RESEARCH-ARTICLE

Towards Cognitive Microservice Orchestration in the Multi-Cloud Continuum

ELISA DRUDI, University of Bologna, Bologna, BO, Italy

GIUSEPPE DI MODICA, University of Bologna, Bologna, BO, Italy

LUCA FOSCHINI, University of Bologna, Bologna, BO, Italy

STEFANO GALANTINO, Polytechnic of Turin, Turin, TO, Italy

FULVIO RISSO, Polytechnic of Turin, Turin, TO, Italy

UMBERTO ARDINGHI, University of Ferrara, Ferrara, FE, Italy

View all

Published: 14 December 2025

Citation in BibTeX format

MIDDLEWARE '25: 26th International Middleware Conference
December 15 - 19, 2025
TN, Nashville, USA

Conference Sponsors:
ACM

Towards Cognitive Microservice Orchestration in the Multi-Cloud Continuum

Elisa Drudi*
Giuseppe Di Modica*
Luca Foschini*
University of Bologna
Bologna, Italy

Stefano Galantino‡
Fulvio Risso‡
Politecnico di Torino
Torino, Italy

Umberto Ardinghi†
Filippo Poltronieri†
Mauro Tortonesi†
University of Ferrara
Ferrara, Italy

ABSTRACT

The proliferation of decentralized applications calls for complex deployments across the highly heterogeneous Cloud Continuum, an overlay infrastructure encompassing traditional public/private clouds, fog clusters, and edge devices. Orchestrating distributed applications within such a non-uniform environment presents significant challenges, particularly in ensuring the fulfillment of critical Quality of Service (QoS) requirements. Achieving the objectives posed by this research challenge requires addressing several key needs, including the development of a robust system for virtualizing heterogeneous resources spanning multiple domains, the design of an intelligent scheduler capable of implementing deployment schemes that respect application-specific QoS requirements, and the provision of an orchestration mechanism that simplifies the task of specifying application functional and non-functional requirements for the application owners while automating the application roll-out. In this work we propose a framework that integrates existing tools that individually address the aforementioned needs by blending them in a unified manner. Preliminary experiments conducted on a software prototype demonstrate the feasibility of the proposed approach.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Software as a service orchestration system**.

KEYWORDS

Cloud Continuum, Multi-cloud, Cluster federation, QoS-aware Orchestration

ACM Reference Format:
Elisa Drudi, Giuseppe Di Modica, Luca Foschini, Stefano Galantino, Fulvio Risso, Umberto Ardinghi, Filippo Poltronieri, and Mauro Tortonesi. 2025. Towards Cognitive Microservice Orchestration in the Multi-Cloud Continuum. In *Proceedings of 3rd International Workshop on Middleware for the Computing Continuum (Mid4CC '25).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3774898.3778038

Mid4CC '25, December 15–19, 2025, Nashville, TN, USA

ACM ISBN 979-8-4007-2301-8/2025/12.
https://doi.org/10.1145/3774898.3778038

1 INTRODUCTION

The evolution of computing has long been guided by the continuous pursuit of efficiency, adaptability, and optimal resource utilization. Over past decades, the computing paradigm has progressively shifted from hardware-specific and tightly coupled systems to virtualized infrastructures and, eventually, to cloud-native architectures. Each evolutionary step has been motivated by the need to abstract physical resources and manage them more flexibly, enabling higher levels of scalability, resilience, and cost-effectiveness. This trajectory has expanded into the *cloud continuum*, which integrates public, private, and hybrid clouds with edge and IoT resources [11]. According to the new paradigm, organizations can freely move workloads across infrastructures without rigid boundaries, leveraging the security, privacy, and control of private environments, the scalability and performance of public clouds, and the real-time responsiveness of edge and IoT devices. By enabling workloads to execute on the most suitable platform across the computing spectrum, the continuum paradigm is expected to further enhance efficiency, ensuring that computational tasks are processed where resources, latency, and energy consumption are optimal.

Although the cloud continuum offers unprecedented flexibility, its heterogeneity also introduces significant management issues [7]. In the continuum, resources differ not only in capacity and performance, but also in access policies and management interfaces, making manual oversight of infrastructures and workloads impractical. For this reason, service orchestrators[1] are essential to achieve target operational and quality objectives. Orchestration solutions such as Kubernetes are becoming useful tools for service providers as they seek to deploy and manage increasingly complex workflows across a wide range of computing resources, including multi-cloud and cloud continuum scenarios. The heterogeneity of the cloud–continuum introduces significant challenges for service management, particularly in the context of identifying convenient deployment strategies for distributed applications with substantial computational demands. Smart deployment calls for the systematic exploration of multiple configuration alternatives, coupled with a rigorous evaluation of the QoS requirements imposed by the requesting applications. To provide a concrete example, inter-site communication latencies across geographically distributed computing environments can significantly hinder the performance of complex workflows, particularly those involving tightly coupled distributed applications. Effectively maintaining these latencies below critical thresholds exemplifies a key task that requires intelligent deployment strategies, continuous monitoring, and dynamic

[1] tools capable of automatically selecting, coordinating, and reconfiguring hardware and software resources based on the current state of the overall infrastructure

adaptation of resource allocations to meet application-specific QoS requirements.

Although several state-of-the-art orchestration tools consider such requirements [6, 13], there remains a lack of approaches that provide application providers with mechanisms to easily specify the QoS requirements of distributed applications, and to seamlessly deploy them across multiple, heterogeneous cloud environments, according to deployment schemes that enforce the desired QoS levels. To address the heterogeneity and complexity inherent in cloud–continuum, we propose an approach based on three key pillars: cloud resource federation, QoS-aware orchestration mechanisms, and the adoption of portable application models. *Resource Federation* addresses the fragmentation and heterogeneity of the cloud–continuum by providing a unified and coherent view of distributed resources. This unification is crucial to ensure seamless accessibility and secure resource sharing across multiple administrative domains and infrastructures. *QoS-Aware Orchestration* focuses on guaranteeing that provisioned applications operate in accordance with their performance requirements—such as latency, throughput, and availability—despite the heterogeneity and dynamicity of the underlying computing environments. *Portable Application Models* introduce a technology-agnostic abstraction layer through which application providers can describe topologies, workflows, and QoS requirements independently of specific cloud stacks. This abstraction conceals runtime complexity while enabling the orchestrator to generate deployment and management strategies that can adapt to diverse infrastructures, thereby mitigating vendor lock-in and promoting interoperability within the cloud–continuum. Building upon these principles, this work integrates open-source tools [2, 10, 17], each individually designed to address one of the aforementioned needs, into a unified framework. The resulting platform provides service providers with a comprehensive and effective solution for the intelligent and QoS-aware roll-out of distributed applications across heterogeneous cloud–continuum environments.

The remainder of this paper is organized as follows. Section 2 reviews the state of the art. Section 3 presents the architecture of the OPTIMUUM framework and describes its main components. Section 4 discusses the results of an experiment conducted in a controlled environment to validate the proposed approach. Finally, Section 5 concludes the paper and outlines directions for future work.

2 RELATED WORK

In this section, we analyze the relevant literature addressing the research topics outlined in the introduction. Concerning *Resource federation*, several initiatives partially tackle the associated challenges but still leave significant gaps unaddressed. KubeEdge [18] extends Kubernetes to support device and edge management, enabling deployment and metadata synchronization across cloud and edge environments. However, it focuses on specific communication patterns and does not generalize to broader multi-cluster scenarios. Karmada [5] targets multi-cluster application management under a master–worker model, yet it lacks true peer-level federation among clusters and relies on external components to enable inter-cluster networking. Liqo [10] directly addresses these limitations by enabling dynamic and seamless multi-cluster Kubernetes management across on-premises, cloud, and edge environments. Through a peering mechanism, each remote cluster is abstracted as a virtual node accessible via the local Kubernetes API, thereby providing a unified resource view and secure inter-cluster connectivity. Regarding *QoS-aware orchestration*, the research community has made several efforts to enhance the management of distributed applications by ensuring that QoS objectives are consistently achieved across heterogeneous infrastructures. Frameworks such as ROMA [6] focus on optimizing resource utilization for microservice-based applications in multi-tier computing and network environments, while hierarchical and analytical models proposed by Pereira et al. [13] aim to improve availability and scalability in edge–cloud scenarios. Other studies [15] adopt model-based techniques to automatically generate and assign multiple software deployment plans to large sets of Internet of Things (IoT) and edge devices operating in continuously evolving contexts. In parallel, Digital Twin (DT) approaches have emerged as powerful tools for performing what-if analyses of applications deployed in multi-cloud environments [3]. DTs enable rapid and parallel exploration of multiple configurations, supporting the simulation of orchestration behaviors such as autoscaling and scheduling, and thereby allowing significant time and cost savings during the optimization process [12]. KubeTwin [2] represents a concrete realization of this paradigm within the Kubernetes ecosystem. It emulates orchestration actions, including pod scheduling, autoscaling, and inter-service communication—allowing developers to evaluate the impact of alternative deployment strategies on latency, throughput, and resource utilization. Focusing on *Portable Application Models*, a number of existing solutions have been proposed to support the modeling of distributed applications. Tools such as Docker Composeand Helm [4] facilitate the description of complex application architectures. However, they are tightly coupled to specific runtimes and consequently lack portability across heterogeneous environments. While infrastructure provisioning and configuration management tools such as Terraform [8] and Ansible [9] can be used to automate the configuration and deployment of diverse infrastructures and applications, they do not provide the high-level, platform-independent modeling features needed to describe portable systems. TOSCA [16] is a widely recognized OASIS standard that defines a metamodel useful for the description of distributed systems in a technology-agnostic manner. It offers a rich syntax for specifying both application topologies and the policies governing their quality and non-functional requirements. Alternative technology-agnostic modeling languages, such as CAMEL [1], are available; however, none yet match TOSCA in terms of maturity, community support, and standardization.

The approaches analyzed provide valuable insights into addressing specific requirements; however, they fall short of delivering a comprehensive solution. The OPTIMUUM platform aims to bridge this gap by integrating some of the aforementioned tools into a complete solution.

3 OPTIMUUM ARCHITECTURE

OPTIMUUM (OPTImized deployment of cloud-native applications in Multi-cloud and cloud continUUM environments) is an open-source[2] platform that offers a comprehensive solution to enable

[2]available at https://gitlab.com/MMw_Unibo/optimuum

efficient deployment and orchestration of cloud-native and distributed applications across multi-cloud and cloud continuum environments. OPTIMUUM fulfills the essential requirements of a QoS-aware, cloud-continuum orchestrator through the integration of three complementary tools: TORCH [17], Liqo [10], and KubeTwin [2]. In particular, Liqo serves as the abstraction layer responsible for federating multiple clusters of computing resources, each governed by a container orchestration system such as Kubernetes. KubeTwin, in turn, uses discrete-event simulation to perform informed decision-making regarding resource allocation, workload balancing, and performance optimization, providing deployment schemas that satisfy QoS objectives and meet user expectations. Finally, TORCH is a TOSCA orchestrator that acts as the integrative element between the end user and the aforementioned components.

The next sections describe the three tools in detail and illustrate the operational workflow of the OPTIMUUM platform.

3.1 Liqo

Liqo is an open-source[3] project that enables seamless workload orchestration across multiple Kubernetes clusters through a peer-to-peer federation model. It allows clusters to autonomously discover each other, establish secure peering sessions, and share computing, storage, and networking resources. Once peered, each remote cluster is collapsed into a virtual node, whose resources (CPU, memory, storage) become part of the local scheduling domain and are fully accessible through the standard Kubernetes API. From the developer's perspective, this abstraction creates a unified infrastructure view in which scheduling and offloading decisions can transparently span both local and remote clusters while preserving full Kubernetes compliance.

To guarantee secure connectivity, Liqo automatically establishes encrypted VPN tunnels using WireGuard, creating a protected network fabric that ensures confidentiality, integrity, and authentication of all inter-cluster communications. This secure overlay network enables workloads offloaded to remote clusters to interact with services and resources as if they were running locally, while allowing administrators to consistently enforce network and security policies across the entire federated environment.

Liqo's native observability and monitoring mechanisms are fundamental to OPTIMUUM's operation. By continuously collecting and exposing metrics on resource utilization, offloading performance, network latency, and policy compliance, Liqo delivers comprehensive visibility over the federated infrastructure. These metrics are then leveraged by the other components of the platform to guide optimization and workload placement decisions across the continuum.

3.2 KubeTwin

KubeTwin[4] is an open-source framework designed to create digital twins of container-based applications running atop Kubernetes. By accurately replicating Kubernetes orchestration mechanisms, including service management, networking, load balancing, and autoscaling, and simulating the behavior of the target distributed application, KubeTwin allows service providers to evaluate various deployment configurations and enables what-if scenario analyses for optimization tasks such as resource allocation, scheduling, and performance tuning [2]. In particular, to identify optimized scheduling decisions KubeTwin employs a metaheuristic-based optimizer that improve the overall performance of the application. Configuration parameters can also be fine-tuned to model additional constraints or simulate specific runtime conditions, increasing the flexibility of the optimization process.

For accurate simulation, KubeTwin requires a detailed description of both the distributed application and the underlying multi-cluster infrastructure. Within OPTIMUUM, this information is obtained partly from user inputs expressed as TOSCA models and partly from observability tools deployed on the infrastructure, such as Prometheus. By combining up-to-date infrastructure metrics with simulation-based evaluation, KubeTwin enables OPTIMUUM to generate deployment plans that satisfy QoS requirements while maintaining resource efficiency.

3.3 Torch

TORCH [17] is an open-source[5] TOSCA orchestrator capable of interpreting TOSCA application models and acting upon them to provision and manage application lifecycles on target infrastructures. It includes a user-friendly dashboard running on a PHP Lavarel framework through which users can specify both functional and non-functional requirements of their distributed applications as well as initiating and monitoring deployments, a set of BPMN-compliant workflows that encapsulate the provisioning logic in a technology- and vendor-independent manner and a modular service layer built around pluggable Java connectors, which separate generic orchestration workflows from the invocation of technology-specific APIs. This design makes TORCH highly extensible, as new technology stacks can be supported simply by developing lightweight plugins that translate abstract orchestration actions into the corresponding API calls of the target environment. As a result, TORCH can easily evolve to accommodate new cloud providers, tools, or infrastructure layers without altering its core logic. Through its support of TOSCA, TORCH allows users to model distributed applications, target infrastructures, and non-functional requirements such as QoS objectives in a portable, technology-agnostic way via TOSCA templates comprising nodes, relationships, and policies.

Overall, within OPTIMUUM, TORCH serves as the orchestration backbone, interpreting TOSCA models and translating them into actionable deployments over Liqo-federated clusters. It integrates seamlessly with KubeTwin, leveraging the optimizer's placement decisions to drive deployments that balance QoS objectives with resource efficiency.

[3] available at https://github.com/liqotech/liqo
[4] available at https://github.com/DSG-UniFE/KubeTwin
[5] available at https://gitlab.com/MMw_Unibo/optimuum with Liqo and KubeTwin integration

3.4 The OPTIMUUM workflow

Within the OPTIMUUM platform, TORCH, KubeTwin and Liqo cooperate within a unified operational workflow that enables federated and QoS-aware deployments across heterogeneous cloud-continuum infrastructures. Fig. 1 depicts how these three components interact.

The orchestration process begins when the user uploads through TORCH of a TOSCA-based specification of the target application and its QoS policies (1). TORCH validates the syntax of the model and translates it into an internal representation capturing both the application topology and the associated optimization policies. This processed TOSCA template is then asynchronously transmitted to KubeTwin (2), which acts as the optimization engine of the system. KubeTwin retrieves up-to-date infrastructure metrics exposed by Liqo, including resource availability and inter-cluster network conditions collected through Prometheus (3). Leveraging its digital twin simulation framework, KubeTwin reenacts deployment behaviours within a virtualized multi-cluster environment that reproduces the characteristics of the real infrastructure. This allows it to perform multiple what-if analyses and evaluate efficient placements according to the QoS and policy constraints defined in the TOSCA model. KubeTwin executes an optimization process based on the *Quantum Particle Swarm Optimization (QPSO)* algorithm [14], which efficiently explores the configuration space to identify the deployment that best satisfies the declared objectives while reducing computational overhead. Once the optimization is complete, KubeTwin enriches the TOSCA model with the optimal placement and replication details and returns it to TORCH (4). At this stage, TORCH takes charge of provisioning the application across the Liqo-federated clusters, relying on Kubernetes and Liqo APIs (5). Specifically, Liqo abstracts the underlying heterogeneity by exposing a unified resource view, enabling TORCH to deploy applications seamlessly across the continuum. Once the application is deployed, TORCH continues to interact with Liqo to monitor the runtime status of the infrastructure (6), ensuring that the deployment remains aligned with the conditions assumed during optimization.

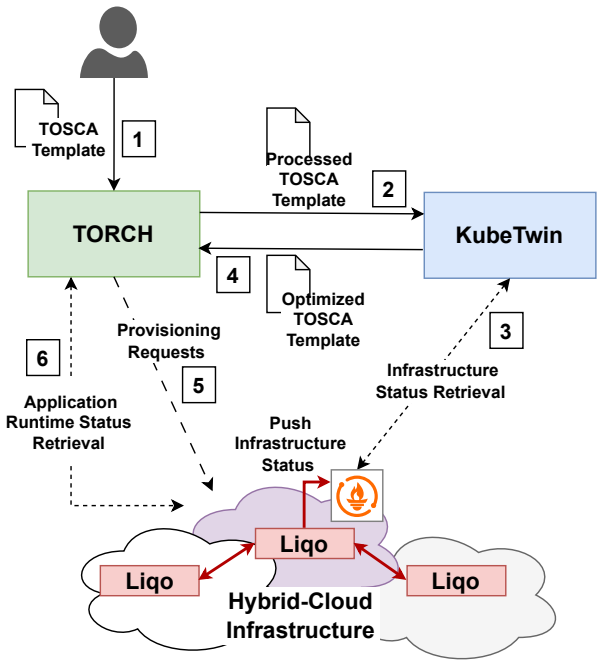

Figure 1: Conceptual Architecture of OPTIMUUM.

4 PRELIMINARY RESULTS

To assess the effectiveness of our proposal, we developed a prototype that covers steps 1 through 5 of the workflow illustrated in Fig. 1 and used it to deploy a distributed application across a multi-cluster infrastructure. We collected metrics to evaluate its ability to satisfy a predefined QoS objective constraining the maximum delay between application components and compared the results with standard baseline scheduling strategies to demonstrate the advantages of the proposed approach.

The experimental evaluation was conducted on a multi-cluster testbed composed of three Kubernetes clusters (C_1, C_2, and C_3) federated through Liqo. Cluster C_1 acts as the root of the federation and is equipped with 12 GB of RAM and eight virtual cores, each running at a base frequency of 2.645 GHz. In contrast, Cluster C_2 and Cluster C_3 each contribute 8 GB of RAM and four virtual cores at 2.645 GHz to the federation. To introduce further heterogeneity into the experimental setup, network latency was injected using the chaos engineering tool Chaos Mesh . Specifically, communication between Cluster C_1 and Cluster C_2 was subjected to an additional round-trip latency of 140 ms, while communication between Cluster C_1 and Cluster C_3 experienced 40 ms. Communication between Cluster C_2 and Cluster C_3 inherited both delays, as secure inter-cluster traffic is routed through Cluster C_1 by Liqo. We used a customized version of Istio's Bookinfo polyglot demo[6] as a reference application for the experimental validation. The application is composed of four microservices: *details*, which provides book metadata; *reviews*, which retrieves user reviews and queries the ratings service; *ratings*, which supplies the rating information associated with each review; and *productpage*, which aggregates data from the details and reviews services to render the final page; The Bookinfo application was chosen because it natively includes timeout mechanisms that trigger a failure when inter-service communication exceeds predefined thresholds, a behavior that can be represented through QoS policies that constrain the response time of specific service chains. To align its behavior with our experimental goals, the application was modified to retain a single timeout mechanism, between *productpage* and *reviews*, with a reduced threshold of 320 ms. In this new configuration, the placement of the microservices across the federation plays a critical role in the overall application performance. Deployments in which requests traverse multiple clusters are more likely to trigger timeouts due to injected latency, while concentrating all services within a single cluster can lead to resource contention under high load, increasing the response times.

To evaluate the OPTIMUUM platform, we modeled the Bookinfo application and its QoS policy, constraining the response time of the microservices, in TOSCA and deployed it over the multi-cluster testbed described above. The prototype, installed on a host equipped with an Intel(R) Core(TM) i7-12650H processor and 16 GB of RAM, took approximately 3 minutes to complete the application

[6] https://istio.io/latest/docs/examples/bookinfo/

Table 1: Deployments proposed by each Scheduling Strategy

Microservices	Baseline Deployments									OPTIMUUM Deployments								
	Standard			Spread			Co-location			At 500 rps			At 1000 rps			At 1500 rps		
	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3	C_1	C_2	C_3
productpage	1	3	3	2	3	2	0	0	7	3	2	2	3	2	2	3	1	3
reviews	1	3	3	2	2	3	0	0	7	0	0	7	1	0	6	3	0	4
ratings	1	3	3	2	3	2	0	1	6	0	0	7	1	0	6	1	1	5
details	1	3	3	2	2	3	0	1	6	3	4	0	3	4	0	2	4	1

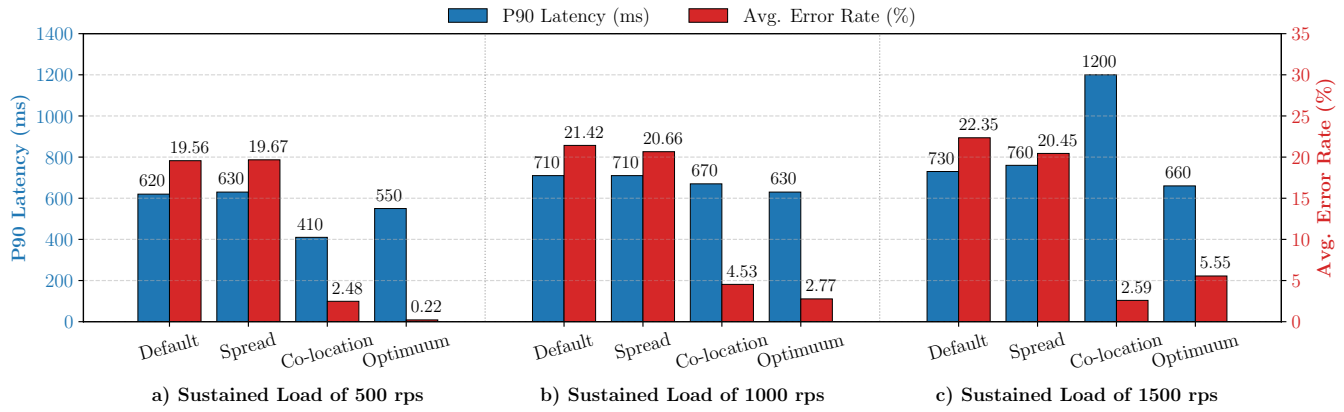

Figure 2: Comparison of Deployments under Different Sustained Load

deployment without accounting for the time required to generate an optimized placement schema. The duration of the optimization process depends on the configuration parameters provided to KubeTwin at installation time, such as the number of replicas per microservice, the expected workload intensity, and the duration of the simulated load. More complex configurations increase the search space explored by the optimizer, resulting in longer decision times. For example, across twenty optimization runs for our experimental setup, the prototype required on average 126.93 seconds to generate a placement schema for a target load of 500 requests per second (RPS), 206.57 seconds for a load of 1000 RPS and 296.11 seconds for a load of 1500 RPS.

To evaluate the quality of the placement decisions made by our platform, we compared its deployments against those produced by three baseline strategies implemented using standard Kubernetes mechanisms on top of the Liqo overlay. The first baseline relied on the Kubernetes default scheduler, which performs placement without any topology awareness. The second used the topology spread constraint feature, configured with a maximum skew of one, to promote balanced distribution of replicas across clusters. Finally, the third one employed pod affinity rules to implement a co-location strategy that favors the placement of all microservices within the same cluster to minimize inter-cluster communication. For a fair comparison, a replica budget of seven instances per microservice was applied to all baseline deployments and to our platform's scheduler, which was fine-tuned to generate three placement schemas optimized to sustain target loads of 500, 1000, and 1500 RPS.

Table 1 summarizes the placements selected by each strategy, highlighting the different deployment patterns produced by the baselines and our proposal. In all strategies, most workloads were assigned to Cluster C_2 and Cluster C_3, although Cluster C_1 had greater nominal capacity. This behavior can be attributed to the additional overhead on C_1, which, as the root of the federation, hosts several supporting services such as Prometheus pods for observability and the Liqo gateway for secure cross-cluster communication. The optimized placements prioritized Cluster C_3, and to a lesser extent Cluster C_1, as a compromise between available resources and cross-cluster latency. At lower loads, C_3 was favored due to its lower communication delay, while higher-load configurations adopted a more balanced distribution, shifting part of the workload to the other clusters to mitigate resource saturation.

To evaluate the runtime behavior, we subjected each deployment to synthetic load tests using Locust , an open-source load-testing tool that simulates concurrent user requests. We configured Locust to generate sustained HTTP request rates of 500, 1000, and 1500 requests per second (RPS) for a duration of 15 minutes per experiment. During each run, we collected aggregated metrics to assess the performance of each deployment. In particular, we measured the error rate, which accounts for the application-level timeouts triggered when service interactions exceed the predefined 320 ms threshold.

This metric reflects the ability of each placement to maintain the required response time for *productpage* service chain under varying load conditions. Additionally, we monitored the full response time as perceived by clients, which provides a more comprehensive view of system responsiveness beyond the built-in timeout logic.

Figure 2 summarizes the error rate and the 90th percentile (P90) of the response time observed across all deployment strategies at each target load. At 500 and 1000 RPS, the optimized placements achieved the lowest error rate among all strategies, including the Co-location strategy. This improvement stems from the optimizer's decision to place latency-sensitive microservices primarily in Clusters C_3 and C_1, a configuration that offers a good balance between available resources and communication overhead. Regarding response times, the optimized deployments consistently outperformed the default and topology-spread strategies, and achieved P90 latencies slightly lower than co-location at 1000 RPS. At 500 RPS, however, the co-location strategy benefits from reduced cross-cluster communication, yielding a better response time due to the limited impact of resource contention at this load level. At 1500 RPS, the optimizer adopted a more distributed placement to mitigate saturation, leading to a slightly higher error rate (5.55%) compared to co-location (2.59%). Nonetheless, it achieved a markedly better P90 latency, approximately half that of the co-location deployment. This behavior is expected, as co-locating all services within a single cluster amplifies resource contention. On the other hand, the error rate remains low in the co-location deployment even at this high load, probably because the application-level timeout is enforced after the productpage service begins processing a request; thus, queued requests contribute to longer response times without necessarily triggering the application timeouts.

Overall, these results show that optimized deployments generated by our proposal produce placements that effectively balance QoS awareness and resource efficiency, achieving low error rates while maintaining competitive response times across varying load conditions.

5 CONCLUSIONS AND FUTURE WORKS

In this paper we presented OPTIMUUM, an orchestration platform designed to enable efficient and QoS-aware deployment and management of applications across the cloud continuum. Building upon the key requirements of resource federation, QoS-driven optimization, and application portability, the platform integrates Liqo for cross-cluster resource federation, KubeTwin for QoS-aware placement optimization, and TORCH for TOSCA-based orchestration, realizing a unified workflow for provisioning distributed applications over heterogeneous infrastructures. Experimental validation on a multi-cluster Kubernetes testbed demonstrated that the proposed platform can streamline the deployment of distributed applications and generate optimized placements that effectively balance QoS awareness and resource efficiency. The resulting deployments achieved low error rates and competitive response times across varying load conditions, highlighting the benefits of QoS-guided optimization compared to common baseline scheduling strategies.

While the current prototype focuses on static deployment-time optimization, future work will extend these capabilities toward runtime monitoring and adaptive re-optimization to dynamically react to changing workloads and infrastructure conditions. Further experiments on larger and more heterogeneous testbeds will also be conducted to evaluate scalability and refine the platform decision-making capabilities.

ACKNOWLEDGMENTS

This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU).

REFERENCES

[1] Achilleas P. Achilleos, Kyriakos Kritikos, Alessandro Rossini, Georgia M. Kapitsaki, Jörg Domaschka, Michal Orzechowski, Daniel Seybold, Frank Griesinger, Nikolay Nikolov, Daniel Romero, and George A. Papadopoulos. 2019. The cloud application modelling and execution language. *Journal of Cloud Computing* 8, 1 (2019), 20. https://doi.org/10.1186/s13677-019-0138-7

[2] Davide Borsatti, Walter Cerroni, Luca Foschini, Genady Ya Grabarnik, Lorenzo Manca, Filippo Poltronieri, Domenico Scotece, Larisa Shwartz, Cesare Stefanelli, Mauro Tortonesi, et al. 2024. Kubetwin: A digital twin framework for kubernetes deployments at scale. *IEEE Transactions on Network and Service Management* 21, 4 (2024), 3889–3903.

[3] Walter Cerroni, Luca Foschini, Genady Ya Grabarnik, Filippo Poltronieri, Larisa Shwartz, Cesare Stefanelli, and Mauro Tortonesi. 2021. Bdmaas+: Business-driven and simulation-based optimization of it services in the hybrid cloud. *IEEE Transactions on Network and Service Management* 19, 1 (2021), 322–337.

[4] Cloud Native Computing Foundation. 2025. Helm: The Kubernetes Package Manager. https://helm.sh/.

[5] Cloud Native Computing Foundation. 2025. Karmada: Kubernetes Multi-Cluster Orchestrator. https://github.com/karmada-io/karmada.

[6] Anousheh Gholami, Kunal Rao, Wang-Pin Hsiung, Oliver Po, Murugan Sankaradas, and Srimat Chakradhar. 2022. Roma: Resource orchestration for microservices-based 5g applications. *arXiv preprint arXiv:2201.11067* (2022).

[7] Panagiotis Gkonis, Anastasios Giannopoulos, Panagiotis Trakadas, Xavi Masip-Bruin, and Francesco D'Andria. 2023. A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues. *Future Internet* 15, 12 (2023). https://doi.org/10.3390/fi15120383

[8] HashiCorp. 2025. Terraform: Automate Infrastructure on Any Cloud. https://developer.hashicorp.com/terraform.

[9] Red Hat. 2025. Red Hat Ansible Automation Platform. https://docs.ansible.com/platform.html.

[10] Marco Iorio, Fulvio Risso, Alex Palesandro, Leonardo Camiciotti, and Antonio Manzalini. 2023. Computing Without Borders: The Way Towards Liquid Computing. *IEEE Transactions on Cloud Computing* 11, 3 (2023), 2820–2838. https://doi.org/10.1109/TCC.2022.3229163

[11] Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, David Hästbacka, and Davide Taibi. 2022. Cloud Continuum: The Definition. *IEEE Access* 10 (2022), 131876–131886. https://doi.org/10.1109/ACCESS.2022.3229185

[12] Giovanni Nardini and Giovanni Stea. 2022. Using network simulators as digital twins of 5G/B5G mobile networks. In *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 584–589.

[13] Paulo Pereira, Carlos Melo, Jean Araujo, Jamilson Dantas, Vinícius Santos, and Paulo Maciel. 2022. Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service. *The Journal of Supercomputing* 78, 3 (2022), 4421–4448.

[14] Filippo Poltronieri, Cesare Stefanelli, Mauro Tortonesi, and Mattia Zaccarini. 2023. Reinforcement learning vs. computational intelligence: Comparing service management approaches for the cloud continuum. *Future Internet* 15, 11 (2023), 359.

[15] Hui Song, Rustem Dautov, Nicolas Ferry, Arnor Solberg, and Franck Fleurey. 2022. Model-based fleet deployment in the IoT–edge–cloud continuum. *Software and Systems Modeling* 21, 5 (2022), 1931–1956.

[16] OASIS TOSCA TC. 2025. TOSCA Version 2.0. https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.html.

[17] Orazio Tomarchio, Domenico Calcaterra, Giuseppe Di Modica, and Pietro Mazzaglia. 2021. TORCH: a TOSCA-Based Orchestrator of Multi-Cloud Containerised Applications. *Journal of Grid Computing* 19, 1 (2021), 5. https://doi.org/10.1007/s10723-021-09549-z

[18] Ying Xiong, Yulin Sun, Li Xing, and Ying Huang. 2018. Extend cloud to edge with kubeedge. In *2018 IEEE/ACM Symposium On Edge Computing (SEC)*. IEEE, 373–377.