An Intelligent Micro-service Chain Orchestrator for Immersive Media Services under the presence of Server Failure Events

Erick Costa[1], Rodrigo Flexa[1], Matheus Brito[1], Hugo Santos[2], Carlos Astudillo[3], Denis Rosário[1], Eduardo Cerqueira[1]

[1]Federal University of Pará (UFPA) – Belém – PA – Brasil

[2]State University of Pará (UEPA) – PA – Brazil

[3]University of Campinas (UNICAMP) – SP – Brazil

{erick.costa, rodrigo.flexa, matheus.moraes.brito}@itec.ufpa.br,

hugo.santos@uepa.br, castudillo@unicamp.br, {denis, cerqueira}@ufpa.br

Abstract. *Immersive Multimedia Services (IMS)s combined with advanced communication technologies, such as beyond 5G (B5G) and 6G, has advanced constraints for low latency and high throughput, offering a fluid and improved user experience. This is achieved by the use of Multi-access Edge Computing (MEC) infrastructure to distribute resource consumption for decentralized service execution between the head-mounted glasses and the central network node. In this context, Micro-service Chain (MSC) orchestration algorithms offer an efficient distribution of resource consumption in the available MEC infrastructure, avoiding interruption in experience and increasing quality. However, it is required robust orchestration strategies for service recovery in cases of connection node failure. This paper introduces an Intelligent MSC Orchestration algorithm for IMS applications under the presence of MEC server failures, called IMSCO. The algorithm uses Genetic Algorithm to determine the optimal routing paths between clients and parallelizable, ordered, and location-aware MSs instantiated at MEC servers, considering computational and network constraints, latency, and client mobility. The comparative results highlight a recovery success 90% in fail events with 5% acceptance ratio improvement to literature algorithms, showcasing advancements ensuring resilient and high-quality immersive service delivery.*

1. Introduction

Computing-intensive Immersive Multimedia Service (IMS), such as Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR), integrate three-dimensional (3D) elements into the user's environment via powerful mobile Head Mounted Glass (HMG) which surpasses traditional on-demand experiences and live broadcasts, requiring more advanced and adaptive solutions [Huang and Simeone 2023]. In addition, supporting multi-user IMS adds another layer of complexity, as synchronizing users' interactions is crucial for achieving a seamless immersive experience. This challenge surpasses traditional on-demand experiences and live broadcasts, requiring advanced wireless communication and powerful computing units to meet stringent latency requirements to avoid motion sickness. In this context, 5G and Beyond 5G (B5G) plays a key role to provide better network connectivity, higher network bandwidth, and lower network latency, enabling a plethora of new IMSs that were previously unfeasible due to the lack of developed technology enablers [Shen et al. 2023].

Beyond 5G (B5G)/6G relies on Multi-access Edge Computing (MEC) paradigm to bring computing resources closer to the network edge to significantly reduce the response time and latency of IMS. In this context, computing-intensive IMS require a significant amount of MEC resources to deploy the service in a monolithic fashion [Zawish et al. 2024]. However, monolithic services can be decomposed into smaller and sequential components (*i.e.*, frame acquisition, object detection, caching, rendering, and transcoding), forming a Micro-service Chain (MSC) of shareable Micro-services (MSs). This decomposition improves scalability and minimizes redundant processing, storage, and transmission, thereby reducing computational overhead and lowering end-to-end latency. For instance, MSC enables resource sharing between users with similar viewpoints, alleviating the computational burden on the MEC servers for duplicate processing and data transmission while ensuring real-time responsiveness. Therefore, it is crucial to consider an efficient orchestrator for the management of MSC, particularly for tasks such as re-instantiating MSs, sharing them, or removing chains.

An intelligent MSC orchestrator is responsible for mapping MSC requests onto the powerful yet constrained computing resources of surrounding MEC infrastructure. However, network condition fluctuations affect throughput or increase latency, while user mobility affects MSC route [Huang and Wu 2022]. In this way, the orchestrator must continuously monitor the network conditions, server workloads, and storage capacities to dynamically reconfigure MSCs to ensure an optimized balance between performance and efficiency. In addition, despite the efficiency and robustness of MEC servers, occasional malfunctions and crashes can still occur within the infrastructure's distributed mesh of servers, directly impacting active MSC requests and availability [Huang and Wu 2022]. Hence, resilience is a fundamental aspect of the MSC orchestration to achieve/maintain the application performance despite sudden software or hardware disruptions of the MEC infrastructure, user mobility, overload, and service stalls due to relocation. To the best of our knowledge, effectively managing MS chains within highly dynamic network environments is crucial for optimizing the utilization of shared SFs, minimizing latency, and improving session acceptance ratios. To the best of our knowledge, these challenges remain open issues and require further research to achieve more efficient solutions.

This paper introduces an Intelligent MSC Orchestration algorithm, called IMSCO, designed to address the challenge of MEC server failures. IMSCO leverages an evolutionary mechanism, particularly Genetic Algorithm (GA), to enhance the adaptability and resilience of the orchestration process in failure scenarios. By utilizing GA, the orchestrator optimizes routing paths between clients and parallelizable, location-aware MSs instantiated at MEC servers, considering key constraints such as computational resources, network capacity, latency, and client mobility. The role of GA is critical for dynamically adjusting the orchestration in response to failure events, providing optimal solutions in complex and changing environments. Furthermore, IMSCO continuously monitors MEC servers, assessing available network and computing resources, and activates recovery mechanisms in the event of failures. The use of GA allows IMSCO to efficiently reorganize MSCs on available MEC servers, thus maintaining service continuity and resilience even during network interruptions or server failures. The evaluation of the proposed approach demonstrates its effectiveness, measuring key resilience metrics such as resource availability, utilization, session acceptance rate, recovery time, and recovery success under MEC failure conditions. Performance results indicate that, on average, the instantiation acceptance ratio decreases by up to 22.5% during failure events compared to existing state-of-the-art solutions, showing the superior adaptability and robustness achieved through orchestration.

The remainder of this paper is structured as follows. Section 2 presents an overview of the literature on MSC with failure scenario approaches and their main drawbacks. Section 3 presents the system model and the failure scenario. Section 4 presents the evaluation method and discusses the results obtained. Finally, Section 5 presents the conclusions and outlines directions for future work.

2. Related Works

[Di Cicco et al. 2024] proposed an approach that leverages Multi-Objective Optimization (MOO) in conjunction with Integer Linear Programming (ILP) to help the orchestrator in identifying all possible "optimal" trade-offs. The authors apply multi-objective evolutionary algorithms to solve the MOO problem, optimizing multiple conflicting objectives simultaneously. However, despite the approach successfully addressing MSC orchestration through mini-cluster Kubernetes (K8s), the authors do not consider the inclusion of low-latency services, such as IMS, or account for resilience in the context of MEC scenarios, where these factors are critical for ensuring robust and responsive system performance.

[Medeiros et al. 2023] proposed a distributed service chain orchestrator to solve the distributed service chain problem, identified as $\mathcal{NP}$-hard problem. The orchestrator considers an Integer Linear Programming (ILP) and a heuristic for decision-making to support a VR service instantiation with offloading, migration, and orchestration considering IMS along an MEC infrastructure simulation. However, the work does not consider resilience in the proposed orchestrator nor applies it a mobile IMS scenario, with high consumption of MEC resources.

[Bai et al. 2023] developed a Multi-Dimensional Semi-Markov model (MDSM) for rejuvenation techniques, avoiding resource degradation in the Internet of Things (IoT)-based services in the MEC infrastructure. The authors aim to improve the availability and reliability of dynamic and heterogeneous IoT services and user devices. Although the proposed scenario considers the guarantee of failure and recovery through behavior identification, it does not address costly services, such as IMS, nor a complete MEC infrastructure with mobility features.

[Huang and Friderikos 2023] proposed a solution for a joint-optimization problem that considers the balance between service delay and energy-consumption cost, evaluating perceived user quality in a mobility event. The authors compare the optimized scheme with a terminal-oblivious scheme called "OptimT", considering the capabilities of user terminals. However, the article does not consider scenarios with failures and disabled server recovery capabilities. Even with the mobility and MSC orchestration approach, only two MSs are considered with constrained mobility.

[Santos et al. 2022] proposed the Multi-User Service Function Chain Orchestrator (MusFiCo) scheme for multi-user VR, which maps an MEC topology and instantiates service functions constrained to latency, CPU, memory, and bandwidth thresholds. Although they proposed an efficient method for orchestrating resources for serving multiple users, the authors do not consider user mobility, a costly immersive service, or a resilient and failure-aware analysis. Afterward, [Santos et al. 2023] extended the previous approach by adding mobility, creating a more dynamic simulation scenario for a multi-user AR service called Mobility-Aware Multi-Criteria Service Function Chaining (MSF) scheme. However, the authors do not consider a failure scenario to supplement the simulation, which can directly affect the Quality of Service (QoS) for users.

Table 1 provides a comprehensive summary of the existing literature, highlighting various approaches in terms of orchestrator techniques, type of service, MEC evaluation environment, resilience, and a comparative analysis of multiple studies. By examining the state-of-the-art works presented, we conclude that no single approach simultaneously addresses all of these critical aspects in a IMS application, leading to partial evaluations that limit the overall scope of the analysis.

Table 1. Comparison features in the literature

State of the art	Service	MEC Environment	Resilience Scenario
[Di Cicco et al. 2024]	K8s	AWS Instances	
[Medeiros et al. 2023]	VR	Simulation	
[Bai et al. 2023]	IoT	Simulation	✓
[Huang and Friderikos 2023]	Metaverse	Simulation	
[Santos et al. 2022]	VR	Simulation	
[Santos et al. 2023]	AR	Simulation	
IMSCO	AR	Simulation	✓

3. An Intelligent MSC orchestrator for IMS in an environment with MEC Server Failures

In this paper, we introduce an intelligent MSC Orchestrator for handling network failures by iteratively evaluating and optimizing request instantiation and re-instantiation in failure scenarios. In the following, we introduce the system model in an environment with MEC server failures. Afterwards, we introduce the orchestration to instantiate multiple MSCs with minimized latencies and efficient use of the network and computational resources.

3.1. System Model

The IMS applications operate in a client-server architecture, where the orchestrator deploys a set of service on edge servers and the cloud server maintains an in-depth analysis of the servers' resources in a specific region. This architecture aims to minimize redundancies by efficiently distributing resources and adapting to the location and mobility of users. In a typical multi-user IMS scenario, a set of mobile clients with HMG devices requests the content *Central Cloud* to set up a session by using any communication network (*i.e.*, 5G, 4G, and WiFi). In this sense, the system ensures continuous connectivity and optimized service delivery, even in highly dynamic environments, by leveraging multiple anchor points such as mobile network antennas, Wi-Fi hotspots, and even satellite-based MEC servers. Each request comprises client and IMS service addresses, a list of MSs, life-cycle duration, and the current client location.

The immersive applications over MSC could decompose a monolithic IMS into ordered, location-aware subtasks known as MSs. Figure 1 illustrates an MSs chain for an IMS [Shen et al. 2023], where the MSC operates on a single MEC server or is distributed between multiple MEC servers, ensuring that the chain order is maintained. In this IMS example, the chain start with the *Synchronization Manager* (SM), which is responsible for feature extraction and preprocessing of input data from cameras, HMG, or multiple sensors embedded in immersion devices. This MS performs global *Simultaneous Localization and Mapping* (SLAM) to prevent accumulated positioning errors while tracking user inputs. Additionally, it synchronizes multi-user interactions, ensuring real-time feedback integration within a single session.

Afterwards, the *Personalized Location* (PL) receives the global mapping of users in a session from SM and instantiates *Virtual Object*s (VOs) placement position, dealing with users' mobility related to requests. The *Depth Sensing* receives each VO positioning from PL, filters foreground and background, classifies the VOs, and defines two flows. The first flow leads to *Personalized* MS to process near VOs with a unique perspective for each user, which presents an object occlusion step for seemingly merging or combining closest objects and hand-tracking for providing in-depth user-hand information. On the other hand, the second flow originated at *Depth Sensing* follows to *General Purpose* (GP) MS, which processes far distance VOs, or the background, and can cache previously rendered VOs to multiple users of the same session. In the final part of the chain, the *Rendering* MS receives both flows and synthesizes the captured scene, overlaying with VOs. The *Encoder* MS converts the processed scene to the video format, and the *Transcoder* MS adapts the video to inconstant network conditions, preventing stalls and re-buffering events.

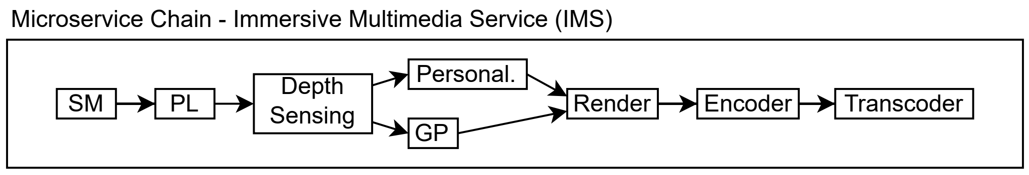


Figure 1. Micro-service chain overview

The orchestrator groups the sessions of mobile users with common views or shared immersive experiences in MS chain flows, allowing instantiation of an MS to multiple users in the same MEC server instance. In this sense, MSs can be shared between multiple users, reducing computational redundancy and improving system efficiency. For example, MS sharing allows multiple users in the same IMS environment to use a single rendering instance, lowering the system's computational overhead and power consumption. The orchestrator must manage the allocation and reallocated of MSs to guarantee essential performance indicators for IMS, such as low latency, high throughput, and reliability despite varying network conditions or resource availability. In addition, the orchestration must iteratively monitors the network to map MSs according to the availability of resources on the MEC servers available in the network, given the resource consumption of each MS. This communication is carried out throughout the network, centralizing orchestration in the defined central server, although it does not represent the starting point of the network.

3.2. Network environment with MEC Failure

We consider an undirected graph $G = (V, E)$ to model the overall architecture, where V representing MEC servers (*e.g.*, cloud servers, micro data centers, and Baseband units) and E denoting their interconnections. Each MS chain is instantiated along these MEC servers, from a source (central cloud) to a destination (MEC server or network anchor point closer to the user), minimizing the accumulated latency $D(v, ms_j)$. Each micro-service s_j consumes computational and bandwidth resources, denoted as C_{s_j} and B_{s_j}, respectively. These resources must be allocated within the capacity of each server $v \in V$, with server v having a total capacity C_v. Three instantiation queues, Q_m for continuous user sessions in motion and Q_i for new user sessions, and Q_b for backup chain, manage the instantiation of MSs chain while adhering to resource and latency constraints. Each queue handles the cumulative latency $D(v, ms_j)$ and the route $R(v, ms_j)$, representing the servers along which a given user consumes micro-services from a set of available services.

We model a failure scenario for an edge node or MS to evaluate the system's resiliency and recovery capabilities, which impacts ongoing sessions and diminishes the overall QoS for users. MEC failures can arise from hardware/software malfunctions, network or power outages, or resource overload, leading to a sudden unavailability of a specific MEC server or MS. Figure 2 shows an example of a MS chain deployed on a set of 6 MEC servers during the timestamp T_1. Afterward, at the timestamp T_2, a specific MEC server presents a fail and drops its set of running MS (*i.e.*, *GP* and *Personal*). In this sense, the orchestrator must continuous monitoring of each MS on the set of MEC server to evaluating the availability of networks and resources, where the orchestrator must reinstantiate based on its reinstantiation policy in a free and close MEC server, reconnecting the previous chain. In our example of Figure 2, the MSs (*i.e.*, *GP* and *Personal*) are reinstantiate on an available MEC server and it is set-up a new chain link.

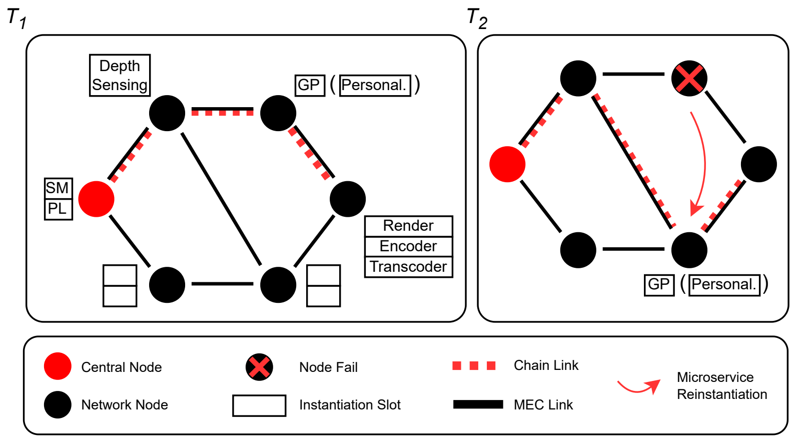


Figure 2. Failure Model

A failure behavior scenario is introduced along with the MEC environment to test the system reaction when a crash occurs, which consists in the probability of failure P_f for an MEC server $P_f(n)$. In this sense, one or more critical MEC servers that host a specific MS could become unavailable during the service execution. For example, in the modeled IMS, we could mean disabling an MEC server that processes real-time rendering or streaming, causing a cascade effects throughout the MS chain, including increased latencies, interrupted user experiences, or failures in dependent MS. These disruptions create a controlled scenario to measure the system's ability to detect, isolate, and mitigate the impact of the failure. Hence, this failure approach ensures that the MSC orchestrator is able to deliver robust and reliable performance, even under adverse conditions, essential for the continuity of IMS that rely on low-latency and high-reliability environments.

3.3. Orchestrator Operations

GAs offer a powerful approach to optimizing the distribution of IMS flows, particularly in dynamic environments such as MEC and 5G networks [Li et al. 2024]. The IMSCO Orchestrator addresses the problem of mapping and routing MSC by continuously monitoring network computing resources and leveraging evolutionary computing techniques to optimize service instantiation. The GA technique is a simple yet fast converging solution, making it suitable for environments with limited resources. It helps balance the exploration of new possibilities with the exploitation of existing knowledge, which is important in large, dynamic problem spaces.

The GA optimizes MSC allocation by determining the optimal routing paths, which minimize cost and latency. The process starts by creating the population *N*, which represent a path solution between a source (src) and a destination (dst) with suitable network nodes for MSC placement, as shown in Algorithm 1. The algorithm analyzes each individual using a fitness function based on network metrics like latency, memory consumption, bandwidth availability, and CPU load to maximize resource reuse. It then refines solutions by crossover and mutation, which combine pieces of multiple solutions with probability p_c and introduce minor random changes to explore new possibilities and avoid local optima with probability p_m. Tournament selection is employed to choose the best candidates for the next generation, balancing the need to preserve high-quality solutions while maintaining genetic diversity to foster continued evolution.

After all generations, the algorithm chooses the most efficient MSC placement and routing configuration as the final solution. It determines the path's overall delay and checks for compliance with the threshold. When delay exceeds the limit, the algorithm fails, indicating no solution. Otherwise, it returns final routing information and delay, optimizing MS deployment for resource efficiency and network performance. This adaptive optimization technique improves network resilience by dynamically addressing outages, congestion, and resource fluctuations, assuring constant service and better QoS guarantees.

This orchestrator's complexity is $O(P \times G \times n^2)$, with P for population, G for generations, and n^2 for fitness function times selection, crossover, and mutation operations. In the event of a failure, the GA orchestrator is more resilient due to its selectivity with the best available solutions, although it has a higher computational cost. The reuse and adaptability functions guarantee efficient resource recovery and management.

4. Evaluation

In this section, we introduce the resilience evaluation method and the results of implementing the failure scenario. Specifically, we analyze the performance of the

Algorithm 1: Genetic Algorithm for MSC Instantiation

```
   Input: $G(V, E)$, $S$
   Output: $R(v, ms_j)$ (route), $D(v, ms_j)$ (accumulated latency)
1  Initialize population with $N$ random individuals based on network state
2  for each generation up to $G_{max}$ do
3      for each individual in the population do
4          Evaluate fitness using total resource cost and path latency
5      while not converged do
6          Apply crossover with probability $p_c$ and mutation with probability $p_m$
7          Calculate fitness for each new solution
8          Select individuals via tournament selection
9          Update the current population with the newly selected individuals
10 $(R, D) \leftarrow SelectBestIndividual()$
11 if $\sum D(v, ms_j) >$ threshold then
12     return Failure
13 else
14     return $R(v, ms_j)$, $\sum D(v, ms_j)$
```

orchestrator in terms of well-known metrics, namely acceptance ratio, CPU and cache utilization, decision time, recovery success index, and time to recover.

4.1. Simulation setup

We considered a NetworkX and Python3-based simulator[1], and we modeled the MEC topology as a graph to represent links, MEC servers, latency, computational/caching resources, and failure behavior as introduced in Section 3. We considered network topology of Luxembourg City with 35 MEC servers and inter-server latencies with a mean one millisecond Poisson distribution [Akhtar et al. 2021]. The network topology starts at a Central Cloud controller located in MEC server 3. We consider one-third of nodes to be edge servers selected through centrality calculation, choosing the most connected ones, and the remaining nodes as base stations. The Central Cloud controller connects these servers through 1 Gbps connections between the nodes. The user mobility applied defines a user speed of up to 60 km/h, as commonly expected in urban centers.

We simulate urban vehicular mobility from Luxembourg City through SUMO (Simulation of Urban Mobility) and OpenStreetMap data. We assume tourism, shopping, or game IMSs sessions that start with four users but can increase or decrease later on according to user mobility or service failure. We simulated 50 sessions with an arrival time based on a Poisson distribution mean of 15 s, lasting 120 s, and a maximum round-trip latency of 12 ms. IMS sessions are accepted if they meet latency and resource requirements, dividing the transmission into general and personalized views. We consider a variable resource availability considering four levels, *i.e.*, 0.95, 0.97, 0.99, and 1. This represents the total resource decrease at failure events, helping to assess the resilience of each orchestrator with increasing unavailability of the MEC topology. Finally, we

[1]https://gitlab.com/gercomlacis/fog-vanet/multi-user-sfc

performed 33 simulations to ensure a 95% confidence interval. Table 2 summarizes all simulation parameters applied for our simulation.

Table 2. Simulation Parameters

Parameter	Value
Total nodes	35 [Akhtar et al. 2021]
MEC servers	$1/3$ Servers [Akhtar et al. 2021]
Latency of inter-server links	uniform $(1, 2)$ ms
Link bandwidth capacity	5 Gbps
User resource load request	$1/3$ CPU and $1/3$ Cache per MEC server
Incoming session rate	Poisson ($\alpha = 20$)
Session request lifetime	Poisson ($\alpha = 120$) sec
Latency threshold per session	6 ms
Session request size	up to 16
GA Population	60 individuals
GA Generations	80 generations
Failure distribution	3 per simulation
CPU cycles / megabit	$10e^6$
Simulation Iterations	30 i.

We compare the performance of the IMSCO orchestrator with existing orchestrators, namely Greedy, MusFiCo, and MSF orchestrators. Specifically, the **Greedy Orchestrator** performs the mapping of the MS chain by iteratively checking the computational resources available in the network, evaluates the available MEC servers with respect to the order of instantiation of the MSs, which aims to minimize the latency and shortest path between src and dst. The **MusFiCo Orchestrator** [Santos et al. 2022] finds the fastest path and check processing (CPU), memory, and bandwidth resources, which tracks the network resources relieves instantiation processes from recalculating their feasibility for MSs on a given MEC server, efficiently completing the MS chain. The **MSF Orchestrator** [Santos et al. 2023] continuously adapts the entire MSF chain on the edge servers but does not implement active resource reuse. Finally, the **IMSCO** Orchestrator iteratively verify possible solutions for request instantiation and reinstantiation in the event of failures. In this sense, it considers a GA to solve the problem of mapping and routing MS by monitoring the network's computing resources, using evolutionary computing techniques to optimize instantiation as introduced in Section 3.3.

Beyond performance analysis, orchestrator resilience is assessed based on their effectiveness in maintaining service continuity under adverse conditions. In this sense, the Greedy Orchestrator lacks re-evaluation of past decisions for instantiation and MEC server verification, and a failed MS chain would reenter the instantiation queue. The MusFiCo Orchestrator only adjusts the route of the existing chain instead of fully re-instantiating the MS chain, limiting its ability to recover when affecting critical MEC servers effectively. The MSF Orchestrator can benefit from the passive reuse and quick re-instantiation strategies in a way to efficiently manage resources and mitigate disruptions in a dynamic scenario. Finally, the IMSCO Orchestrator is more resilient due to its selectivity with the best available solutions, although it has a higher computational cost. The reuse and adaptability functions guarantee efficient resource recovery and management.

We consider well-known metrics for evaluating the orchestrators, namely acceptance ratio, CPU usage, cache usage, bandwidth consumption, recovery success rate, time to recover, and latency degradation. Specifically, acceptance ratio means the number of accepted sessions divided by the total number of sessions. CPU Utilization denotes the CPU usage by accepted MSC requests versus total edge server resources. Cache usage means the ratio of memory usage for all MSC requests with the sum of all edge servers. Bandwidth utilization measures the link usage by accepted MS chains versus total edge connection resources. The time to recover captures the time it took the failed event until the service fully recovered in another topology MEC server or queue advance, while the recovery success ratio denotes the proportion of failed services that were successfully restored. Finally, the latency degradation metric evaluates the latency impact and how the orchestrator handles the failure event, with higher than zero values representing a latency decrease and positive effect and the opposite effect for lower than zero values.

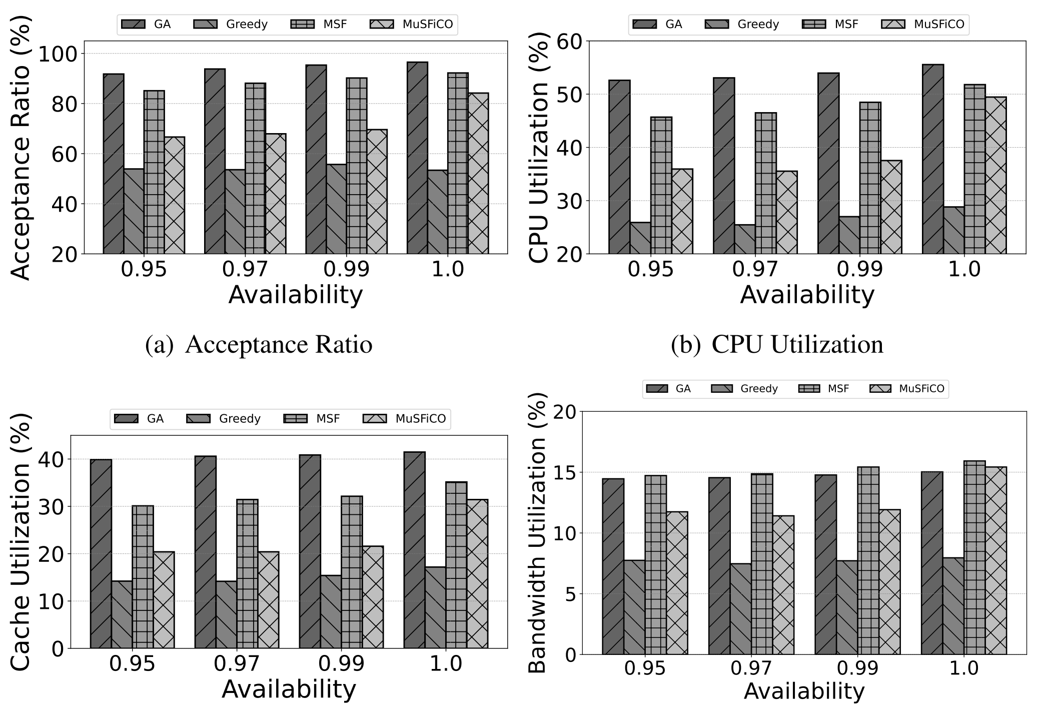
4.2. Results

Figure 3 shows the performance of the orchestration algorithms in scenarios with variable resource availability during failures. By analyzing the results of Figure 3(a), we observe that the IMSCO orchestrator outperforms in terms of acceptance ratio the greedy, MusFiCo, and MSF orchestrator by 43%, 13%, and 5%, respectively. This reflects the capacity of the IMSCO orchestrator to manage sessions efficiently, since higher acceptance ratio indicates a robust mechanism capable of handling numerous requests, thereby enhancing user satisfaction. This behavior of IMSCO orchestrator is attributed to its meta-heuristic, which enables a more comprehensive evaluation of the network, maximizing resource reuse and optimizing resource allocation. The IMSCO orchestrator mitigates convergence to local optima, an obvious limitation of the greedy orchestrator, which has the lowest acceptance rate. On the other hand, MSF performs well in terms of acceptance ratio, with results above 92% in scenarios without failures, benefiting from passive reuse and efficient service reinstantiation. However, it suffers degradation of up to around 5% in contexts of reduced availability. On the other hand, MuSFiCO has a higher rejection rate of around 54%, as it does not efficiently adapt the MSs after connection changes. This method only adjusts the routes from the last MS to the mobile users, introducing additional latency and increasing the likelihood of blocked sessions. This problem is exacerbated by MEC server failures and reduced availability, resulting in a drop of up to 15% in the availability change.

Figure 3(b) shows the CPU utilization for the evaluated orchestration algorithms in scenarios with variable resource availability during failures. By analyzing the results, we conclude that the IMSCO orchestrator with the consistently highest CPU utilization is a more costly orchestrator to deal with the service due to the orchestration functionality of genetic operation. However, the CPU utilization maintains a variation of around 3%, representing efficient service maintenance. Besides, due to passive reuse, the second highest CPU consumption is MSF, presenting a variation of around 4%, similarly to MusFiCo with above 12% of variation. The greedy orchestrator presents the lowest CPU utilization due to low utilization and incomplete orchestration. Finally, let us analyze the computational complexity of the analyzed MSC orchestrator. The Greedy Algorithm

Orchestrator has computational complexity $O(m \times V)$, with m representing the total MSs and V representing the MEC topology. The MusFiCo Orchestrator complexity is $O(m \times V \times (V-1))$, where V represents all MEC servers, and $V-1$ represents that the orchestrator skips the same MEC server. The MSF Orchestrator complexity is $O(m \times V^2)$. Finally, the IMSCO Orchestrator complexity is $O(P \times G \times n^2)$, with P for population, G for generations, and n^2 for fitness function times selection, crossover, and mutation operations.

Figure 3(c) depicts the cache utilization for the evaluated orchestration algorithms in scenarios with variable resource availability during failures. The results present a similar performance with CPU utilization because of network resource behavior, where the IMSCO orchestrator uses around 40% of the total available cache with a slight variation of 1%. The second best performance is the MSF, followed by MusFiCo, and the greedy performance is the worst for cache utilization. This result occurs due to better leverage of available resources in the topology from the compared orchestrators.



(a) Acceptance Ratio (b) CPU Utilization

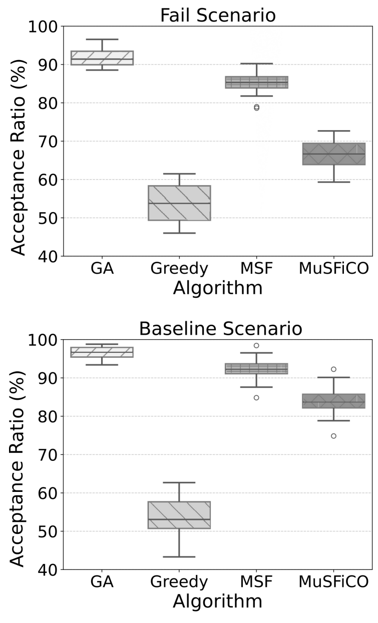
(c) Cache Utilization (d) Bandwidth Utilization

Figure 3. Orchestrators resource availability performance in failure events

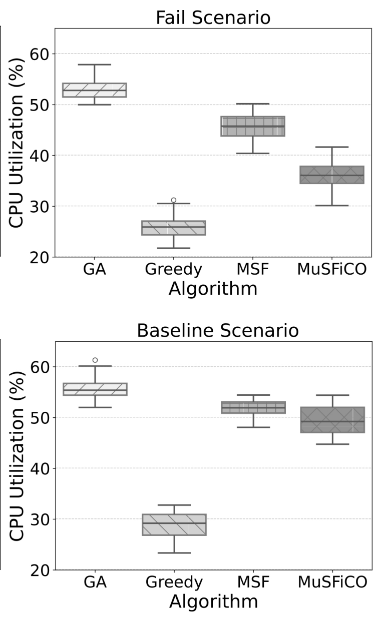
Figure 3(d) presents the bandwidth utilization in the connection link between topology and users to provide insight into the network performance and data transfer efficiency. The results reveal a proximate performance for the IMSCO orchestrator and MSF due to better performance with user mobility along the simulation, with less variation with less resource availability in failure events. Conversely, MusFiCo presents a great variation of almost 3% due to no mobility mechanism for reuse. In summary, the IMSCO orchestrator has a better performance for session acceptance ratio and less resource utilization variation with less availability in failure events, but demands more resources due to higher complexity than other, without reaching the maximum value in the proposed scenario.

Figure 4 shows the evaluation results for a scenario with failure and without fail events. By analyzing the results of Figure 4(a), we conclude that the session acceptance ratio of the IMSCO orchestrator varies around 5% in the median value regardless of the scenario with or without MEC server failure, which presents less variation in the acceptance distribution (*i.e.*, around 10%). On the other hand, MSF and MusFiCo are second and third better performance for acceptance ratio, obtaining 6% and 18%, respectively, with higher varying distribution values. Finally, Greedy orchestration presents a worst acceptance ratio stabler at around 55% in baseline and fail scenarios.

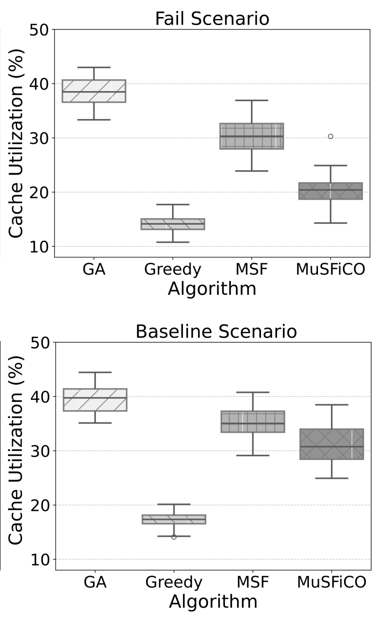
Figure 4(b) presents the CPU utilization in a scenario with MEC failure, showing a slight median value difference of 1% for the IMSCO orchestrator and a 10% distribution variation. This result depicts a stable behavior of IMSCO orchestrator, with a 6% and 13% median fluctuation for MSF and MusFiCo, respectively. As the worst value for CPU usage, the greedy orchestrator presents a median value of around 6% in both cases. Similarly, in Figure 4(c), the IMSCO orchestrator surpasses the second-best orchestrator, MSF, with a 1% variation compared to 5%. MusFiCo has a greater variation of 10% for median value, and the greedy orchestrator stays under 20% of cache utilization. In summary, these results show greater stability of IMSCO orchestrator with a session acceptance ratio of nearly 100% in the overall simulation results, but consuming a computational resource similar to MSF in some metrics, evidencing a valid trade-off.



(a) Acceptance Ratio



(b) CPU Utilization

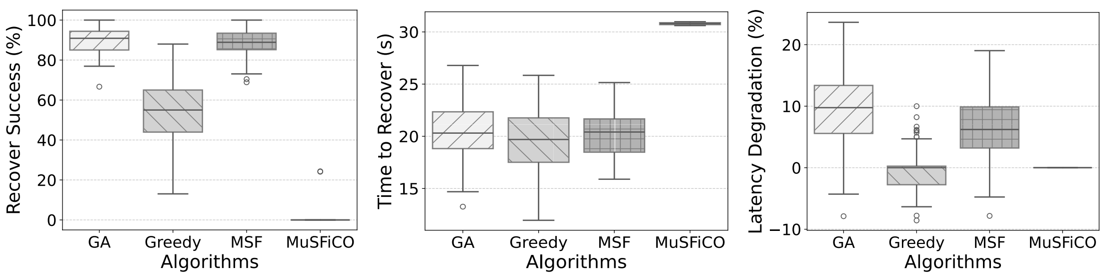


(c) Cache Utilization

Figure 4. Comparison of different orchestrators with and without failure events

Figure 5 presents the resilient recovery comparison for the evaluated orchestrators. For instance, Figure 5(a) shows the recovery success ratio as an indicator of the resilience, where high restoration rates of failed services contribute to the overall dependability of the system. By analyzing the results, we can see a similar variation for the IMSCO orchestrator and MSF for a median value around 90% due to similar management mechanisms for reuse and MEC server verification. Even with similar performance for recovery, IMSCO orchestrator has a higher acceptance ratio than MSF, being an attractive alternative for orchestration. Greedy orchestrators have a wider distribution of around 80% due to instability in a recovery opportunity between resource utilization and queue management, leading to uncertain behavior. MSF does not apply a recover mechanism and maintains an off state in a lost MEC server, and the outlier value present in the result appeared due to MEC server fast reconnection after fail.

Figure 5(b) shows a time spent to service reinstantiation after recovery, which is a crucial consideration for maintaining service continuity. A shorter recovery time can significantly influence the perception of reliability and QoS from the user's perspective. The results reveal a similar median of 20 seconds for the IMSCO orchestrator, greedy, and MSF. This result occurs due to the service reinstantiation procedure executing in queue reentry and MS relocation. MusFiCo stays above 30 seconds because of the nonexistence of a recovery mechanism for queue reentry, resulting in the outlier event time. Finally, Figure 5(c) shows a greater median value for the IMSCO orchestrator in below 10%, representing a lower latency sustain and less impact to service maintenance, owing to greater resource reuse in MS instantiation. Due to passive reuse, the second best value is MSF of around 6% and more stable degradation. The greedy orchestrator presents a negative degradation distribution, causing a higher latency in the simulations in many failure events caused by the least efficient path selection and greater cascade effect in an instantiated chain.



(a) Recover Success (b) Time to Recover (c) Latency Degradation

Figure 5. Orchestrator resilience performance

5. Conclusion

This paper introduces an intelligent MSC orchestration algorithm for IMS under failure scenarios, called IMSCO. Specifically, the IMSCO orchestrator relies on a GA to enhance failure handling, which adapts dynamically and minimizes service disruptions, offering a robust backup mechanism and ensuring minimal latency increases. Evaluation results showed that IMSCO increases acceptance ratio in 43%, 13%, and 5% compared to Greedy, MusFiCo, and MSF, respectively. Moreover, IMSCO algorithm obtain a 90% recovery rate and a median latency degradation around 10% than other algorithms - representing decreasing latency with desirable result, minimizing downtime and enhancing latency after a failure event, compared to literature. The findings underscore the importance of adopting more advanced orchestration approaches, such as the IMSCO, which offers a balanced trade-off between computational cost and superior service quality maintenance and recovery behavior. For future work, we are planning to combine both GA for optimization tasks and Machine Learning (ML), such as Reinforcement Learning and Deep Learning, for dynamic learning and adaptation. While GA can provide effective solutions for long-term planning and resource distribution, ML can enhance the system's reactivity and resilience, ensuring optimal orchestration under varying network conditions.

Acknowledgements

This research was partially sponsored by São Paulo Research Foundation (FAPESP) grant 2023/00673-7, and also by CNPq grant 405940/2022-0 and CAPES grant 88887.954253/2024-00

References

Akhtar, N., Matta, I., Raza, A., Goratti, L., Braun, T., and Esposito, F. (2021). Managing chains of application functions over multi-technology edge networks. *IEEE Transactions on Network and Service Management*, 18(1):511–525.

Bai, J., Chang, X., Machida, F., Trivedi, K. S., and Li, Y. (2023). Model-driven dependability assessment of microservice chains in mec-enabled iot. *IEEE Transactions on Services Computing*, 16(4):2769–2785.

Di Cicco, N., Poltronieri, F., Santos, J., Zaccarini, M., Tortonesi, M., Stefanelli, C., and De Turck, F. (2024). Multi-objective scheduling and resource allocation of kubernetes replicas across the compute continuum. In *2024 20th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE.

Huang, H. and Wu, W. (2022). Hypersfp: Fault-tolerant service function chain provision on programmable switches in data centers. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE.

Huang, Z. and Friderikos, V. (2023). Optimal mobility-aware wireless edge cloud support for the metaverse. *Future Internet*, 15(2):47.

Huang, Z. and Simeone, O. (2023). *Edge Cloud and Network Optimization for Mobile Augmented Reality*. PhD thesis, King's College London.

Li, J., Pang, J., and Fan, X. (2024). Optimization of 5g base station coverage based on self-adaptive mutation genetic algorithm. *Computer Communications*, 225:83–95.

Medeiros, A., Di Maio, A., Braun, T., and Neto, A. (2023). Tenet: Adaptive service chain orchestrator for mec-enabled low-latency 6dof virtual reality. *IEEE Transactions on Network and Service Management*.

Santos, H., Martins, B., Rosário, D., Cerqueira, E., and Braun, T. (2023). Mobility-aware service function chaining orchestration for multi-user augmented reality. In *2023 IEEE 48th Conference on Local Computer Networks (LCN)*, pages 1–9. IEEE.

Santos, H., Rosario, D., Cerqueira, E., and Braun, T. (2022). Multi-criteria service function chaining orchestration for multi-user virtual reality services. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 6360–6365. IEEE.

Shen, X., Gao, J., Li, M., Zhou, C., Hu, S., He, M., and Zhuang, W. (2023). Toward immersive communications in 6g. *Frontiers in Computer Science*, 4:1068478.

Zawish, M., Dharejo, F. A., Khowaja, S. A., Raza, S., Davy, S., Dev, K., and Bellavista, P. (2024). Ai and 6g into the metaverse: Fundamentals, challenges and future research trends. *IEEE Open Journal of the Communications Society*, 5:730–778.