

Minimax Optimization for Adaptive Cache Orchestration in Edge-Fog-Cloud Streaming Microservices

OLHA SHPUR¹, MARIAN SELIUCHENKO¹, IVAN DEMYDOV¹, MYKHAILO KLYMASH¹, SVITLANA SACHENKO²

¹Department of Information and Communication Technology, Lviv Polytechnic National University, Lviv, 79013, Ukraine

²Department for Management and Administration, Ivano-Frankivsk Educational and Scientific Institute for Management, West Ukrainian National University, Ternopil, 46009 Ukraine

Corresponding author: Olha Shpur (e-mail: olha.m.shpur@lpnu.ua).

This research was supported by the Ukrainian government project #0126U001332 “Development of the Physically Consistent Multimodal AI Models for Robotic Systems”.

ABSTRACT Streaming microservice applications in multi-level Edge-Fog-Cloud infrastructures require low content delivery latency despite limited cache resources at peripheral nodes. This paper proposes a minimax optimization model for adaptive balancing of local and remote caching in streaming applications. Unlike existing request-oriented approaches, the method integrates streaming traffic dynamics, topological user affinity, and volatile edge cache consumption into a unified optimization criterion. The solution co-optimizes microservice and mobile cache placement to minimize the worst-case value of a composite objective function. A modified genetic algorithm (MGA) is introduced with tournament selection, two-point crossover, stochastic encoding, adaptive penalty handling, and a fitness stagnation stopping criterion. Experimental evaluation included Python/NumPy simulations with Zipf-distributed requests and a simulation software complex under realistic Edge and Fog constraints. The proposed MGA reduced average edge cache utilization from 61.77% to 34.84%, decreased peak cache load by 51.84%, and lowered mean delivery latency from 247.3 ms to 158.4 ms. The share of requests satisfying the 200 ms QoS threshold increased from 41.1% to 78.9%.

KEYWORDS cache placement; microservices; Edge-Fog-Cloud; minimax optimization; modified genetic algorithm; NSGA-II; QoS; dynamic orchestration.

I. INTRODUCTION

The contemporary development of information and communication technologies is accompanied by a rapid growth in the number of connected devices and in the volume of data generated by streaming applications. These include real-time services such as video analytics, monitoring systems, intelligent transportation systems, industrial IoT platforms, and systems involving virtual and augmented reality. Such systems are characterized by high data arrival rates, continuous processing, and strict requirements on latency, reliability, and quality of service.

Traditional centralized cloud architectures prove insufficient for supporting these scenarios due to significant data transmission delays, communication channel congestion, and the limited scalability of data centers. This has driven the active development of hybrid computing models, in particular the Edge, Fog, and Cloud paradigms, which distribute data processing according to spatial proximity to the data sources. Within such architectures, the key challenge is no longer

limited to the distribution of computational workload – it also encompasses the placement of auxiliary processing components, among which caches play a particularly important role. In microservice applications, caching has traditionally been viewed as a mechanism for reducing data-access latency and for offloading services. However, in streaming microservice systems, the cache ceases to be a static repository and instead acquires the characteristics of a separate managed resource that can be dynamically deployed, migrated, or replicated across different parts of the infrastructure.

Unlike approaches that focus on the optimization of computational components or on the application of static caching strategies, in streaming environments the problem of optimal cache placement relative to the data sources and the consumers of processed results becomes critical. This is because intermediate computation results in streaming pipelines are actively reused by downstream microservices, and their availability directly affects the processing latency of the entire stream. The problem is especially relevant under the

resource constraints of edge nodes, where available memory, memory access speed, and network bandwidth are all limited. Under such conditions, a balance must be struck between local caching – which minimizes data-access latency – and remote caching, which uses scarce resources more efficiently but may increase access time due to network delays. Consequently, choosing the optimal cache location – local, remote, or hybrid – becomes a complex problem that depends on workload characteristics, the topology of microservice interactions, data access frequencies, network parameters, and QoS/QoE requirements. In this context, the cache must be regarded as an active element of the orchestration system, whose placement should be adaptively optimized together with microservice deployment.

A. PROBLEM STATEMENT

With the continued evolution of computing technologies and the growth of data volumes in streaming microservice systems, cache optimization and load distribution have become central problems for ensuring high performance and minimizing latency. The conceptual foundations for partitioning computation between the cloud, fog, and edge levels were laid in [1,2], which argued for moving processing closer to the data source in order to achieve low latency, geographic/topological awareness, and support for real-time streaming applications. Building on these ideas, a broad line of research has emerged that focuses on optimizing the placement of computational components across multi-level hierarchical architectures.

Subsequent studies [3-6] address the optimization of container and microservice placement in Edge/Fog/Cloud environments. For instance, Sharma *et al.* [3] proposed multi-objective container-allocation strategies that minimize resource usage, network traffic, and latency.

Guerrero *et al.* [4] proposed a service placement model for fog infrastructures using evolutionary algorithms – WSGA, NSGA-II, and MOEA/D. Experiments on random topologies showed that NSGA-II provides the best Pareto-quality solutions, MOEA/D is the fastest, and WSGA offers no significant advantage. Despite these strengths, the approach has several limitations. First, the model does not account for data caching or the management of local and remote caches. Services are treated as isolated computational units, with no analysis of how data are stored and reused, which makes it impossible to evaluate the trade-off between memory usage and system performance. Second, the experiments rely on static topologies with a fixed number of services, which does not reflect the dynamic nature of streaming microservice applications. Third, the approach does not integrate cache-efficiency metrics such as hit rate, cached data volume, access latency, or network bandwidth utilization, which limits its applicability in systems where low-latency access to intermediate streaming results is critical.

Studies [7-12] have largely focused on cache optimization for static or request-oriented systems, in which cache accesses are discrete and independent events. The situation in streaming microservices is fundamentally different: the cache is inherently dynamic, and its state evolves over time depending on the rate of data arrival, the achieved processing throughput, and the underlying microservice-orchestration mechanisms (e.g., Kubernetes or Istio). As a result, classical policies such as LRU, LFU, FIFO, or ARC do not provide optimality under dynamic streaming workloads. This motivates the development of an adaptive caching-strategy model that is able to:

- predict the efficiency of local and remote caching for different stream configurations within a microservice architecture;
- dynamically select the cache location during microservice deployment or scaling;
- minimize the worst-case (maximum admissible) trade-off between memory consumption and performance loss, while preserving the stability of stream processing.

Balancing – or, equivalently, predicting the relative efficiency of remote versus local caching – for streaming applications under a minimax memory/performance criterion is a challenging optimization problem. Its solution must account for the trade-off between memory consumption and operational efficiency within an information and communication system, particularly for applications that process large volumes of data in real time. Caching must therefore be optimized so that:

- system memory consumption is minimized (subject to resource constraints such as available RAM);
- system performance is maximized, which in turn implies minimizing latency and maximizing the data-access throughput.

Two principal caching strategies are typically considered:

- local caching, which stores data on the client side or at resources close to the request sources. This enables fast access but significantly increases memory consumption at every node;
- remote caching, which stores cached data on remote servers or in the cloud. This is advantageous when large data volumes must be retained, but it may introduce additional latency due to network requests and increased network traffic.

From the standpoint of cache access in microservice systems, we have previously identified two principal approaches in our preliminary work [13-15]. In the first approach, a microservice uses a local cache located on the same compute node (Edge) on which the service runs. In our formulation, the local cache is constructed not as an isolated store dedicated to a single microservice, but as a shared resource for a group of services deployed on the same node. Specifically, when a predefined RAM-utilization threshold is exceeded, the contents of the individual microservice caches are analyzed to identify duplicated data. The detected shared content is migrated to a unified local cache and then removed from the individual caches. The local cache thus operates on a “one-for-many” principle, which avoids data duplication and reduces edge-node memory consumption without sacrificing data availability. After migration, the microservices route client requests directly to the shared cache, falling back to the originating service only when the data are not found in the cache.

The second approach uses a remote cache, formed by migrating the shared cache from the Edge level to the Fog level of the infrastructure. This migration is triggered when the CPU-load threshold on the corresponding edge node is exceeded, which indicates excessive computational load associated with cache-request handling. The previously formed local cache is then relocated to Fog nodes, where it functions as a remote shared store for a group of edge nodes. After relocation, all microservices update their data-access configuration and forward subsequent client requests to the remote cache. This approach reduces the load on edge resources by offloading cache management to a higher infrastructure level while still

enabling shared use of cached data across multiple services and nodes.

Taken together, these approaches realize a dynamic cache-management model in which the cache location is determined by the current state of infrastructure resources (memory and CPU), and the cache itself is regarded as a mobile, managed object capable of adaptively changing its placement to maintain an optimal balance between data-access latency and resource utilization.

In this context, the optimization problem is formulated under a minimax criterion that bounds the worst-case values of system metrics. Specifically, the goal is to:

- maximum memory utilization across infrastructure nodes;

- minimize the access latency to cached data.

This formulation ensures stable system behavior under non-uniform load and constrained resources.

Summarizing the above, a streaming microservice system can be represented as a multi-level hierarchical Edge-Fog-Cloud infrastructure (Figure 1), in which microservices are deployed on edge nodes and the cache is treated as a separate logical resource whose location can vary with the system state. In this model, the local cache is formed at the Edge level as a shared store for a group of microservices, whereas the remote cache is instantiated at the Fog level through migration of the shared cache when edge resources become overloaded.

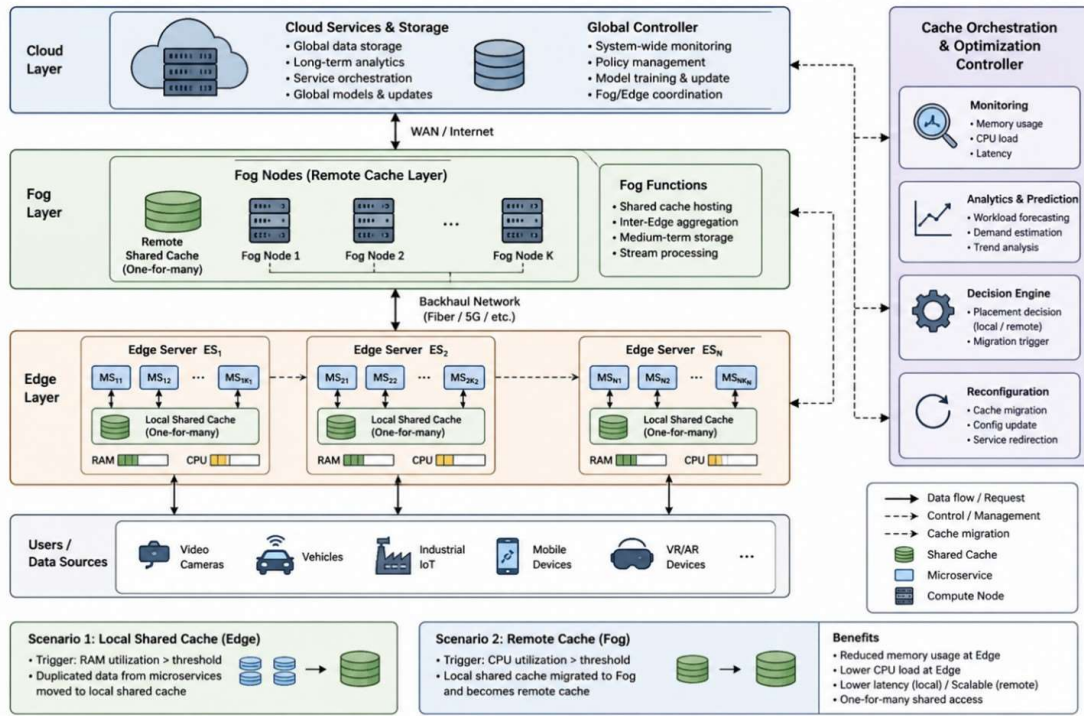


Figure 1. Multi-level hierarchical Edge-Fog-Cloud architecture with dynamic cache placement across edge, fog, and cloud nodes

Under this scheme, access to cached data is performed dynamically: depending on the current cache placement, requests may be served locally or forwarded to a remote level. The cache location therefore directly affects both the data-access latency and the consumption of computational and memory resources

II. CACHE PLACEMENT OPTIMIZATION MODEL FOR STREAMING APPLICATIONS

Given the architecture and the local/remote cache-formation mechanisms described above, it is natural to formalize the problem as one of selecting the placement of cached data in a multi-level infrastructure subject to resource constraints and performance requirements. The goal of optimization is to determine a distribution of cached data between local (Edge) and remote (Fog) caches that minimizes the use of scarce memory resources while satisfying the data-access latency requirements of streaming applications.

Consider an edge-computing scenario with N edge servers (ES) and M users. Let us denote the set of edge servers $ES = \{es_1, es_2, \dots, es_N\}$, and the set of users

$MU = \{m_1, m_2, \dots, m_M\}$, and assume that user m_i is associated with the j -th edge server $m_i \in es_j$. Each user m_i uses a set of service types S , each composed K of a set of components $S = \{s_1, s_2, \dots, s_k\}$; the components correspond to different applications and their data, which may also be encapsulated as individual microservices s_k . If user m_i requires a component k , then $m_i \in s_k$ this is reflected in the placement variables. If the i -th service type is cached on the j -th ES, we denote this as $s_i \in es_j$ a placement variable. The size of data items varies for different service types (microservices s_k) that process them and is represented as the set $DS = \{d_{s_1}, d_{s_2}, \dots, d_{s_k}\}$.

ES capabilities, both computational and storage-related, are limited. Let each ES have a maximum memory budget available for the local cache M_{local} and M_{remote} for the

remote cache. The access (latency) cost for the k -th d_{s_k} element when read from the local cache is given $T_{local}(d_{s_k})$ by one parameter and from the remote cache $T_{remote}(d_{s_k})$ by another. Let the amount of system memory required to process an element d_{s_k} cached locally $S_{local}(d_{s_k})$ and remotely - $S_{remote}(d_{s_k})$ be denoted by the corresponding variables. For

$$f = \min_{\{x_h, y_h\}} \max \left(\sum_{h=1}^n x_h * S_{local}(d_{s_k}) + \sum_{h=1}^n y_h * S_{remote}(d_{s_k}), \sum_{h=1}^n x_h * T_{local}(d_{s_k}) + \sum_{h=1}^n y_h * T_{remote}(d_{s_k}) \right) \quad (1)$$

where h denotes the iteration of the optimization algorithm. That is, we minimize the maximum value of the objective function with respect to the following criteria:

- total memory utilization, including both the local (M_{local}) and remote caches (M_{remote});
- total cached-data access latency, including access time to both the local ($T_{local}(d_{s_k})$) and remote caches ($T_{remote}(d_{s_k})$).

The admissible solutions must satisfy the following constraints:

- memory capacity constraints;

$$\sum_{h=1}^n x_h * S_{local}(d_{s_k}) \leq M_{local}$$

$$\sum_{h=1}^n y_h * S_{remote}(d_{s_k}) \leq M_{remote}$$

- the constraint that each element d_{s_k} be stored in only one cache, for all iterations of the algorithm.

$$x_h + y_h = 1, \quad \forall h = 1, 2, \dots, N,$$

where N – the total number of iterations

A. GENETIC ALGORITHM FOR THE MULTI-DIMENSIONAL MINIMAX PROBLEM

The problem can be solved using a variety of optimization methods. Unfortunately, analytical solutions are extremely difficult to obtain due to the inherently combinatorial nature of the underlying processes. Given the large solution space and the need for step-by-step evaluation of the objective (fitness) function during search – together with possible adjustments as the optimization conditions change – genetic algorithms (GAs) are a natural choice for exploring this solution set [16-20].

The optimization problem is reformulated as follows:

- each candidate solution at iteration h of the GA is encoded as a chromosome $[x_h, y_h]$ that represents a distribution of elements d_{s_k} across the caches, consistent with the fitness function based on (1);
- the fitness function based on (1) must minimize the maximum values of memory utilization (M_{total}) and data-access latency (D_{total});
- genetic operators (crossover and mutation) generate new candidate solutions.

The genetic algorithm must therefore be adapted to solve

each element d_{s_k} , we must decide whether it is stored in the local cache (let such storage be denoted $x_h = 1$) or in the remote cache (let such storage be denoted $y_h = 1$). The optimization minimizes the maximum value of the objective function simultaneously with respect to memory utilization and latency. The objective function of the minimax problem is formulated as follows:

the multi-dimensional minimax problem based on (1), with the following input variables:

1.1. Access latency to the local cache $T_{local}(d_{s_k})$;

1.2. Access latency to the remote (fog or cloud) store - $T_{remote}(d_{s_k})$ – variables in class 1.X correspond to inverse-proportional system performance for streaming data processing.

2.1. Maximum capacity of the local cache - M_{local} ;

2.2. Maximum capacity of the allocated remote (cloud or fog) environment - M_{remote} – variables in class 2.X correspond to available system resources.

3.1. System memory required to process a locally cached element ($S_{local}(d_{s_k})$);

3.2. System memory required to process a remotely (cloud) cached element - $S_{remote}(d_{s_k})$.

Variables in class 3.X correspond to the combination of sequential and parallel processing of data elements in the system. We assume that the CPU power allocated for cloud processing is ten times higher than the local capacity. We also impose the logical caching constraint that any given element may be stored in the system at most once.

The chromosome encoding (i.e., the solution structure) of the genetic optimization algorithm is defined as follows: each chromosome element is a bit (or marker) corresponding to a specific data element:

- 1 – cache the element (locally or remotely);
- 0 – do not cache the element.

The most compact representation at step h uses at least two mutually exclusive feature bits per element - $[x_h, y_h]$, where $x_h = 1$ - one bit indicates local caching, another indicates - $y_h = 1$ - remote caching, and the absence of any “1” indicates that the element is not cached. Only one feature bit set to “1” is admissible per element.

The fitness function (minimax), under (1) and the above assumptions, is:

$$F = \min_h \max (M_{total,h}, D_{total,h}), \quad (2)$$

$$\text{where } M_{total} = \sum_h (S_{local}(d_{s_k}) * x_h + S_{remote}(d_{s_k}) * y_h),$$

$$D_{total} = \sum_h (T_{local}(d_{s_k}) * x_h + 0,1 * T_{remote}(d_{s_k}) * y_h).$$

The goal of the genetic algorithm is to minimize the fitness function (2) subject to the additional condition that the selected element is the closest available cacheable element d_{s_k} for the given user's service.

Let us define the constraints for the fitness function (2):

$$\begin{aligned} M_{total} &\leq M_{max}, \\ D_{total} &\leq D_{max}, \\ x_h + y_h &\leq 1. \end{aligned} \quad (3)$$

In these constraints for the fitness function condition $x_h + y_h \leq 1$ means that an element cannot be cached in two locations simultaneously.

For our optimization, the penalty function is defined as:

$$\begin{aligned} V_M &= \max[0, M_{max} - M_{total}], \\ V_D &= \max[0, D_{max} - D_{total}], \\ V &= V_M + V_D, \\ Penalty &= 0.15 * V. \end{aligned} \quad (4)$$

We adopt a penalty intensity coefficient of 15% in accordance with [18].

Violations of (3) are therefore penalized in the fitness function (2) by an additive penalty term based on (4):

$$F = \min_h \max(M_{total,h}, D_{total,h}) + Penalty. \quad (5)$$

The GA is implemented with a population of 50–200 candidate solutions, generated at random under constraints (3) and (4). Selection is performed via tournament selection [17–19]. We use two-point feature crossover, after which constraints (3) and (4) are verified. We adopt a low mutation rate of up to 5% in line with the recommendations of [21], in order to maintain solution diversity without destroying good solutions. The storage location of each element (local or remote) is determined stochastically in every cycle of the GA.

The stopping criterion of the proposed GA is the absence of any improvement of the fitness function (5), under constraints (3) and (4), for three consecutive iterations. We thereby identify the optimal cache-hosting node – according to our proposed approach – together with a verification that this node is as close as possible to the user (according to the principle of the Affinity Degree Determination Algorithm [22]).

A high-level block diagram of the algorithm is shown in Figure 2.

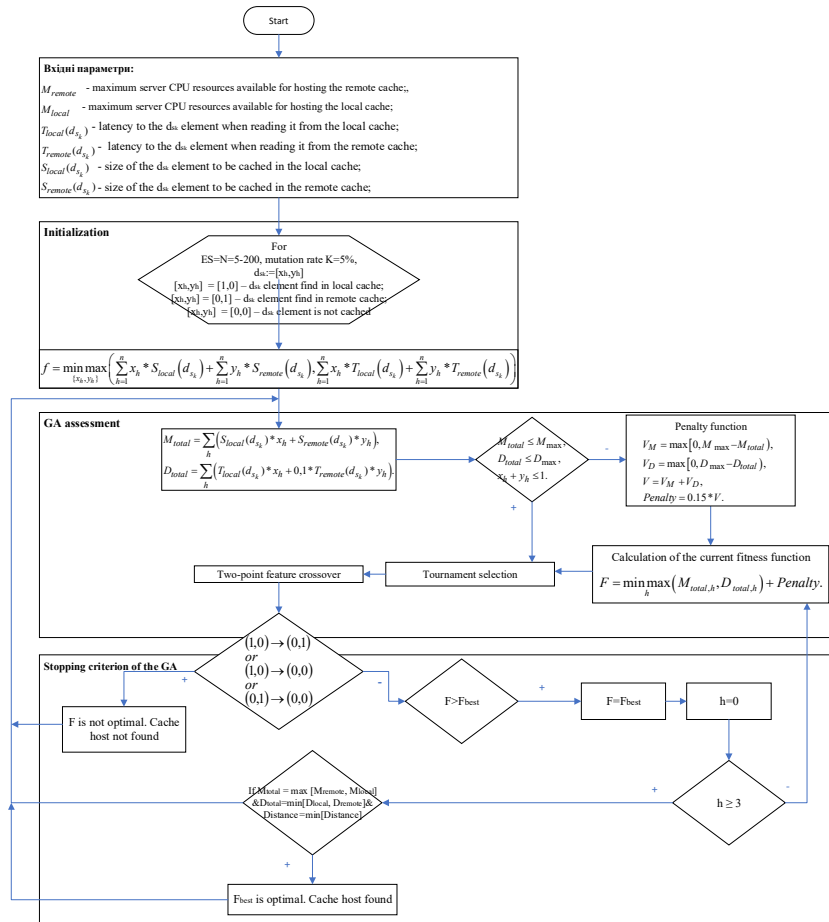


Figure 2. High-level block diagram of the modified genetic algorithm (MGA) for balancing local

III. EXPERIMENTAL EVALUATION

A. MATHEMATICAL MODELING OF THE PROPOSED ALGORITHM

To establish the effectiveness of the proposed genetic algorithm (GA) in solving the multi-dimensional minimax problem, we performed a comprehensive mathematical simulation. The software implementation was carried out in Python 3, using the NumPy library for matrix computations and Matplotlib for visualization.

The experiment is based on a three-level Edge–Fog–Cloud hierarchical architecture. To reproduce a realistic data-request pattern, we generated an input stream of 5,000 requests over 1,000 unique data elements. The content popularity distribution follows a Zipf law with skewness $\alpha = 1.0$, which is a standard approach for modeling caching processes in modern networks. In the context of the experiment, this parameter governs how often users access different microservices.

The computational infrastructure was modeled across three processing levels:

- Edge level: modeled as a set of nodes with variable memory capacity (NODE_CAP) in the range from 50 to 900 units, enabling evaluation of the algorithm’s behavior under resource scarcity.
- Fog level: served as a remote store (REM_CAP) with fixed capacity values (100, 200, and 400 units).
- Cloud level: assumed to provide unlimited resources, with compute power ten times higher than that of the local edge servers but with the highest access latency.

For adequacy assessment and comparative evaluation, we simulated both the proposed modified genetic algorithm (MGA) and the classical multi-objective NSGA-II. For both methods, the population size (total number of nodes) varied between 50 and 200 individuals. Selection was performed via tournament selection, and crossover via the two-point operator. The mutation rate was capped at 5%. The penalty intensity for memory-constraint violations or data duplication was 15%. A distinctive feature of the MGA is its stopping criterion based on the absence of objective-function improvement over three consecutive iterations, whereas NSGA-II was run for a fixed budget of 100 iterations to ensure convergence to the Pareto set. This approach made it possible to obtain statistically reliable data regarding the average latency (latency_avg), the number of cache hits (hits), and the load balancing across system nodes. The simulation results for the MGA are presented in Figures 3–5.

A key feature of the proposed algorithm is its dynamic penalty function (Penalty), which is integrated into the fitness function (5). Figure 5 shows that the penalty values oscillate within the range of 6,000 to 11,500 units. This amplitude indicates a high sensitivity of the algorithm to memory constraints $S_{local}(d_{s_k})$ and $S_{remote}(d_{s_k})$. A comparative analysis of Figures 4 and 5 reveals a direct correlation: a rise in penalty values coincides with an increase in cache misses, which confirms the adequacy of the chosen penalty model for inefficient cache usage.

The mean access latency across architecture levels (Figure 3) shows that using the fitness-function stopping criterion yields the lowest latency under tight memory budgets at the Edge level (up to 200 units).

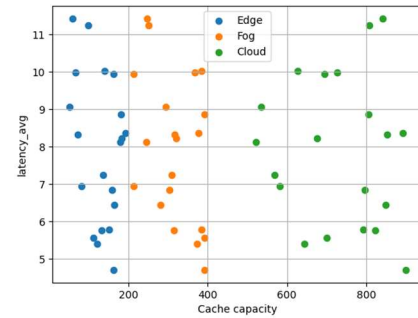


Figure 3. Average latency by architecture level.

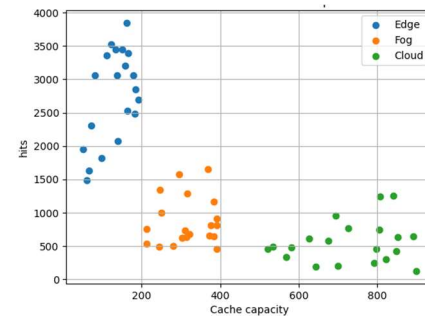


Figure 4. Distribution of local cache hits by architecture level.

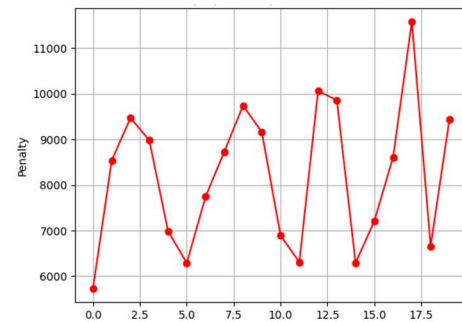


Figure 5. Dynamics of the penalty score.

This demonstrates the ability of the MGA to identify the critical microservices that should be cached locally first. Use of the Fog level allows the latency to remain within acceptable bounds when local memory becomes saturated. Despite the substantial compute resources at the Cloud level, its use increases latency due to network overhead, making it preferred only for low-frequency or large-volume data. The three-iteration stagnation stopping criterion enables fast convergence to an optimal solution, which is critical for dynamic environments where network load varies in real time. The results confirm that the algorithm maximizes the Affinity Degree by placing data as close as possible to the end user, thereby satisfying the performance requirements of streaming data processing.

The simulation confirms that the modified GA delivers an effective trade-off between memory utilization and access latency. The combination of stochastic placement and tournament selection reduces the overall system latency by minimizing the worst-case value of the objective function, consistent with the minimax optimization concept.

For comparison of the obtained values, we simulated another genetic algorithm — NSGA-II with the proposed fitness function. When modeling NSGA-II, instead of a minimax

scalarization, two sub-objectives are optimized in parallel: $f1$ = total memory utilization; $f2$ = total effective latency.

The result is a set of Pareto-optimal cache configurations, from which the most balanced solution can be selected. The simulation results for NSGA-II are shown in Figures 6–9.

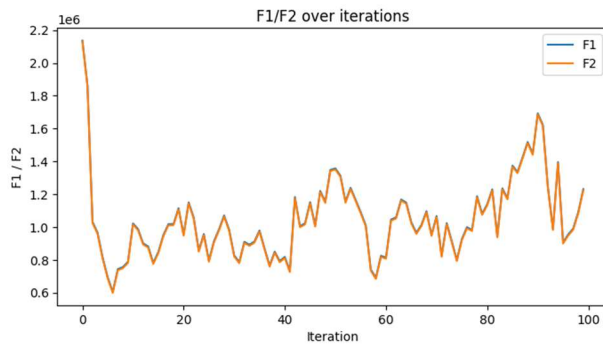


Figure 6. Dynamics of the sub-objectives $f1$ and $f2$ over iterations (NSGA-II).

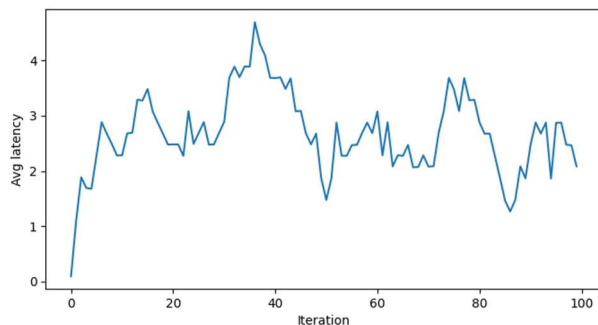


Figure 7. Dynamics of latency over iterations (NSGA-II).

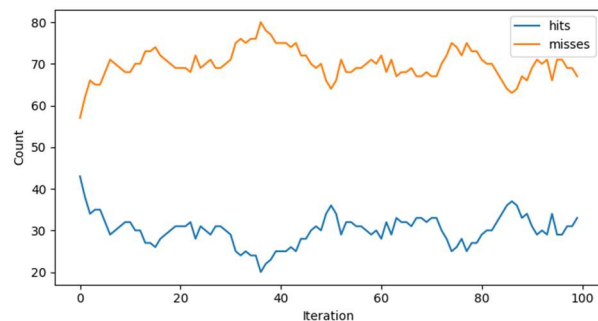


Figure 8. Cache hits vs. misses over iterations

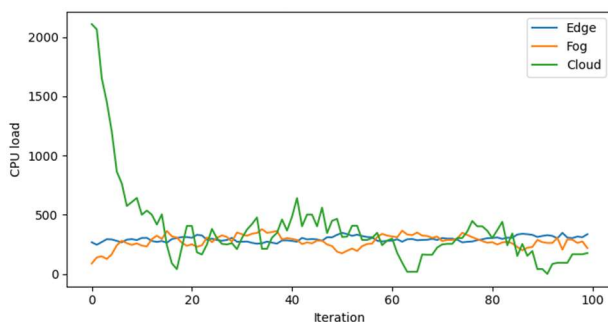


Figure 9. Mean node load by architecture level (NSGA-II).

As we can see in Figure 6, the dynamics of the sub-objectives $f1$ and $f2$ over iterations for the NSGA-II algorithm shows an identical trajectory of the memory-utilization ($f1$) and latency ($f2$) criteria. This confirms the hypothesis of a strong correlation between these parameters under the given architecture: any

restriction on local-cache access at the Edge level automatically results in a proportional increase in latency due to requests being forwarded to the Fog or Cloud level. The sharp drop over the first 10 iterations, followed by oscillations within $[0.6 \times 10^6, 1.6 \times 10^6]$, indicates active exploration of the Pareto trade-off front by the algorithm.

The ratio of successful requests (hits) to cache misses as a function of optimization iterations (Figure 8) shows a stable distribution: the number of hits remains in the range of 25–40 per iteration, while misses range from approximately 65 to 75. This indicates that NSGA-II, in contrast to aggressive single-objective methods, prioritizes the survival of solutions that satisfy stringent memory constraints, even at the cost of a moderate reduction in data locality.

The per-tier load dynamics shown in Figure 9 deserve particular attention:

- Cloud level (green line): in the early iterations, the cloud absorbs the bulk of the requests (load > 2000), ensuring system stability while the algorithm explores. Its load then drops rapidly and stabilizes at the level of the other levels.
- Edge and Fog levels (blue and orange lines): the algorithm maintains balanced load across edge and fog nodes within 250–350 units of CPU load. This indicates the effectiveness of the non-dominated sorting mechanism in preventing overload of individual edge servers.

A series of experiments was conducted to evaluate both optimization strategies under identical multi-level infrastructure conditions. Whereas the modified GA pursues a latency-minimization strategy through rapid data localization, NSGA-II emphasizes system stability and balanced memory utilization. A comparative summary of the algorithms' performance — based on the key metrics of mean latency and computational efficiency — is presented in Table 1.

The data in Table 1 indicate that the proposed modified GA offers better performance in terms of mean latency for time-critical applications. Owing to the three-iteration stagnation stopping criterion, the algorithm fixes a state with the highest data localization at the Edge level, which corresponds to the largest number of hits.

Table 1. Comparative analysis of latency and efficiency metrics for the two algorithms

Critical metric	Modified GA (minimax approach)	NSGA-II (Pareto optimization)
Average latency (Avg Latency)	Low and stable, due to Edge-level prioritization and the use of Affinity Degree.	Variable (range 1.5–4.5 ms), with a downward trend after the 80th iteration.
Latency dynamics	Uniform latency, owing to the strict stopping condition under fitness-function stagnation.	Nonlinear dynamics with spikes due to the search for a memory-vs.-speed trade-off.
Data localization (hits)	High ($\approx 4,000$ hits), minimizing the transport component of latency.	Moderate (25–40 hits), oriented toward global system stability.
Effect of load	Latency is minimized by selecting the closest available node.	Latency is stabilized by redistributing peak load from the Cloud level to the Edge/Fog levels.
Convergence speed	High; the optimal latency is reached in a small number of iterations.	Gradual; a full cycle of 100 iterations is required to minimize $f2$.

The NSGA-II results illustrate a more complex dynamic: the mean latency (f2) initially exhibits high variance because of Cloud-level dominance (Cloud load > 2,000), but it later stabilizes as the algorithm redistributes data toward the Fog level. This confirms that, for real-time systems, the minimax modification of the GA is more appropriate, whereas NSGA-II is better suited to scenarios in which a flexible balance between latency and scarce memory resources is required.

B. EVALUATION USING A PACKET-LEVEL SOFTWARE SIMULATOR

To verify the obtained results, a discrete-event simulator was developed, oriented toward the simulation modeling of distributed service infrastructures (Edge and Fog layers) and caching mechanisms. The simulator allows for the simulation of service provisioning processes, the dynamics of node computing resource allocation, and service competition for shared cache memory with microsecond precision. Network interaction, implemented on the basis of SDN protocols (OpenFlow) [23-26], plays the role of a subordinate transport environment in this model to ensure connectivity between the elements of the distributed system.

The simulation model includes key network components such as hosts, switches, network interfaces, communication links, packets, and an SDN controller, extended with computing nodes, a DNS-based balancing system, RAM memory models, and service-oriented host roles to support modern distributed architectures. The class diagram of the model is presented in Figure 10 and also includes SDN-specific elements such as flows, flow tables, and forwarding rules.

The SDN controller serves as the central network-management entity operating under the OpenFlow standard [27-34]. It maintains network topology information, registers hosts, switches, and links, computes forwarding paths using SPF, CSPF, and optimization-based methods, installs forwarding rules, and manages traffic flows. The controller also supports traffic engineering, rerouting flows when QoS requirements are violated, and monitors delays, transferred data volumes, and SLA compliance.

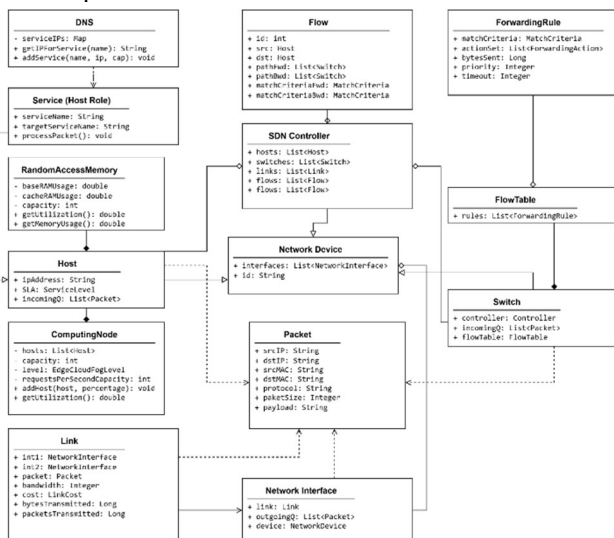


Figure 10. Class diagram showing the relationships and properties of the main network components of the model.

Hosts act as traffic generators and consumers, supporting both ping-based communication and stochastic multi-service traffic generation. Hosts may also operate as services deployed on computing nodes representing Edge or Fog infrastructure with limited processing capacity. A RAM memory model dynamically reflects resource usage, while a DNS-based balancing mechanism distributes requests according to available computational resources. Communication links model physical connections between hosts and switches, considering bandwidth, delay, load, and packet-transmission time.

Switches, implemented according to the OpenFlow standard, process and forward packets using flow tables and support arbitrary network topologies and configurable interface speeds.

To establish the effectiveness of the proposed MGA in solving the multi-dimensional minimax problem, we investigated an orchestration system for microservice operation. Unlike approaches that rely on static or reactive strategies, the goal of this experiment was to evaluate the MGA's ability to proactively determine the optimal cache location with respect to data sources and end users. The central idea of the experiment is that the MGA performs a directed search for the cache-hosting node, guided by two fundamental principles:

- Optimization of the target criteria: the search continues until a trade-off is reached between the lowest achievable content-delivery latency and the highest admissible (efficient) computational load of a given node.
- Stopping criterion: the search terminates as soon as the algorithm localizes the solution on a node with maximal geographic (topological) proximity to the user, provided that this node can sustain the streaming load without performance degradation.

We simulated a request transmission and processing system in a microservices environment.

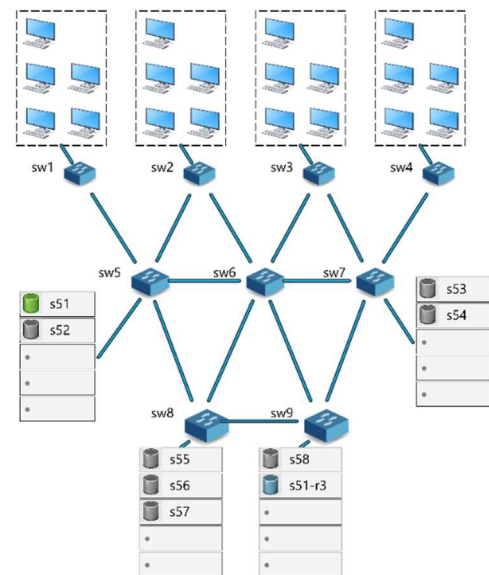


Figure 11. Network architecture used in the simulation experiment.

To formalize and solve the multi-dimensional minimax resource-management problem, we designed a two-level hierarchical distributed-computing architecture (Figure 11). Each structural element of this topology directly reflects a constraint or variable of the optimization model (1)–(5), maintaining the balance between scarce memory resources and

QoS requirements. For conducting experimental research, the simulation modeling software complex was deployed on a dedicated server running Windows 11 Pro. The hardware configuration of the testbed included an 11th Gen Intel® Core™ i7-11850H processor (2.50 GHz) and 32 GB of DDR5 RAM. To adequately replicate the resource constraints inherent in hierarchical distributed systems, the computing capabilities of individual nodes were software-limited. Specifically, for the edge nodes (Edge), the maximum throughput was set to 100 requests per second, and the available cache size was limited to 100 MB. For the intermediate layer nodes (Fog), these metrics were set at 500 requests per second and 200 MB of cache memory, respectively.

Within the framework of the proposed model, each deployed microservice, regardless of its hierarchy level, was guaranteed to reserve 10% of the host node's total computing capacity. Conversely, strict quotas for cache memory usage were not established for individual microservices within a single node. The cache operated as a shared resource, for which the microservices competed in real-time until its complete exhaustion, theoretically allowing a single service's content to monopolize the entire cache capacity. Regarding the cloud layer (Cloud), no artificial software limitations were applied, which aligns with the core paradigm of cloud computing regarding the use of theoretically infinitely scalable resources. The correspondence between the architectural levels and the parameters of the optimization problem is defined as follows:

- Edge level: composed of switches SW5–SW7 and the low-power servers attached to them. This level provides minimum data-access time from the local cache, which is critical for streaming applications. However, the memory capacity $S_{local}(d_{s_k})$ of these nodes is tightly bounded. Within the optimization problem, this level operates on the Boolean variable $x_h = 1$ “cache locally” and is preferred by the MGA, since it has zero geographic/topological distance to clients and provides the lowest data-access latency.
- Fog level: built on aggregation switches SW8 and SW9 and high-performance servers. Compute capacity and memory capacity $S_{remote}(d_{s_k})$ are higher than at the edge, which enables remote-cache access. However, the transport latency - D_{total} - to this level is greater. The Fog level acts as a buffer that prevents system degradation when edge nodes become saturated.

Network load: clients connect to the infrastructure through four access switches (SW1–SW4). In the optimization context, this zone defines the set of users that generate the intense input stream of microservice and data requests. As a result, the designed architecture transforms the GA's search space into a problem of minimizing the maximum objective-function deviation. The modified GA satisfies the capacity constraints of the switches, eliminates content duplication, and keeps the overall system latency within suboptimal bounds.

Services s51–s58 are deployed on the servers. Microservice s51 is designated as the target object of orchestration: the modified GA performs a dynamic search for the optimal cache and replica placement for s51 in accordance with the minimax criteria (Equations 1–5). Services s52–s58 generate the

background system load. They are shown in gray on the plots so as not to distract attention.

Our simulations show that a single replica of the target service s51 on an edge node can stably handle a streaming load of up to 10 req/s (marked by a red line on the result plots). As soon as the request intensity grows, the system spawns a new local replica on the same edge server in order to keep the latency low (Figure 13 illustrates the cache memory usage by the created replicas of microservice s51-r1 and s51-r2.). Because the MGA stopping criterion favors maximal topological proximity to the user, the algorithm keeps the cache at the Edge level as long as possible, loading its resources to the limit. Scaling, however, continues only while the 10-second moving average of edge-server RAM utilization remains below the critical threshold of 80%. Crossing this threshold signals to the algorithm that any further attempt to preserve topological proximity would trigger an avalanche-style degradation of the node itself (through the penalty term). At this point, the MGA adaptively shifts the cache: it removes redundant edge replicas (s51-r_1 and s51-r_2), keeping only the baseline replica, and transfers the main load deeper into the hierarchy — to the Fog level (switch SW9). A new s51_r3 replica is deployed on a fog server, which, owing to its greater capacity, reserves 10% of the total CPU and single-handedly handles up to 50 req/s.

The experimental results are shown in Figures 12 and 13.

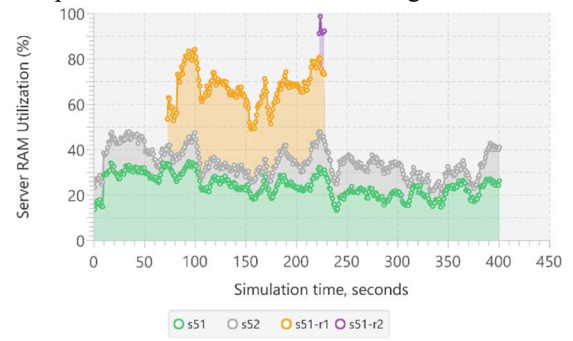


Figure 12. Edge-node RAM utilization per service instance during the simulation.

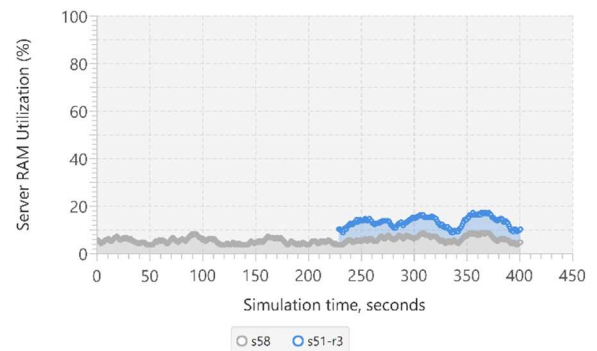


Figure 13. Fog-node RAM utilization per service instance during the simulation.

The dynamic search for the optimal node based on MGA criteria made it possible to find a compromise between topological distance and resource constraints. Quantitative indicators of RAM utilization optimization on the edge node are presented in Table 2.

Table 2. Optimization metrics for edge-node RAM utilization

Metric (RAM)	Before optimization (Edge)	After optimization (MGA)	Effect
Average load	61.77%	34.84%	↓ nearly two fold
Peak load	94.12%	42.28%	−51.84%

Consequently, the average cache utilization at edge nodes (Edge) decreased from 61.77% to 34.84%. At the same time, the load on the intermediate server (Fog) cache increased marginally: the average value rose from 7.18% to 10.75%, while its peak value reached only 14.54%. This 26.93% reduction in Edge node utilization effectively freed up additional cache resources to serve other latency-sensitive streaming services, amidst a perfectly acceptable overhead on the Fog layer. Importantly, this optimization was achieved without any degradation in the Quality of Service (QoS) metrics of the target service under study, as requests continued to be seamlessly processed by both the user-adjacent edge layer and the capacities of the intermediate layer.

As shows in Figure 14, the three parallel local replicas of the target service (s51-r_1, s51-r_2, and s51-r_3) operated under severe resource scarcity, with mean throughputs of 8.37, 8.58, and 6.62 req/s, respectively. Attempting to keep the cache at the closest topological position to the user caused the total input request rate to repeatedly approach the critical red performance line (10 req/s per replica), triggering avalanche-style depletion of edge-server RAM.

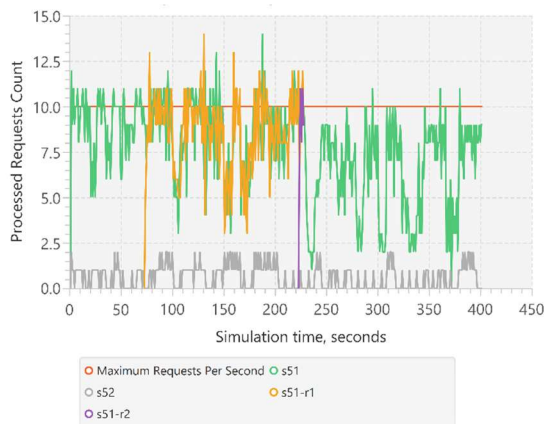


Figure 14. Per-instance request throughput on edge nodes during the simulation.

Upon reaching the 80% RAM trigger, the MGA proactively migrated the computational pipeline to the higher level of the hierarchy (Figure 15). Thanks to higher compute surplus at the fog node, this replica alone provided stable processing of the input stream with a mean throughput of 34.90 req/s and observed peak bursts of up to 45 req/s. Notably, even at peak load, system throughput did not exceed the 50 req/s ceiling, which confirms the high robustness and balance of the MGA solution.

The main criterion for optimization success was defined as the system's ability to maintain response latency within the critical time threshold, which for this type of streaming application is 200 ms (indicated by the red line in the overall latency analysis graph in fig.17).

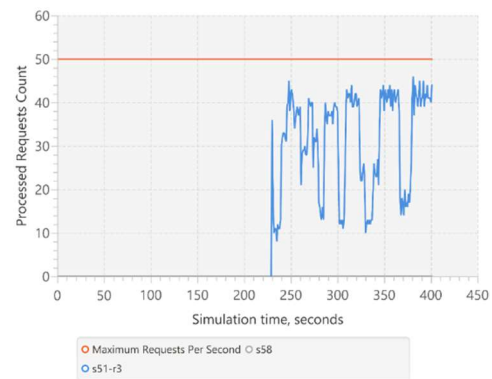


Figure 15. Per-instance request throughput on fog nodes during the simulation.

To verify the obtained results, we performed a comparative statistical analysis of the cumulative distribution functions (CDFs) of cached-content delivery latency before and after the orchestration algorithm took effect. The results are shown in Figures 16 and 17 and summarized in Table 3.

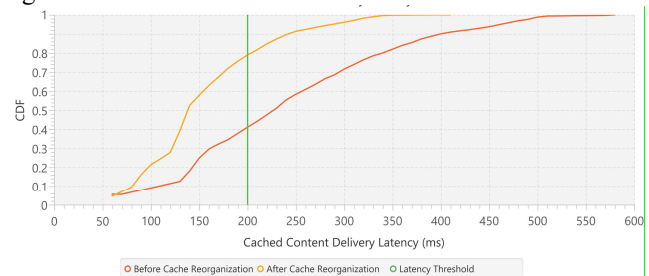


Figure 16. Cumulative distribution functions (CDFs) of cached-content delivery latency before (yellow line) and after (red line) the optimization algorithm takes effect.

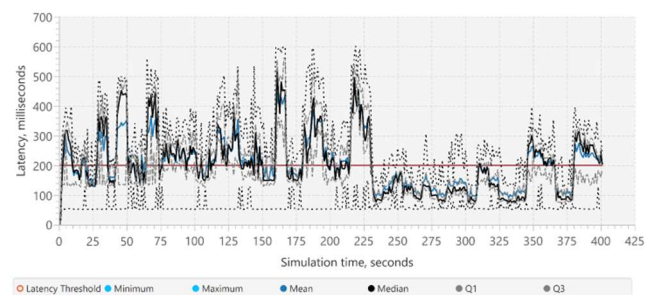


Figure 17. Client-side content-delivery latency before and after optimization.

Table 3. Content-delivery latency parameters before and after optimization

Latency statistic	BEFORE OPTIMIZATION (EDGE)	After optimization (MGA Edge-Fog)	System effect
Mean latency	247.3 ms	158.4 ms	Reduction of 88.9 ms
Median latency	230.0 ms	140.0 ms	Reduction of 90.0 ms

The results show that due to the overload of edge processors and inefficient cache allocation, only 41.1% of the total number of user requests were processed on time. The remaining requests experienced significant delays caused by processing queues within the service. Owing to the adaptive cache migration to the Fog layer, the number of requests successfully meeting the QoS latency threshold increased to 78.9%, thereby ensuring stable

and uninterrupted service delivery. The 88.9 ms reduction in mean latency and the shift of the median to 140.0 ms indicate that the majority of requests have moved into the safe, user-comfortable operating range.

Thus, the proposed minimax model, instantiated through a modified genetic algorithm, treats the cache not as a static repository but as a flexible, controllable orchestration instrument, which yields a substantial improvement in QoS/QoE under the tight resource constraints of edge infrastructure.

IV. DISCUSSION

The experimental results confirm that the proposed minimax formulation, coupled with the modified genetic algorithm (MGA), produces a measurable improvement in the operational characteristics of streaming microservice systems deployed over hierarchical Edge-Fog-Cloud infrastructures. Under the resource constraints imposed on the testbed – namely, a throughput ceiling of 100 requests per second and 100 MB of cache memory for Edge nodes, and 500 requests per second with 200 MB of cache memory for Fog nodes – the MGA reduced the average cache utilization at the Edge tier from 61.77% to 34.84%, decreased the peak cache load by 51.84% (from 94.12% to 42.28%), and stabilized the 95th-percentile utilization at 40.02%. The offloading of peripheral resources translated directly into a reduction of the average content delivery latency from 247.3 ms to 158.4 ms and of the median latency from 230.0 ms to 140.0 ms, increasing the share of requests satisfying the 200 ms threshold typical of streaming applications from 41.1% to 78.9%.

A comparative analysis against NSGA-II indicates that, although NSGA-II maintains a stable Pareto front under a fixed budget of 100 iterations and is therefore well suited to scenarios requiring flexible trade-offs between latency and scarce memory, the proposed MGA achieves lower and less dispersed latency in time-critical settings at a lower computational cost. The stagnation-based stopping criterion – terminating execution after three consecutive iterations without improvement of the fitness function – allows the algorithm to consistently capture configurations characterized by maximal data localization at the Edge tier, which correlates with the highest cache hit ratios. In contrast, NSGA-II exhibits high variance of the average latency objective at early stages, driven by the dominance of cloud-tier computation, before stabilizing as the algorithm redistributes content to the Fog level. These observations support the interpretation of cache placement as a dynamic orchestration task rather than a passive deployment-phase decision, and validate the choice of the minimax criterion for systems operating under strict QoS requirements.

The proactive migration mechanism, triggered when Edge RAM utilization exceeds the 80% threshold, proved to be an effective control point. It preempts the avalanche-like degradation of peripheral nodes while preserving their topological proximity to end users for as long as feasible, and it does so without the workload-specific parameter tuning typically associated with classical cache-replacement policies.

Several limitations should nonetheless be acknowledged. First, the evaluation assumed a Zipf-distributed request popularity with a fixed exponent ($\alpha = 1.0$), whereas production workloads exhibit non-stationary popularity, concept drift, and bursty arrival patterns that may shift the optimal placement

boundary. Second, the Fog tier was modeled as homogeneous, while operational deployments comprise nodes with heterogeneous CPU, memory, and network profiles. Third, the experimental topology, although packet-accurate, was confined to a two-level Edge-Fog arrangement with a single target service (s51); the scalability of the MGA to topologies containing hundreds of services and pronounced inter-service dependencies has not been quantified. Fourth, the proposed approach is reactive: cache migration is initiated upon observed crossings of resource thresholds rather than in anticipation of forthcoming demand.

V. FUTURE RESEARCH

Three directions of further research are envisaged. The first concerns the integration of time-series machine-learning models – in particular, predictors based on LSTM and Transformer architectures – to support proactive cache repositioning before resource thresholds are breached, thereby converting the present reactive policy into an anticipatory one. The second direction involves extending the minimax formulation to heterogeneous multi-tenant Fog environments and incorporating energy consumption as an additional objective, which would allow joint optimization of latency, memory pressure, and operational cost. The third direction is empirical validation of the proposed mechanism on a production-grade Kubernetes/Istio testbed using real streaming workloads, such as video analytics and industrial telemetry, in order to assess algorithmic behavior under realistic network jitter, container startup latency, and cross-tenant interference.

VI. CONCLUSIONS

This paper has addressed the problem of dynamic cache placement in streaming microservice systems deployed over multi-tier Edge-Fog-Cloud infrastructures, where the trade-off between the limited memory of peripheral nodes and end-to-end delivery latency determines the feasibility of stable real-time service operation. A minimax optimization model has been proposed in which cache placement and microservice placement are jointly determined under a single composite criterion that combines peak memory utilization and access latency, subject to node capacity and exclusive-placement constraints. Unlike prior approaches that treat service placement and cache management as independent query-oriented processes, the proposed formulation integrates streaming traffic dynamics, topological user proximity, and volatile memory consumption at the Edge tier within one objective, treating the cache as a mobile and centrally managed resource.

To solve the resulting combinatorial problem under near-real-time conditions, a modified genetic algorithm has been developed, combining tournament selection, two-point crossover, stochastic encoding of placement decisions, an adaptive penalty mechanism with a 15% intensity coefficient, and a stagnation-based stopping criterion. Mathematical modeling in Python/NumPy with a Zipf-distributed request stream (5000 requests over 1000 unique items, $\alpha = 1.0$), supplemented by packet-level simulation on a dedicated server, confirmed that the proposed algorithm reduces average Edge cache utilization by nearly a factor of two, lowers peak load by more than half, and decreases average delivery latency from 247.3 ms to 158.4 ms, raising the proportion of requests

meeting the 200 ms streaming threshold from 41.1% to 78.9%. The 80% RAM threshold for proactive Edge-to-Fog migration was shown to be an effective control point that prevents avalanche-like degradation of peripheral nodes without workload-specific tuning. The findings substantiate the principal claim of the study: treating cache placement as a dynamic orchestration task, rather than a passive deployment decision, yields a measurable improvement of QoS and QoE indicators under the tight resource constraints characteristic of edge environments.

ACKNOWLEDGMENT

This research was supported by the Ukrainian government project #0126U001332 “Development of the Physically Consistent Multimodal AI Models for Robotic Systems”.

During the preparation of this work, the authors used Google Gemini and Grammarly for English proofreading and grammar correction of the original article text. Some images in the paper were generated by Google Gemini, with detailed instructions and further editing by authors.

References

- [1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things,” in *Proceedings of the 1st MCC Workshop Mobile Cloud Computing*, Helsinki, Finland, 2012, pp. 13–16. <https://doi.org/10.1145/2342509.2342513>.
- [2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016. <https://doi.org/10.1109/JIOT.2016.2579198>.
- [3] C. Guerrero, I. Lera, and C. Juiz, “Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures,” *Future Gener. Comput. Syst.*, vol. 97, pp. 131–144, 2019. <https://doi.org/10.1016/j.future.2019.02.056>.
- [4] D. Sharma, S. Singhal, A. Rai, and A. Singh, “Adam-Ladybug Beetle Optimization enabled multi-objective service placement strategy in fog computing,” *Concurrency Computat. Pract. Exper.*, vol. 36, no. 22, Art. no. e8239, 2024. <https://doi.org/10.1002/cpe.8239>.
- [5] S. Pallevatta, V. Kostakos, and R. Buyya, “A two-tier multi-objective service placement in container-based fog-cloud computing platforms,” *Cluster Comput.*, vol. 27, pp. 4131–4156, 2024. <https://doi.org/10.1007/s10586-023-04183-8>.
- [6] M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane, “Quality of experience improvement and service time optimization through dynamic computation offloading algorithms in multi-access edge computing networks,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 16, no. 4, pp. 1–16, 2024. <https://doi.org/10.5815/ijcnis.2024.04.01>.
- [7] N. Zhang, W. Wang, P. Zhou, and A. Huang, “Delay-optimal edge caching with imperfect content fetching via stochastic learning,” *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 338–352, 2022. <https://doi.org/10.1109/TNSM.2021.3118946>.
- [8] D. Gupta, S. Rani, B. Tiwari, and T. Gadekallu, “An edge communication based probabilistic caching for transient content distribution in vehicular networks,” *Sci. Rep.*, vol. 13, Art. no. 3614, 2023. <https://doi.org/10.1038/s41598-023-30315-6>.
- [9] I. Byzov, S. Yakovlev, “Information technology for automation of server infrastructure management using DevOps tools,” *Radioelectronic and Computer Systems*, vol. 2025, no. 3, pp. 257–269, 2025. <https://doi.org/10.32620/reks.2025.3.18>.
- [10] Y. Liu, Y. Bin, N. Chen, S. Zhu, “Caching placement optimization strategy based on comprehensive utility in edge computing,” *Applied Sciences*, vol. 13, issue 16, 9229, 2023. <https://doi.org/10.3390/app13169229>.
- [11] M. D. E. Gouaouri, S. Ouahouah, M. Bagaa, M. A. Ouameur, D. Massicotte and A. Ksentini, “Diktos: A two-stage framework for joint container-based microservice placement and distributed volume allocation on cloud-edge networks,” *IEEE Transactions on Cloud Computing*, vol. 14, no. 2, pp. 931–948, 2026. <https://doi.org/10.1109/TCC.2026.3676203>.
- [12] X. He, H. Xu, X. Xu, Y. Chen and Z. Wang, “An efficient algorithm for microservice placement in cloud-edge collaborative computing environment,” *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 1983–1997, 2024. <https://doi.org/10.1109/TSC.2024.3399650>.
- [13] O. Shpur, M. Seliuchenko, M. Klymash, and N. Seliuchenko, “Method for improving cache memory utilization in multilayer Edge–Fog–Cloud infrastructures,” in *Proc. IEEE 18th Int. Conf. Advanced Trends Radioelectron., Telecommun. Comput. Eng. (TCSET)*, Lviv, Ukraine, 2026, pp. 1–6. <https://doi.org/10.1109/TCSET65181.2026.11460406>.
- [14] V. Faychuk, O. Lavriv, M. Klymash, V. Zhebka, and O. Shpur, “Investigation of the Bellman-Ford algorithm enhanced for remote execution,” in *Proc. 3rd Int. Conf. Advanced Information Commun. Technol. (AICT)*, Lviv, Ukraine, 2019, pp. 204–208. <https://doi.org/10.1109/AIACT.2019.8847727>.
- [15] M. Seliuchenko, M. Beshley, O. Shpur, N. Seliuchenko, M. Klymash, and H. Beshley, “Software-defined multi-access Edge/Cloud computing for 5G/6G time-critical services,” in *Proc. 17th Int. Conf. Experience Designing Application CAD Syst. (CADSM)*, Jaroslaw, Poland, 2023, pp. 1–4. <https://doi.org/10.1109/CADSM58174.2023.10076524>.
- [16] B. L. Miller and D. E. Goldberg, “Genetic algorithms, tournament selection, and the effects of noise,” *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.
- [17] M. Sohrabi, A. M. Fathollahi-Fard, and V. A. Gromov, “Genetic Engineering Algorithm (GEA): An efficient metaheuristic algorithm for solving combinatorial optimization problems,” *Autom. Remote Control*, vol. 85, pp. 252–262, 2024. <https://doi.org/10.1134/S000511792403007X>.
- [18] M. Elkawagy, I. A. Elgendy, S. A. Chelloug, and H. Elbeh, “Genetic algorithm-driven joint optimization of task offloading and resource allocation for fairness-aware latency minimization in mobile edge computing,” *IEEE Access*, vol. 13, pp. 118237–118248, 2025. <https://doi.org/10.1109/ACCESS.2025.3584971>.
- [19] N.K. Verma, K.J. Naik, “Optimized fog community framework with advanced genetic algorithm for enhanced performance dynamics,” *Journal Supercomputing*, vol. 80, pp. 8202–8235, 2024. <https://doi.org/10.1007/s11227-023-05769-0>.
- [20] F. Talavera, I. Lera, C. Juiz, and C. Guerrero, “Optimizing fog colony layout and service placement through genetic algorithms and hierarchical clustering,” *Expert Systems with Applications*, vol. 254, p. 124372, 2024. <https://doi.org/10.1016/j.eswa.2024.124372>.
- [21] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Cham, Switzerland: Springer, 2015. <https://doi.org/10.1007/978-3-662-44874-8>.
- [22] O. Shpur, M. Seliuchenko, and M. Klymash, “Control of automated deployment of microservice applications with QoS/QoE support for sustainable development of information and communication networks,” in *Networks and Sustainability*, A. Luntovskyy, M. Klymash, I. Melnyk, M. Beshley, and D. Gütter, Eds. Cham, Switzerland: Springer, 2025, pp. 365–378. https://doi.org/10.1007/978-3-032-02272-1_31.
- [23] T. K. Das, S. K. Das, S. Bissoyi, D. K. Patel, “Self-adaptive resource allocation in fog-cloud systems using multi-agent deep reinforcement learning with meta-learning,” *International Journal of Intelligent Systems and Applications (IJISA)*, vol. 18, no. 1, pp. 107–118, 2026. <https://doi.org/10.5815/ijisa.2026.01.08>.
- [24] Luo, S., Xu, H., Lu, C., Ye, K., Xu, G., Zhang, L., et al.: An in-depth study of the microservice call graph and runtime performance. *IEEE Trans. Parallel Distrib. Syst.* 33, 3901–3914, 2022. <https://doi.org/10.1109/TPDS.2022.3174631>.
- [25] N. Šatkauskas, A. Venčkauskas, “Multi-agent dynamic fog service placement approach,” *Future Internet*, vol. 16, issue 7, 248, 2024. <https://doi.org/10.3390/fi16070248>.
- [26] M. Beshley, N. Kryvinska, H. Beshley, O. Yaremko, J. Pyrih, “Virtual router design and modeling for future networks with QoS guarantees,” *Electronics*, vol. 10, issue 10, 1139, 2021. <https://doi.org/10.3390/electronics10101139>.
- [27] M. Elsharkawey, H. Refaat “MLRTS: Multi-Level Real-Time Scheduling Algorithm for Load Balancing in Fog Computing Environment,” *Int. J. Mod. Educ. Comput. Sci.* 10, 1–15, 2018. <https://doi.org/10.5815/ijmecs.2018.02.01>.
- [28] K. Aruna, G. Pradeep “Container-to-fog Service Integration using the DIS-LC Algorithm,” *Int. J. Comput. Netw. Inf. Secur.* 16, 85–96, 2024. <https://doi.org/10.5815/ijcnis.2024.01.07>.
- [29] A. Nicolás-Plata, J. González-Compean, V. Sosa-Sosa, “A service mesh approach to integrate processing patterns into microservices applications,” *Cluster Comput.* 27, 7417–7438, 2024. <https://doi.org/10.1007/s10586-024-04342-5>.
- [30] A. Samanta, J. Tang, “Dyme: Dynamic Microservice Scheduling in Edge Computing Enabled IoT,” *IEEE Internet Things J.* 7(7), 6164–6174, 2020. <https://doi.org/10.1109/JIOT.2020.2981958>.
- [31] S. He, A. Xiong, M. Zhang, C. Li, “Microservice deployment strategy based on interaction weight in the edge environment,” *Res. Square*, 1–13, 2024. <https://doi.org/10.21203/rs.3.rs-3916188/v1>.

- [32] A. Volokyta, A. Kogan, O. Cherevatenko, D. Korenko, D. Oboznyi, Y. Kulakov, "Traffic Engineering with Specified QoS Parameters in Software-defined Networks," *Int. J. Comput. Netw. Inf. Secur.* 16, 1–13, 2024. <https://doi.org/10.5815/ijenis.2024.05.01>.
- [33] R. Alakberov, "Clustering Method of Mobile Cloud Computing According to Technical Characteristics of Cloudlets," *Int. J. Comput. Netw. Inf. Secur.* 14, 75–87, 2022. <https://doi.org/10.5815/ijenis.2022.03.06>.
- [34] S. Tidake, P. Mulkalwar, "Efficient Resource Allocation to Enhance the QoS in Cloud Computing," *Int. J. Comput. Netw. Inf. Secur.*, 17, 68–81, 2025. <https://doi.org/10.5815/ijenis.2025.01.06>.



OLHA SHPUR, PhD, associate professor of the department of "Information and Communication Technology" at Lviv Polytechnic National University. Here research interests: improving the efficiency of network traffic management in data center networks, analyzing the efficiency of Big Data flow processing and management, and adapting data center operations to the integration of multitasking CloudNFV/Big Data structure



MARIAN SELIUCHENKO, PhD, associate professor of the department of "Information and Communication Technology" at Lviv Polytechnic National University. His research interests: computer networks and protocols, methods for ensuring quality of service (QoS) and quality of experience (QoE), and the modeling of software-defined networks and cloud systems.



IVAN DEMYDOV, Doctor of Engineering Sciences (DSc), professor of the department of "Information and Communication Technology" at Lviv Polytechnic National University. His research interests: ICT systems, telecommunication networks, technology transfer, innovation ecosystems, MedTech/Digital Health campus development, and commercialization of research outputs.



MYKHAILO KLYMASH is now the Chief of Information and Communication Technology Department, Lviv Polytechnic National University, Ukraine. He received his PhD in optical data transmission, location and processing systems from Bonch-Bruевич Saint-Petersburg State University of Telecommunications, Saint Petersburg, Russia, in 1994 Honored member of Ukrainian Communications Academy.

The topics of his current interest of research include distributed networks, cloud computing, convergent mobile networks, big data, software defined networks and 5G heterogeneous networks.



ASSOCIATE PROFESSOR SVITLANA SACHENKO, PhD, Department of Management and Administration, Ivano-Frankivsk Educational and Scientific Institute of Management, West Ukrainian National University. Research interests: control, audit, economic expertise, internal control, knowledge visualization.

...