das folhas. Com a maior capacidade de folha, a diferença mantém-se inalterada, devido ao número reduzido de pontos dentro de cada agrupamento que a contém, o que origina um baixo número de expansões, mesmo no caso intermédio. Deste modo, é possível concluir que a criação de novos nós tem maior impacto espacial que o aumento de tamanho dos *buckets*.

Nas RLCs com raio inicial igual a 2776 as conclusões anteriores mantêm-se válidas, sendo ainda mais visíveis do que na situação transacta, devido à contínua diminuição da percentagem dos nós. Em comparação com o da Figura 7.22, o gráfico de 7.23 correspondente a uma capacidade de folha 4, apresenta um valor superior da variável "Nós", para o mesmo exemplo. Embora a raiz seja de menor dimensão, a menor distribuição neste nó, implica a criação de alguns suplementares, o provoca um aumento do tamanho total. Com folhas maiores, a menor distribuição inicial dos dados é compensada, originando uma menor cardinalidade e o seu melhor preenchimento.

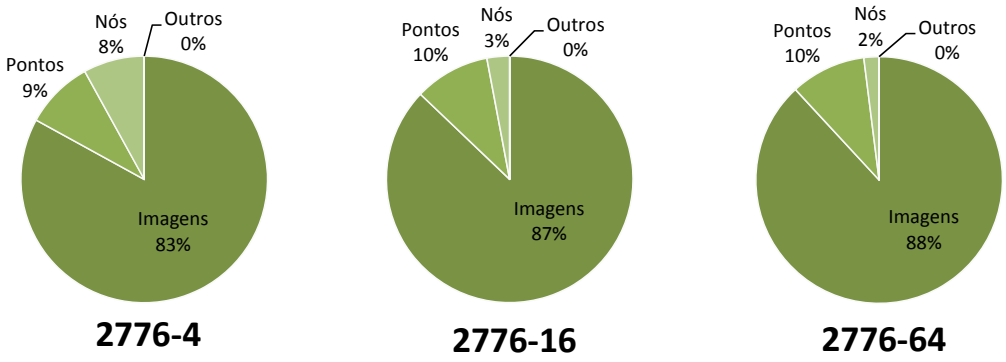


Figura 7.23: Gráficos representativos do espaço ocupado pelos objectos nas RLCs de raio inicial 2776 em memória distribuída

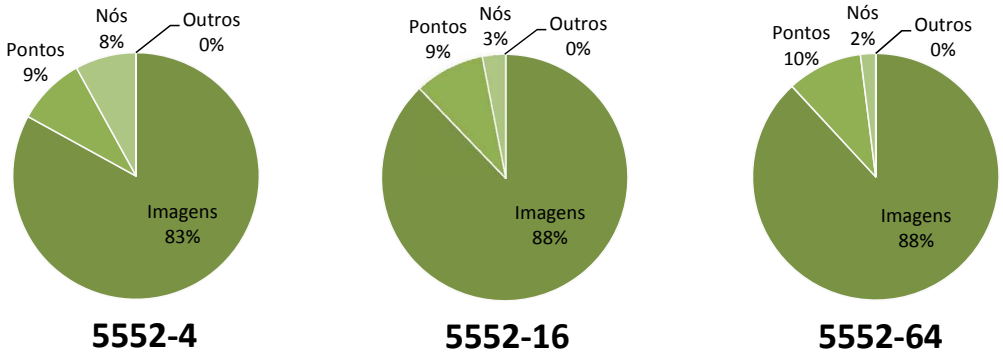


Figura 7.24: Gráficos representativos do espaço ocupado pelos objectos nas RLCs de raio inicial 5552 em memória distribuída

As mesmas características são verificadas nos gráficos da Figura 7.24, embora neste âmbito (número de imagens, dimensão elevada de raio e folhas) as diferenças para o exemplo anterior não sejam tão notórias.