Paprika: Replica Placement for Multi-Region Edge Publish-Subscribe Systems

Xavier Pacheco
Pedro Vieira*
NOVA School of Science and Technology
NOVA University Lisbon
Portugal

Hervé Paulino
NOVA LINCS, NOVA School of Science and Technology
NOVA University Lisbon
Portugal
herve.paulino@fct.unl.pt

Abstract

Efficient data replica placement at the network edge is crucial for minimizing data access latency and enhancing user experience. Current solutions are mostly popularity oriented, assuming that users in close proximity share similar interests. While this strategy is effective for widely accessed data, it overlooks the specificity of user interests and their affinities, especially in diverse environments managed by different edge servers: data popular in region A may not be relevant in region B but be highly valuable in region C.

We introduce Paprika, an online heuristic-based hybrid algorithm that combines the strengths of Genetic Algorithms and Tabu Search to address replica selection and placement in edge computing environments. Paprika takes into account data popularity, pair-wise affinity between regions, and server storage capacity. Our evaluation demonstrates that the hybrid approach outperforms traditional heuristic methods by better balancing user interests across regions and favoring regions with stronger affinities.

CCS Concepts

• **Information systems** → **Data replication tools**; • **Software and its engineering** → **Publish-subscribe / event-based architectures**; • **Computer systems organization** → *Peer-to-peer architectures.*

Keywords

Edge computing, Replica placement, Popularity, Affinity, Capacity

ACM Reference Format:
Xavier Pacheco, Pedro Vieira, and Hervé Paulino. 2025. Paprika: Replica Placement for Multi-Region Edge Publish-Subscribe Systems. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3672608.3707924

1 Introduction

The edge-computing paradigm represents a significant shift in how data is processed and stored, moving these functions closer to the data source to enhance system performance and enable real-time capabilities. This approach, however, introduces new complexities, particularly in efficiently sharing data among users distributed across various geographical regions and architectural settings.

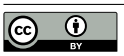
The placement of data replicas on nodes at the network edge is a crucial strategy to enhance user experience by significantly reducing data access latency. However, this strategy comes with inherent challenges due to the limited storage capacity of edge servers compared to cloud servers. Effective data placement at the edge must, therefore, carefully balance the need to minimize latency with the constraints of available storage.

Current replica placement solutions typically focus on popular content [2, 13, 14, 17], operating under the assumption that users in close proximity are often interested in related information. While this strategy works well for widely accessed data, it tends to overlook: **The specificity of user interests and their affinity** – user interests can vary significantly, specially in environments where user populations are diverse and spread across multiple regions, managed by different edge servers. Data popular in one region may not be relevant in a nearby region but could be highly valuable in another. **Less popular yet still valuable information** – the focus on popular content often neglects the availability and effective sharing of niche or less popular data.

In this paper, we introduce Paprika, a replica placement solution designed for publish/subscribe systems in edge computing environments spanning multiple regions. We define region to be an edge server plus the set of connected (potentially mobile) devices. Paprika is designed to meet the unique requirements of execution and storage at the network edge, basing its placement decisions on key factors such as data popularity, the affinity of data across different regions, and the storage capacity of the servers supporting these regions. Our design aims to achieve a fine balance: addressing potential biases in data placement without sacrificing the availability of highly demanded content, while also minimizing the computational overhead needed to maintain a high-quality solution.

Given that the replica placement problem is NP-Complete, the current state-of-the-art solutions prioritize different aspects. Algorithms like Holistic All [14] focus on improving the quality of approximations, often at the expense of efficiency, while approaches such as D-Rep [2] emphasize efficiency over approximation accuracy. In contrast, Paprika aims to achieve both accuracy and effectiveness by being a decentralized online heuristic-based hybrid algorithm that leverages the exploratory prowess of Tabu Search (TS) and the exploitative efficiency of Genetic Algorithm (GA). This combination ensures a balanced search strategy that can effectively navigate complex and large-scale search spaces. Specifically, TS aids in swiftly converging to a promising region of the solution space, while the GA meticulously explores this refined region to

*At the time of his contribution to this work.



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707924

discover high-quality solutions. The online nature of the algorithm allows it to adapt efficiently to the dynamism of edge environments, where user interests and data demands are continually changing.

We implemented Paprika in the context of the Gardenbed framework [12], and developed a comprehensive simulation platform that models all relevant aspects of Gardenbed atop PureEdgeSim [10]. This platform facilitates rigorous testing and comparison of various heuristic algorithms employed by Paprika, particularly in the data placement phase. The performance evaluation shows that the hybrid approach followed by Paprika consistently delivers superior solutions by accurately reflecting user interests and efficiently disseminating niche data from its origin region to others where demand is higher. This ensures optimal data placement, enhancing system performance and improving user satisfaction across multiple regions. The contributions of this work are thus:

An online Popularity, Capacity and Affinity oriented replica placement algorithm. We propose a novel online replica placement algorithm designed to improve the discoverability and distribution of data in edge environments (Section 4). Unlike traditional methods that primarily focus on popular content, our approach facilitates personalized and region-specific data access for diverse user groups, tailoring placement based on the affinity of the users' interests on a given region. The online nature of the algorithm permanently adapts the placement decisions to the changes in the popularity and/or affinity factors.

Heuristic approach to data placement. We propose a heuristic method for efficiently addressing the replica placement problem by considering our three key factors: data popularity, affinity between regions, and the storage constraints of edge servers (Section 5). This approach ensures effective data placement while accommodating the computation limitations inherent to edge environments.

2 Related Work

In this section, we provide an overview of data placement systems for publish/subscribe systems with a particular focus on edge computing environments.

Popularity Oriented: In [14], the authors present four distributed cache management algorithms for information-centric networking. The emphasis is on minimizing overall network traffic by strategically placing data replicas based on real-time popularity and proximity to consumers. It operates with a global view of the network and adjusts replicas through coordinated, iterative rounds. The Holistic-All variant is particularly noted for balancing network traffic reduction with scalability challenges.

The work presented in [13], designed for content-centric publish-subscribe networks, minimizes network traffic by selecting optimal nodes for storing replicas based on data popularity and network topology. It uses a two-phase process: planning, which selects storage nodes, and assignment, which distributes topics to these nodes.

D-ReP [2] operates in a fully decentralized manner, focusing on pushing content closer to users at the edge, considering both latency and storage costs across different IaaS providers. It adapts to dynamic network conditions with minimal information, prioritizing low computational overhead and cost-effective data placement. D-Rep is particularly relevant in the edge/fog paradigm, as it is fully decentralized and accounts for factors often neglected by other methods, such as storage resource costs. However, its decision-making is limited by a narrow information horizon.

The algorithm proposed in [17] maps data items and servers to a virtual plane, placing the most popular data centrally to minimize access latency. It also includes load-balancing strategies to manage storage limitations, ensuring popular data is replicated and distributed efficiently across servers.

Capacity Oriented: Greedy Routing for Edge Data (GRED) [18] is a DHT-based approach that reduces index table space while providing low latency and load balancing by mapping network nodes onto a virtual plane where Euclidean distances correspond to actual network distances. It uses Delaunay triangulation to guide routing, ensuring that data is forwarded to the closest node. GRED's placement is centralized, offering simplicity and efficiency but lacks considerations for data popularity.

The algorithm proposed in [9] optimizes user experience and task scheduling by considering data popularity, storage capacity, and replacement costs. It calculates the optimal data placement using a value function and solves the problem as a knapsack problem with the Tabu algorithm. This method, however, relies on a centralized edge orchestrator and does not account for user mobility.

Cathode [4] is a decentralized algorithm focused on minimizing costs related to storage and replica consistency, adapting to dynamic access patterns without prior knowledge. It calculates optimal data placement in regular intervals (epochs) by analyzing operation frequencies and potential benefits of new replicas. Cathode excels in balancing time efficiency, scalability, and cost-effectiveness while allowing developers to specify consistency levels for operations.

Affinity Oriented: The Associated Data Placement (ADP) algorithm [19] addresses data affinity by optimizing data placement, minimizing the separation of related data, and reducing the distance between data and requesting nodes. In this work, affinity is defined in a geographic context, meaning the closer two servers are, the more affinity they share. This reflects the likelihood that geographically close edge servers will request the same dataset.

Despite its effectiveness, ADP suffers from scalability issues due to its centralized approach and high computational overhead.

3 System Model

We are targeting systems with, at least, two layers: device and edge; the cloud layer common in edge computing systems, is optional. The device layer handles data creation and consumption, with inter-device communication being achieved by standard Wi-Fi or peer-to-peer protocols like Bluetooth, Wi-Fi Direct, or Zigbee. Devices also connect to wireless access points (APs), which are linked to edge servers in a many-to-one relationship. These edge servers, that constitute the edge layer, may run on the APs themselves or on nearby computers like cloudlets. We refer to the ensemble of an edge server and its connected (potentially mobile) devices, referred to as nodes, as a *Region*.

Edge servers may connect to peers in reach to form a distributed system at the edge that provides a cross-region, topic-based publish/subscribe abstraction. Our system thus spans across geographically dispersed regions $R = R_0, R_1, \ldots, R_n$, where each region R_i is managed by an edge server e_i. Each of these servers has a known capacity, represented by $capacity(e_i)$, in bytes.

Although it may generalized, Paprika was mainly designed to work with topic-based publish/subscribe systems. To prevent naming conflicts, it assumes that topics are organized into namespaces. These namespaces allow data to be segregated across different applications or distinct users within the same application. For example, in a photo-sharing app, a namespace can represent a gallery, enabling each photo to be shared under any topics within its gallery namespace, without conflicting with photos in other galleries.

Each data object includes metadata that contains at least three key components: a unique *objectId*, the topics associated with the publication, and the operation's *namespace*. Similarly, each subscription must include the list of topics to subscribe to and the operation's *namespace*.

4 Paprika

The goal of Paprika is to place replicas of data published by devices dispersed across regions into the edge servers of those regions, while optimizing three key parameters:

1. Popularity Defined as the number of downloads an object receives.

2. Affinity Defined as the likelihood that geographically distributed edge servers will request the same dataset. In contrast to ADP [19], we do not define affinity based on geographic proximity. Instead, we define affinity between regions R_i and R_j as a measure of how popular the content generated in R_i is within R_j. Thus, if content on an edge server in R_i becomes highly popular in R_j, the greater the affinity R_j has with respect to R_i.

The goal is to enhance data object discoverability without overwhelming storage or network resources. Affinity is hence calculated as the ratio of object downloads within a region R_i that originated from Region R_j, denoted as $\#Downloads_{R_i} \leftarrow R_j$, relative to the total downloads in an epoch k, denoted as: $\#Downloads_{R_i,k}$. The affinity from region R_j to R_i in epoch k is therefore defined as:

$$affinity(R_i, R_j, k) = \frac{\#Downloads_{R_i} \leftarrow R_j}{\#Downloads_{R_i,k}}$$

It is important to emphasize that the affinity from region R_j to region R_i may not be symmetric, as the content of region R_i may be more or less appealing to the users in region R_j.

3. Server Capacity This is a crucial consideration due to the varying and limited storage capabilities of edge servers. To maximize user downloads, it may often be more effective to prioritize sending multiple smaller data objects with lower popularity scores rather than a single large object with a higher score.

4.1 Architecture

Building upon work developed in the context of the Gardenbed [12] framework, each edge server comprises three caches. The **Local Popularity Cache (LPC)** stores replicas of objects generated within the region, with the aim of providing faster and convenient access to the region's more popular data objects at a given time. By doing so, it shifts energy consumption from the nodes (typically mobile devices) to the edge, enhancing efficiency and reducing the energy burden on individual devices. Additionally, the LPC serves as the source for content to be shared with other regions through the replica placement algorithm.

By leveraging the LPC to provision content we significantly improve network utilization and overall performance by ensuring that only the most relevant (i.e., popular) data is transmitted. This approach eliminates the need to retrieve items on-demand from the devices, allowing for in-memory lookups, which directly reduces latency across the entire process.

The **Remote Prefetch Cache (RPC)** stores data and metadata entries for content proactively sent from remote regions by the replica placement algorithm. The content of this cache is periodically updated by the algorithm on an epoch-based schedule (more details in Section 4.2).

The **Remote Content Cache (RCC)** stores entries that were initially in the RPC and later deemed relevant in the edge server's region (i.e., effectively downloaded) by at least one of the region's nodes. Given the volatility of the RPC, moving content considered relevant to the RCC ensures greater persistence for future access and discovery by other nodes.

4.2 Algorithm

The replica placement algorithm is composed of four main phases, executed in epochs from the time the server is turned on. Each edge server has an independent time to trigger each one's epoch since they are not dependent on one another.

Phase 1 - Data Collection. In this phase, the edge server gathers information for the objects being published in its region, namely a *popularity* measure (given by the number of downloads) and the *size* of each object. To this end, the nodes periodically report (to the server) the popularity of the objects they manage. The server, in turn, continuously collects and stores this data, computing what we call the region's *Object Info*, for later use in the subsequent phases of the algorithm.

It is important to emphasize that there are no synchronization points that mark the beginning and end of this phase. Instead, the information and its processing occur continuously, allowing the object info to be refined throughout the duration of the algorithm's epoch. Each subsequent phase operates upon the state of the popularity information at the time it begins.

Phase 2 - Popularity Oriented Replica Selection. In the *Replica Selection* phase the algorithm processes the object info to determine which objects should be cached in the server's LPC and later, when relevant, shared (placed) with neighboring regions.

As recognized by the community [4, 15, 17], server/replica selection and placement in edge environments can be modeled as a Knapsack problem. The objective of this problem is to maximize the total value of items placed in a knapsack with limited capacity. Each item has a weight and a value, and the goal is to select a combination of items that maximizes the total value without exceeding the knapsack's weight limit. Similarly, in our scenario, we seek to maximize the utility of selected objects while adhering to our resource constraints, such as storage capacity.

Given the dynamic nature of edge systems, characterized by high churn rates of mobile users and rapidly changing interests, we determined that a heuristic approach would be most effective. In Section 5, we will delve into the chosen methodologies.

Phase 3 - Popularity and Capacity Oriented Replica Placement. To access content from other regions of interest, each edge

server sends *Subscription Catalog Dissemination* messages to its reachable peers. These messages are structured as follows: $\langle \overline{RPC}, \overline{Sub_R}, C, A, T \rangle$, where $\overline{RPC}$ is an optional value, sent in the first interaction, that denotes the capacity of the edge server's RPC; $\overline{Sub_R}$ holds the total amount of subscriptions, across all namespaces, made by the region's nodes up until the moment the message is built; C defined as $map\langle namespace_{id}, map\langle topic, \overline{Sub_t} \rangle \rangle$ represents the subscription catalog of the region's nodes, by namespace, which, in turn, contains the total amount of subscriptions ($\overline{Sub_t}$) for each topic; A represents the number of *Affinity Tuning* messages to be sent between epochs (see Phase 4); and, lastly, T represents the period between *Subscription Catalog Dissemination* messages.

Upon receiving this message, each peer will generate a *Subscription Catalog Reply* message containing the objects it deems relevant, fetched from its own LPC, and adjusted according to the available space on the requesting server. The replies are tailor made to each subscription catalog by cross referencing the received map of subscriptions C, and the *LPC*. We optimize the provisioning process by considering the subscription count for each topic, which reflects the popularity or relevance of that topic. This allows us to allocate space more effectively within the reply, ensuring that popular topics receive more entries, while less popular ones may receive fewer or no entries at all. This dynamic allocation ensures that the most relevant content is prioritized during transmission.

Additionally, in heterogeneous deployments, the *RPC* size for each server may differ between servers. This can, naturally, lead to situations where a region sends more data than the destination can accommodate, potentially overloading the network for no gain. To mitigate this, we adjust the data transmission based on the current capacity of *RPC* at the destination server, trimming excess entries to fit within the cache's limits.

Phase 4 - Affinity Oriented Tuning This final phase addresses the affinity objective by sending additional content through extra *Subscription Catalog Reply* messages to the regions that share a higher affinity with the current. The reasoning behind this approach is to address the interests of specific regions by broadcasting content that could never be considered popular enough to be sent. So, these additional messages broaden the scope of selection beyond the LPC; the sole source for the initial reply message. This requires reevaluating the object info to determine if there is content of interest in the region that has not been previously sent. The objects already sent are naturally excluded.

The resulting new content may include objects that have not made to the LPC and, hence, must be retrieved from the region's nodes on-the-fly. Therefore, in order for this solution to be useful, a balance must be struck between processing time, network resource utilization, and epoch timing and object dissemination. The affinity degree (A) is defined to capture this balance. It is computed at the source, sent in the *Subscription Catalog Dissemination* messages, and can assume only three values:

(0) $\rightarrow$ Low affinity: no additional messages are sent.
(1) $\rightarrow$ Regular affinity: one additional message is sent at the midpoint of the epoch $\left(\frac{T}{2}\right)$.
(2) $\rightarrow$ High affinity: two additional messages are sent; at the $\left(\frac{T}{3}\right)$ and $\left(\frac{2 \cdot T}{3}\right)$ marks.

The A value for each peers is derived from a ranking table computed during the epoch. Peers are first ranked based on the total number of downloads for items originating from their region. Then, during the epoch, each time an item is moved from the RPC to the RCC, the ranking of the peer from which the item originated is incremented. When its time to compute A for each peer, a prefix sum of the ranking table is done and two thresholds determined: 30% and 50% of the total downloads. Peers whose combined download counts fall below or equal the 30% threshold are assigned a A value of 2. Peers whose combined download counts fall below or equal the 50% threshold but above the 30% threshold are assigned a A value of 1. The remaining peers are assigned a A value of 0. For example, consider ten peer edge servers with the following download counts:

$$\{e_0 : 6, e_1 : 10, e_2 : 5, e_3 : 15, e_4 : 6, e_5 : 3, e_6 : 18, e_7 : 6, e_8 : 9, e_9 : 12\}$$

After ordering by download counts, we get:

$$[e_6 : 18, e_3 : 15, e_9 : 12, e_1 : 10, e_8 : 9, e_0 : 6, e_4 : 6, e_7 : 6, e_2 : 5, e_5 : 3]$$

The prefix sum of these counts is $[18, 33, 45, 55, 64, 70, 76, 82, 87, 90]$. Hence, with a total number of downloads of 90, the 30% and 50% thresholds are, respectively, 27 and 45, meaning that only the highest ranked peer (e_6) is assigned a A value of 2, and the next two (e_3 and e_9) are assigned a A value of 1.

5 Heuristic-based Replica Selection

The problem to solved in the *Replica Placement Phase* can be defined as follows: given a finite set of items $\mathcal{I} = \{1, 2, \ldots, n\}$, where each item $i \in \mathcal{I}$ has an associated size s_i and regional popularity p_i, the objective is to select a subset $S \subseteq \mathcal{I}$ that:

$$maximize \quad \sum_{i \in S} p_i \qquad \text{with constraint} \sum_{i \in S} s_i \leq capacity(e)$$

where $capacity(e)$ represents the storage capacity of edge server e. This formulation aligns with the well-known *0-1Multiple Dimension Knapsack Problem (MKP))*, a class of NP-Hard problems extensively studied in the literature [5, 6, 16].

In addition to size and value constraints, another critical factor must be considered: *time*. Our solution must operate not just once but across a series of closely spaced epochs. Hence, it is neither feasible nor necessary to seek a perfect solution at each run; a near-optimal solution will suffice. Consequently, we focus on heuristic algorithms, renowned for their efficiency in producing high-quality approximate solutions within reasonable time constraints.

Metaheuristic techniques are recognized for their effectiveness in addressing NP-Hard combinatorial optimization problems [8]. For instance, the University Course Timetabling Problem (UCTP) [7] exhibits structural and complexity similarities with the Knapsack Problem, making it a relevant domain for applying these methods.

Metaheuristic algorithms generally fall into two broad categories [1]. *Population-based algorithms*, such as GAs, initiate with a pool of potential solutions and explore the search space by combining and evolving these solutions over time. These methods tend to explore larger areas of the solution space and are less prone to getting stuck in local optima, although they may lack precision. On the other hand, *local search algorithms*, such as TS, begin with a single solution (often randomized) and iteratively refine it by evaluating

incremental improvements based on a fitness function. While local search methods offer more precision, they are more susceptible to getting trapped in suboptimal regions of the search space. Given the strengths and weaknesses of both approaches, hybrid methods that leverage the strengths of each are often preferred for complex problems like the MKP and UCTP [1, 3, 7].

Motivated by these insights, we propose a hybrid algorithm combining GA and TS to effectively address the replica selection problem. This approach leverages the exploratory capabilities of GAs and the exploitative precision of TS, aiming to achieve superior solution quality and computational efficiency.

5.1 Genetic Algorithm

GAs are inspired by the principles of natural selection and genetics. They are highly effective for solving optimization problems where the search space is vast and multimodal. GAs operate by evolving a population of candidate solutions through processes of selection, crossover, and mutation, gradually enhancing the population's overall fitness.

A pivotal component of GAs is the *fitness function*, which evaluates how well a solution meets the problem's objectives. The fitness function guides the selection process by assigning higher probabilities of survival to fitter solutions. Consequently, GAs iteratively refine the population towards optimal or near-optimal solutions.

For the Knapsack Problem, where the goal is to maximize the total popularity of selected items within a fixed storage capacity, the fitness function evaluates the quality of a solution based on its total popularity and the size of the selected items. In our specific scenario, considering the storage capacity of an edge server e, denoted by $capacity(e)$, and the set of selected items S, where each item $i \in S$ has a popularity value p_i and a size s_i, the fitness function $\text{Fitness}(e, S)$ is defined as:

$$\text{Fitness}(e, S) = \begin{cases} \sum_{i \in S} p_i & \text{if} \quad \sum_{i \in S} s_i \leq capacity(e), \\ 0 & \text{otherwise.} \end{cases}$$

This formulation ensures that solutions exceeding the storage capacity ($capacity(e)$) receive a fitness score of zero, effectively penalizing infeasible solutions and guiding the Genetic Algorithm towards feasible regions of the search space.

5.2 Tabu Search

Unlike GAs, TS operates on a single solution, iteratively improving it by exploring neighboring solutions. To prevent cycling back to previously visited solutions, TS utilizes a memory structure known as the *tabu list*, which stores recent moves or solutions, rendering them temporarily inaccessible. Additionally, *aspiration criteria* allow the search to override tabu restrictions if a move results in a solution that surpasses the best previously found.

TS leverages both short-term and long-term memory structures to maintain diversity and prevent premature convergence. These memory structures guide the search by retaining valuable information about previously explored regions, ensuring a more thorough examination of the solution space.

The efficacy of TS fundamentally depends on the precise definition and modeling of the evaluation function. The *evaluate* function assesses the quality of a solution $S \subseteq \mathcal{I}$ (where $\mathcal{I}$ is the set of all possible items) by calculating the total weight and value of items selected for the knapsack. The weight and value are computed as follows:

$$\text{Weight}(S) = \sum_{i \in S} s_i \qquad \text{Value}(S) = \sum_{i \in S} I(n_i) \cdot p_i$$

where s_i is the size (weight) of item i, p_i is the popularity of item i, n_i denotes the namespace to which item i belongs, and, lastly, $I(n_i)$ represents the interest level associated with namespace n_i.

When a move $M = (i, o)$, where i is an entry item and o is an exit item, is applied, the updated weight and value of the new solution S' are given by:

$$\text{Weight}(S') = \text{Weight}(S) + s_i \cdot \mathbb{1}_{\{i \notin \text{ns}(S)\}} - s_o \cdot \mathbb{1}_{\{\text{count}(o,S) \geq 2\}}$$
$$\text{Value}(S') = \text{Value}(S) + I(n_i) \cdot p_i - I(n_o) \cdot p_o$$

where $\text{ns}(S)$ returns all namespaces to which objects in S belong and $\mathbb{1}_{\{condition\}}$ is 1 if $condition$ is true, and 0 otherwise. If the updated weight $\text{Weight}(S')$ exceeds the maximum capacity $W_{\max}$, the solution is penalized by setting $\text{Value}(S') = -\infty$ to discourage infeasible solutions.

5.3 Hybrid Approach

The hybrid algorithm, presented in Algorithm 1, combines the exploratory strengths of TS with the evolutionary optimization capabilities of GAs. This combination aims to enhance solution quality and computational efficiency by leveraging the complementary advantages of both heuristic approaches.

The algorithm begins with a pre-processing step, on which it initiates TS with the initial search space S to obtain a preliminary solution (*initialSolution*) based on the region's interests P, defined as a map from namespaces to sets of *object info* entries, containing the size and popularity of the objects published.

To mitigate the tendency of TS to get trapped in local optima, while ensuring diversity and the potential for better solutions, the algorithm, at line 2, evaluates the initial solution to identify excluded objects, referred to as *leftOutObjects*, which may present opportunities for improving the solution. The selection process involves identifying the least represented namespaces within the solution. From these namespaces, α% of their objects are chosen in a round-robin fashion, prioritizing from the least popular to the most popular, until the initial population size of the GA is no more than β% of the LPC. Besides overcoming the limitations of TS, this approaches also ensures that the initial solution for the GA remains manageable in size, making it solvable within the resource constraints of the edge servers. Both α and β configurable at deployment time. From our experiments, we have fine-tuned both to 25%. This means that 25% of the objects from the least represented namespaces are chosen and the initial population size of the GA is restricted to 25% of the LPC.

Algorithm 1 Hybrid TS and GA

```
1: function HybridAlgorithm(S,P)
2:     initialSolution ← TabuSearch(S,P)
3:     leftOutObjects ← IdentifyLeftOutObjects(initialSolution, S)
4:     augmentedSpace ← initialSolution ∪ leftOutObjects
5:     return GA.rank(augmentedSpace, P)
6: end function
```

The algorithm proceeds at line 4 by incorporating these additional objects into the initial solution, thereby expanding the search space. The integration works in the same way as the *mutation* step in GAs. Finally, at line 5, the GA is applied over this augmented search and the best solution from the optimized population is returned as the function's result.

6 Evaluation

In this section, we evaluate our proposal Paprika with the aim to answer the following questions:

- How fast do Paprika solutions converge on a solution and how do they use the available space?
- What are the quality of deployments provided by Paprika?
- Does the *Affinity Oriented Tuning* phase enhance data discoverability?

To achieve this, we developed a simulation environment that leverages: 1) the PureEdgeSim framework [10] to run the (actual) Java code of the edge servers and simulate their interactions, and 2) a previously developed trace-based simulator for mobile devices [11], which defines the operations performed by the set of mobile devices that belong to a region. Hence, in each experiment we deploy a PureEdgeSim setup that runs 1 or more interconnected edge servers, each asynchronously running the trace-based device simulator.

For each experiment, we define the number of regions, the capacity of the edge servers, the number of mobile devices, the number of namespaces and, distributions for data publication and subscription.

Publication Distribution defines the distribution of publications across namespaces within each region. It indicates the total number of objects in a region and how many of these belong to a particular namespace. For a region R, this distribution is denoted as $P^R\langle x_0, ..., x_n\rangle$, with $0 \leq x_i \leq 1, \forall i \in [0, n] \subset \mathbb{Z}$ representing the percentage distribution of each namespace in R. For example, $P^R\langle 0.1, 0.5, 0.4\rangle$ denotes a region R comprising three namespaces, with 10% of the objects belonging to Namespace 0, 50% to Namespace 1, and 40% to Namespace 2.

Subscription Distribution defines the distribution of subscriptions across namespaces within each region, i.e., the popularity of each namespace. It is denoted as $S^R\langle x_0, ..., x_n\rangle$, with $0 \leq x_i \leq 1, \forall i \in [0, n] \subset \mathbb{Z}$. representing the percentage distribution for each namespace in R.

6.1 Replica Selection

These set of experiments evaluate the performance between TS, GA and the Hybrid algorithms under varying server capacities, representative of edge computing environments, where both memory and computational resources are constrained.

After a series of parameterization tests, we identified that the optimal configuration for the selectors in the GA is the combination of the *Monte Carlo* parent selector and the *Roulette* child selector. This pairing has consistently demonstrated the highest effectiveness for our specific use case. However, even with this configuration, our initial findings are that GA alone frequently struggles to generate good solutions within the limited time frame of an epoch. In our experiments, the GA was unable to produce a solution unless the cache capacity was at least half the size of the total object set. Notably, the largest test case we considered featured a cache size

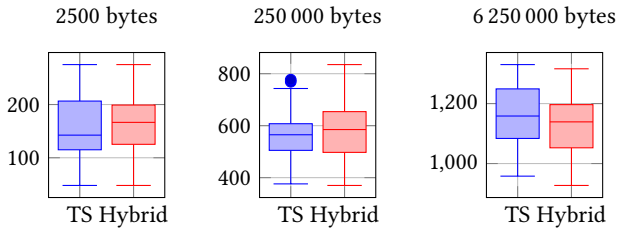


Figure 1: TS vs. Hybrid: Popularity score (# Downloads)

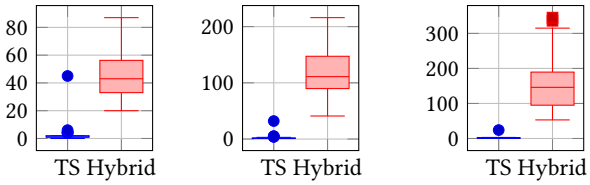


Figure 2: TS vs. Hybrid: Execution time (ms)

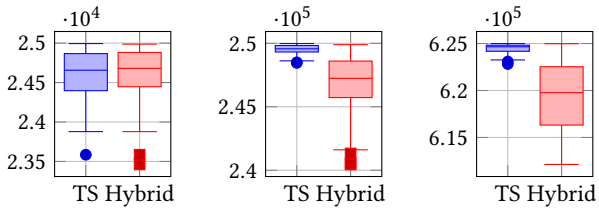


Figure 3: TS vs. Hybrid: Space used (bytes)

approximately one-quarter of the object size. Consequently, we reached the conclusion—aligned with the findings of [7]—that GA is unable to yield an optimized solution without some form of preprocessing, which sustains the development of the hybrid approach.

Given such conclusion, in this section we limit our experiments to the TS and the Hybrid algorithms, focusing on three primary metrics: *Score* (solution quality), *Execution Time*, and *Space* (memory usage in bytes). The results presented are based on 50 independent runs, conducted with 1 000 objects ranging from 500 to 50 000 bytes in size, and three server capacity configurations: 2 500, 250 000 and 6 250 000 bytes.

As illustrated in Figure 1, the findings consistently highlight the superiority of the Hybrid approach in terms of solution quality across the tested capacities. In particular, for the smallest capacity (2 500 bytes), the Hybrid method significantly outperformed TS, producing consistently better solutions. This advantage remained as server capacity increased, though the performance gap between the two algorithms narrowed. By the time we evaluated the 6 250 000-byte capacity, both approaches produced similar scores, with the Hybrid method losing a slight edge due to the bigger search space.

In terms of *Execution Time*, however, the TS consistently outperformed the Hybrid method across all configurations. As shown in Figure 2, the execution time for TS was significantly lower, particularly for smaller problem sizes, where it completed in mere milliseconds. In contrast, the Hybrid approach demonstrated much higher execution times, particularly in larger configurations. This variation in runtime reflects the increased complexity and overhead

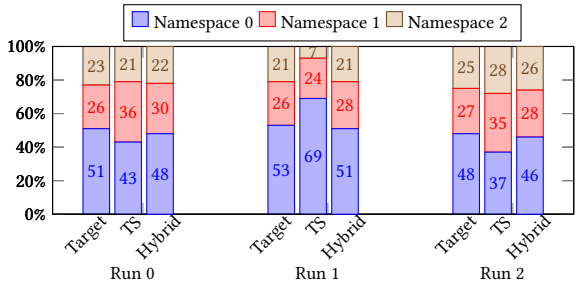


Figure 4: Three runs of the TS and Hybrid algorithms across three namespaces and 25 000 bytes of storage capacity.

of the Hybrid algorithm, which seeks higher solution quality at the expense of computational time.

In terms of *Space*, both the TS and Hybrid algorithms demonstrated the capacity to effectively utilize the available memory, a crucial factor in edge computing environments where maximizing data transfer is paramount. However, important distinctions emerge between the two approaches, particularly when memory availability is constrained, as evidenced in Figure 3. In memory-constrained environments, the Hybrid algorithm consistently demonstrates better memory efficiency and higher solution quality compared to TS, particularly in smaller memory configurations. However, as memory availability increases and the search space grows, Hybrid's performance diminishes, sometimes failing to provide optimal solutions. In contrast, TS shows greater stability across various memory configurations, though with lower overall solution quality.

Figure 4 shows the result for three runs of the algorithms. For each run, we present the target (perfect) score, this is, the number of downloads by namespace for that run and, next to it, both the TS and the Hybrid score. The TS algorithm produces competitive results. However, it is vulnerable to becoming trapped in local optima. This is particularly evident in the second run, where TS deviated by 16% from the perfect target for Namespace 0. Despite also being susceptible to this problem when computing the initial solution, the Hybrid algorithm consistently mitigated such shortcomings in the next phases. Hence, while TS achieved commendable scores, the Hybrid approach regularly surpassed it by integrating corrective mechanisms, leading to a more refined and accurate solution.

In conclusion, the Hybrid algorithm demonstrates superior performance in terms of solution quality, especially in smaller, resource-constrained environments typical of edge computing. However, this improvement in solution quality comes at the cost of increased execution time and variable memory consumption. TS, while faster and more memory-efficient, produces lower-quality solutions, particularly in environments with strict capacity limits. Given these findings, the Hybrid method represents a balanced trade-off between computational efficiency and solution quality, making it a more suitable choice for edge computing applications where memory and processing power are limited, but solution quality is paramount.

6.2 Replica Placement

In this section, we examine how Paprika responds to the specific interests of distinct regions. To achieve this, we conducted an experiment across three regions, each characterized by unique content distributions and varying interests across three namespaces. Each region's server has a predefined storage capacity, while the total volume of available data exceeds the combined storage capacities of the regions by a factor of five.

This setup enables the standalone GA to generate a solution that requires prioritizing the most popular data, while excluding the least popular namespace from each region's local cache. As a result, each region's LPC cache holds only a small subset of the most popular data, leaving a significant portion of frequently accessed content outside the cache. By enforcing these constraints, we are able to rigorously evaluate the efficiency of Paprika in adapting to regional demands, while ensuring that the cache management strategies are aligned with the specific interests and content consumption patterns of each region. Accordingly, the publication and subscription distributions for the three regions are the following:

$$P^{R_0}\langle 70\%, 30\%, 0\%\rangle, \qquad S^{R_0}\langle 100\%, 0\%, 0\%\rangle,$$
$$P^{R_1}\langle 50\%, 25\%, 25\%\rangle, \qquad S^{R_1}\langle 50\%, 25\%, 25\%\rangle,$$
$$P^{R_2}\langle 30\%, 0\%, 70\%\rangle, \qquad S^{R_2}\langle 30\%, 0\%, 70\%\rangle.$$

In this framework, we analyze the cache contents of edge servers in the three regions, denoted as LPC_X for region X, and the data provisioned in each *Subscription Catalog Reply* message sent to region Y (SCR_Y). For the messages, the percentages indicate the portion of total message objects allocated to a specific namespace, representing the percentage of slots dedicated to that namespace.

We have that Region 0 has a subscription distribution that only requests content from Namespace 0, despite also existing publications in Namespace 2. As a result, its cache content and catalog replies follow distributions:

$$\text{LPC}_{R_0}\langle 100\%, 0\%, 0\%\rangle$$
$$\text{SCR}_{R_1}\langle 100\%, 0\%, 0\%\rangle \qquad \text{SCR}_{R_2}\langle 100\%, 0\%, 0\%\rangle$$

Region 1 maintains a balanced cache across all namespaces. It provides data based on the subscriptions of the other regions, leading to the following cache and catalog reply distributions:

$$\text{LPC}_{R_1}\langle 47\%, 28\%, 25\%\rangle$$
$$\text{SCR}_{R_0}\langle 100\%, 0\%, 0\%\rangle \qquad \text{SCR}_{R_2}\langle 20\%, 0\%, 80\%\rangle$$

Only Namespace 0 data is sent to Region 0, while for Region 2, 80% of the response corresponds to Namespace 2 content, aligning with its strong interest in that namespace.

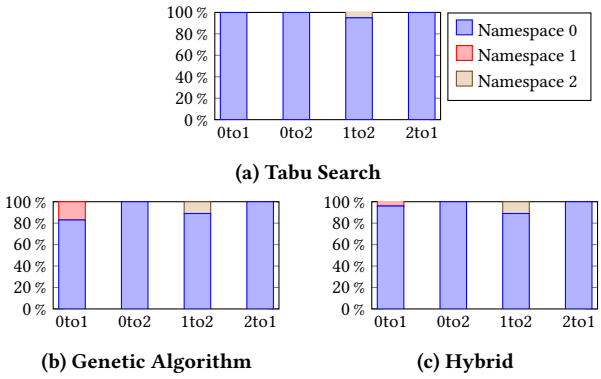
Region 2's subscription distribution shows a strong preference for content from Namespace 2. The catalog replies from Region 1 align with the subscription distribution of that region:

$$\text{LPC}_{R_2}\langle 35\%, 0\%, 65\%\rangle$$
$$\text{SCR}_{R_0}\langle 100\%, 0\%, 0\%\rangle \qquad \text{SCR}_{R_1}\langle 30\%, 0\%, 70\%\rangle$$

This experiment illustrates how cache contents influence response behavior and highlights the diverse content preferences and availability across the regions.

6.3 Affinity Tuning

This study investigates the performance of a proposed hybrid algorithm in enhancing the discoverability of niche namespaces within a constrained storage environment, in comparison to traditional TS and GA methods. We want to assess if (1) the hybrid approach



(a) Tabu Search

(b) Genetic Algorithm **(c) Hybrid**

Figure 5: Affinity Tuning

provide measurable advantages over standalone algorithms and (2) Affinity Tuning enhance the visibility of niche namespaces. To that end, we build on the example of the previous section, setting the affinity degree A to 1 between all regions, and analyzing the contents of the complementary *Subscription Catalog Reply* messages.

Region 0 is only interested in Namespace 0 and, as expected, only received content for such namespace from other regions. However it managed to serve other region's interests, for example Region 1 that was interested in content from Namespace 1, which Region 0 had. For such cases, the results shown in Figure 5 reveal significant differences between the algorithms.

The performance of TS (Figure 5a) shows limitations in overcoming local optima, particularly when storage constraints lead to the omission of a namespace. Conversely, GA (Figure 5b) managed to provide a response more aligned with the remote user's interests, but as seen in prior experiments, its performance is inconsistent in more realistic, uncontrolled scenarios. When the workload increases to reflect real-world conditions, GA often fails to produce a valid solution. The hybrid approach (Figure 5c), however, demonstrated greater adaptability by dynamically adjusting cache contents, particularly in Region 2, which has a strong demand for Namespace 2 content. This algorithm successfully overcame the limitations of TS and produced valid solutions that more accurately reflected the remote user's interests.

We may thus conclude that the hybrid approach not only outperforms the TS and GA algorithms but also expands the discoverability of niche namespaces in constrained environments. The Affinity Tuning mechanism achieved a more balanced and accurate data distribution, especially in scenarios where regions exhibited diverse namespace interests. This confirms that the hybrid solution is better suited to complex multi-region systems, where affinity-based adjustments are crucial for improving overall system efficiency and responsiveness to user needs.

7 Conclusion

In this paper, we proposed Paprika, an novel online heuristic-based hybrid algorithm designed to enhance data replica placement in edge computing environments. By considering data popularity, regional affinities, and server storage constraints, and by effectively combining the strengths of GA and TS, Paprika addresses the challenges associated with diverse user interests in multi-region edge publish-subscribe systems, ensuring a more tailored placement of objects in the regions' edge servers.

The complementary relationship between GA and TS enhances both the speed and quality of the optimization process, making this hybrid approach particularly effective. Performance evaluations demonstrate that Paprika outperforms traditional heuristic methods, achieving a better balance between user interests across regions, while prioritizing regions with higher affinity.

We can thus conclude that Paprika makes a meaningful contribution to the ongoing efforts to improve data management at the network edge, helping to enhance user experiences in dynamic and diverse environments.

Acknowledgments

This work is supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP.

References

[1] Mohammed Azmi Al-Betar and Ahamad Tajudin Khader. 2012. A harmony search algorithm for university course timetabling. *Ann. Oper. Res.* 194, 1 (2012), 3–31.
[2] Atakan Aral and Tolga Ovatman. 2018. A Decentralized Replica Placement Algorithm for Edge Computing. *IEEE Trans. Netw. Serv. Manag.* 15, 2 (2018), 516–529.
[3] Vincent Barichard and Jin-Kao Hao. 2003. Genetic tabu search for the Multi-Objective knapsack problem. *Tsinghua Science and Technology* 8, 1 (2003), 8–13.
[4] Leonardo Epifâneo. 2021. *Cathode: A Consistency Aware Data Placement Algorithm for the Edge.* Master's thesis. University Of Lisbon.
[5] Fred W. Glover. 1989. Tabu Search - Part I. *INFORMS J. Comput.* 1, 3 (1989), 190–206.
[6] Fred W. Glover. 1990. Tabu Search - Part II. *INFORMS J. Comput.* 2, 1 (1990), 4–32.
[7] Sadaf Naseem Jat and Shengxiang Yang. 2011. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. *J. Sched.* 14, 6 (2011), 617–637.
[8] Rhydian Lewis. 2008. A survey of metaheuristic-based techniques for University Timetabling problems. *OR Spectr.* 30, 1 (2008), 167–190.
[9] Chunlin Li, Jingpan Bai, and Tang Jianhang. 2019. Joint optimization of data placement and scheduling for improving user experience in edge computing. *J. Parallel Distributed Comput.* 125 (2019), 93–105.
[10] Charafeddine Mechalikh, Hajer Taktak, and Faouzi Moussa. 2021. PureEdgeSim: A simulation framework for performance evaluation of cloud, edge and mist computing environments. *Comput. Sci. Inf. Syst.* 18, 1 (2021), 43–66.
[11] João A. Silva, Filipe Cerqueira, Hervé Paulino, João M. Lourenço, João Leitão, and Nuno M. Preguiça. 2021. It's about Thyme: On the design and implementation of a time-aware reactive storage system for pervasive edge computing environments. *Future Gener. Comput. Syst.* 118 (2021), 14–36.
[12] João A. Silva, Pedro Vieira, and Hervé Paulino. 2020. Data Storage and Sharing for Mobile Devices in Multi-region Edge Networks. In *International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM 2020).* 40–49.
[13] Vasilis Sourlas, Paris Flegkas, Georgios S. Paschos, Dimitrios Katsaros, and Leandros Tassiulas. 2011. Storage planning and replica assignment in content-centric publish/subscribe networks. *Comput. Networks* 55, 18 (2011), 4021–4032.
[14] Vasilis Sourlas, Lazaros Gkatzikis, Paris Flegkas, and Leandros Tassiulas. 2013. Distributed Cache Management in Information-Centric Networks. *IEEE Trans. Netw. Serv. Manag.* 10, 3 (2013), 286–299.
[15] Vaibhav Tiwari, Chandrasen Pandey, Abisek Dahal, Diptendu Sinha Roy, and Ugo Fiore. 2024. A Knapsack-based Metaheuristic for Edge Server Placement in 5G networks with heterogeneous edge capacities. *Future Gener. Comput. Syst.* 153 (2024), 222–233.
[16] Pamela H. Vance. 1993. Knapsack Problems: Algorithms and Computer Implementations (S. Martello and P. Toth). *SIAM Rev.* 35, 4 (1993), 684–685.
[17] Xinliang Wei and Yu Wang. 2023. Popularity-Based Data Placement With Load Balancing in Edge Computing. *IEEE Trans. Cloud Comput.* 11, 1 (2023), 397–411.
[18] Junjie Xie, Chen Qian, Deke Guo, Xin Li, Shouqian Shi, and Honghui Chen. 2019. Efficient Data Placement and Retrieval Services in Edge Computing. In *IEEE International Conference on Distributed Computing Systems, ICDCS 2019.* 1029–1039.
[19] Boyang Yu and Jianping Pan. 2015. Location-aware associated data placement for geo-distributed data-intensive applications. In *IEEE Conference on Computer Communications, INFOCOM 2015.* IEEE, 603–611.