Journal of Systems Architecture 176 (2026) 103787

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Distributed replica allocation and load balancing for Edge–Cloud FaaS☆ A cooperative multi-agent orchestration approach

Federica Filippini [a,*], Marin Lujak [b], Michele Ciavotta [a]

[a] University of Milano-Bicocca, Milan, Italy
[b] CETINIA, University Rey Juan Carlos, Madrid, Spain

ARTICLE INFO

Keywords:
Function-as-a-Service
Multi-agent orchestration
Load balancing
Distributed optimization
Privacy-preserving coordination
Edge–Cloud continuum

ABSTRACT

The Function-as-a-Service (FaaS) paradigm supports many Cloud-native applications, but rising demand for low-latency services exceeds what the Cloud alone can deliver. Edge computing addresses this limitation; however, its heterogeneity and fragmented administrative domains, together with the workload dynamicity, greatly complicate resource coordination and function replica allocation across the Edge–Cloud continuum. This paper addresses these challenges through two complementary contributions. First, we formalize the Function Replica Allocation and Load Balancing (FRALB) problem in the Edge–Cloud continuum as a Distributed orchestration problem referred to as DiFRALB. The formulation jointly optimizes function placement, horizontal offloading among Edge nodes, and vertical offloading toward Cloud resources. Computation offloading plays a central role in this setting, as it enables workloads to be distributed across heterogeneous Edge and Cloud infrastructures while meeting performance requirements. Second, recognizing that centralized orchestration approaches suffer from scalability limitations and raise privacy concerns in multi-stakeholder environments, we propose FaaS-MACrO, a distributed multi-agent orchestration architecture designed to solve the DiFRALB problem. In FaaS-MACrO, each Edge node operates as an independent agent making local decisions, while a lightweight coordinator ensures global consistency by iteratively updating offloading prices to resolve conflicts among neighboring nodes. Crucially, coordination requires only minimal information exchange, thereby preserving operational privacy. Our solution approach jointly optimizes processing and offloading decisions by capturing the trade-offs among local execution efficiency, horizontal offloading latency, and vertical offloading costs. Extensive experiments across heterogeneous node configurations, diverse network topologies, and varying function characteristics demonstrate that FaaS-MACrO achieves solutions within 0.03–12.14% of the centralized optimum on average while significantly improving scalability, reducing solution times by up to three orders of magnitude in large-scale deployments with 200 nodes.

1. Introduction

Modern digital services are expected to react quickly, scale unpredictably, and remain available under variable demand. Serverless computing addresses these requirements by abstracting the infrastructure management from developers, allowing them to focus on application logic while the platform handles scaling, resource allocation, and fault recovery [1]. Within this paradigm, Function as a Service (FaaS) provides the core execution paradigm, enabling applications to run as small, stateless functions that are triggered on demand [2,3]. By offering rapid elasticity and a pay-per-use pricing model [4], FaaS shortens development cycles and adapts seamlessly to diverse workload patterns. Therefore, it has become a key enabler of Cloud-native application development and has proven effective in centralized Cloud environments.

However, the original assumptions of serverless computing break down for latency sensitive and context-aware services, of which industrial IoT, emergency medical assistance services, smart mobility, and augmented or virtual reality [5,6] are relevant illustrative examples. These applications rely on data generated by geographically distributed devices and require mobility support and locality awareness [7]. Executing functions exclusively in centralized Cloud infrastructures often introduces communication delays that violate strict Quality-of-Service (QoS) requirements. In such settings, deciding where functions should

☆ This article is part of a Special issue entitled: 'AI4ORC' published in Journal of Systems Architecture.

* Corresponding author.
E-mail address: federica.filippini@unimib.it (F. Filippini).

https://doi.org/10.1016/j.sysarc.2026.103787
Received 24 November 2025; Received in revised form 4 March 2026; Accepted 26 March 2026
Available online 27 March 2026

run, how resources are shared, and how load is balanced across heterogeneous infrastructures becomes a key systems challenge. As a consequence, computation is increasingly migrating closer to data sources and end users, motivating the extension of the FaaS model toward the Edge, exploiting the emerging Edge-Cloud computing continuum [8,9].

Although the deployment of serverless functions closer to the data sources promises reduced latency, decreased network traffic, and improved bandwidth utilization [10,11], this paradigm shift introduces a number of unresolved orchestration challenges. The FaaS model was originally designed for Cloud environments, which are relatively homogeneous and usually over-pro-visioned. Instead, Edge nodes exhibit limited and heterogeneous computational capacities, operate under different administrative entities, and experience highly dynamic workloads. Coordinating function execution across such a fragmented ecosystem requires continuous adaptation to changing conditions, workload patterns, and node availability. These difficulties are further amplified at scale, as the short-lived and bursty nature of function invocations complicates efficient scheduling and load balancing across Edge environments [6].

Among other challenges, effective workload management is critical in this context to optimize resource utilization across heterogeneous nodes while maximizing request throughput and meeting performance requirements. Without intelligent distribution mechanisms, individual nodes experience resource contention, resulting in increased response latency, performance degradation, and potential violations of Service Level Agreements. To address these challenges, orchestration strategies must span the entire computing continuum, featuring both local Edge deployments and Cloud-based resources, thereby combining latency-aware execution at the periphery with the increased processing capacity of Cloud nodes. Within this continuum, computational tasks can be offloaded *vertically*, across different tiers (e.g., Edge-to-Cloud), and/or *horizontally*, among peer Edge nodes, to balance the traffic and mitigate local overloads. This dual offloading mechanism enables FaaS workloads to achieve greater responsiveness, scalability, and resilience in various deployment scenarios [12].

Recent research efforts have led to the development of FaaS frameworks specifically designed for Edge environments, including Serverledge [12], DFaaS [13], and GeoFaaS [14], as comprehensively surveyed in [2,15–17]. However, despite these advances, the literature reveals a clear gap: existing approaches rarely achieve, at the same time, *scalability*, *privacy-preserving coordination*, *quantifiable near-optimality*, and *decision-making autonomy* at local Edge nodes, where autonomy denotes the ability of each node to independently decide replica placement and request routing based on local information, rather than merely executing decisions prescribed by centralized control.

To rigorously define this workload management challenge, in this paper we formalize the Function Replica Allocation and Load Balancing (FRALB) problem in a FaaS-enabled Edge–Cloud continuum through a centralized Mixed-Integer Linear Programming (MILP) model that addresses workload orchestration through both horizontal and vertical offloading. Beyond determining how requests should be distributed across the continuum, our MILP model identifies the optimal number of function replicas to instantiate on each Edge node, ensuring that deployed services can accommodate the incoming load.

However, while the MILP formulation provides a rigorous problem definition and serves as an optimization benchmark, centralized workload management approaches are ill-suited for highly dynamic, geo-distributed Edge environments in practice. Platforms employing centralized approaches, such as funcX [18] and FunLess [19], which are typical of Cloud-based scheduling frameworks, suffer from scalability bottlenecks and introduce single points of failure, while also requiring broad visibility over the infrastructure state. Even systems that feature distributed execution capabilities often retain centralized scheduling components or require extensive state sharing to support vertical and/or horizontal offloading decisions, limiting their ability to adapt to the dynamic and unpredictable conditions typical of Edge environments. Furthermore, most existing approaches do not adequately address privacy concerns arising from the need to share sensitive infrastructure information across administrative boundaries. This is a critical requirement when Edge nodes are managed by independent and possibly competing stakeholders, who may be reluctant to delegate their resource management decisions to a central authority. These limitations motivate the need for scalable distributed multi-agent orchestration mechanisms, where autonomous agents collaboratively negotiate, offload, and balance workloads across the Edge–Cloud continuum while minimizing the information sharing among peers and with a centralized management system.

Addressing these requirements, we formalize the FRALB problem as a Distributed orchestration problem (DiFRALB), and propose FaaS-MACrO (Multi-Agent Coordinated Orchestration), a privacy-preserving cooperative multi-agent architecture to solve it. Our approach distributes decision-making across participating Edge nodes, treating them as autonomous agents. Each agent independently decides how to allocate requests between local processing and horizontal offloading and determines the number of function replicas to activate to handle the incoming load. More importantly, agents make these local decisions without any visibility into the state, characteristics, or decisions of other nodes in the network. Agents share their offloading decisions with a central coordinator, which verifies whether the horizontal offloading allocation is globally feasible; when this would cause nodes to exceed their capacity, it determines the vertical offloading to the Cloud required at each node to prevent request rejection, resolving conflicts arising from the limited visibility local nodes have of the state of the neighbors.

The proposed cooperative multi-agent orchestration mechanism balances autonomy and coordination, enabling each node to make individually rational decisions that align with system-wide objectives. Following the paradigm of social welfare optimization, distributed agent decisions are coordinated to maximize an aggregate utility reflecting the collective performance of the system. Rather than optimizing isolated node-level metrics, decisions are evaluated based on their global impact on computational and networking efficiency. Accordingly, we define an objective function that integrates the benefits and costs of local execution, horizontal offloading, and vertical offloading, promoting cooperative behavior under distributed control.

FaaS-MACrO is designed to operate effectively under heterogeneous and dynamic conditions, accommodating Edge nodes with diverse computational capabilities, different network topologies, and serverless functions with various execution demands.

In summary, the main contributions of this paper are:

- We formalize the Function Replica Allocation and Load Balancing (FRALB) problem in a FaaS-enabled Edge–Cloud continuum, encompassing both horizontal and vertical offloading
- We formulate a centralized MILP model to address the FRALB problem, providing an optimization benchmark for its distributed version (DiFRALB) with quality of solution guarantees.
- We develop FaaS-MACrO, a privacy-preserving multi-agent workload management architecture that distributes decision-making across Edge nodes while ensuring global consistency through minimal information exchange.
- We define a social welfare optimization objective that captures the trade-offs between local processing, horizontal and vertical offloading, balancing efficiency, latency, and communication costs.
- We demonstrate the robustness and scalability of the proposed approach across heterogeneous nodes, network conditions, and function types, scaling up to 200 nodes and 50 functions, and showing performance close to the centralized optimum with acceptable coordination overhead.

Table 1
Taxonomy of FaaS Orchestration Approaches: Centralized vs Distributed vs Decentralized.

	Centralized	Distributed	Decentralized
Vertical	[20–23]	[21,24]	[14,21,25–27]
Horizontal	[28–35]	[34,36]	[13,37–42]
Hybrid	[43–49]	✗	[12,50–52]

The remainder of this paper is organized as follows. Section 2 reviews the related work and positions our contribution within the state of the art. Section 3 formalizes the FRALB problem and defines its key concepts and parameters. Section 4 presents the centralized mathematical formulation of the FRALB problem, whereas Section 5 presents a distributed formulation for scenarios in which Edge nodes make local decisions coordinated by a lightweight central coordinator. In the same section, we describe the FaaS-MACrO architecture proposed to tackle the DiFRALB problem. Section 6 details the simulation setup and discusses the obtained results. Finally, Section 7 summarizes the main findings and outlines directions for future research.

2. Related work

The rapid expansion of serverless computing has stimulated extensive research activity, as documented in several comprehensive surveys [2,15–17]. Although a large body of work on task allocation and scheduling in the computing continuum addresses comparable challenges, in this section, we restrict our attention to studies that explicitly adopt the FaaS execution model. We review contributions on FaaS deployment, scheduling, and offloading across heterogeneous infrastructures, grouping them into three architectural categories: *centralized* orchestration, where decisions are computed by a single system-wide controller; *distributed* orchestration, in which autonomous node-level agents make local decisions under the coordination of a lightweight system coordinator, and *decentralized* orchestration where peer nodes coordinate directly in a fully peer-to-peer manner without any system-wide coordinator. Within each category, we further classify works according to the offloading dimension they enable, namely vertical (across different tiers of the computing continuum), horizontal (across peer nodes within the same tier), or hybrid (both).

Table 1 summarizes this taxonomy, organizing representative studies by orchestration paradigm and supported offloading direction.

2.1. Centralized orchestration

Centralized orchestration approaches employ a single controller with complete system visibility to compute placement and offloading decisions for FaaS workloads. We organize these works by their supported offloading dimensions.

2.1.1. Vertical offloading

Among the works that focus specifically on vertical offloading from Edge to Cloud, Das et al. [20] address the problem of scheduling serverless pipeline execution either at the Edge or in the Cloud. The approach they propose enables users to specify latency and cost requirements, and determines the execution location based on task duration prediction models, making binary placement decisions per pipeline without considering horizontal cooperation among Edge peers. In a similar setting, Costless [22] reduces cost by merging multiple functions into a single composite unit before deployment on Edge or Cloud nodes, thereby mitigating inter-function communication overhead. Ascigil et al. [21] formulate the FaaS orchestration problem in Edge–Cloud hierarchies as a time-slotted Mixed Integer Program capturing admission control, Earliest Deadline First (EDF)-based scheduling, cold-start effects, and multi-class queue dynamics. Under the fully centralized variant, a global controller has complete visibility of all cloudlets and determines both provisioning decisions and the optimal execution point for each request along the vertical Edge-to-Cloud path. Although used primarily as an upper bound due to its computational complexity, this centralized design illustrates the potential gains of globally coordinated vertical offloading under strict latency and deadline constraints. Yao et al. [23] propose a Deep Reinforcement Learning (DRL) approach with population-guided parallel policy search to optimize vertical function offloading from IoT devices to a serverless edge (EFaaS) layer. Offloading is formulated as a centralized decision problem in which a single controller, assuming global system state observability, determines per-function execution at the device or edge tier to minimize latency, energy consumption, and monetary cost. While effective in reducing delay, the approach relies on centralized control and extensive offline training and does not address multi-edge placement or load balancing across heterogeneous edge nodes. Finally, E2FIS [35] focuses on minimizing energy consumption. It considers joint optimization of node activation and concurrency control. Integrated with Serverledge, E2FIS formulates a centralized MILP that runs periodically (15-60 min epochs) to determine which Edge nodes should remain active and their concurrency levels. Experiments demonstrate energy savings of up to 92% compared to EDF scheduling.

2.1.2. Horizontal offloading

Some centralized approaches focus exclusively on horizontal offloading among Edge nodes. In particular, Palade et al. [28] apply Ant Colony Optimization to serverless function placement in federated Multi-access Edge Computing (MEC) systems. Despite using bio-inspired swarm intelligence, the model performs horizontal offloading among MEC nodes coordinated by a central controller that solves placement decisions per function request. Results on federated setups with up to 20 servers demonstrate reduced latency with sub-second decision times, though the approach does not employ function replication. Implemented in OpenWhisk, LaSS [29] integrates function placement with auto-scaling decisions, using queuing models to predict performance under different allocation scenarios and adapt resource provisioning to workload fluctuations. Wang et al. [30] propose a framework combining a Hawkes Process Temperature-Driven (HPTD) scaler with a Dual-Level Orchestrator (DLO) scheduler. The HPTD component predicts workload bursts using temporal self-excitation, enabling proactive scale-out decisions, while the DLO scheduler performs fine-grained per-invocation bin-packing. Implemented in a centralized master-worker architecture, the approach achieves up to 900% improvement in Quality-Price Ratio. Chen et al. [32] introduce *pCache*, a cross-Edge orchestration framework that combines online request scheduling with probabilistic container caching. Built on Knative and Kubernetes, pCache features a centralized controller that optimizes request routing and container eviction decisions across multiple Edge nodes, taking into account per-node memory constraints. Finally, Russo Russo et al. [33] explore adaptive load balancing via a Multi-Armed Bandit framework, where the load balancer periodically selects among competing policies according to aggregated performance metrics.

2.1.3. Horizontal and vertical offloading

A large body of work on centralized FaaS orchestration supports both offloading dimensions. Among these, HEFTLess [44] addresses serverless workflow orchestration across the Edge-Fog-Cloud continuum by formulating a bi-objective optimization problem that jointly minimizes execution cost and completion time. The system employs a hybrid approach combining Binary Linear Programming with HEFT-based heuristics to compute placement decisions for workflow batches, enabling both horizontal and vertical offloading but requiring global visibility of all resources. EnergyLess [45] extends the optimization

scope to incorporate energy consumption and carbon emissions alongside latency and monetary cost. The framework formulates a multi-objective Mixed-Integer Nonlinear Programming model solved via an anchor-based heuristic scheduler that achieves sub-second orchestration times even for systems with hundreds of nodes. The authors of [46] address the problem of minimizing carbon emissions and long-term cost under latency constraints; they propose a hierarchical model for cross-Edge orchestration that performs horizontal offloading among multiple Edge nodes using Lyapunov-based online control, with decisions recomputed once per time slot (approximately 1 h).

Deng et al. [31] propose a proactive algorithm to distribute traffic among Edge nodes jointly optimizing functions placement and data stream mapping.

A number of studies also investigate the optimal placement of serverless functions across heterogeneous infrastructures. For instance, in [47] the authors address the placement of interdependent functions within workflow Directed Acyclic Graphs (DAGs), aiming to minimize application completion time by balancing processing latency and communication overhead.

Mahdizadeh and Abrishami [43] address Edge FaaS scenarios where users submit DAGs of dependent functions. Their system operates in fixed 10-second rounds, with the EFaaS Manager collecting ready-task requests and allocating them to Edge servers through bidding-based assignment models inspired by the course allocation problem, aiming to minimize the Average Normalized Completion Time. Peri et al. [48] propose a two-tier scheduling strategy in which the first tier selects between public Cloud and private Edge based on a cost-aware heuristic, while the second tier performs fine-grained placement among Edge nodes. Raith et al. [49] present a pressure-based framework for adaptive serverless function management across Edge and Cloud nodes, designed to handle device mobility. The system implements both horizontal and vertical offloading through a centralized controller (global function placement and replicas) coordinated with local modules integrated with Kubernetes. Decisions are guided by a quantitative pressure metric combining request distribution, round-trip time, and resource usage. The pressure-based heuristic lacks formal optimality guarantees, and threshold-based triggers may react slowly to rapid workload surges.

2.2. Distributed orchestration

In distributed orchestration approaches, individual nodes make local decisions while sharing information with a central coordinator, which ensures global consistency across the system and/or takes higher-level decisions in a hierarchical setting.

Focusing on *vertical offloading*, In their coordinated variant, Ascigil et al. [21] adopt a distributed architecture where local nodes perform request-level admission using only on-node queue simulations, while a central controller periodically orchestrates provisioning across the hierarchy. The decision-making burden is split: the controller manages slow-timescale provisioning, and Edge nodes handle fast-timescale per-request admission. This hybrid scheme substantially reduces coordination overhead while achieving performance close to the centralized upper bound. Tütüncüoğlu et al. [24] formulate a Stackelberg game in which a profit-driven FaaS Edge operator acts as the leader, jointly optimizing pricing, radio/compute resource allocation, and service caching under complete and centralized knowledge of all system parameters. Wireless devices play as followers and individually decide, through a closed-form threshold rule, whether to offload or compute locally.

For *horizontal offloading*, NEPTUNE [36] partitions Edge infrastructure into autonomous *communities*, each managed by a local orchestrator solving Mixed Integer Programs for placement and routing. Higher-level controllers coordinate inter-community interactions and handle topology reconfiguration. Within each community, routing distributes requests according to computed fractions, while local control loops dynamically adjust CPU and GPU resource allocations. Aslanpour et al. [34] investigate load balancing in heterogeneous serverless edge environments through a performance-driven, empirical weight-tuning approach. Edge nodes are profiled with respect to throughput, energy efficiency, latency, and cost, and the resulting weights are used to steer a weighted round-robin scheduler. The approach is primarily centralized, but the authors also evaluate a coordinated distributed configuration in which identical load-balancing policies are enforced by multiple edge-level gateways.

2.3. Decentralized orchestration

In decentralized approaches, autonomous nodes make decisions based solely on local information and peer-to-peer interactions, without any central coordination.

2.3.1. Vertical offloading

Among the decentralized frameworks that focus exclusively on vertical offloading, Jinu et al. [26] present ComFaaS, a platform enabling vertical offloading between Edge and Cloud layers. Each Edge node independently decides at runtime whether to execute or offload functions. The system demonstrates through empirical evaluation that Edge execution can outperform Cloud offloading when data transfer costs dominate. Li et al. [25] propose AttentionFunc, a multi-agent reinforcement learning framework for decentralized vertical offloading in Edge–Cloud serverless systems. Each Edge node is modeled as an autonomous agent that, on a per-invocation basis, decides whether to execute a function locally or offload it to the Cloud. The framework adopts a centralized training with decentralized execution paradigm: during training, agents exchange attention-weighted latent representations to enable cooperative policy learning, while at runtime each agent performs fully local offloading decisions based solely on its own observations and limited peer information, without relying on a central controller. Wu et al. [27] extend DRL by explicitly addressing cold-start latency through CSODQN, a distributed framework combining a warm-instance pool with importance-sampling-enhanced double-dueling Deep Q-Network (DQN). Each device independently decides on offloading actions among local, Edge, or Cloud execution, with the DQN incorporating cold-start risk explicitly. While the training of the CSODQN algorithm is distributed (central memory is used to accelerate and stabilize agent learning), execution is decentralized. Ascigil et al. [21] also present fully decentralized policies in which each cloudlet independently manages its provisioning and admission control using only local statistics such as utilization, missed utilization, and deadline adequacy. Decisions are made autonomously and vertical offloading emerges naturally as rejected requests climb the hierarchy toward higher-capacity nodes. The proposed policies operate at low computational cost and, despite the absence of coordination, achieve satisfaction rates close to centralized strategies. GeoFaaS [14] is a geo-aware Edge-to-Cloud FaaS platform where each node autonomously decides, for each request, whether to execute it locally or offload it vertically to the Cloud based on availability and latency considerations.

2.3.2. Horizontal offloading

Several frameworks enable horizontal peer-to-peer offloading among Edge nodes. In particular, DFaaS [13] federates multiple OpenFaaS nodes at the Edge through an overlay network, enabling horizontal, peer-to-peer offloading between federated Edge nodes without relying on a centralized controller, achieving dynamic and autonomous load balancing. Function offloading is performed through a weighted Round-Robin policy whose weights are dynamically adjusted at runtime by local agents employing heuristic or DRL strategies [53,54], enabling adaptive and self-optimizing behavior under varying workload conditions. Similarly, P2PFaaS [38] supports horizontal offloading through gossip-based peer discovery. Each node independently decides whether to execute, reject, or forward incoming requests based on its current load, with RL policies continuously updated online to improve decision quality over time. In FaaS@Edge [39], a fully peer-to-peer solution

built on the InterPlanetary File System for decentralized resource discovery, offloading decisions are made on a per-request basis via randomized peer selection among nodes advertising compatible memory availability, ensuring fairness and decentralization without any global scheduler. Experimental results across 40 Edge nodes report low submission latency (≈ 110 ms) and over 98% successful execution rates.

Cicconetti et al. [37] propose an Internet Protocol-inspired algorithm where each Edge router autonomously assigns function requests to nearby executors based on dynamically updated latency weights. The system employs weighted Round-Robin for proportional fairness and supports hierarchical scaling through inter-router forwarding, though it operates purely reactively without explicit optimization. Carlini et al. [40] propose a fully decentralized horizontal offloading mechanism using the Marginal Computing Cost per User (MCU) as a fairness metric. Each node periodically selects one neighbor and evaluates potential workload migrations, accepting migrations only if the destination residual capacity remains positive. This pairwise negotiation iteratively equalizes MCU values across the network. Chen et al. [41] introduce Ekko, a fully decentralized FaaS scheduling system built on a Pastry Distributed Hash Table. Nodes autonomously assume roles as workers or shadow schedulers, with scheduler count growing sub-linearly through gossip-based self-election. Experiments on up to 100,000 emulated nodes demonstrate significant latency reductions. Finally, a multi-agent DQN scheduler for horizontal offloading within data-center-scale FaaS clusters is introduced in [42]. Each worker node acts as an autonomous agent observing local resources and communicating with peers to cooperatively decide function placement. Experiments on a 10-node simulated cluster demonstrate up to 33% higher CPU usage compared to heuristic baselines.

2.3.3. Horizontal and vertical offloading

Several decentralized approaches support both offloading dimensions. Among these, Serverledge [12] is an open-source FaaS framework that employs a decentralized architecture to execute serverless functions in Cloud-to-Edge environments. Each node autonomously decides whether to execute or offload requests based on local state and neighbor latency estimates obtained through the Vivaldi algorithm. Serverledge utilizes Docker containers for function execution and supports both horizontal and vertical offloading through reactive, rule-based policies. Russo Russo et al. [50] extend Serverledge with a decentralized QoS-aware offloading strategy where each node solves a local Linear Programming model to optimize offloading probabilities, maximizing expected utility under local resource and cost constraints. Nodes periodically exchange lightweight aggregated information about capabilities and load states. Russo Russo et al. [51] adopt a DQN-based offloading policy within Serverledge, featuring centralized training and decentralized per-invocation decision making at Edge nodes across local execution, horizontal/vertical offloading, and request dropping. The approach requires limited state information, including local resource availability and coarse neighbor-availability indicators. Finally, Daniëlse et al. [52] compare centralized, federated, and decentralized offloading architectures for OpenFaaS-based systems. In their decentralized variant, each node independently decides which request to execute locally, offload to peers, or offload to the Cloud using weighted Round-Robin with empirically updated weights based on observed response times.

2.4. Positioning of this work

Our review shows that the literature is extremely limited when it comes to *distributed* FaaS orchestration. In particular, to the best of our knowledge, there are no distributed approaches that jointly support both *vertical* and *horizontal* offloading (Table 1). In the few works labeled as distributed, Edge nodes often perform only trivial admission decisions, while substantive scheduling and provisioning logic remains centralized. Local nodes neither coordinate with each other nor engage in iterative negotiation with the orchestrator.

Centralized hybrid offloading solutions typically require full system visibility to compute placement, provisioning, and routing decisions. This makes them poorly suited to federated scenarios where nodes belong to different administrative domains and detailed state sharing is not feasible (Table 2). Conversely, fully decentralized coordination can improve flexibility, scalability, and resilience to failures or intermittent connectivity by avoiding dependence on a single global decision maker. However, these benefits often come at the cost of potentially significant deviations from globally optimal solutions. Moreover, decentralized coordination is not inherently privacy-preserving, as it may still require substantial information exchange. Nonetheless, the authors of the works analyzed in Table 2 explicitly account for this issue by designing coordination mechanisms that minimize the amount of shared information, thereby enabling more privacy-aware operation. Table 2 also highlights that most existing approaches are evaluated on relatively small problem instances, often involving only a limited number of nodes and functions.

Overall, the analysis identifies a clear gap: existing approaches rarely achieve at the same time scalability, privacy-preserving coordination, quantifiable near-optimality, and decision-making autonomy at local Edge nodes. This work fills this gap by proposing a distributed orchestration model in which Edge nodes are the main actors in an iterative decision-making process. They independently determine both the volume of requests to execute locally or offload and the number of function replicas to instantiate, while a lightweight coordinator exchanges only minimal signals to ensure global feasibility. While learning-based approaches (particularly DRL) are increasingly explored for distributed orchestration, they typically require extensive training and exploration, offer limited deterministic guarantees on constraint satisfaction, and may suffer from convergence instability and limited explainability in highly coordinated multi-agent environments. In contrast, our model-based framework provides coordination stability, explicit feasibility guarantees, and transparent decision logic, enabling near-optimal and privacy-aware distributed orchestration without relying on unsafe exploration or long offline training phases.

The framework formulates orchestration as a social welfare maximization problem that we solve through an iterative coordination mechanism. The problem definition is inspired by Lagrangian relaxation, which enables near-optimal operation and allows quantifying the distance from an optimality bound. Finally, our experimental analysis demonstrates scalability up to 200 nodes and 50 functions, well beyond the scale considered in most of the analyzed works, showing that strong performance and privacy-aware operation can be achieved even in the presence of a central coordinator.

The architectural principle is inspired by Giordani et al. [55], who proposed a two-level distributed framework for production planning in manufacturing systems. In their formulation, mobile robot agents autonomously solve local scheduling problems, while a central coordinator ensures that the set of local schedules collectively satisfies global resource and capacity constraints by updating Lagrangian multipliers. Although developed for a different domain, [55] proposes a decomposition logic that inspires the design of FaaS-MACrO. In our case, this principle is adapted to the dynamic, latency-sensitive, and privacy-aware nature of FaaS workloads operating across the Edge-to-Cloud continuum.

3. Problem description

We study and propose the Function Replica Allocation and Load Balancing (FRALB) problem in a reference scenario consisting of a network of geo-distributed, heterogeneous computing nodes in the Edge–Cloud continuum. The infrastructure includes Edge devices and

Table 2
Comparison of works jointly supporting horizontal and vertical offloading, including privacy aspects and experimental scale.

	Centralized					
	[43]	[44,45]	[46]	[47]	[48]	[49]
Privacy	✗	✗	✗	✗	✗	✗
# nodes	10	325	15	6	3	9
# func.	115	17	5	200	10	1

	Decentralized			
	[12]	[50]	[51]	[52]
Privacy	✓	✓	✓	?
# nodes	4	11	3	6
# func.	4	150	5	2

micro-datacenters with diverse computational capabilities and communication characteristics. The system is modeled as an undirected graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ denotes the set of computing nodes and $\mathcal{E}$ denotes the set of communication links between them.

Each node $i \in \mathcal{N}$ hosts a FaaS platform that orchestrates the execution of serverless functions. Nodes are constrained by finite processing capacity and a limited memory capacity $\overline{RAM}_i$. We adopt a discrete-time model where client function execution requests arrive in time slots. Each client accesses the system through their closest Edge node, which acts as the initial ingress point for function execution requests, as illustrated in Fig. 1. Requests may be executed locally at the ingress Edge node, forwarded to neighboring Edge or micro-datacenter nodes through horizontal offloading, or delegated to the Cloud layer via vertical offloading, depending on the availability of computational resources and prevailing network conditions.

Communication links $(i, j) \in \mathcal{E}$ are characterized by fixed bandwidth and transmission latency parameters, capturing the heterogeneity in inter-node connectivity. This abstraction enables the joint modeling of computational and network-level constraints that influence function replica placement, load distribution, and coordination across the Edge–Cloud continuum. All nodes in $\mathcal{N}$ are connected to a centralized Cloud layer C. Following other proposals (e.g., [46]), the Cloud is assumed to possess sufficient computational capacity to absorb excess workload whenever Edge nodes are saturated. This assumption ensures that function requests are never rejected and avoids the need for over-provisioning at the Edge, thereby reducing energy consumption and operational costs.

Following the serverless paradigm, a predefined set of functions $\mathcal{F}_i$ is deployed on each Edge node $i \in \mathcal{N}$ as well as on the Cloud. We assume that the set of deployable functions is fixed within the time horizon considered and that all nodes support compatible execution environments. For the purposes of this work, we use the following function definition.

Definition 1. A *function* $f \in \mathcal{F} = \bigcup_{i \in \mathcal{N}} \mathcal{F}_i$ is a discrete, stateless computational unit that encapsulates a specific microservice or task. Functions are invoked independently in response to client requests and can be instantiated as multiple *function replicas* (typically deployed as containerized instances [44,46]) on each node $i \in \mathcal{N}$ and on the Cloud.

We model functions as deterministic and stateless entities with consistent runtime behavior across invocations. Specifically, we assume that each function f exhibits predictable execution characteristics on a given node i, which enables precise estimation of CPU utilization. Under this assumption, the execution of a single request of function f on node i is characterized by a constant average service time (or *demand*) D_i^f. Here, D_i^f represents the execution time required to process a single request in isolation, i.e., when a single replica of function f is deployed on node i without resource contention from other functions. Additionally, each replica of function f is assumed to require a deterministic and fixed amount of memory RAM^f, independent of the execution node. This abstraction neglects transient memory fluctuations and runtime variability, allowing memory constraints to be modeled through static capacity limits at each node. Although these assumptions are seemingly restrictive, they are commonly adopted in the serverless computing literature and generally hold in practical applications. Indeed, function replicas are usually deployed as containerized services (e.g., Docker containers), whose memory requirement can be statically estimated at design time depending on the container image size and the data structures required for execution. On the other hand, the service demand D_i^f is estimated without resource contention; therefore, its average value primarily depends on the hardware characteristics of the node i where requests are executed [56].

To capture system dynamics, we model the system using discrete time periods t, over which system parameters, including function request arrival rates and network conditions, are considered quasi-stationary, while computational capacities at each node remain constant throughout the entire time horizon. Replica allocation and load-balancing decisions are maintained throughout each period but are recomputed across successive intervals to adapt to changing conditions. The duration of period t must satisfy two competing requirements: it should be short enough to accurately capture transient workload fluctuations and ensure that average request rates approximate actual arrival processes, yet long enough to allow execution of the coordination algorithms that determine function placement and routing decisions [35]. In practice, t may range from a few seconds to several minutes depending on system dynamics and the computational overhead of the decision process [35,43,46].

Definition 2. The *input workload* $\lambda_i^f(t)$ represents the average arrival rate of execution requests (measured in requests per second) for function $f \in \mathcal{F}_i$ at node $i \in \mathcal{N}$ during time period t, originating from external clients as depicted in Fig. 2.

We emphasize that both the service demand parameters D_i^f and the workload descriptors $\lambda_i^f(t)$ are treated in our work as configurable inputs rather than application-specific constants. By appropriately setting these parameters, the model can capture heterogeneous execution characteristics, request volumes, and latency sensitivities associated with different application domains. In this sense, the proposed approach provides an infrastructure-level orchestration framework that is application-agnostic, yet can be specialized to specific scenarios through suitable parameterization of workload intensity, computational demand, and resource requirements.

In this work, we focus on optimizing system behavior within a single time period t, assuming the average workload λ_i^f is known or can be estimated [35,57]. To simplify notation, we henceforth omit the explicit time dependency in variables and parameters, with the understanding that all quantities refer to the time period under consideration. While approximating workload as a known average introduces some error, particularly in scenarios with high workload volatility (as discussed in Section 6.1), it does not significantly impact our approach. We assume that any excess workload beyond the expected average is absorbed through vertical offloading to the central Cloud, leveraging its virtually unlimited computational capacity to prevent overload at the Edge [46].

Given the resource characteristics and computational constraints described above, two sets of decisions must be made for each node i during time period t:

- **Replica allocation:** Determine the optimal number of function replicas r_i^f to deploy for each function $f \in \mathcal{F}_i$, thereby enabling concurrent request processing.
- **Load distribution:** Decide how to partition the incoming request rate λ_i^f among three processing strategies: local execution (x_i^f), forwarding to neighboring nodes (y_{ij}^f), or vertical offloading to the Cloud (z_i^f).

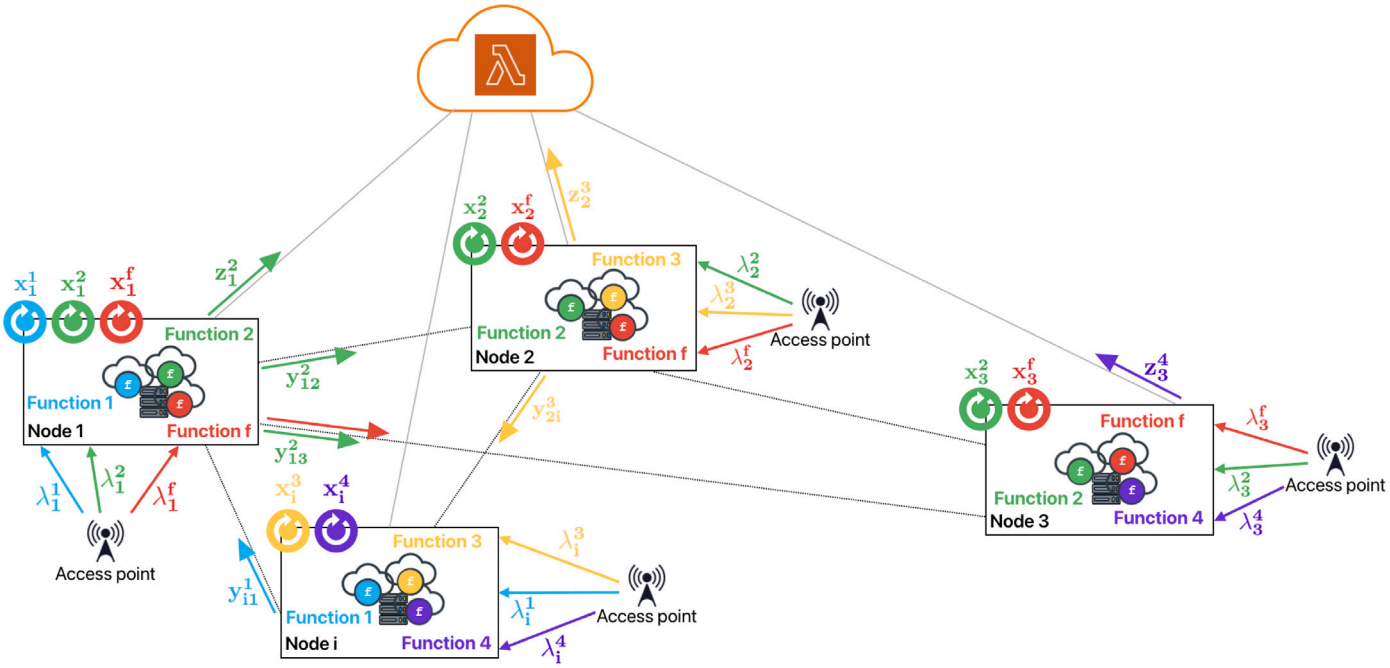


Fig. 1. Architecture of a geo-distributed Function-as-a-Service (FaaS) system modeled as an undirected graph $G = (\mathcal{N}, \mathcal{E})$. Edge nodes host containerized function replicas with finite computational resources, processing incoming workloads $\lambda_i^f(t)$ from geographically distributed clients via wireless access points. Inter-node workflows y_{ij}^f represent dynamic request routing across the network topology.

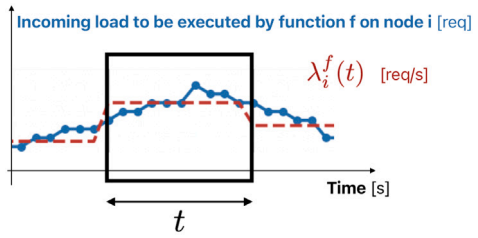


Fig. 2. Illustration of workload arrival rate $\lambda_i^f(t)$ for function f at node i over time, showing time-varying request rates from external clients. The blue line records actual measurements, while the red dashed line indicates the expected average. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

These decisions are subject to two fundamental constraints that ensure system feasibility and stability: memory capacity and overload prevention constraints.

Memory capacity constraints: The allocation of function replicas at each node is limited by available memory:

$$\sum_{f \in \mathcal{F}_i} r_i^f \cdot \mathrm{RAM}^f \leq \overline{\mathrm{RAM}}_i \qquad \forall\, i \in \mathcal{N}. \tag{1}$$

Overload prevention constraints: To formalize these requirements, we model the execution dynamics at each node as represented in Fig. 3. Requests assigned to node i comprise locally processed requests x_i^f and horizontally offloaded requests $\sum_j y_{ji}^f$ from neighboring nodes. These requests enter a dedicated queue for function f and are distributed among r_i^f function replicas using Round-Robin scheduling. Each replica provides a processing rate of $1/D_i^f$ requests per second, yielding a total processing capacity of r_i^f / D_i^f for function f on node i[1] [29].

Definition 3. The *resource utilization* U_i^f is defined as the ratio of current load to available capacity:

$$U_i^f = \frac{D_i^f}{r_i^f} \left(x_i^f + \sum_{j:(j,i)\in\mathcal{E} \wedge f \in \mathcal{F}_i} y_{ji}^f \right), \tag{2}$$

where U_i^f represents the average fraction of time each replica remains occupied.

Definition 4. A node i operating function f is in a *stable regime* if its utilization satisfies $U_i^f < 1$. When $U_i^f \geq 1$, the node enters an *overloaded state*, characterized by request arrival rates that exceed processing capacity, resulting in unbounded queue growth and divergent response times.

In practical deployments, overloaded states lead to request loss and QoS degradation. To prevent overload, resource providers typically enforce a maximum utilization threshold $\overline{U}_{\max}^f$,[2] maintaining resources sufficiently far from saturation:

$$U_i^f \leq \overline{U}_{\max}^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i. \tag{3}$$

Although our model does not explicitly optimize end-to-end request response time, the resource utilization Constraints (3), forcing the system to operate in a stable regime, implicitly promotes acceptable latency behavior.

Having established the system constraints, we now define the objective function that guides optimization decisions. Each distribution option (i.e., local execution, horizontal and vertical offloading) carries distinct performance characteristics and resource implications that must be balanced to achieve an aggregate, system-wide profit.

[1] The first time a request is assigned for execution to a new function replica (i.e., a new container instance), it must wait for the container to be fully initialized, thus experiencing a potential delay (referred to as *cold start*). However, since this additional waiting time is typically of the order of a few milliseconds to some seconds [27,50], we can assume, as in other proposals [46], that the control time interval t is long enough to offset its impact and to ensure that all incoming requests safely terminate the execution.

[2] Typical values in practical applications range in $[0.5, 0.8]$ [58], with lower thresholds providing larger safety margins against transient workload spikes.

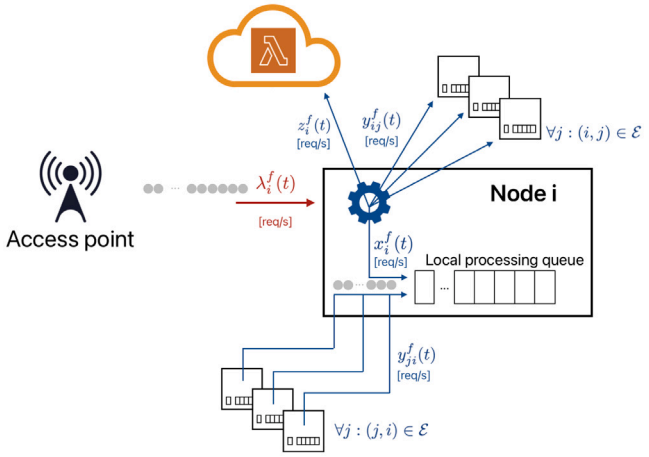


Fig. 3. Node-level workload management decisions for function $f \in \mathcal{F}_i$ during control period t. Node i must decide how to partition incoming requests among local processing, horizontal, and vertical offloading.

Definition 5. The *local profit* for workload management for function f at node i quantifies the economic or performance benefit of request processing decisions, expressed as:

$$\alpha_i^f x_i^f + \sum_{j:(i,j)\in\mathcal{E}\wedge f\in\mathcal{F}_i} \beta_{ij}^f y_{ij}^f - \gamma_i^f z_i^f, \tag{4}$$

where α_i^f represents the unit profit from locally executing a rate of 1 req/s for function f on node i, β_{ij}^f captures the unit profit from offloading requests horizontally to a neighboring node j, and γ_i^f represents the unit cost of vertical offloading to account for network transfer time and Cloud service costs.

In the rest of this work, we tackle the FRALB problem outlined in this section by initially adopting a centralized perspective (Section 4), considered then as a starting point and performance upper bound for a cooperative multi-agent distributed approach (Section 5) where a privacy-preserving coordinator mediates local decisions taken by individual Edge nodes.

4. Centralized problem formulation

We now present a centralized Mixed-Integer Linear Programming (MILP) formulation for the FRALB problem that jointly optimizes replica allocation and load distribution across all Edge nodes to maximize system-wide profit.

Overall, the centralized workload management problem reads:

$$\max \sum_{i\in\mathcal{N}} \sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f} \left(\alpha_i^f x_i^f + \sum_{j:(i,j)\in\mathcal{E}\wedge f\in\mathcal{F}_j} \beta_{ij}^f y_{ij}^f - \gamma_i^f z_i^f \right) \tag{P1a}$$

subject to:

subject to :

$$\sum_{f\in\mathcal{F}_i} r_i^f \cdot RAM^f \le \overline{RAM}_i \qquad \forall\, i \in \mathcal{N} \tag{P1b}$$

$$x_i^f + \sum_{j:(i,j)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ij}^f + z_i^f = \lambda_i^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1c}$$

$$\sum_{j:(i,j)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ij}^f \le \lambda_i^f \mu_i^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1d}$$

$$\sum_{j:(j,i)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ji}^f \le \Lambda^f \xi_i^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1e}$$

$$\xi_i^f + \mu_i^f \le 1 \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1f}$$

$$D_i^f \left(x_i^f + \sum_{j:(j,i)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ji}^f \right) \le r_i^f\, \overline{U}_{\max}^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1g}$$

$$D_i^f \left(x_i^f + \sum_{j:(j,i)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ji}^f \right) \ge \left(r_i^f - 1 \right) \overline{U}_{\max}^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1h}$$

$$x_i^f, z_i^f \ge 0 \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1i}$$

$$r_i^f \in \mathbb{N}_0 \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i \tag{P1j}$$

$$y_{ij}^f \ge 0 \qquad \forall (i,j) \in \mathcal{E},\ f \in \mathcal{F}_i \cap \mathcal{F}_j \tag{P1k}$$

$$\xi_i^f, \mu_i^f \in \{0,1\} \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}_i. \tag{P1l}$$

In each time period t, solving this problem means determining the most appropriate number of replicas r_i^f to deploy for each function $f \in \mathcal{F}_i$ on all nodes $i \in \mathcal{N}$ (under the memory Constraints (P1b)), and how to manage the incoming load. The formulation maximizes the aggregate operational profit (P1a), defined as the sum over all nodes and functions of the terms introduced in Eq. (4), normalized by considering the total rate of incoming requests λ_i^f. While our model is agnostic to specific parameter values, their relative magnitudes encode the execution strategy that the optimization is designed to favor. The most common approach in FaaS-enabled Edge systems prioritizes local execution over horizontal offloading, and horizontal offloading over vertical offloading to the Cloud, since request forwarding incurs network communication overhead and execution delays. This preference can be encoded by setting $\beta_{ij}^f < \alpha_i^f$ for all nodes, making local execution most profitable when feasible. However, different application scenarios may warrant alternative strategies: for functions with loose response time constraints, vertical offloading may be preferable to local execution on resource-constrained nodes, as it frees resources to serve latency-sensitive requests. Such scenarios can be represented by setting $\gamma_i^f < 0$, converting the vertical offloading penalty into a positive profit that favors Cloud execution. Moreover, the local execution profit α_i^f typically reflects function priority levels. Functions are often grouped into priority classes (e.g., *basic*, *premium*, and *gold*), where $\alpha_i^f = \alpha_i^c$ for all functions f in class c.

Each node i receives the main flow of requests from clients connected to the node access point, which come at the rate of λ_i^f requests per second directed to function $f \in \mathcal{F}_i$. As explained in Section 3, these can be processed locally (x_i^f), forwarded to neighboring nodes ($\sum_j y_{ij}^f$), or offloaded to the Cloud (z_i^f) according to the node computational capacity and trying to maximize the resulting profit. Constraints (P1c) enforce flow conservation for the λ_i^f incoming requests, ensuring that every request is either processed locally or horizontally/vertically offloaded, thus preventing traffic loss.

Moreover, node i receives, for each function $f \in \mathcal{F}_i$, an auxiliary flow coming from neighbors, represented by $\sum_j y_{ji}^f$. In our model, these requests cannot be further offloaded neither horizontally, redirecting them to other neighboring nodes, nor vertically to the Cloud. Indeed, Constraints (P1c) implies that workload management decisions at each node account only for the main load arriving directly at that node from the access point, while the auxiliary flow $\sum_j y_{ji}^f$ coming from neighbors is always explicitly included in the local processing queue through Constraints (P1g)–(P1h), where it appears at the left-hand side together with the requests x_i^f. Therefore, we can say that Constraints (P1c), (P1g) and (P1h) together establish a *one-hop constraint* that prevents routing cycles and bounds end-to-end latency by limiting the forwarding chain length.

At the same time, the formulation adopts a stricter assumption to prevent inefficient routing chains, guaranteeing that a node i cannot simultaneously offload requests for a function f to its neighbors while also accepting requests for the same function from other neighbors. Although such behavior may increase the objective value in certain cases, it typically leads to unnecessary forwarding and higher average latency. Constraints (P1d)–(P1f) formalize this restriction: in particular, the two binary variables ξ_i^f and μ_i^f, for each node $i \in \mathcal{N}$ and function

Table 3
Parameters and variables for the centralized formulation of the FRALB problem.

Set	Symbol	Description
Nodes	$\mathcal{N}$	Set of computing nodes $i \in \mathcal{N}$.
Arcs	$\mathcal{E}$	Set of node connections (arcs) $(i,j) \in \mathcal{E}$.
Functions	$\mathcal{F}_i$	Set of functions $f \in \mathcal{F}_i$ executed on node i.
Parameter	**Symbol**	**Description**
Memory capacity	$\overline{RAM}_i$	Memory capacity of node $i \in \mathcal{N}$ in megabytes [MB].
Input workload	λ_i^f	Rate of incoming requests for function f at node i [req/s].
RAM demand	RAM^f	RAM demand of function $f \in \mathcal{F}$ in megabytes [MB].
Service demand	D_i^f	Average execution time of function $f \in \mathcal{F}$ at node i without resource contention [s].
Maximum utilization threshold	$\overline{U}_{\max}^f$	Maximum acceptable utilization level for function f to avoid saturation, $\overline{U}_{\max}^f \in [0,1)$.
Local execution profit	α_i^f	Profit for processing 1 req/s of function f locally on node i.
Offloading profit	β_{ij}^f	Profit for offloading 1 req/s of function f from node i to node j.
Cloud offloading penalty	γ_i^f	Cost for offloading to the Cloud 1 req/s of function f on node i.
Workload upper bound	Λ^f	"Large-enough" constant used to impose an upper bound on the number of requests that a node can receive from neighbors
Variable	**Symbol**	**Description**
Locally processed requests	x_i^f	Rate of function f requests processed at node i [req/s], $x_i^f \geq 0$.
Forwarded requests	y_{ij}^f	Rate of function f requests sent from node i to node j [req/s], $y_{ij}^f \geq 0$.
Requests offloaded to the Cloud	z_i^f	Rate of function f requests offloaded to the Cloud by node i [req/s], $z_i^f \geq 0$.
Function replicas	r_i^f	Number of function f replicas deployed on node i, $r_i^f \in \mathbb{N}_0$.
Routing variable (1)	ξ_i^f	1 if node i receives requests of function f from any neighbor, 0 otherwise.
Routing variable (2)	μ_i^f	1 if node i offloads requests of function f to any neighbor, 0 otherwise.

$f \in \mathcal{F}_i$ are 1 if node i receives from (or offloads to, respectively) any neighbor requests of function f. By Constraints (P1f), at most one of the two can be true (have value of 1), meaning that a node cannot forward requests for a specific function if it receives them from others (and vice versa). Note that Λ^f in Constraints (P1e) is a large-enough constant defined, e.g., as $\Lambda^f = \sum_j \lambda_j^f$ for each function f.

Finally, Constraints (P1g)–(P1h) ensure that the effective load to be served at node i for function f is matched by an appropriate number of replicas r_i^f. Specifically, Constraint (P1g) enforces stability by requiring $U_i^f \leq \overline{U}_{\max}^f$ as discussed in Section 3, whereas Constraint (P1h) establishes an upper bound on r_i^f thus imposing that an additional replica is started for function f only if the incoming load is large-enough to guarantee that doing so does not lead to a drop in utilization, effectively maintaining replicas close to their maximum efficiency.[3]

Table 3 provides a comprehensive overview of all parameters and variables used in the formulation.

5. Cooperative multi-agent orchestration

As mentioned in Section 3, the FRALB problem can be tackled as a distributed problem (hereafter referred to as

DiFRALB) by considering a multi-agent architecture where nodes make local management decisions under the coordination of a central controller. In this scenario, the formulation proposed in Section 4 for the centralized case must be adapted to account for the lack of global information about node capacities.

In this section, we propose the DiFRALB problem and the corresponding distributed FaaS-MACrO architecture, which operates through an iterative coordination process. At iteration h, the central coordinator broadcasts a price vector $\Pi^h = [\pi^f(h)]_{f \in F}$ indicating the price for offloading 1 req/s of each function f. Given Π^h, each node i solves the local subproblem (P2) to determine (i) the number of replicas to deploy for each function (r_i^f), (ii) the locally served request rate (x_i^f), and (iii) the rate ω_i^f it asks the network to process (without specifying destinations). The coordinator then collects the locally-optimal solution identified by each node, denoted as $\langle \hat{x}_i^f, \hat{\omega}_i^f, \hat{r}_i^f \rangle^h$, and solves the coordination problem (P3) to compute fine-grained offloading decisions y_{ij}^f, to enforce global consistency and feasibility across nodes. Prices are subsequently updated to form Π^{h+1}, and the process repeats until a stopping criterion is met.

With respect to fully decentralized approaches [50,54] where nodes make processing decisions independently on the social welfare, the presence of a coordinator in our multi-agent approach creates a *virtual market* where the additional offloading prices Π^h guide the system towards a solution that is hopefully preferable from the perspective of the global network. While optimality guarantees cannot be established, the comparison with the centralized approach discussed in our experimental campaign (see Section 6) highlights reasonably low deviations in the solution quality.

As shown in Fig. 4, each iteration h[4] of the FaaS-MACrO algorithm covers 6 subsequent steps. Table 4 provides a complete summary of the additional parameters and variables introduced in this section. Moreover, since they act as decision makers in this distributed scenario, the terms *nodes* and *agents* are used interchangeably in the rest of the section.

Step 1: The central coordinator communicates the vector of offloading prices $\Pi^h = [\pi^f(h)]_{f \in F}$ to all nodes.

Step 2: Each node i solves the local subproblem (P2) considering the requests λ_i^f coming from the access point, determining how many function replicas to deploy (r_i^f), and the rate of requests it can serve locally (x_i^f) or it would send to neighbors (ω_i^f), so that $x_i^f + \omega_i^f = \lambda_i^f$. The locally-optimal solution $\langle \hat{x}_i^f, \hat{\omega}_i^f, \hat{r}_i^f \rangle$, including information on the *tentative offloading*, is shared with the central coordinator.

Step 3: By solving problem (P3), the central coordinator checks whether the tentative offloading decisions $\hat{\omega}_i^f$ comply with the residual capacity of all nodes, which depends on the rate of requests $\hat{x}_i^f$ they decided to process locally and on the number of replicas $\hat{r}_i^f$ already

[3] This design choice is particularly suited to resource-constrained Edge nodes, where starting an underutilized replica would waste limited computational capacity that could be better allocated to functions with higher load or execution profit. In such settings, offloading a small number of requests is more efficient than maintaining a poorly utilized instance. Conversely, in powerful or Cloud environments with abundant resources, this constraint may be unnecessarily restrictive, as maintaining maximum utilization is less critical; however, it can be easily relaxed by defining an acceptable utilization range $[\overline{U}_{\min}^f, \overline{U}_{\max}^f]$. Moreover, when the Edge nodes are generally close to saturation, the potential inefficiency caused by this constraint does not arise, since all replicas naturally operate near their utilization limit.

[4] In the following description, the iteration index h is omitted from variables where doing so does not compromise clarity.

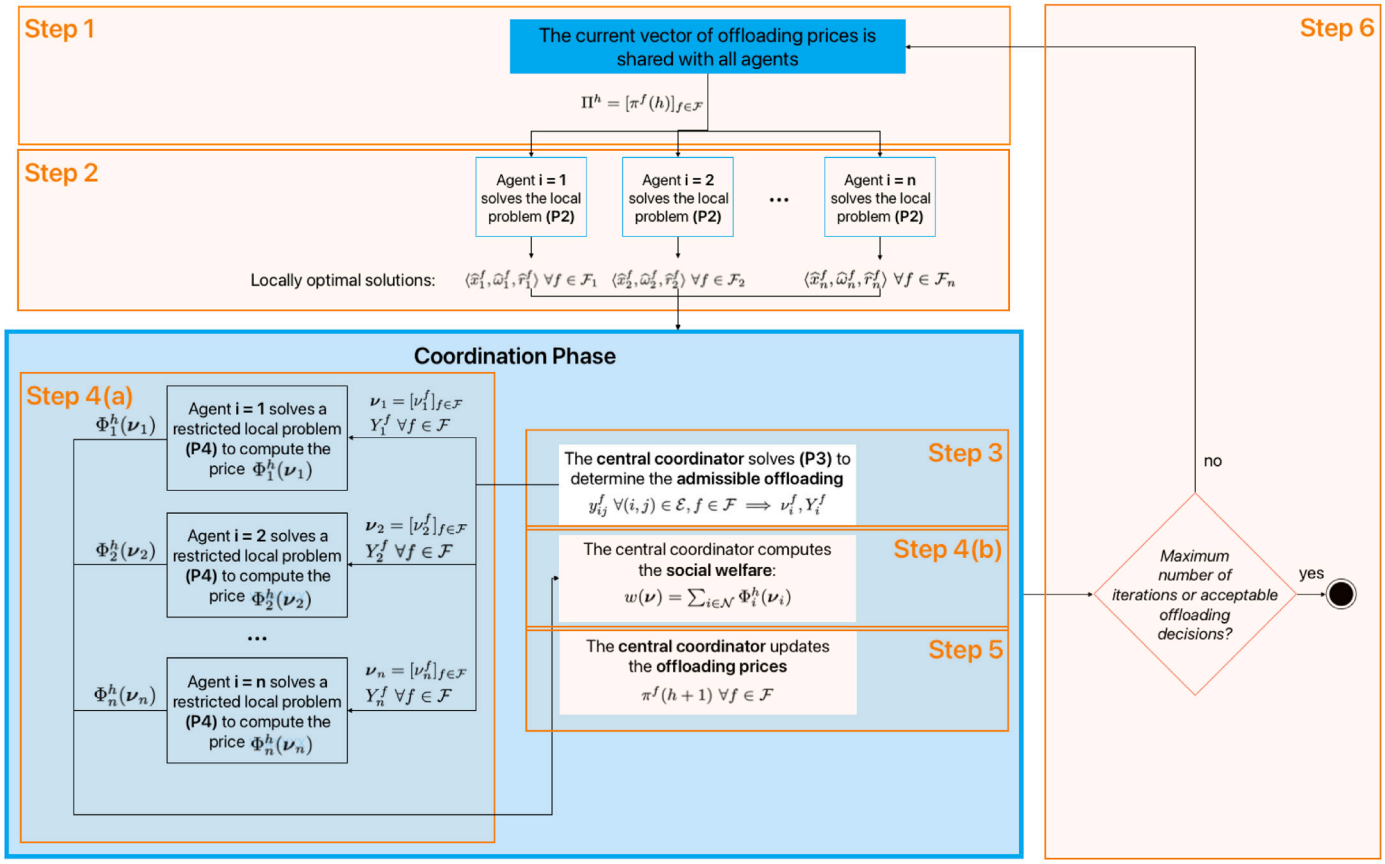


Fig. 4. Multi-agent iterative decision process implemented in FaaS-MACrO.

deployed for each function. Accordingly, it determines whether and how the requests tentatively offloaded by nodes can be distributed across the network (y_{ij}^f) to guarantee that the global solution is feasible (i.e., no node exceeds its processing capacity).

Step 4: The coordinator offloading decisions y_{ij}^f are used to:

(a) Compute the *assigned offloading* $v_i^f \le \widehat{\omega}_i^f$, i.e., the rate of requests for function f that node i can actually forward horizontally without violating global feasibility, as well as the incoming request rate Y_i^f that node i receives from neighbors. Based on these values, each node solves a restricted optimization problem (P4) to determine the overall cost related to the incoming requests management.

(b) Evaluate the *social welfare* of the network. The coordinator collects the costs computed by each node through problem (P4) and aggregates them to assess the overall system performance.

Step 5: Based on the social welfare, the coordinator updates the offloading prices to encourage (or discourage) load sharing in the next iterations.

Step 6: The iterative process terminates when either a maximum number of iterations is reached or the global solution is deemed acceptable by all nodes, according to criteria that will be detailed in Section 5.3.

Note that, to simplify the discussion, the steps above and problems (P2)–(P4) are formulated assuming that the coordinator has access to complete information on the nodes, the incoming requests and the local processing decisions. However, the coordination problem (P3) can be easily modified to account for privacy preservation, as detailed in Section 5.2.5.

5.1. Local subproblem formulation

The local subproblem (P2) solved by each node i in step 2 of iteration h reads:

$$\max \quad \sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f} \left[\alpha_i^f x_i^f + \delta_i^f \omega_i^f - \pi^f \omega_i^f \right] \tag{P2a}$$

subject to :

$$\sum_{f\in\mathcal{F}_i} r_i^f \cdot RAM^f \le \overline{RAM}_i \tag{P2b}$$

$$x_i^f + \omega_i^f = \lambda_i^f \qquad \forall f \in \mathcal{F}_i \tag{P2c}$$

$$D_i^f x_i^f \le r_i^f \overline{U}_{\max}^f \qquad \forall f \in \mathcal{F}_i \tag{P2d}$$

$$D_i^f x_i^f \ge \left(r_i^f - 1\right) \overline{U}_{\max}^f \qquad \forall f \in \mathcal{F}_i \tag{P2e}$$

$$x_i^f, \omega_i^f \ge 0 \qquad \forall f \in \mathcal{F}_i \tag{P2f}$$

$$r_i^f \in \mathbb{N}_0 \qquad \forall f \in \mathcal{F}_i. \tag{P2g}$$

The objective function (P2a) includes two different terms. The first term, $\sum_f (\alpha_i^f x_i^f + \delta_i^f \omega_i^f)$, represents the total profit related to the local execution and horizontal offloading of the incoming requests. Its definition derives from the one in Eq. (4); however, in this context, node i does not actively decide to offload requests vertically to the Cloud: since it is unaware of the processing capacity of the rest of the network, it can always assume that all requests it cannot process locally will be accepted by some neighbor. The parameter δ_i^f plays a similar role to the one of β_{ij}^f in Eq. (4), i.e., it corresponds to the profit associated with offloading 1 req/s of function f; notably, since node i does not have information on which neighbor receives the forwarded requests, the unit profit here only depends on f and i.

Table 4
Additional parameters and variables for the DiFRALB problem addressed by FaaS-MACrO (the other parameters and variables were reported in Table 3).

Problem	Parameter	Symbol	Description
(P2), (P4)	Offloading profit	δ_i^f	Profit associated with offloading 1 req/s of function f from node i to any other node.
(P2)	Offloading cost	π^f	Cost related to offloading 1 req/s for function f.
(P4)	(Actual) forwarded requests	v_i^f	Rate of requests of function f node i offloads according to the coordinator decision.
(P4)	Received requests	Y_i^f	Rate of requests of function f node i receives from the rest of the network according to the coordinator decision.
Problem	**Variable**	**Symbol**	**Description**
(P2)	(Required) forwarded requests	ω_i^f	Rate of requests of function f node i requires to offload, $\omega_i^f \geq 0$.
(P3)	Additional number of function replicas	a_i^f	Additional number of replicas of function f deployed on node i according to the coordinator decision, $a_i^f \in \mathbb{N}_0$.

The second term in (P2a) corresponds to the additional offloading price that is paid to the central coordinator according to the values of $\pi^f(h)$ in the current iteration.

Similarly to the centralized problem (P1), node i optimizes the values of the decision variables under memory (P2b), load conservation (P2c) and utilization constraints (P2d)–(P2e). However, it is worth noting that the latter do not include information on requests possibly offloaded to i by its neighbors, since such information is unavailable to local nodes before coordination occurs. Moreover, as mentioned, offloading decisions ω_i^f are taken without any information on the rest of the network, namely without knowing how many neighbors are available and what their computational capacity is. Instead, decisions are only guided by the offloading prices π^f shared by the central coordinator at the beginning of the current iteration h, so that the optimal solution of problem (P2), denoted as $\langle \hat{x}_i^f, \hat{\omega}_i^f, \hat{r}_i^f \rangle$, is likely unfeasible from a global perspective.

To simplify the subsequent discussion, we will express the objective function (P2a) as a minimization function, i.e., we write:

$$\min \quad -\sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f}\left(\alpha_i^f x_i^f + \delta_i^f \omega_i^f\right) + \sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f}\pi^f \omega_i^f. \tag{5}$$

This allows to express the value of the objective function (5) at the locally-optimal solution $\langle \hat{x}_i^f, \hat{\omega}_i^f, \hat{r}_i^f \rangle$ of problem (P2) as:

$$\Phi_i^h(\hat{\boldsymbol{\omega}}_i) + p(\Pi^h, \hat{\boldsymbol{\omega}}_i), \tag{6}$$

where $\hat{\boldsymbol{\omega}}_i$ is the vector of offloading variables $\hat{\boldsymbol{\omega}}_i = [\hat{\omega}_i^f]_{f\in\mathcal{F}_i}$ at iteration h. Note that $\Phi_i^h(\hat{\boldsymbol{\omega}}_i)$ is the intrinsic costs of local execution and offloading, i.e.:

$$\Phi_i^h(\hat{\boldsymbol{\omega}}_i) = -\sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f}\left(\alpha_i^f x_i^f + \delta_i^f \omega_i^f\right). \tag{7}$$

On the other hand, $p(\Pi^h, \hat{\boldsymbol{\omega}}_i)$ is the total offloading price paid by node i, which depends on the vector of offloading prices Π^h shared by the coordinator at the beginning of iteration h, and on the vector of offloading variables $\hat{\boldsymbol{\omega}}_i$:

$$p(\Pi^h, \hat{\boldsymbol{\omega}}_i) = \sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f}\pi^f(h)\hat{\omega}_i^f. \tag{8}$$

We note that, if we consider a reduced version of problem (P2) where a node considers only local processing and scaling decisions (i.e., we keep only variables x_i^f and r_i^f), this can be easily mapped to the structure of a Knapsack Problem, considering functions f as the classes of "items" to select, and x_i^f as the quantity of selected items, which are limited by capacity constraints that depend on the number of deployed replicas r_i^f. Since the Knapsack Problem is a known NP-hard problem, we can conclude that also our problem (P2) is NP-hard.

5.2. Coordination phase

From the perspective of the global system, the coordination phase aims to ensure that the total number of requests offloaded by each node i towards the rest of the network is properly divided among neighbors j (that is, to determine fine-grained offloading variables y_{ij}^f) without exceeding their residual computational capacity, i.e., the rate of incoming requests they can process without exceeding $\overline{U}_{\max}^f$ once $\hat{x}_i^f$ requests per second are enqueued. At each iteration h, this is achieved by applying three subsequent steps, denoted as steps 3–5 in Fig. 4 and detailed in Sections 5.2.1, 5.2.2 and 5.2.3, respectively.

5.2.1. Computing the assigned offloading and deviation

Once the coordinator receives from each node $i \in \mathcal{N}$ the information on the number of replicas $\hat{r}_i^f$ it needs to deploy to serve locally $\hat{x}_i^f$ requests per second, it can distribute the residual memory capacity among the different functions $f \in \mathcal{F}_i$ by determining a number of *additional* replicas a_i^f so that:

$$\sum_{f\in\mathcal{F}_i} a_i^f \cdot RAM^f = \overline{RAM}_i - \sum_{f\in\mathcal{F}_i} \hat{r}_i^f \cdot RAM^f \qquad \forall i \in \mathcal{N}. \tag{9}$$

Accordingly, if $\sum_j y_{ji}^f$ is the total number of requests offloaded to node i by all its neighbors $j \in \mathcal{N}$ and considering $\mathcal{F} = \bigcup_j \mathcal{F}_j$, we can write:

$$D_i^f\left(\hat{x}_i^f + \sum_{j\in\mathcal{N}} y_{ji}^f\right) \leq \left(\hat{r}_i^f + a_i^f\right)\overline{U}_{\max}^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}, \tag{10}$$

or, equivalently,

$$\sum_{j\in\mathcal{N}} y_{ji}^f \leq \left(\hat{r}_i^f + a_i^f\right)\frac{\overline{U}_{\max}^f}{D_i^f} - \hat{x}_i^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}. \tag{11}$$

Summing both sides of Inequalities (11) over all nodes $i \in \mathcal{N}$, we get:

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} y_{ji}^f \leq \sum_{i\in\mathcal{N}}\left(\hat{r}_i^f + a_i^f\right)\frac{\overline{U}_{\max}^f}{D_i^f} - \hat{x}_i^f \qquad \forall f \in \mathcal{F}, \tag{12}$$

which entails:

$$\sum_{j\in\mathcal{N}} \hat{\omega}_j^f \leq \sum_{i\in\mathcal{N}}\left(\hat{r}_i^f + a_i^f\right)\frac{\overline{U}_{\max}^f}{D_i^f} - \hat{x}_i^f \qquad \forall f \in \mathcal{F}. \tag{13}$$

These constraints are likely violated when considering the tentative offloading variables $\hat{\omega}_j^f$ determined by all nodes solving the subproblem (P2).

By introducing the *function residual capacity*:

$$N^f = \sum_{i\in\mathcal{N}}\left(\hat{r}_i^f + a_i^f\right)\frac{\overline{U}_{\max}^f}{D_i^f} - \hat{x}_i^f, \tag{14}$$

it is possible to characterize the deviation, for each function f, between the required number of forwarded requests and the existing capacity at the current iteration h as:

$$dev^f(h) = \sum_{j\in\mathcal{N}} \hat{\omega}_j^f - N^f \qquad \forall f \in \mathcal{F}. \tag{15}$$

To guarantee a globally-feasible solution, the central coordinator may therefore solve, at iteration h, the following optimization problem (P3):

$$\max \sum_{\substack{(i,j)\in\mathcal{E} \\ f\in\mathcal{F}_i\cup\mathcal{F}_j}} \beta_{ij}^f y_{ij}^f - \sum_{\substack{i\in\mathcal{N} \\ f\in\mathcal{F}_i}} \gamma_i^f \left(\hat{\omega}_i^f - \sum_{j:(i,j)\in\mathcal{E}} y_{ij}^f\right) \tag{P3a}$$

subject to:

subject to :

$$\sum_{f\in\mathcal{F}_i} a_i^f \cdot RAM^f = \overline{RAM}_i - \sum_{f\in\mathcal{F}_i} \hat{r}_i^f \cdot RAM^f \qquad \forall\, i\in\mathcal{N} \tag{P3b}$$

$$\sum_{j:(i,j)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ij}^f \le \hat{\omega}_i^f \mu_i^f \qquad \forall i\in\mathcal{N},\ f\in\mathcal{F}_i \tag{P3c}$$

$$\sum_{j:(j,i)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ji}^f \le \Lambda^f \xi_i^f \qquad \forall i\in\mathcal{N},\ f\in\mathcal{F}_i \tag{P3d}$$

$$\xi_i^f + \mu_i^f \le 1 \qquad \forall i\in\mathcal{N},\ f\in\mathcal{F}_i \tag{P3e}$$

$$\sum_{j:(i,j)\in\mathcal{E}\wedge f\in\mathcal{F}_j} y_{ji}^f \le \left(\hat{r}_i^f + a_i^f\right)\frac{\overline{U}_{\max}^f}{D_i^f} - \hat{x}_i^f \qquad \forall i\in\mathcal{N},\ f\in\mathcal{F} \tag{P3f}$$

$$y_{ij}^f \ge 0 \qquad \forall\, i,j\in\mathcal{N},\ f\in\mathcal{F} \tag{P3g}$$

$$a_i^f \in \mathbb{N}_0 \qquad \forall\, i\in\mathcal{N},\ f\in\mathcal{F} \tag{P3h}$$

$$\xi_i^f, \mu_i^f \in \{0,1\} \qquad \forall i\in\mathcal{N},\ f\in\mathcal{F}_i. \tag{P3i}$$

thus determining the *assigned offloading* $v_i^f = \sum_j y_{ij}^f$ and the rate of requests $Y_i^f = \sum_j y_{ji}^f$ received by each node $i\in\mathcal{N}$. We can easily observe that the assigned offloading can never exceed the tentative offloading (Constraints (P3c)), so that the values of v_i^f are determined aiming at saturating as much as possible the overall residual capacity of each node.

Note that the last term $\hat{\omega}_i^f - \sum_j y_{ij}^f$ in the objective function (P3a) corresponds to the rate of requests that are vertically offloaded to the Cloud. Unlike in the centralized problem (P1), we do not introduce here an explicit variable z_i^f to model this decision. This allows us to keep the formulation simpler, since, in the absence of local processing decisions, the optimal values of y_{ij}^f and the rate of requests sent to the Cloud are so closely related to each other.

Moreover, the term Λ^f in Constraints (P3d) is conceptually equivalent to the one introduced for the analogous Constraints (P1e) in problem (P1); however, its value can be reevaluated in this context to provide a tighter upper bound and thus avoid numerical instability, e.g., by setting $\Lambda^f = \sum_j \hat{\omega}_j^f$.

5.2.2. Computing the social welfare

The social welfare, i.e., the quality of the overall processing decisions for the whole system, can be determined by considering the execution cost $\Phi_i^h(\boldsymbol{v}_i)$ under the centralized offloading decisions $\boldsymbol{v}_i = [v_i^f]_{f\in\mathcal{F}}$ for all nodes and functions. In particular, we define:

$$w(\boldsymbol{v}) = \sum_{i\in\mathcal{N}} \Phi_i^h(\boldsymbol{v}_i), \tag{16}$$

where $\Phi_i^h(v)$ is defined as in (7), i.e., it is the cost computed solving a restricted version of problem (P2), where each node i offloads exactly v_i^f and receives a rate of Y_i^f requests per second as determined by the central coordinator. The restricted problem (P4) reads:

$$\min \quad -\sum_{f\in\mathcal{F}_i} \frac{1}{\lambda_i^f}\left(\alpha_i^f x_i^f + \delta_i^f v_i^f\right) \tag{P4a}$$

subject to :

$$\sum_{f\in\mathcal{F}_i} r_i^f \cdot RAM^f \le \overline{RAM}_i \tag{P4b}$$

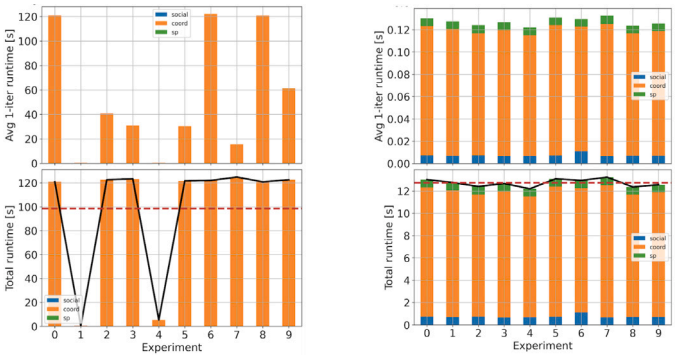
$$x_i^f + v_i^f \le \lambda_i^f \qquad \forall f\in\mathcal{F}_i \tag{P4c}$$

$$D_i^f\left(x_i^f + Y_i^f\right) \le r_i^f \overline{U}_{\max}^f \qquad \forall f\in\mathcal{F}_i \tag{P4d}$$

$$D_i^f\left(x_i^f + Y_i^f\right) \ge \left(r_i^f - 1\right)\overline{U}_{\max}^f \qquad \forall f\in\mathcal{F}_i \tag{P4e}$$

$$x_i^f \ge 0 \qquad \forall f\in\mathcal{F}_i \tag{P4f}$$

$$r_i^f \in \mathbb{N}_0 \qquad \forall f\in\mathcal{F}_i. \tag{P4g}$$



(a) The coordinator solves (P3) (b) The coordinator applies Algorithm 1

Fig. 5. Average per-iteration and total runtime of the FaaS-MACrO iterative process in 10 sample experiments. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

A lower bound for the social welfare $w(v)$ under the current vector of offloading prices Π^h can be derived by considering:

$$w_L(\Pi^h) = \sum_{i\in\mathcal{N}} \left(\Phi_i^k(\hat{\boldsymbol{\omega}}_i) + p(\Pi^h, \hat{\boldsymbol{\omega}}_i)\right) - \sum_{f\in\mathcal{F}} \pi^f(h), \tag{17}$$

since $\sum_f \pi^f(h)$ is the highest additional price the nodes collectively pay to the coordinator.

5.2.3. Updating the offloading prices

Following [55], having defined the deviation $dev^f(h)$ for all functions $f\in\mathcal{F}$ as in Eq. (15) and the lower bound for the social welfare $w_L(\Pi^h)$ as in Eq. (17), we can denote by w^* the minimum value of $w(v)$ found so far during the iterative process and write:

$$\pi^f(h+1) = \max\left\{0, \pi^f(h) + \psi(h)\frac{w^* - w_L(\Pi^h)}{\sum_{f\in\mathcal{F}}(dev^f(h))^2} dev^f(h)\right\}. \tag{18}$$

The parameter ψ^h controls how aggressive the price variation is in iteration h. As in [55], we let its value start at $\psi^0 = 2$ and progressively set $\psi^{h+1} = \psi^h/2$ if the value of w^* does not improve for a predefined number of iterations, defined by the *patience* parameter Θ.

5.2.4. Greedy coordination approach

When the system size increases, the time required by the coordinator to solve problem (P3) becomes too large for practical applications. As a motivating example, we conducted 10 experiments over randomly-generated instances featuring systems with 50 nodes and 5 functions deployed on each node, and we ran the iterative process assuming that FaaS-MACrO has a total time budget of 120 s to determine a globally-feasible solution (i.e., that the process is stopped after 120 s even if it does not reach convergence). Fig. 5(a) reports the average per-iteration runtime (above) and the total runtime of the iterative process (below) across the 10 experiments, computed as the sum over all iterations of the time the average agent takes to solve problem (P2), plus the time it takes to solve (P4), plus the time required by the coordinator to solve (P3). We can observe not only that the coordinator runtime (orange) is dominant, but also that in the 30% of experiments the coordinator takes the whole budget of 120 s to solve the problem (P3), forcing the process to stop after the first iteration with a possibly strong negative impact on the global solution quality.

Therefore, we proposed a *greedy* algorithm to determine the assigned offloading in reasonable computing time. Starting from the local solutions $\langle \hat{x}_i^f, \hat{\omega}_i^f, \hat{r}_i^f\rangle$ shared by each node $i\in\mathcal{N}$ and considering all functions $f\in\mathcal{F}_i$, the method proceeds as reported in Algorithm 1. First of all, the coordination algorithm generates a list $\mathbf{S}$ by sorting the required offloading variables and possible target nodes according to the associated profit. Each element of $\mathbf{S}$ is a tuple $\langle i_s, f_s, \hat{\omega}_{i_s}^{f_s}, T_{i_s}\rangle$, where

$i_s \in \mathcal{N}$ is a node that tentatively offloads $\widehat{\omega}_{i_s}^{f_s} > 0$ req/s of function f_s as a result of the optimization in (P2). T_{i_s} is a list of targets for the offloading, i.e., it includes all the neighbors $\tau : (i_s, \tau) \in \mathcal{E} \wedge f_s \in \mathcal{F}_{i_s} \cap \mathcal{F}_\tau$; these are sorted in non-increasing order of offloading profit, i.e., τ_1 precedes τ_2 in T_{i_s} if $\beta_{i_s\tau_1}^{f_s} \widehat{\omega}_{i_s}^{f_s} \geq \beta_{i_s\tau_2}^{f_s} \widehat{\omega}_{i_s}^{f_s}$.[5]

For each target node τ, the *residual memory capacity* ρ_τ is initially computed as:

$$\rho_\tau = \overline{RAM}_\tau - \sum_{f \in \mathcal{F}_\tau} \widehat{r}_\tau^f \cdot RAM^f. \tag{19}$$

Accordingly, by Eq. (9), at most $\frac{\rho_\tau}{RAM^f}$ additional replicas can be started on node τ to execute a generic function $f \in \mathcal{F}_\tau$ (assuming that the whole residual memory capacity is devoted to this function). Therefore, it is possible to estimate the maximum acceptable request rate for function f as:

$$\Omega_\tau^f = \max\left\{0, \left(\widehat{r}_\tau^f + \frac{\rho_\tau}{RAM^f}\right) \frac{\overline{U}_{\max}^f}{D_\tau^f} - \widehat{x}_\tau^f - \sum_{j \in \mathcal{N}} y_{j\tau}^f\right\}. \tag{20}$$

If the desired offloading $\widehat{\omega}_{i_s}^{f_s}$ does not exceed the maximum rate $\Omega_\tau^{f_s}$ (lines 11–15), the algorithm adds $\widehat{\omega}_{i_s}^{f_s}$ to the value of variable $y_{i_s\tau}^{f_s}$. Moreover, it computes the actual number of additional replicas (denoted in line 13 as $A_\tau^{f_s}$) that node τ needs to instantiate to support the execution of the new requests, and it updates the residual memory capacity accordingly.

A similar procedure is followed if the value of the required offloading variable $\widehat{\omega}_{i_s}^{f_s}$ exceeds $\Omega_\tau^{f_s}$ (lines 16–20). In this case, $\Omega_\tau^{f_s}$ requests per second are assigned to node τ, the number of additional replicas required to execute them is precisely equal to the upper bound $\frac{\rho_\tau}{RAM^f}$, and the residual memory capacity of node τ is exhausted.

Fig. 5(b) reports the per-iteration and total runtime of the same 10 experiments observed in Fig. 5(a) when the coordinator applies Algorithm 1. We can easily observe that, while still dominant with respect to the execution time of (P2) (green) and (P4) (blue), the coordinator runtime is reduced by up to three orders of magnitude (it reaches at most 0.12 s against the 120 s required, e.g., in experiment 0), allowing the process to run for a larger number of iterations, if needed, within the same time budget.

A deep analysis and comparison of the performance obtained when tackling the coordination problem by solving (P3) or applying Algorithm 1, both in terms of execution time and solution quality, will be discussed in Section 6.

5.2.5. Privacy-preserving coordination

Although, in a cooperative context, managing contracts with a unique, central entity is easier than defining protocols to interact with multiple, decentralized actors, adopting the proposed multi-agent workload orchestration approach in practical scenarios may introduce privacy concerns if nodes are managed by different stakeholders (e.g., companies) that usually want to minimize the amount of shared information.

While the models reported in the previous sections were developed assuming the coordinator has full access to the locally-optimal solution $\langle \widehat{x}_i^f, \widehat{\omega}_i^f, \widehat{r}_i^f \rangle \ \forall\ f \in \mathcal{F}_i$ determined by each node $i \in \mathcal{N}$, as well as to details about the service times D_i^f and utilization thresholds $\overline{U}_{\max}^f$ imposed for all functions, the formulation of (P3) can be easily adapted to consider only a minimal level of information sharing.

Indeed, we can consider the definition of residual memory capacity ρ_i introduced in Eq. (19) and denote by:

$$\sigma_i^f = \widehat{r}_i^f \frac{\overline{U}_{\max}^f}{D_i^f} - \widehat{x}_i^f \tag{21}$$

[5] Note that an alternative sorting rule may be generated by considering only $\beta_{i_s\tau}^{f_s}$.

Algorithm 1 Greedy Coordination

1: **Input:** $sol = \{\langle \widehat{x}_i^f, \widehat{\omega}_i^f, \widehat{r}_i^f \rangle \ \forall i \in \mathcal{N},\ f \in \mathcal{F}_i\}$, sorting rule
2: $y_{ij}^f \leftarrow 0 \quad \forall\ i, j \in \mathcal{N},\ f \in \mathcal{F}_i \cap \mathcal{F}_j$
3: $z_i^f, a_i^f, \xi_i^f, \mu_i^f \leftarrow 0 \quad \forall\ i \in \mathcal{N},\ f \in \mathcal{F}_i$
4: **S** = `sort_by_offloading_profit(`*sol*`, sorting rule)`
5: Compute the residual memory capacity ρ_i of all nodes (Eq. (19))
6: **for** each $i_s, f_s, \widehat{\omega}_{i_s}^{f_s}, T_{i_s}$ in **S** **do**
7: **while** $\widehat{\omega}_{i_s}^{f_s} > 0$ **and** T_{i_s} is not empty **do**
8: Pick the first target node $\tau \in T_{i_s}$
9: **if** τ has residual memory capacity ($\rho_\tau > 0$) **and** does not offload f_s ($\mu_\tau^{f_s}$ is 0) **then**
10: Compute the maximum acceptable request rate $\Omega_\tau^{f_s}$ (Eq. (20))
11: **if** $\widehat{\omega}_{i_s}^{f_s} < \Omega_\tau^{f_s}$ **then**
12: $y_{i_s\tau}^{f_s} \leftarrow y_{i_s\tau}^{f_s} + \widehat{\omega}_{i_s}^{f_s}$
13: $A_\tau^{f_s} \leftarrow \min\left\{\frac{\rho_\tau}{RAM^{f_s}}, \left(\sum_j y_{j\tau}^{f_s} + \widehat{x}_\tau^{f_s}\right)\frac{D_\tau^{f_s}}{\overline{U}_{\max}^{f_s}} - \widehat{r}_\tau^{f_s} - a_\tau^{f_s}\right\}$
14: $\rho_\tau \leftarrow \rho_\tau - A_\tau^{f_s} \cdot RAM^{f_s}$
15: $\widehat{\omega}_{i_s}^{f_s} \leftarrow 0$
16: **else**
17: $y_{i\tau}^f \leftarrow y_{i\tau}^f + \Omega_\tau^{f_s}$
18: $A_\tau^{f_s} \leftarrow \frac{\rho_\tau}{RAM^{f_s}}$
19: $\rho_\tau \leftarrow 0$
20: $\widehat{\omega}_{i_s}^{f_s} \leftarrow \widehat{\omega}_{i_s}^{f_s} - \Omega_\tau^{f_s}$
21: **end if**
22: $a_\tau^{f_s} \leftarrow a_\tau^{f_s} + A_\tau^{f_s}$
23: $\mu_{i_s}^{f_s} \leftarrow 1$ and $\xi_\tau^{f_s} \leftarrow 1$
24: **else**
25: $T_{i_s} \leftarrow T_{i_s} \setminus \{\tau\}$
26: **end if**
27: **end while**
28: **if** $\widehat{\omega}_{i_s}^{f_s} > 0$ **then**
29: $z_{i_s}^{f_s} \leftarrow \widehat{\omega}_{i_s}^{f_s}$
30: **end if**
31: **end for**

the *residual computational capacity* of the replicas deployed on node i to serve function f as a result of the local optimization. Furthermore, we can characterize the *computational impact* of function f on node i as:

$$\varphi_i^f = \frac{\overline{U}_{\max}^f}{D_i^f}. \tag{22}$$

As a result, Constraints (P3b) and (P3f) can be rewritten as:

$$\sum_{f \in \mathcal{F}_i} a_i^f \cdot RAM^f = \rho_i \qquad \forall\ i \in \mathcal{N} \tag{P5b}$$

and

$$\sum_{j:(i,j) \in \mathcal{E} \wedge f \in \mathcal{F}_j} y_{ji}^f \leq \sigma_i^f + \varphi_i^f a_i^f \qquad \forall i \in \mathcal{N},\ f \in \mathcal{F}, \tag{P5f}$$

respectively, effectively avoiding the need to share with the coordinator any potentially privacy-sensitive information.

Other than enhancing confidentiality, adopting σ_i^f and φ_i^f in the problem formulation significantly reduces the number of parameters that need to be shared between each node and the central coordinator, thus limiting the communication overhead over the network. This aspect is critical in Edge computing, since, as already mentioned, the links connecting nodes in the network are usually characterized by limited bandwidth.

If we denote by $|\mathcal{N}|$ the number of nodes in the network and by $|\mathcal{F}|$ the number of deployed functions, the overall impact of information sharing in each iteration of the coordination process is of the order of:

$$\mathcal{O}\left(|\mathcal{N}||\mathcal{F}|\pi^f + |\mathcal{N}||\mathcal{F}|(\widehat{\omega}_i^f + \sigma_i^f + \varphi_i^f) + |\mathcal{N}||\mathcal{F}|(\nu_i^f + \Phi_i^h(\boldsymbol{\nu}_i))\right),$$

where the first term is due to the central coordinator sharing with individual nodes the vector of offloading prices, the second term is due to nodes informing the coordinator about the local processing decisions, and the third term represents information sharing required to compute the social welfare (see Section 5.2.2). This can be expressed as $\mathcal{O}\left(|\mathcal{N}||\mathcal{F}|\kappa\right)$, where κ denotes the size of each shared parameter. As an example, if these are represented by floating point numbers and therefore occupy 4 bytes each, tackling a system featuring 100 Edge nodes and 5 functions deployed on each node introduces a communication overhead of roughly $12Kb$ of data on average at each iteration over the whole network.

5.3. Stopping criteria

The iterative process illustrated in Fig. 4 and described in the previous sections terminates when either a maximum number of iterations is reached or any of the following conditions is satisfied:

- The locally-optimal solution $\langle \hat{x}_i^f, \hat{\omega}_i^f, \hat{r}_i^f \rangle$ is globally feasible;
- The offloading prices π^f for all functions $f \in \mathcal{F}$ keep increasing for a predefined number of iterations Θ, meaning that the coordination process does not help nodes advance towards an improved global solution. This occurs, e.g., when the overall incoming load $\sum_i \lambda_i^f$ exceeds the total computational capacity of the network for all functions f, entailing that some vertical offloading to Cloud is inevitable (i.e., even solving the centralized system of Section 4 would result in an optimal solution with $z_i^f > 0$ for at least one node i and function f).
- The deviation between the optimal load management cost identified by the local nodes solving (P2), i.e., $\sum_i \left(\Phi_i^h(\hat{\boldsymbol{\omega}}_i) + p(\Pi^h, \hat{\boldsymbol{\omega}}_i)\right)$, and the social welfare $w(\boldsymbol{\nu})$ remains smaller than a threshold for Θ consecutive iterations.
- The deviation between the last Θ values of the best social welfare w^* is smaller than a threshold.

If, at the end of the last iteration, $\nu_i^f < \hat{\omega}_i^f$ (i.e., the process terminates without identifying a globally-feasible solution), the nodes set $z_i^f = \hat{\omega}_i^f - \nu_i^f$, i.e., offload to the Cloud all requests that could not be successfully processed by neighbors.

6. Experimental analysis

This section presents and discusses the results obtained validating the centralized problem (P1), referred to in the following as the Load Management Model (LMM), and the FaaS-MACrO approach presented in Section 5.

A preliminary FaaS workload traces analysis, reported in Section 6.1 was conducted to evaluate the impact of considering in our model the average incoming rate of requests $\lambda_i^f(t)$ as an approximation of the actual load. Sections 6.3, 6.4 and 6.5 report the results of the comparison and scalability analysis of LMM and FaaS-MACrO on various problem instances generated as described in Section 6.2. In particular, the experiments were designed to evaluate the performance of the two methods, the FaaS-MACrO capacity to preserve near-optimal performance, the trade-off between solution quality and runtime, and the resilience of our approach under different network dimensions and topologies, and various levels of node and function heterogeneity.

6.1. FaaS traces analysis

To assess the accuracy of adopting the average over a fixed time period t as an approximation of real workloads, we analyzed 6385 traces extracted from Microsoft Azure Functions Trace dataset [59], selecting only those functions for which continuous data were available for the entire 14-day observation window. To the best of our knowledge, this is the only publicly available dataset focused on FaaS execution. Fig. 6 illustrates a representative workload trace extracted from this dataset, depicting the number of requests per minute arriving at a serverless function over a 14-day period. The blue markers represent the raw workload data, while the red and green lines indicate, respectively, a moving average with a 5-minute window and a non-overlapping 5-minute average. The orange line reports the corresponding 75th percentile, also computed over non-overlapping 5-minute windows. Note that the choice of considering time intervals t of 5 min aligns with some state-of-the-art approaches [60], while control periods range from tens of seconds to several minutes or even one hour in the general literature [35,43,46].

For each trace, we computed the Mean Absolute Percentage Error (MAPE) between the original per-minute workload and its approximation using the 5-minute non-overlapping average (the green curve in Fig. 6). Results show that in 77% of the functions, the MAPE remains below 30%, which is generally regarded in the performance modeling literature as a threshold for reasonable accuracy (see, e.g., [57]). This indicates that short-term workload averaging can effectively capture the main dynamics of serverless workloads.

However, in the context of workload orchestration and resource provisioning, underestimation errors are significantly more critical than overestimations. Overestimating workload leads to conservative resource allocation, potentially inefficient but safe, whereas underestimating may result in node saturation and degraded performance. To better characterize this asymmetric risk, we introduced a metric called the Mean Percentage Error of Underestimation (MPEU), defined as:

$$\text{MPEU} = \frac{1}{N}\sum_{i=1}^{N}\frac{\max(y_i - \hat{y}_i, 0)}{y_i}, \tag{23}$$

where we denote by y_i the real workload value and by $\hat{y}_i$ the corresponding prediction. This metric captures only the relative magnitude of underestimations, ignoring overestimation contributions. Across the 6385 analyzed traces, we observed that the maximum MPEU did not exceed 26%, and that for 75% of the functions the value remained below 8%. These results confirm that a 5-minute average can be effectively employed as an approximation strategy in the design of adaptive workload orchestration and Edge resource management policies without incurring significant risk of under-provisioning.

As shown in Fig. 6, considering the 75th percentile as a predictor instead of the average would further reduce the risk of underestimation (and therefore make the estimate more conservative); such an approach may be followed, e.g., on Edge nodes that struggle or cannot implement vertical offloading strategies (due, e.g., to network disconnections).

6.2. Experimental setup and methodology

To compare the performance of the centralized and distributed approaches, we conducted several sets of experiments, considering fully- or partially-connected networks of Edge nodes $\mathcal{N}$ and assuming that there exists a unique set of functions $\mathcal{F}$ deployed on all nodes (i.e., $\mathcal{F}_i = \mathcal{F}\ \forall\ i \in \mathcal{N}$). In all experiments, we set $\overline{U}_{\max}^f = 0.8$, and we considered a variable number of nodes $|\mathcal{N}|$, hereafter referred to as the *network dimension*, and a variable number of functions $|\mathcal{F}|$. For each experiment, characterized by specific values of $|\mathcal{N}|$ and $|\mathcal{F}|$, we generated random problem instances as described in the following.

Scenario A (fully-connected networks). For the first series of experiments (see Section 6.3), we assumed that all nodes in $\mathcal{N}$ are connected to each other, i.e., that $\exists\ (i,j) \in \mathcal{E}\ \forall\ i,j \in \mathcal{N}$. On the one hand, this is a simplifying assumption with respect to real Edge systems, since the increased number of connections entails that more neighbors may be available to absorb the excess load of saturated nodes. On the other hand, however, this scenario challenges the scalability of both LMM and FaaS-MACrO, since a large number of edges makes it harder to identify the optimal request distribution, particularly in a multi-agent setting with limited visibility on the nodes state.

We analyzed the performance of our approaches in four different cases, as listed in the following:

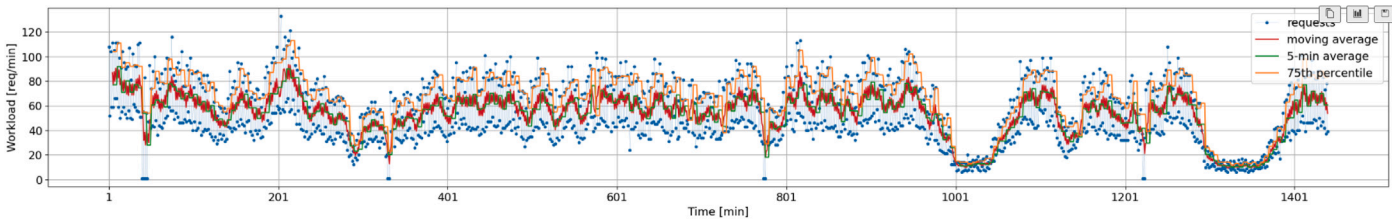


Fig. 6. Sample workload trace from [59]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

A.1: The first case, whose results are discussed in Section 6.3.1, evaluates the impact of *varying the network dimension*, progressively increasing the number of nodes $|\mathcal{N}| = 4, 10, 20, 50, 100, 200$, assuming that they are homogeneous in terms of hardware characteristics and, therefore, have the same memory and computational capacity.

Following other literature proposals [33,50], we considered a fixed memory capacity $\overline{RAM}_i = 4Gb \; \forall \, i \in \mathcal{N}$, and we assumed for the service demand D_i^f that there may exist $f_1, f_2 \in \mathcal{F}$ for which $D_i^{f_1} \neq D_i^{f_2}$, but $D_i^f = D^f \; \forall \, i \in \mathcal{N}$, sampling D^f uniformly at random from the interval $[0.1, 0.5]$ s. Moreover, each function $f \in \mathcal{F}$ had a memory requirement RAM^f sampled uniformly between 128 Mb and 1 Gb.

As mentioned in Section 3, while we do not model explicitly the functions initialization time or the network latency, these parameters are used to determine the values of the objective function weights α_i^f, β_{ij}^f, γ_i^f and δ_i^f. In particular, for each function f, the local execution profit α_i^f (equal for all nodes $i \in \mathcal{N}$) is inversely proportional to the initialization time, which is sampled uniformly at random from the interval $[0.25, 0.75]$ s. To determine the impact of offloading, as in other literature proposals [60,61] we modeled the network transfer time b_{ij}^f as:

$$b_{ij}^f = \text{access delay} \; \cdot \frac{\text{data size}}{\text{bandwidth}}. \tag{24}$$

As in [50], we sampled the input data size of each function $f \in \mathcal{F}$ from a truncated normal distribution, with mean and standard deviation equal to $1Kb$, and support (100 b, 5 Mb). Moreover, we considered an access delay of 5 ms for each node $i \in \mathcal{I}$ and a bandwidth of 100 Mbps. The offloading benefit β_{ij}^f for each function f and pair of nodes $(i, j) \in \mathcal{E}$ is defined as inversely proportional to the sum of the initialization time of function f and the transfer time b_{ij}^f. The parameter δ_i^f used in FaaS-MACrO (problems (P2) and (P4) in Section 5) is set to $\frac{1}{|\mathcal{E}|} \sum_j \beta_{ij}^f$ for each node $i \in \mathcal{N}$ and function $f \in \mathcal{F}$. Finally, the penalty γ_i^f associated with offloading requests to the Cloud is defined to be directly proportional to the corresponding network transfer time. This is computed as in Eq. (24), assuming a bandwidth of 10, Mbps and sampling the access delay uniformly at random from the set $\{50, 100, 200\}$, ms.

In all scenarios, once a base problem instance is generated by appropriately setting the network size and the values of D_i^f, α_i^f, β_{ij}^f, γ_i^f, and δ_i^f, this is solved under 10 different configurations of incoming load λ_i^f for each node $i \in \mathcal{I}$ and function $f \in \mathcal{F}$, with variations of up to 300% between different experiments. This results in a total of 30 problem instances for each network size, for a grand total of 180 experiments in scenario **A.1**. The values of λ_i^f are selected to ensure that the network operates in a regime that is neither under- nor over-saturated; in other words, nodes cannot rely solely on local processing but must distribute incoming load among one another to limit the amount of traffic ultimately offloaded to the Cloud.

A.2: For the second set of experiments (see Section 6.3.2), we kept varying the network dimension $|\mathcal{N}|$ between 4 and 200, but assumed, as in other literature proposals [33,62], that nodes belong to three classes with *different hardware characteristics*, as reported in Table 5. According to this, nodes have different computational capacities, which entails that the service demand of each function f may vary when this is executed on different nodes (i.e., there may exist i_1 and i_2 such that $D_{i_1}^f \neq D_{i_2}^f$ if the nodes belong to different classes). Accordingly, for each of the 5 functions in $\mathcal{F}$, we generated D_i^f uniformly at random from the interval $[0.2, 0.5]$ s and then applied a speed-up factor [33] as reported in Table 5. All the other parameters were generated as in scenario **A.1**, for a total number of 180 experiments.

Table 5
Classes of Edge nodes and relative speed-up factors.

Node class	RAM	Speed-up factor	Proportion in the network
light	4 Gb	0.5×	12.5%
	8 Gb	1.0×	12.5%
intermediate	16 Gb	1.2×	37.5%
high	24 Gb	1.4×	25%
	32 Gb	1.4×	12.5%

A.3: The third set of experiments, discussed in Section 6.3.3, evaluates the impact of *varying the number $\mathcal{F}$ of deployed functions*. We considered $|\mathcal{N}| = 50$ Edge nodes with heterogeneous hardware characteristics (as those generated in scenario **A.2**), and evaluated $|\mathcal{F}| = 3, 5, 8, 10, 15$ and 20, for a total number of 180 experiments.

A.4: Finally, the last set of 120 experiments with fully-connected networks (see Section 6.3.4) concerns networks of heterogeneous Edge nodes with $|\mathcal{N}| = 20, 50, 100$ or 200 (generated as in scenario **A.2**), subject to *light-load conditions*. Accordingly, instead of choosing λ_i^f to keep the nodes on the verge of saturation as it was done in scenarios **A.1**–**A.3**, we constrained the generated values to keep $\sum_f \lambda_i^f$ close to the rate of requests that each node $i \in \mathcal{N}$ can process locally.

Scenario B (different network topologies). To evaluate the impact of the network topology on the models scalability and on the Edge networks capacity to handle the incoming load, Section 6.4 discusses the impact of varying the patterns in node connectivity. In particular:

B.1: In the first set of experiments, we considered the same problem instances generated for the heterogeneous nodes analysis (scenario **A.2**) and progressively reduced the *degree* k of each node: while, in a fully-connected network of $|\mathcal{N}| = 200$ nodes, each $i \in \mathcal{N}$ has exactly $k = |\mathcal{N}| - 1$ neighbors, we considered four scenarios with k equal to 3, 5, 10, 25, 50, 100 or 150. For each value of k, we generated three problem instances by letting each node select k peers at random as its neighbors (ensuring that the graph remains connected), for a total number of 210 experiments. The random graph generation was performed using the NetworkX[6] Python library. The results are reported in Section 6.4.1.

B.2: Finally, we run a last set of experiments by assuming that only a fraction of Edge nodes – that we called the *edge exposed fraction* (eef) – is connected to an access point (and therefore receives a workload λ_i^f from clients), while all the others can only process requests offloaded by neighbors. We considered $|\mathcal{N}| = 200$ nodes as in **B.1** and varied eef between 0.5% (that is, only one node is connected to an access point) and 99.5% (i.e., all nodes except one are connected to an access point).

[6] https://networkx.org

For each value of eef, experiments were repeated considering fully-connected networks, networks whose nodes have degree $k = 100$, and networks with $k = 10$, for a total number of 810 experiments. Results are reported in Section 6.4.2.

Comparison metrics:. In each of the following sections, to evaluate the difference between objective function values obtained by LMM and FaaS-MACrO we computed the percentage *objective deviation*:

$$dev_O = \frac{O_{FaaS-MACrO} - O_{LMM}}{O_{LMM}} \cdot 100, \tag{25}$$

where O_{method} is the value of (P1a). This value is yielded directly by LMM as part of the optimization process, whereas for FaaS-MACrO it is computed a posteriori applying Eq. (P1a) to the solution obtained at the end of the iterative procedure. Evaluating both methods using Eq. (P1a) guarantees that they are compared against the same metric employed for the centralized solution. It is important to note that dev_O takes negative values when the solution produced by FaaS-MACrO is *worse* than that of LMM, which is consistent with the fact that Eq. (P1a) is maximized in the centralized load management problem.

The difference in terms of time-to-solution is measured through the *runtime deviation*:

$$dev_R = \frac{R_{FaaS-MACrO}}{R_{LMM}}, \tag{26}$$

where R_{method} denotes the runtime of the method. A value of $dev_R > 1$ means that FaaS-MACrO is dev_R times slower than LMM in identifying the final solution. The time-to-solution for FaaS-MACrO is computed as the sum over all iterations of the time the average agent takes to solve problem (P2), plus the time it takes to solve (P4), plus the time required by the coordinator to solve either (P3) or the greedy Algorithm 1, as already mentioned in Section 5.2.4.

Finally, we compared the percentage rate of requests that are offloaded to the Cloud in the final solution identified by LMM and FaaS-MACrO, i.e., the value of $\sum_{i,f} \frac{z_i^f}{\lambda_i^f}$. We defined the *Cloud offloading deviation*:

$$dev_C = \left(\sum_{i,f} \frac{z_i^f}{\lambda_i^f} \cdot 100 \right)_{FaaS-MACrO} - \left(\sum_{i,f} \frac{z_i^f}{\lambda_i^f} \cdot 100 \right)_{LMM}. \tag{27}$$

While the rates of vertical offloading can be indirectly compared by observing dev_O (since z_i^f appear in the objective function (P1a)), a more direct comparison can be relevant when discussing the performance of LMM and FaaS-MACrO. Indeed, only the coordinator problem (P3) explicitly considers vertical offloading in its optimization (see Section 5.2.1), while individual nodes only act through local processing and horizontal offloading. As a result, it may be more challenging for FaaS-MACrO to identify good-quality solutions when the overall network is close to saturation, because a globally-optimal solution involving a significant percentage of vertical offloading would be considered by the iterative process only as a last resort.

Problems (P1)–(P4) are solved using the Gurobi Optimizer[7] version 12.0.3. The MIP gap is set to 10^{-5} for all models except (P3), for which we considered 10^{-3}. All experiments were run on Azure Virtual Machines *Standard B16ls v2*, equipped with 16 vCPUs and 32Gb of RAM.[8]

All the data and results discussed in the next sections are publicly available on Zenodo.[9] The implementation of LMM and FaaS-MACrO is open-source on GitHub.[10]

[7] https://www.gurobi.com/solutions/gurobi-optimizer/

[8] https://azure.microsoft.com/en-gb/pricing/details/virtual-machines/linux/#pricing

[9] https://doi.org/10.5281/zenodo.17670071

[10] https://github.com/unimib-datAI/DFaaSOptimizer/tree/25.11.22

6.3. Fully-connected networks

The results reported in this section concern the comparison and scalability analysis of the centralized LMM (P1) and FaaS-MACrO when considering fully-connected networks of Edge nodes generated as described in Section 6.2 (scenario A). In particular, each problem instance is solved once by LMM considering a time limit of 120 s, and twice by FaaS-MACrO to consider the differences, in terms of objective function deviation and time-to-solution, between addressing the coordination step by solving the optimization problem (P3) and applying the greedy algorithm presented in Section 5.2.4. In the first case, the maximum number of iterations is set to 100, while in the second case, we considered 100 or 500 iterations; the patience parameter Θ and the tolerance used to assess the termination criteria defined in Section 5.3 are always equal to 10 and 0.001, respectively. To guarantee a fair comparison with LMM, we set a maximum time limit of 120 s also for FaaS-MACrO.

In the following, Section 6.3.1 discusses the results obtained on networks of homogeneous nodes and functions (scenario **A.1**), while Section 6.3.2 evaluates the impact of hardware heterogeneity (scenario **A.2**). Then, we discuss in Section 6.3.3 the specific challenges introduced when considering a larger number of functions deployed on each node (scenario **A.3**), and we present in Section 6.3.4 an analysis conducted under light-load conditions (scenario **A.4**).

6.3.1. Homogeneous nodes

Fig. 7 illustrates the comparison between LMM and FaaS-MACrO when considering the optimal coordinator (P3). As detailed in Section 6.2 (scenario A.1), three random problem instances are generated for each network dimension (or number of agents), and each problem instance is solved considering 10 different values of incoming workload λ_i^f, for a total number of 30 experiments. For a fixed number of agents $|\mathcal{N}|$, each violin plot in the first row represents the distribution of results obtained with LMM or FaaS-MACrO across the 30 experiments, with horizontal dashed lines representing the quartiles. The bars in the second row report the average, minimum and maximum value of dev_O, dev_R and dev_C, respectively (average deviations are also reported for clarity in Table 6).

From Fig. 7 (top right corner) and the first line of Table 6, we can observe that the objective deviation (dev_O) is between −6.94% and −0.24%, which allows us to conclude that FaaS-MACrO, although designed to aim for global feasibility rather than global optimality, gets reasonably close to the solution determined by LMM.

For small networks (when $|\mathcal{N}|$ is 4 or 10), the time required by the iterative process to determine the final solution is significantly larger than the time required by LMM, with an average deviation between one and two orders of magnitude. However, both methods are still reasonably fast, with FaaS-MACrO requiring less than 2 s for $|\mathcal{N}| = 4$ and around 4 s for $|\mathcal{N}| = 10$ on average to determine the solution (we recall that, as discussed in Section 6.1, a reasonable duration for the control period t is in the order of 5 min).

For intermediate networks (when $|\mathcal{N}|$ is 20 or 50), we can observe from the second line of Fig. 7 that the time-to-solution of LMM has very large variability, ranging, e.g., from 1.3 s to 120 s, with an average value of 52.2 s when $|\mathcal{N}|$ is 20. Instead, the variability is significantly less noticeable in FaaS-MACrO, whose runtime ranges between 1.01 s and 13.6 s, with an average of 9.4 s. Due to this, we also observe a large variability in dev_R, since FaaS-MACrO outperforms LMM in terms of runtime when the latter approaches the time limit, but takes longer on instances that are easily solved by LMM (see Fig. 8).

Finally, both LMM and FaaS-MACrO reach the time limit of 120 s when considering a large network of 100 nodes. This is due, in the case of FaaS-MACrO, to the large time required by the coordinator to solve problem (P3) at each iteration. An example is reported in Fig. 9(a): considering a sample instance with $|\mathcal{N}| = 100$ nodes and 10 different values of input workload λ_i^f (represented on the x axis

Table 6
Comparison among dev_O, dev_R and dev_C obtained by FaaS-MACrO for different network dimensions when tackling the coordination problem by solving (P3) or by Algorithm 1, and when considering a maximum number of 100 or 500 iterations.

$\lvert\mathcal{N}\rvert$		4	10	20	50	100	200
(P3), 100	dev_O	−4.85%	−1.91&	**−1.84%**	**−0.24%**	−6.94%	–
	dev_R	18.57×	9.49×	2.05×	2.41×	1.0×	–
	dev_C	8.74%	0.50%	0.66%	0.18%	3.79%	–
Alg. 1, 100	dev_O	−4.92%	−2.91%	−3.46%	−1.36%	−6.30%	−2.52%
	dev_R	10.02×	5.56×	0.97×	0.71×	0.49×	0.93×
	dev_C	9.34%	0.98%	1.80%	0.55%	2.48%	2.01%
Alg. 1, 500	dev_O	**−3.52%**	**−1.41%**	−1.94%	−0.98%	**−5.15%**	0.6%
	dev_R	36.19×	15.34×	3.75×	1.31×	0.93×	3.78×
	dev_C	7.02%	0.60%	1.90%	0.34%	1.76%	−1.27%

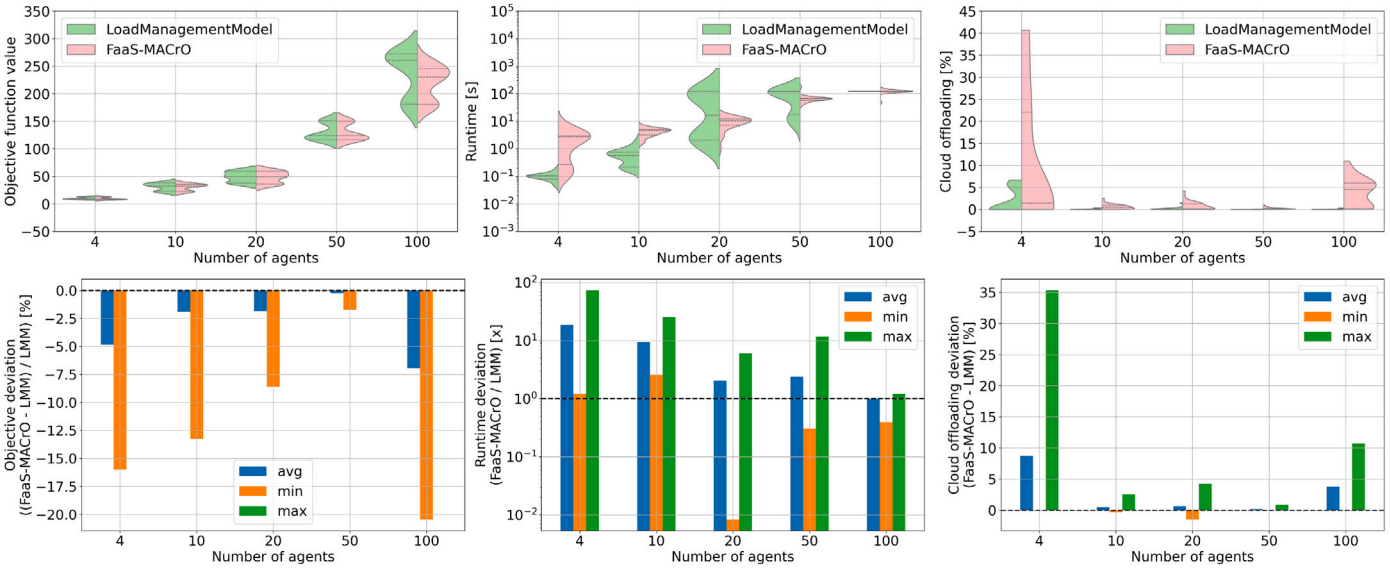


Fig. 7. Comparison between the value of the objective function (P1a), the time-to-solution and the percentage rate of requests offloaded to Cloud obtained with LMM and FaaS-MACrO; the coordinator solves problem (P3); the maximum number of iterations is 100.

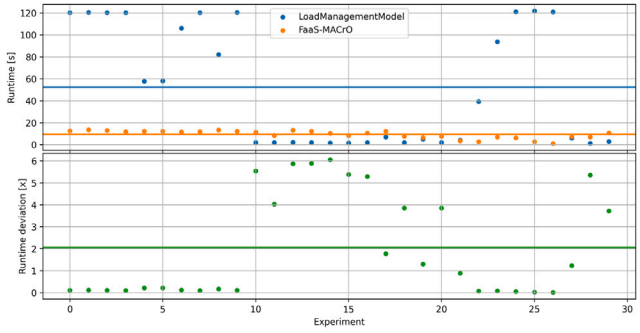
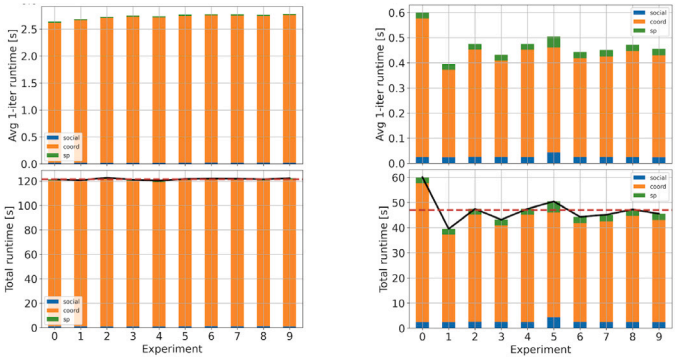


Fig. 8. Detailed comparison between LMM and FaaS-MACrO times-to-solution when the coordinator solves problem (P3) and considering $|\mathcal{N}| = 20$ nodes. Each dot represents the value in a specific experiment, while horizontal solid lines report the average values.



(a) The coordinator solves (P3) (b) The coordinator applies Algorithm 1

Fig. 9. Average per-iteration and total runtime of FaaS-MACrO in a sample problem instance with $|\mathcal{N}| = 100$ nodes and considering 10 different values of λ_i^f. The figure reports the time-to-solution of (P2) (in green), the coordination step (orange) and (P4) (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

as 10 different experiments), we can observe that the time-to-solution of the coordination problem (P3) dominates the average per-iteration execution time, taking almost 3 s. On the other hand, Fig. 9(b) shows the FaaS-MACrO runtime when executing the same experiments by solving the coordination problem through the greedy Algorithm 1. We can observe that, even though the coordination time is still significantly larger than the time required by each node to solve (P2) and (P4), it never exceeds 0.6 s on average in a single iteration, reducing the total runtime to less than 50 s on average in the considered instances.

A detailed comparison between the performance of LMM and FaaS-MACrO when applying Algorithm 1 is shown in Fig. 10, with average deviations reported in lines three to five of Table 6. We can observe

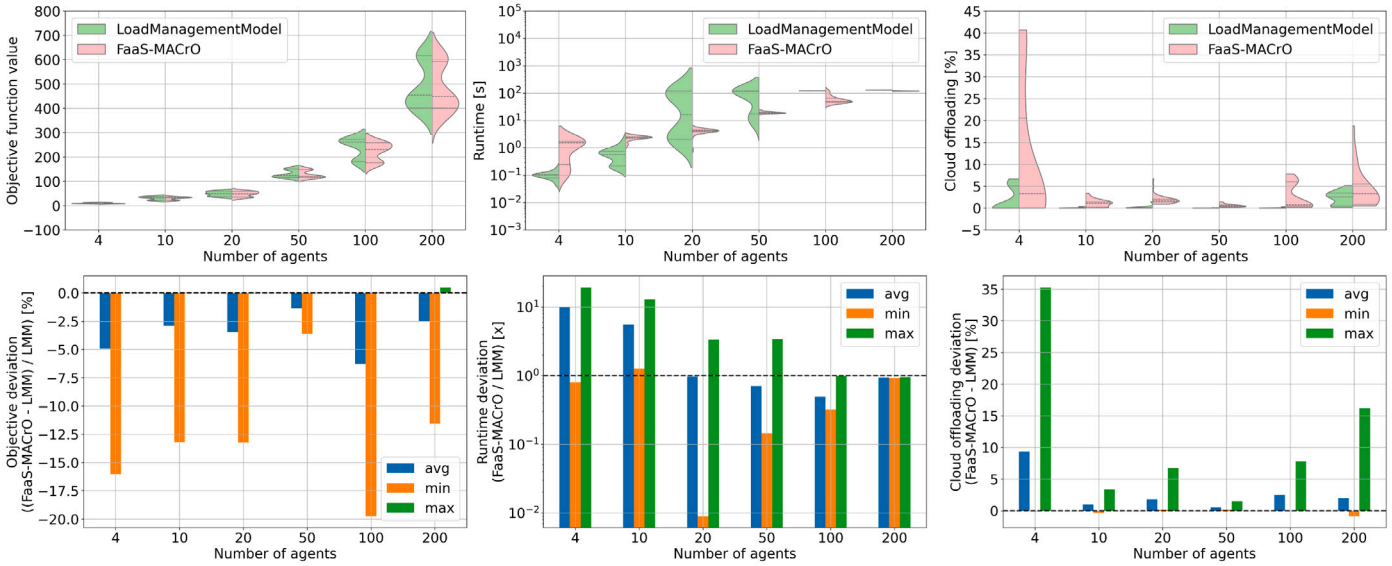


Fig. 10. Comparison between the value of the objective function (P1a), the time-to-solution, and the percentage rate of requests offloaded to Cloud obtained with LMM and FaaS-MACrO; the coordinator applies the greedy Algorithm 1; the maximum number of iterations is 100.

that dev_O maintains acceptable values on average (between −1.36 and −6.30%), but the runtime of FaaS-MACrO is significantly reduced, steadily outperforming LMM even for intermediate networks of 20 nodes ($dev_R < 1$ in the fifth row of Table 6 and the central-right plot in Fig. 10). Furthermore, when $|\mathcal{N}|$ is 100, LMM reaches the time limit of 120 s without determining the optimal solution, while the average runtime of FaaS-MACrO is still around 60 s (both methods stop at the time limit, instead, for $|\mathcal{N}|$ equal to 200).

Finally, we have evaluated the impact of considering a maximum number of 500 iterations for FaaS-MACrO when applying the greedy Algorithm 1 to solve the coordination problem. As illustrated in Fig. 11 and the last three rows of Table 6, the average dev_O significantly reduces in absolute value, even outperforming the other FaaS-MACrO versions in some scenarios, at the price of a significantly longer execution time, particularly for small systems of 4 or 10 nodes. Overall, even if increasing the maximum number of iterations may be beneficial in some instances (FaaS-MACrO identifies a globally-feasible solution in 80% of the cases using between 100 and 200 iterations in a sample instance with 50 nodes), the gain in terms of objective function value does not justify the longer execution times in the considered scenarios.

A separate discussion is needed for the case considering $|\mathcal{N}| = 200$. Indeed, since in this context the execution time of FaaS-MACrO reached 120 s already when executing at most 100 iterations (see Fig. 10), we had to remove the time limit to observe its behavior with a maximum number of 500 iterations. This explains why sometimes FaaS-MACrO outperforms LMM in terms of objective function value (the average dev_O is greater than 0 in Table 6), since the latter stops at the time limit without necessarily identifying the optimal solution.

6.3.2. Heterogeneous nodes

The results of a comparison between LMM and FaaS-MACrO considering variable-sized networks of heterogeneous Edge nodes (scenario **A.2**) are reported in Fig. 12, while the average values of dev_O, dev_R and dev_C are listed in Table 7. As in scenario **A.1**, the networks are fully connected, i.e., $(i, j) \in \mathcal{E}$ for each pair of nodes $i, j \in \mathcal{N}$.

Despite the additional complexity introduced by considering different classes of nodes and their impact on the service demands D_i^f, we can observe from Fig. 12 (first line) that the behavior of FaaS-MACrO in terms of the value of objective function (P1a) is still reasonably aligned with the one of LMM, with an average deviation dev_O between −7.55% and −1.79%.

Table 7
Comparison among dev_O, dev_R and dev_C obtained by FaaS-MACrO for variable-sized networks of heterogeneous nodes, tackling the coordination problem by Algorithm 1 and considering a maximum number of 100 iterations.

$\|\mathcal{N}\|$		4	10	20	50	100	200
Alg. 1, 100	dev_O	−1.79%	−4.36%	−5.11%	−5.86%	−7.55%	−7.03%
	dev_R	17.63×	14.51×	10.18×	7.21×	2.63×	0.96×
	dev_C	1.53%	2.79%	5.07%	2.48%	3.70%	7.07%

As in the homogeneous case discussed in Section 6.3.1, the iterative process generally takes longer than the centralized model LMM to identify the final solution, particularly for small networks. The deviation is even larger in the heterogeneous case, because a problem instance with heterogeneous parameters is less likely to experience symmetries that make it harder for the centralized model to determine whether a solution is optimal. As a consequence, the time-to-solution of LMM approaches the time limit of 120 s only for the largest systems with $|\mathcal{N}| = 200$, while it remains consistently below 60 s for networks of up to 100 nodes (and below 6 s for up to 50 nodes).

It is relevant to observe that, although significantly larger than for LMM on average, as testified by the values of dev_R reported in Table 7, the maximum time-to-solution observed by FaaS-MACrO for systems of up to 100 nodes is 50.11 s, i.e., lower than the maximum value required by LMM across the same set of instances.

6.3.3. Variable number of deployed functions

Figs. 13–15 compare LMM and FaaS-MACrO applied to solve systems of $|\mathcal{N}| = 50$ heterogeneous Edge nodes characterized by a variable number $|\mathcal{F}|$ of deployed functions (under the assumption that there is a unique set of functions deployed on all nodes – scenario A.3). In particular, Fig. 13 considers the case in which the coordination problem (P3) in FaaS-MACrO is solved to optimality, while Figs. 14 and 15 refer scenarios where the greedy Algorithm 1 is applied (with 100 or 500 maximum iterations).

As in the other cases, we can observe by comparing the average values of dev_O and dev_R (reported also in Table 8), that solving (P3) to optimality guarantees significantly better performance at the price of a considerably longer execution time, particularly as the number of deployed functions increases. This, on one hand, demonstrates that

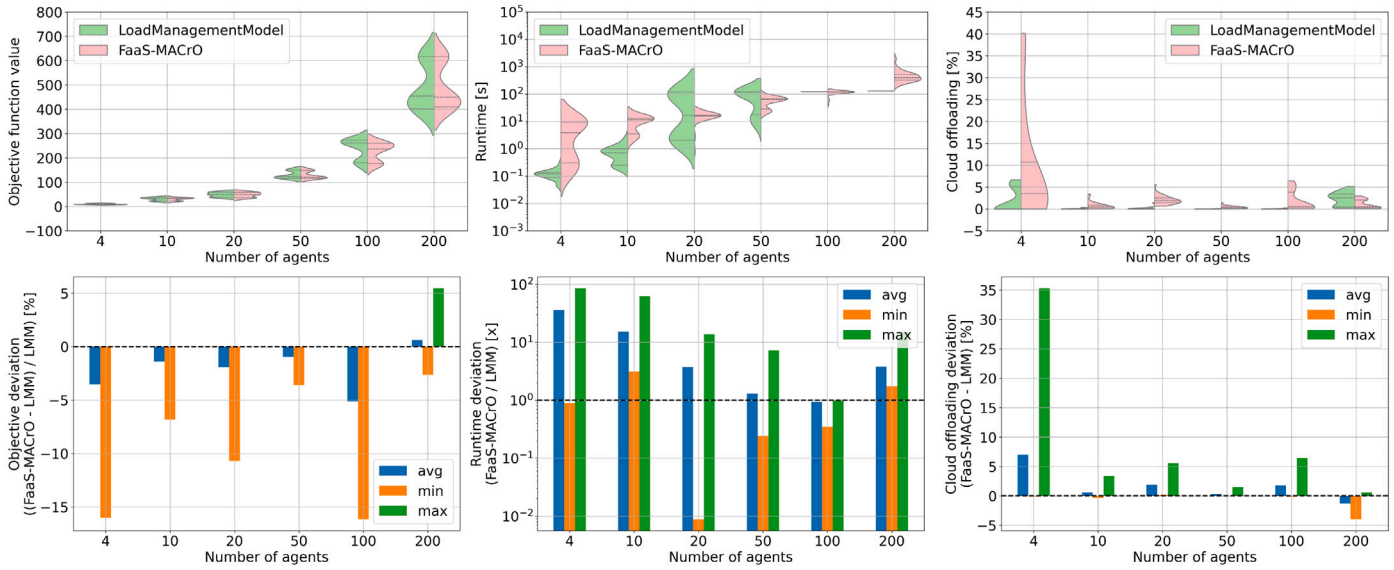


Fig. 11. Comparison between the value of objective function (P1a), the time-to-solution and the percentage rate of requests offloaded to Cloud obtained with LMM and FaaS-MACrO; the coordinator applies the greedy Algorithm 1; the maximum number of iterations is 500.

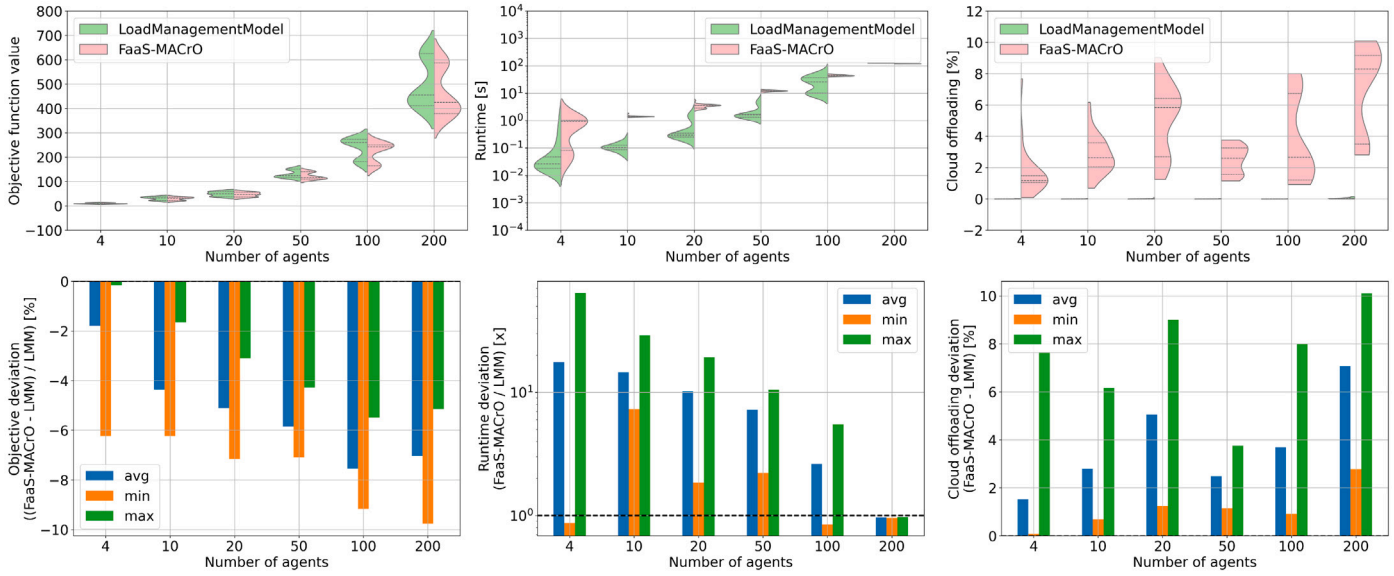


Fig. 12. Comparison between LMM and FaaS-MACrO considering networks of *heterogeneous* nodes; the coordinator applies Algorithm 1; the maximum number of iterations is 100.

FaaS-MACrO with an optimal coordinator achieves a near-globally-optimal solution even if the iterative process does not have global optimality as an explicit goal. On the other hand, an average dev_O below 8.6% in absolute value lets us conclude that considering a greedy coordination process guarantees us reasonably good performance in terms of global objective function value while reducing by up to two-thirds the time-to-solution with respect to the centralized process.

6.3.4. Light-load conditions

To further validate the FaaS-MACrO approach, we run a set of experiments considering heterogeneous nodes as in Section 6.3.2, but observing the system in light-load conditions, i.e., when the value of $\sum_f \lambda_i^f$ for each node $i \in \mathcal{N}$ is close to the rate of requests that node i can process locally without having to rely on horizontal or vertical offloading (scenario A.4).

Fig. 16 reports the comparison between LMM and the distributed approach considering the greedy coordination Algorithm 1. We can observe that the two methods reach almost identical solutions (dev_O is always lower than 0.01% in absolute value on average), but FaaS-MACrO is significantly faster, always reaching the final solution in less than 0.04 s (with an average deviation dev_R of up to 3 orders of magnitude) in large-scale networks of 200 nodes. It is relevant to observe that since at 200 nodes LMM sometimes reaches the maximum time limit, stopping before identifying the global optimum, FaaS-MACrO manages to slightly outperform it in those instances (dev_O is greater than 0). On the other hand, due to the fact that nodes are rarely overloaded, FaaS-MACrO needs only one iteration to converge, i.e., the first locally-optimal solution identified by all nodes solving problem (P2) is always globally feasible.

6.4. Different network topologies

This section reports the results of two sets of experiments meant to analyze the impact of changing the Edge nodes network topology (the

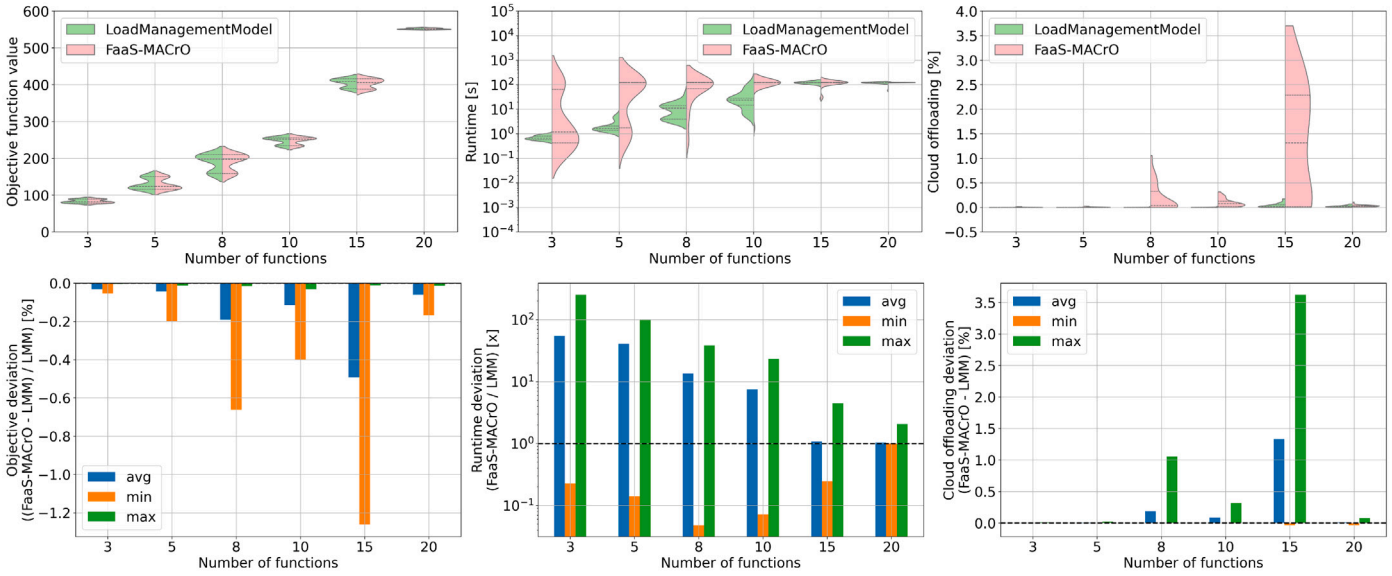


Fig. 13. Comparison between LMM and FaaS-MACrO considering $|\mathcal{N}| = 50$ heterogeneous nodes and a variable number of functions; the coordinator solves (P3); the maximum number of iterations is 100.

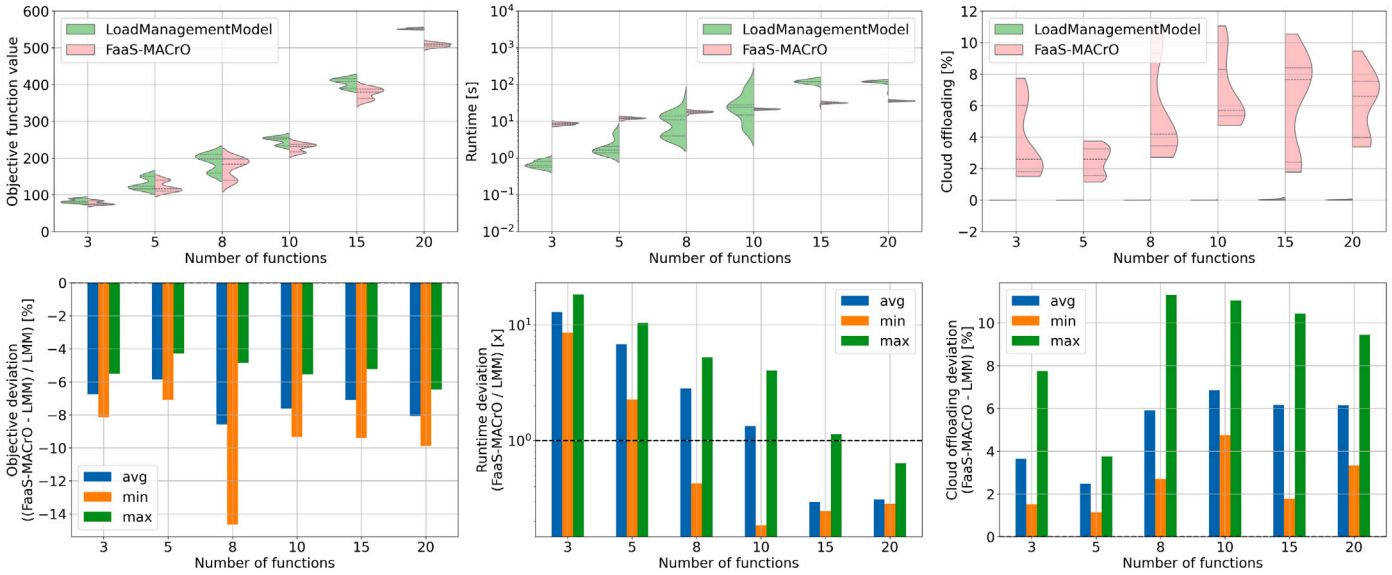


Fig. 14. Comparison between LMM and FaaS-MACrO considering $|\mathcal{N}| = 50$ heterogeneous nodes and a variable number of functions; the coordinator applies Algorithm 1; the maximum number of iterations is 100.

Table 8
Comparison among dev_O, dev_R and dev_C obtained by FaaS-MACrO considering $|\mathcal{N}| = 50$ heterogeneous nodes and a variable number of functions, when tackling the coordination problem by solving (P3) or by Algorithm 1, and considering a maximum number of 100 or 500 iterations.

$\lvert\mathcal{F}\rvert$		3	5	8	10	15	20
(P3), 100	dev_O	−0.03%	−0.04%	−0.19%	−0.12%	−0.49%	−0.06%
	dev_R	54.54×	41.13×	13.53×	7.50×	1.09×	1.05×
	dev_C	0.002%	0.005%	0.19%	0.09%	1.33%	0.01%
Alg. 1, 100	dev_O	−6.75%	−5.86%	−8.58%	−7.62%	−7.09%	−8.08%
	dev_R	12.92×	6.82×	2.83×	1.34×	0.29×	0.31×
	dev_C	3.64%	2.48%	5.91%	6.85%	6.16%	6.15%
Alg. 1, 500	dev_O	−6.33%	−5.46%	−8.50%	−7.36%	−7.00%	−8.57%
	dev_R	68.20×	35.35×	14.15×	6.87×	1.11×	1.02×
	dev_C	3.39%	2.35%	5.83%	6.44%	5.97%	6.58%

scenario **B** described in Section 6.2). In particular, we considered the same instances generated in scenario **A.2** for fully-connected networks of $|\mathcal{N}| = 200$ heterogeneous Edge nodes. In Section 6.4.1, we report the results obtained by progressively reducing the degree k of each

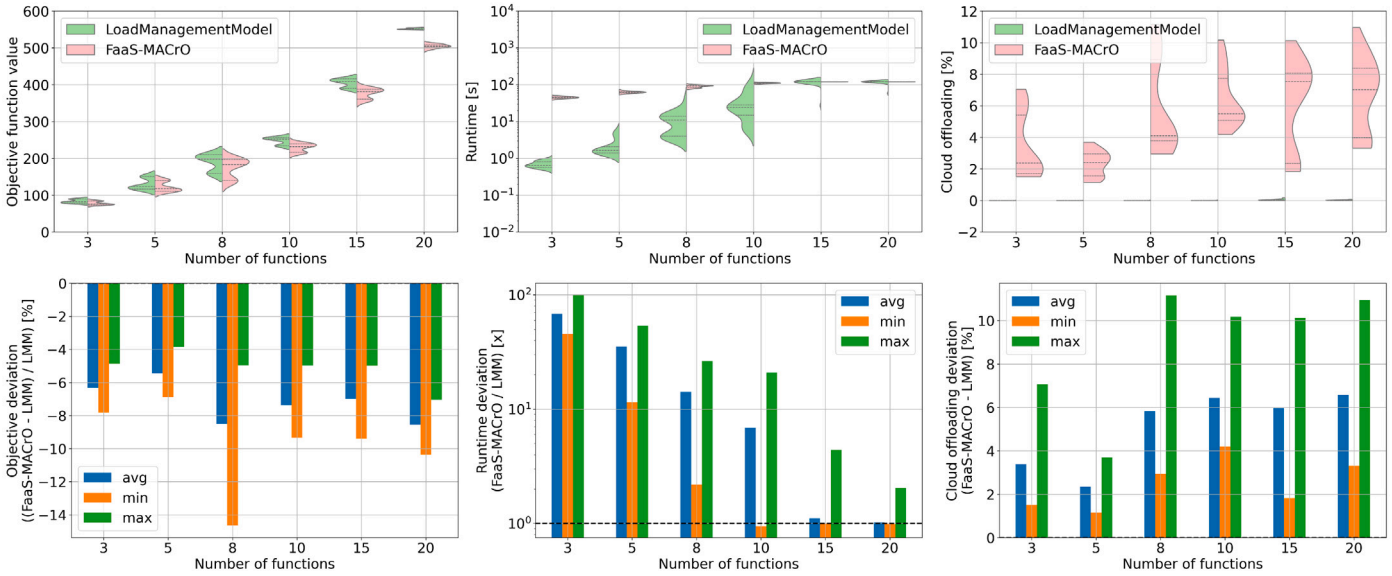


Fig. 15. Comparison between LMM and FaaS-MACrO considering $|\mathcal{N}| = 50$ heterogeneous nodes and a variable number of functions; the coordinator applies Algorithm 1; the maximum number of iterations is 500.

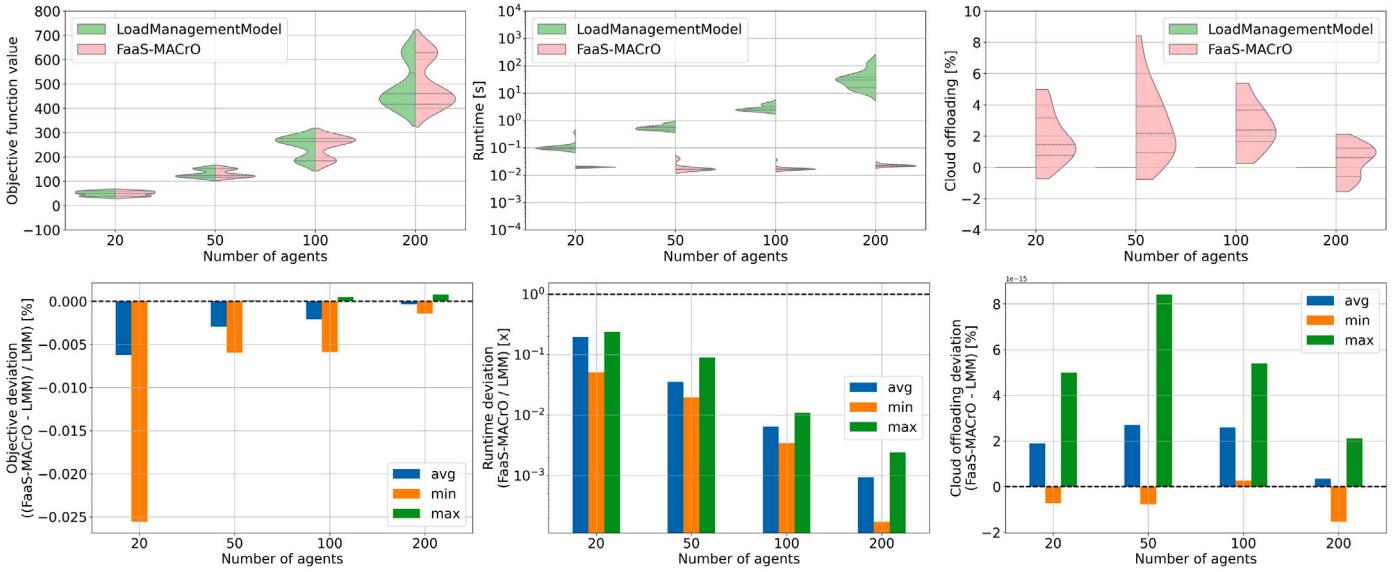


Fig. 16. Comparison between the value of objective function (P1a), the time-to-solution and the percentage rate of requests offloaded to Cloud obtained with LMM and FaaS-MACrO in *light-load* conditions; the coordinator applies Algorithm 1.

node, up to selecting a minimum number of 3 neighbors per node (scenario **B.1**). In Section 6.4.2, instead, we fixed the degree k and progressively reduced the *edge-exposed* fraction, i.e., the number of nodes connected to an access point and receiving load λ_i^f from connected clients (scenario **B.2**).

Figs. 17(a)–17(c) show a sample network of 200 Edge nodes with degree k equal to 100, 10 and 3 when eef is equal to 1.0, i.e., all nodes are connected to an access point. For $k = 10$, instead, Figs. 17(d)–17(f) report the same network while decreasing eef to 0.95, 0.75 and 0.25, respectively (red nodes are those without connections to an access point).

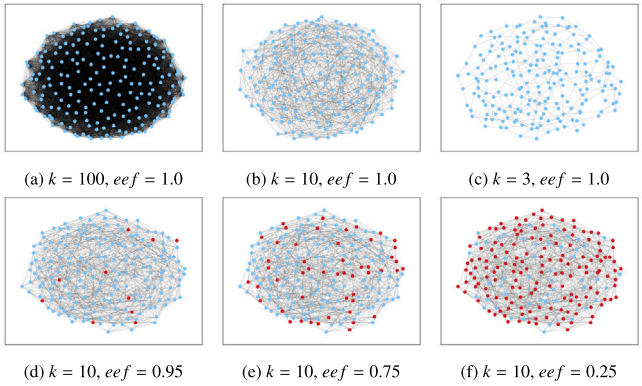
It is relevant to observe that, since the load λ_i^f directed to all the other nodes does not change while decreasing eef, the network observes a progressively lower number of incoming requests. Consequently, the load management problem becomes increasingly easier to solve.

6.4.1. Variable number of neighbors

Fig. 18 and the first three lines of Table 9 report the results obtained by LMM and FaaS-MACrO (using Algorithm 1) for different values of node degree k. Comparing the objective function deviation dev_O with the corresponding values obtained on fully-connected networks of $|\mathcal{N}| = 200$ nodes (shown in Fig. 12 and in the last column of Table 9), we observe that FaaS-MACrO maintains relatively stable performance as the node degree decreases to $k = 10$. However, the quality of the solution identified by the iterative process drops significantly in poorly-connected networks. This performance degradation stems from two interconnected phenomena. First, when fewer neighbors are available, selecting a suboptimal target for horizontal offloading has a stronger negative impact than when many alternatives exist. Second, we observe that when k is 3 or 5, the centralized LMM also relies on vertical offloading to optimally manage the incoming traffic, which, as already mentioned, creates a more challenging scenario for FaaS-MACrO.

Table 9
dev_O, dev_R and dev_C obtained by FaaS-MACrO for a network of 200 heterogeneous nodes with varying k when tackling the coordination problem by (P3) or Algorithm 1, and when considering a maximum number of 100 or 500 iterations. The last column (Fully Connected) reports for comparison the corresponding values from Table 7.

$\|k\|$		3	5	10	25	50	100	150	FC
Alg. 1, 100	dev_O	−16.43%	−13.65%	−8.05%	−6.63%	−6.99%	−7.04%	−6.98%	**−7.03%**
	dev_R	0.007×	0.009×	0.12×	0.73×	1.13×	1.46×	0.98×	**0.96×**
	dev_C	8.40%	8.93%	6.57%	6.67%	8.40%	7.08%	6.96%	**7.07%**
(P3), 100	dev_O	−12.14%	−8.89%	−0.54%	−0.05%	–	–	–	–
	dev_R	0.98×	0.98×	0.95×	2.82×	–	–	–	–
	dev_C	6.76%	6.00%	0.33%	0.02%	–	–	–	–
Alg. 1, 500	dev_O	−16.34%	−13.49%	−8.00%	−6.56%	–	–	–	–
	dev_R	0.07×	0.08×	0.66×	0.96×	–	–	–	–
	dev_C	8.36%	8.80%	6.51%	6.64%	–	–	–	–



(a) $k = 100$, $eef = 1.0$ (b) $k = 10$, $eef = 1.0$ (c) $k = 3$, $eef = 1.0$
(d) $k = 10$, $eef = 0.95$ (e) $k = 10$, $eef = 0.75$ (f) $k = 10$, $eef = 0.25$

Fig. 17. Sample network of 200 Edge nodes with different degree k and edge-exposed fraction (red nodes are those that are not connected to an access point and only receive offloaded requests from neighbors). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

A partial comparison between LMM and FaaS-MACrO when the latter solves problem (P3), with k between 3 and 25, is reported in Fig. 19. We can observe that, when the network connectivity level is reduced, choosing the most appropriate target for offloading becomes increasingly relevant. As a consequence, the objective function deviation becomes significantly smaller in absolute value when the coordinator solves (P3) to optimality instead of applying the greedy Algorithm 1, as testified by the average values of dev_O reported in Table 9. On the other hand, the time-to-solution is considerably longer when the coordinator solves (P3), approaching the same time limit reached by LMM already when k is between 3 and 10.

Similarly, Fig. 20 and the last three lines of Table 9 report the comparison between LMM and FaaS-MACrO when applying Algorithm 1 but running the iterative process for a maximum of 500 iterations, considering k between 3 and 25. The reported results highlight how, in this context, increasing the number of iterations does not have a significant benefit on the FaaS-MACrO performance.

6.4.2. Edge-exposed fraction

As mentioned, we analyzed in this section the impact of considering some *intermediate* nodes in the network, which are not connected to an access point (and therefore do not receive any load from clients) but have computational capacity to share with neighbors to help them process the incoming requests.

In particular, we have analyzed networks of $|\mathcal{N}| = 200$ nodes with different levels of connectivity, from fully-connected networks to networks with $k = 100$ or $k = 10$, and progressively varying the edge-exposed fraction (eef). The results are reported in Figs. 21, 22 and 23, respectively.

It is relevant to note that in all three scenarios we have tackled the coordination problem by applying Algorithm 1 and by running a maximum number of 100 iterations. Moreover, we must observe that we slightly varied the expression of the objective function (P1a) we evaluate here: indeed, having some *intermediate* nodes with $\lambda_i^f = 0 \; \forall f \in \mathcal{F}$, we had to apply a different normalization process by considering $\sum_{i,f} \lambda_i^f$ instead of simply λ_i^f as a denominator. This is the reason why in this section the objective function values reported in the top-left corner of each figure assume values in $[0, 1]$.

For highly-connected networks (Figs. 21 and 22), we observe that FaaS-MACrO achieves optimal performance in terms of objective function value when $eef \leq 90\%$, yielding an average deviation dev_O below 0.003% in absolute value. At the same time, FaaS-MACrO proves to have very good performance in terms of time-to-solution in this context, always reaching a globally feasible solution in less than 0.32 s, with a minimum value of 0.02 s, while LMM takes between 5.2 s and 48.6 s.

On the other hand, the problem becomes more challenging when eef increases to 95% and 99.5% (i.e., when 190 or 199 out of the 200 nodes in the network are connected to an access point). In these scenarios, we observe that the deviation between FaaS-MACrO and LMM in terms of objective function value increases, reaching almost −4% on average.

Fig. 23 highlights the additional challenges introduced in this context when the network is poorly connected (i.e., each node has only 10 neighbors). Indeed, reducing the nodes degree entails that, even if some *intermediate* nodes are present, not all the nodes will be connected to them and able to take advantage of the additional processing capacity they offer. In this scenario we observe therefore a higher dev_O value, up to −2.3%, even when eef is 75% or 90%. On the other hand, we can note that the reduced connectivity does not have a negative impact on FaaS-MACrO performance when eef increases, with dev_O still remaining above −4% on average in all experiments.

6.5. Convergence dynamics and time-bounded performance

Due to the bursty nature of FaaS workloads and the fact that function requests are usually short-lived and characterized by tight response time constraints, some applications may require the control time period t to be reduced, making a time-to-solution of 120 s hardly feasible in practice.

To account for this need, we have analyzed how the solution quality improves over time, both for LMM and throughout the iterative process of FaaS-MACrO, to assess the impact of possibly imposing a stricter time limit. In particular, we have considered sample problem instances featuring either networks of $|\mathcal{N}| = 100$ homogeneous Edge nodes (as those analyzed in scenario **A.1**) or $|\mathcal{N}| = 200$ heterogeneous nodes (as those from scenario **A.2**). For LMM, we have solved multiple times each problem instance, forcing Gurobi Optimizer to stop at the first feasible solution, the second feasible solution, or after a maximum execution time of 30 s, 60 s, 120 s and 1 h. Similarly, we have recorded the value

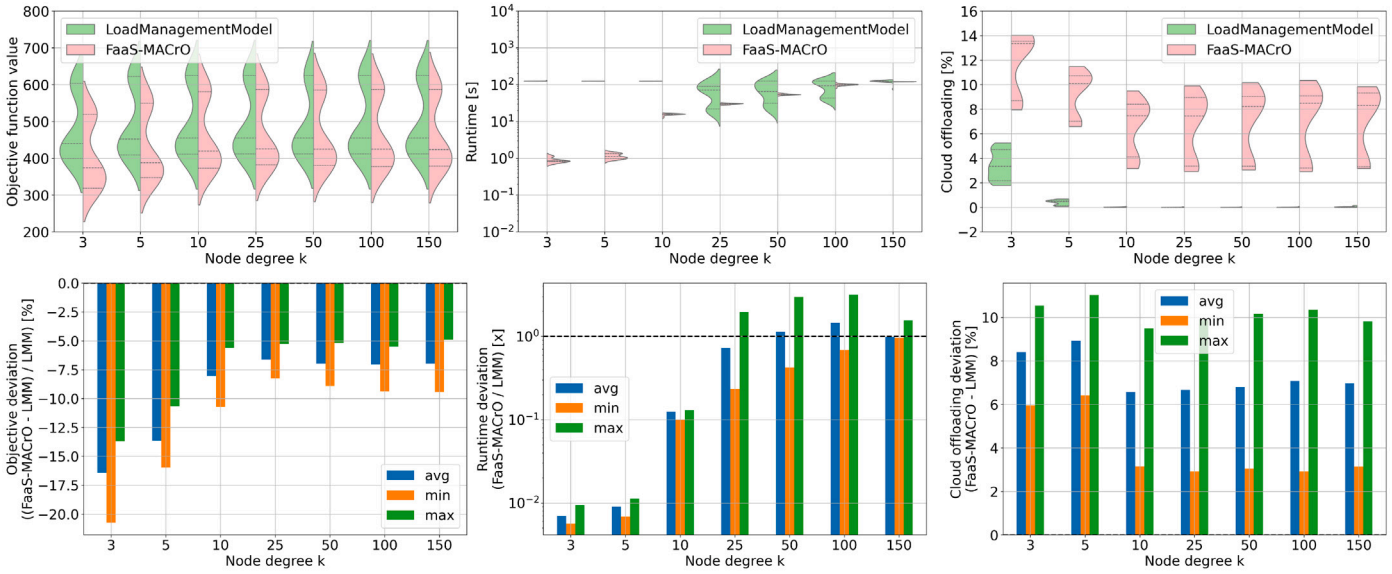


Fig. 18. Comparison between LMM and FaaS-MACrO for a network of 200 nodes with varying k; the coordinator applies Algorithm 1; the maximum number of iterations is 100.

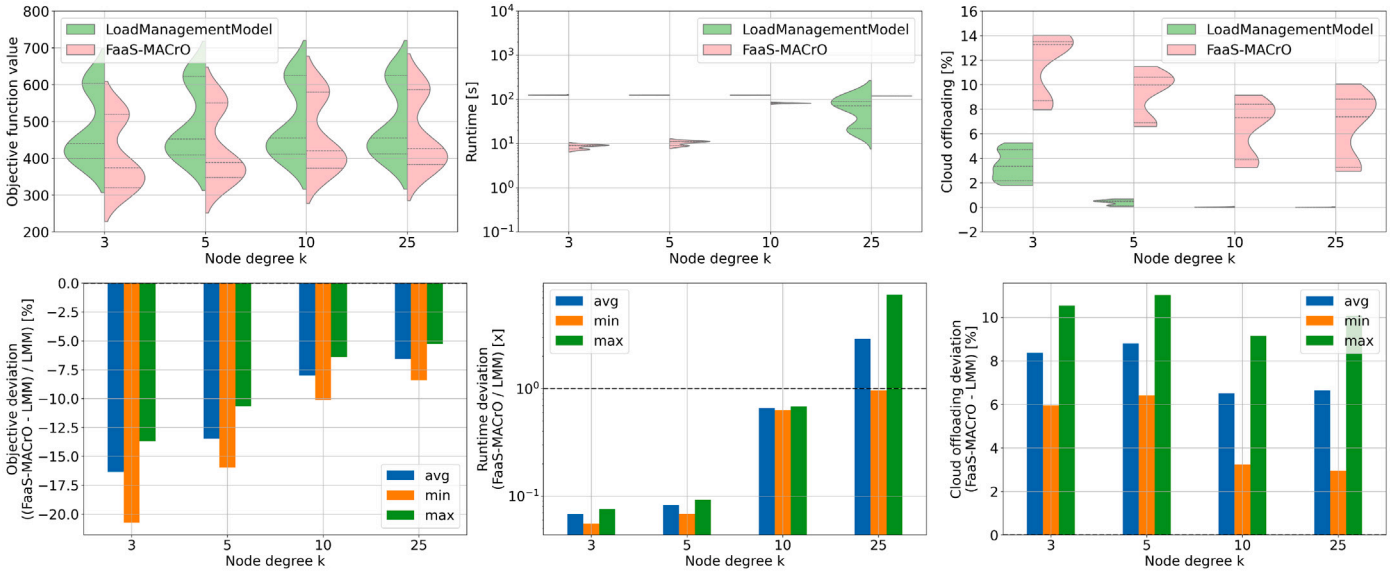


Fig. 19. Comparison between LMM and FaaS-MACrO for a network of 200 nodes with varying k; the coordinator solves (P3); the maximum number of iterations is 100.

of the objective function (P1a) for the intermediate solutions identified by FaaS-MACrO during the iterative process.

The results for LMM are reported in Table 10. In particular, for both the homogeneous and heterogeneous nodes scenarios the table reports the minimum, maximum and average value of the actual time-to-solution, and the minimum, maximum and average objective function value when the solver stops after identifying the first or second feasible solution, or considering different time limits.

In the homogeneous case, as discussed in Section 6.3.1, LMM does not identify the optimal solution within the original time limit of 120 s, even if this is actually found in the 50% of cases in less than 1 h (the asterisks in the last column mark scenarios where the actual time-to-solution is lower than the time limit). However, the objective function value remains essentially stable once hitting the 30 s mark.

The same pattern is observed in the heterogeneous case, however the optimal solution is always found in less than 1 h because heterogeneity reduces symmetries in the solution structure. We can thus conclude that a time limit of 30 s is enough to obtain a good-quality centralized solution, with an average deviation of −0.12% with respect to the global optimum.

To analyze the convergence dynamics of FaaS-MACrO, we kept track, throughout the iterative process, of every time the new candidate solution improves the previous recorded one in terms of (P1a) value. The corresponding values of dev_O for the same homogeneous and heterogeneous cases analyzed with LMM are reported as blue dots in Fig. 24.

Then, we have divided the maximum runtime of 120 s in 4 intervals, setting intermediate time limits at 10 s, 30 s and 60 s (the vertical dashed lines in Fig. 24), and we have computed the minimum, maximum and average values of dev_O in each subinterval. The results are reported in Table 11, while the average values are also marked as red stars in Fig. 24. Note that, in Table 11, the "maximum" deviation is intended as the most negative value of dev_O.

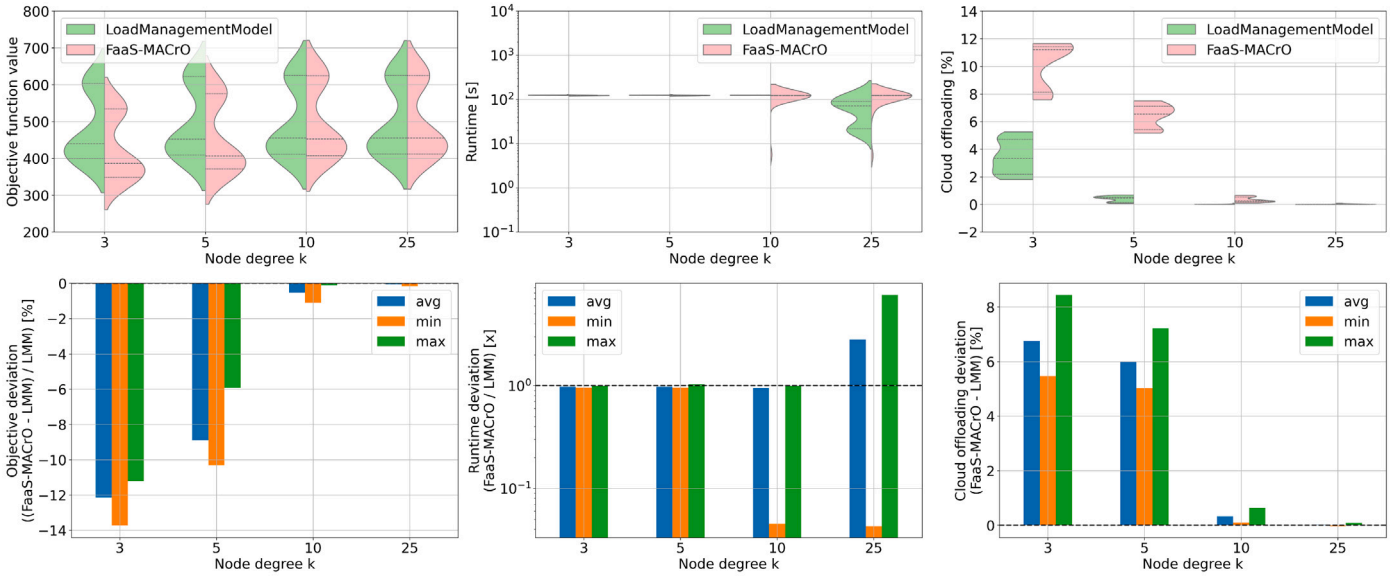


Fig. 20. Comparison between LMM and FaaS-MACrO for a network of 200 nodes with varying k; the coordinator applies Algorithm 1; the maximum number of iterations is 500.

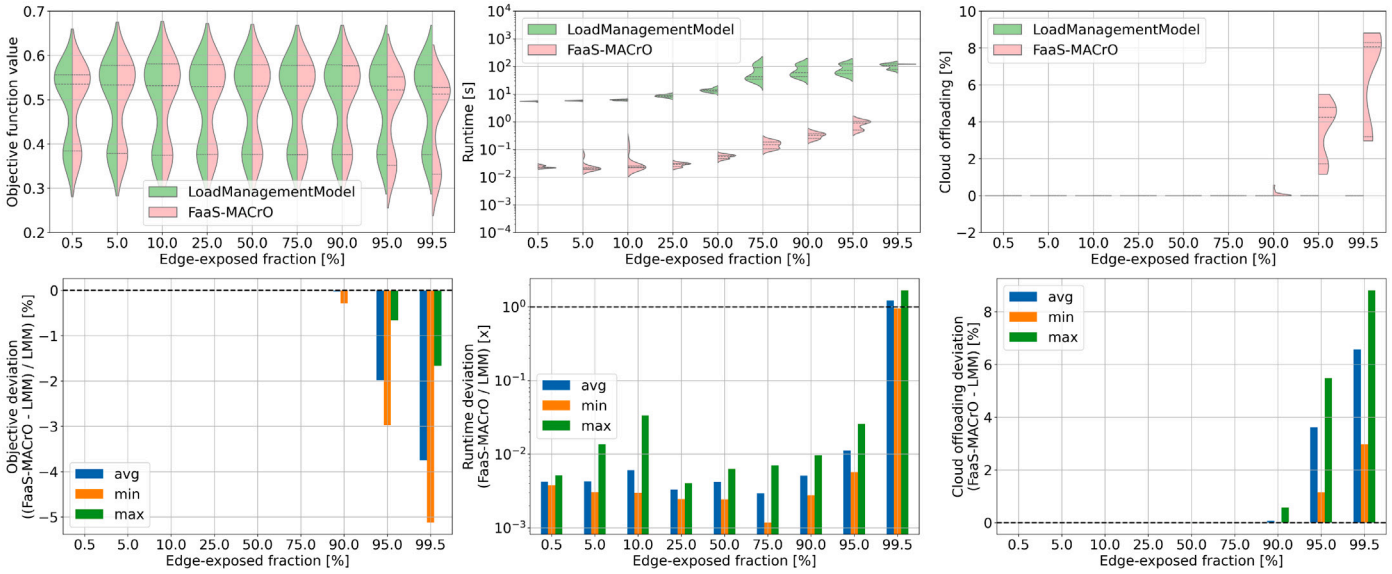


Fig. 21. Comparison between LMM and FaaS-MACrO for a fully-connected network of 200 nodes, varying the edge-exposed fraction eef; the coordinator applies Algorithm 1; the maximum number of iterations is 100.

Table 10
Runtime and objective function value yielded by LMM when stopping at the first or second feasible solution, or considering an increasing time limit.

Scenario			1st feas. sol.	2nd feas. sol	30 s	60 s	120 s	3600 s
Homogeneous nodes	Runtime	min	2.20 s	3.10 s	32.19 s	62.28 s	122.21 s	419.35* s
		max	2.35 s	3.32 s	32.64 s	62.67 s	122.99 s	3603.43 s
		avg	2.27 s	3.20 s	32.41 s	62.40 s	122.47 s	2778.63* s
	Obj. value	min	−293.67	136.55	179.11	180.75	180.75	180.77
		max	−289.33	197.86	272.23	272.54	272.59	272.61
		avg	−290.90	168.99	236.77	237.81	238.07	238.10
Heterogeneous nodes	Runtime	min	5.30 s	6.86 s	35.63 s	65.36 s	125.37 s	217.53* s
		max	5.82 s	7.19 s	35.99 s	65.74 s	125.80 s	866.37* s
		avg	5.55 s	7.02 s	35.82 s	65.59 s	125.50 s	544.55* s
	Obj. value	min	−579.75	−579.75	410.00	411.65	411.66	411.71
		max	−565.71	−565.71	625.02	625.35	625.37	625.41
		avg	−574.14	−574.14	487.70	488.18	488.24	488.28

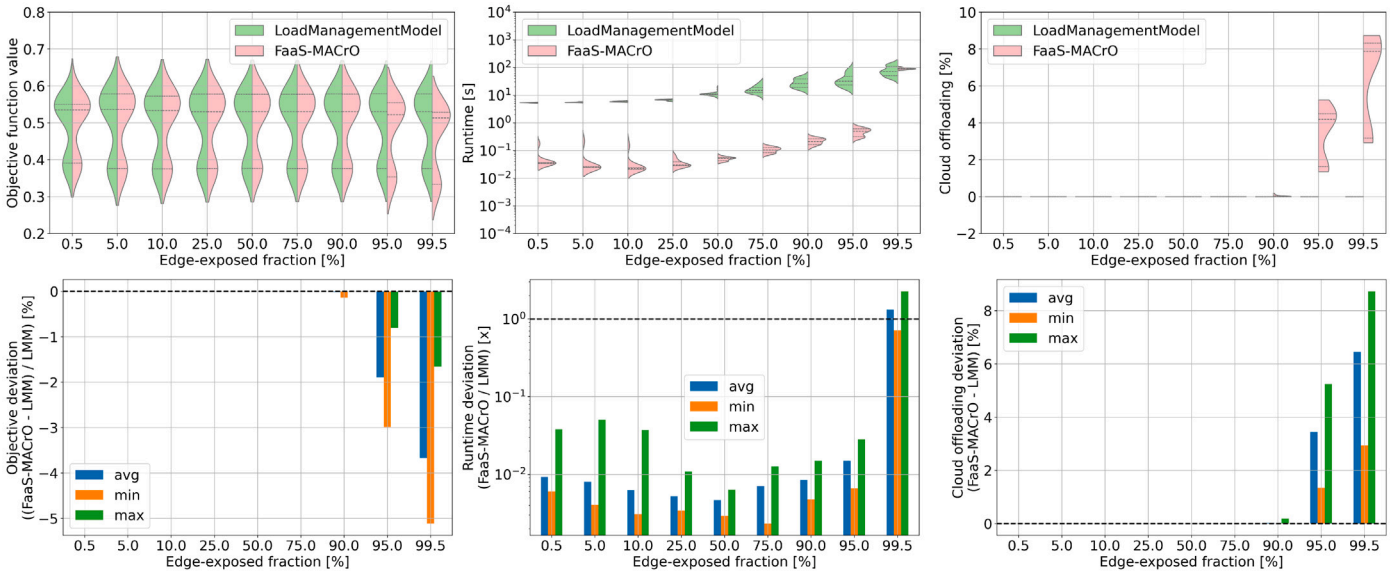


Fig. 22. Comparison between LMM and FaaS-MACrO for a network of 200 nodes with $k = 100$, varying the edge-exposed fraction eef; the coordinator applies Algorithm 1; the maximum number of iterations is 100.

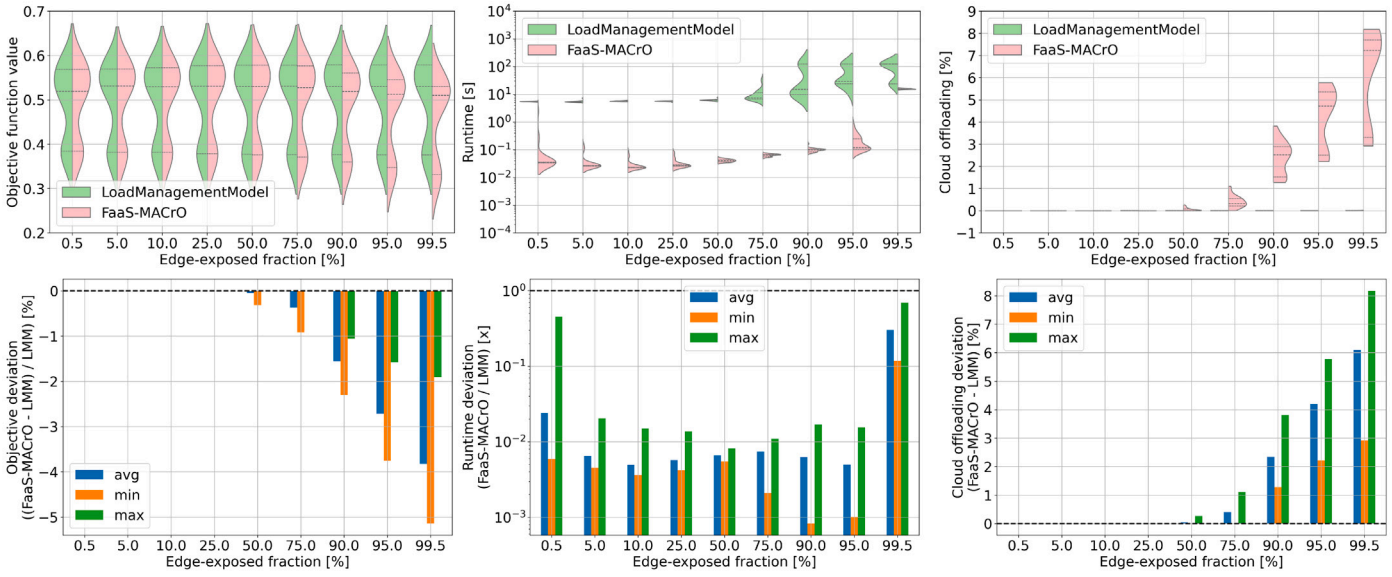


Fig. 23. Comparison between LMM and FaaS-MACrO for a network of 200 nodes with $k = 10$, varying the edge-exposed fraction eef; the coordinator applies Algorithm 1; the maximum number of iterations is 100.

For the homogeneous case, we can observe that the last objective function values are recorded between 60 s and 80 s, thus significantly before the maximum time limit. Moreover, the value of dev_O steadily improves over time, increasing from the −10.26% recorded within the first 10 s to −2.06% on average.

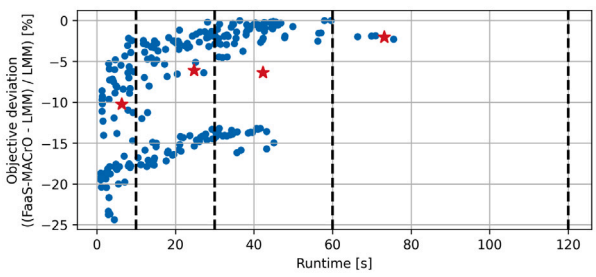
On the other hand, in the heterogeneous case we note that the performance of FaaS-MACrO are more stable, with the average value of dev_O increasing only marginally after the 30 s milestone. Hence, as for LMM we can conclude that terminating the iterative process earlier would not negatively impact the solution quality, while being preferable in highly dynamic settings where a shorter control period should be considered.

Table 11
Runtime and objective function deviation yielded by FaaS-MACrO over time.

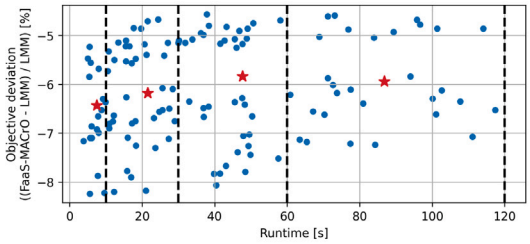
Scenario			10 s	30 s	60 s	120 s
Homogeneous nodes	Runtime	min	1.35	11.86 s	30.30 s	70.95 s
		max	9.26 s	29.99 s	59.58 s	75.53 s
		avg	6.40 s	24.74 s	42.33 s	73.24 s
	dev_O	min	−2.18%	−0.18%	0.01%	−1.86%
		max	−20.41%	−14.47%	−14.95%	−2.27%
		avg	−10.26%	−6.11%	−6.37%	−2.06%
Heterogeneous nodes	Runtime	min	5.02 s	11.91 s	31.82 s	60.84 s
		max	9.86 s	29.97 s	58.07 s	117.37 s
		avg	7.53 s	21.53 s	47.71 s	86.84 s
	dev_O	min	−5.23%	−4.67%	−4.56%	−4.59%
		max	−8.23%	−8.18%	−7.51%	−7.21%
		avg	−6.43%	−6.18%	−5.83%	−5.94%

7. Conclusions

This work formalized the Function Replica Allocation and Load Balancing (FRALB) problem in the Edge–Cloud continuum and addressed



(a) Network of $|\mathcal{N}| = 100$ homogeneous nodes



(b) Network of $|\mathcal{N}| = 200$ heterogeneous nodes

Fig. 24. Progressive values of dev_O over time throughout the FaaS-MACrO iterative process. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

it as a Distributed orchestration problem, termed DiFRALB. To solve it, we introduced FaaS-MACrO (Multi-Agent Coordinated Orchestration), a scalable privacy-preserving architecture designed to coordinate FaaS workloads across heterogeneous and geo-distributed cooperative Edge nodes. This work fills a gap highlighted by our review: distributed FaaS orchestration remains extremely limited, and existing proposals rarely combine scalability, privacy-preserving coordination and quantifiable near-optimality, or leave only very limited autonomy to local agents.

FaaS-MACrO distributes decision-making across Edge nodes by treating each one as an autonomous agent, while a lightweight coordinator ensures global feasibility through iterative updates of the offloading prices. This mechanism aligns self-interested local decisions with global objectives without requiring agents to disclose private information, thereby preserving privacy, reducing communication overhead, and supporting environments with multiple stakeholders. As an optimization benchmark, we derived a centralized MILP formulation that jointly optimizes replica allocation and offloading decisions under memory and utilization constraints, providing an upper bound on achievable global performance. To avoid the coordinator becoming a bottleneck at scale, we introduced a greedy coordination algorithm that preserves solution quality while substantially reducing computation time.

Extensive experiments confirmed the effectiveness and scalability of FaaS-MACrO. Solutions remained within 0.03–12.14% of the centralized optimum on average in absolute value, while reducing computation time by up to three orders of magnitude in deployments with 200 nodes. Moreover, we demonstrated scalability up to 200 nodes and 50 functions, well beyond the scale considered in most existing works.

Several simplifying assumptions limit the scope of this study. We optimize decisions over single control periods using average workloads and deterministic service demands, and we restrict horizontal offloading to one hop. Moreover, while near-optimality is empirically observed, formal convergence and optimality guarantees remain open.

In future work, we will consider extending the Faas-MACrO architecture to embrace multi-period and stochastic formulations, richer offloading models, and mechanisms for overloaded regimes (e.g., load shedding and prioritization). Moreover, we will consider intelligent mechanisms to exploit potential correlations between functions to optimize latency through locality awareness, and to adapt to highly-bursty conditions. Finally, removing the coordinator is a natural step toward fully decentralized orchestration, raising challenges in convergence, fairness, and incentive compatibility under strategic or adversarial behavior, while learning-based price adjustment and energy-aware objectives represent promising directions for real deployments.

CRediT authorship contribution statement

Federica Filippini: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Marin Lujak:** Writing – review & editing, Visualization, Validation, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Michele Ciavotta:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Funding

The work of Marin Lujak is supported by the grant VAE: TED2021-131295B- C33 funded by MCIN/AEI/10.13039/501100011033 and by the "European Union NextGenerationEU/PRTR", by the grant COSASS: PID2021-123673OB-C32 funded by MCIN/AEI/10.13039/501100011033 and by the project grant EVASAI: PID2024-158227NB-C32 funded by MICIU/AEI/10.13039/501100011033 /FEDER, UE.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Ghorbian, M. Ghobaei-Arani, L. Esmaeili, A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends, Clust. Comput. 27 (5) (2024) 5571–5610.
[2] C. Calavaro, V. Cardellini, F. Lo Presti, G. Russo Russo, Beyond cloud: Serverless functions in the compute continuum, SN Comput. Sci. 6 (3) (2025) 194.
[3] M. Ghorbian, M. Ghobaei-Arani, R. Asadolahpour-Karimi, Function placement approaches in serverless computing: A survey, J. Syst. Archit. 157 (2024) 103291.
[4] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N.J. Yadwadkar, R.A. Popa, J.E. Gonzalez, I. Stoica, D.A. Patterson, What serverless computing is and should become: The next phase of cloud computing, Commun. ACM 64 (5) (2021) 76–84.
[5] M.S. Aslanpour, A.N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S.S. Gill, R. Gaire, S. Dustdar, Serverless edge computing: vision and challenges, in: Proceedings of the 2021 Australasian Computer Science Week Multiconference, 2021, pp. 1–10.
[6] R. Xie, Q. Tang, S. Qiao, H. Zhu, F.R. Yu, T. Huang, When serverless computing meets edge computing: Architecture, challenges, and open issues, IEEE Wirel. Commun. 28 (5) (2021) 126–133.
[7] S. Smolka, L. Wissenberg, Z.A. Mann, EdgeDecAp: An auction-based decentralized algorithm for optimizing application placement in edge computing, J. Parallel Distrib. Comput. 175 (2023) 22–36.
[8] M. Nardelli, G. Russo Russo, V. Cardellini, Compute continuum: What Lies ahead? in: European Conference on Parallel Processing, Springer, 2023, pp. 5–17.
[9] C. Puliafito, O. Rana, L.F. Bittencourt, H. Wu, Serverless computing in the cloud-to-edge continuum, Future Gener. Comput. Syst. 161 (2024) 514–517.
[10] A. Hall, U. Ramachandran, An execution model for serverless functions at the edge, in: Proceedings of the International Conference on Internet of Things Design and Implementation, 2019, pp. 225–236.
[11] P. Gackstatter, P.A. Frangoudis, S. Dustdar, Pushing serverless to the edge with webassembly runtimes, in: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing, CCGrid, IEEE, 2022, pp. 140–149.
[12] G. Russo Russo, T. Mannucci, V. Cardellini, F. Lo Presti, Serverledge: Decentralized function-as-a-service for the edge-cloud continuum, in: 2023 IEEE International Conference on Pervasive Computing and Communications, PerCom, IEEE, 2023, pp. 131–140.
[13] M. Ciavotta, D. Motterlini, M. Savi, A. Tundo, DFaaS: Decentralized function-as-a-service for federated edge computing, in: 2021 IEEE 10th International Conference on Cloud Networking, CloudNet, IEEE, 2021, pp. 1–4.
[14] M. Malekabbasi, T. Pfandzelter, T. Schirmer, D. Bermbach, Geofaas: An edge-to-cloud faas platform, in: 2024 IEEE International Conference on Cloud Engineering, IC2E, IEEE, 2024, pp. 66–71.

[15] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, M. Guo, The serverless computing survey: A technical primer for design architecture, ACM Comput. Surv. 54 (10s) (2022) 1–34.

[16] Y. Li, Y. Lin, Y. Wang, K. Ye, C. Xu, Serverless computing: state-of-the-art, challenges and opportunities, IEEE Trans. Serv. Comput. 16 (2) (2022) 1522–1539.

[17] A. Mampage, S. Karunasekera, R. Buyya, A holistic view on resource management in serverless computing environments: Taxonomy and future directions, ACM Comput. Surv. 54 (11s) (2022) 1–36.

[18] Z. Li, R. Chard, Y. Babuji, B. Galewsky, T.J. Skluzacek, K. Nagaitsev, A. Woodard, B. Blaiszik, J. Bryan, D.S. Katz, et al., funcX: Federated function as a service for science, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 4948–4963.

[19] G. De Palma, S. Giallorenzo, J. Mauro, M. Trentin, G. Zavattaro, Funless: Functions-as-a-service for private edge cloud systems, in: 2024 IEEE International Conference on Web Services, ICWS, IEEE, 2024, pp. 961–967.

[20] A. Das, S. Imai, S. Patterson, M.P. Wittie, Performance optimization for edge-cloud serverless platforms via dynamic task placement, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID, IEEE, 2020, pp. 41–50.

[21] O. Ascigil, A.G. Tasiopoulos, T.K. Phan, V. Sourlas, I. Psaras, G. Pavlou, Resource provisioning and allocation in function-as-a-service edge-clouds, IEEE Trans. Serv. Comput. 15 (4) (2021) 2410–2424.

[22] T. Elgamal, A. Sandur, K. Nahrstedt, G. Agha, Costless: Optimizing cost of serverless computing through function fusion and placement, in: 2018 IEEE/ACM Symposium on Edge Computing, SEC, IEEE, 2018, pp. 300–312.

[23] X. Yao, N. Chen, X. Yuan, P. Ou, Performance optimization of serverless edge computing function offloading based on deep reinforcement learning, Future Gener. Comput. Syst. 139 (2023) 74–86.

[24] F. Tütüncüoğlu, G. Dán, Joint resource management and pricing for task offloading in serverless edge computing, IEEE Trans. Mob. Comput. 23 (6) (2023) 7438–7452.

[25] K. Li, S. Nastic, Attentionfunc: balancing FaaS compute across edge-cloud continuum with reinforcement learning, in: Proceedings of the 13th International Conference on the Internet of Things, 2023, pp. 25–32.

[26] J.J. Lee, J. Nava, H. Lee, ComFaaS distributed: Edge computing with function-as-a-service in parallel cloud environments, in: 2024 7th International Conference on Information and Computer Technologies, ICICT, IEEE, 2024, pp. 133–138.

[27] P. Wu, H. Chen, T. Wu, K. Gu, Y. Xia, Cold-start-aware offloading and resource allocation by importance sampling-based double dueling DQN in serverless edge computing, IEEE Internet Things J. 12 (19) (2025) 40190–40205, http://dx.doi.org/10.1109/JIOT.2025.3588727.

[28] A. Palade, A. Mukhopadhyay, A. Kazmi, C. Cabrera, E. Nomayo, G. Iosifidis, M. Ruffini, S. Clarke, A swarm-based approach for function placement in federated edges, in: 2020 IEEE International Conference on Services Computing, SCC, IEEE, 2020, pp. 48–50.

[29] B. Wang, A. Ali-Eldin, P. Shenoy, Lass: Running latency sensitive serverless computations at the edge, in: Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing, 2021, pp. 239–251.

[30] P. Wang, Y. Li, C. Fang, Y. Zhong, Z. Ding, Optimizing serverless performance through game theory and efficient resource scheduling, IEEE Trans. Comput. (2025).

[31] S. Deng, H. Zhao, Z. Xiang, C. Zhang, R. Jiang, Y. Li, J. Yin, S. Dustdar, A.Y. Zomaya, Dependent function embedding for distributed serverless edge computing, IEEE Trans. Parallel Distrib. Syst. 33 (10) (2021) 2346–2357.

[32] C. Chen, M. Herrera, G. Zheng, L. Xia, Z. Ling, J. Wang, Cross-edge orchestration of serverless functions with probabilistic caching, IEEE Trans. Serv. Comput. 17 (5) (2024) 2139–2150.

[33] G. Russo Russo, E. D'Alessandro, V. Cardellini, F.L. Presti, Towards a multi-armed bandit approach for adaptive load balancing in function-as-a-service systems, in: 2024 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion, ACSOS-C, 2024, pp. 103–108, http://dx.doi.org/10.1109/ACSOS-C63493.2024.00039.

[34] M.S. Aslanpour, A.N. Toosi, M.A. Cheema, M.B. Chhetri, M.A. Salehi, Load balancing for heterogeneous serverless edge computing: A performance-driven and empirical approach, Future Gener. Comput. Syst. 154 (2024) 266–280, http://dx.doi.org/10.1016/j.future.2024.01.020.

[35] F. Righetti, B. Cornacchia, G. Russo Russo, N. Tonellotto, V. Cardellini, C. Vallati, Energy-efficient function invocation scheduling for edge FaaS platforms , in: 2025 IEEE International Conference on Smart Computing, SMARTCOMP, IEEE Computer Society, Los Alamitos, CA, USA, 2025, pp. 18–25, http://dx.doi.org/10.1109/SMARTCOMP65954.2025.00057.

[36] L. Baresi, D.Y.X. Hu, G. Quattrocchi, L. Terracciano, NEPTUNE: A comprehensive framework for managing serverless functions at the edge, ACM Trans. Auton. Adapt. Syst. 19 (1) (2024) 1–32.

[37] C. Cicconetti, M. Conti, A. Passarella, A decentralized framework for serverless edge computing in the internet of things, IEEE Trans. Netw. Serv. Manag. 18 (2) (2020) 2166–2180.

[38] G. Proietti Mattia, R. Beraldi, P2PFaaS: A framework for faas peer-to-peer scheduling and load balancing in fog and edge computing, SoftwareX 21 (2023) 101290.

[39] C. Gonçalves, J. Simão, L. Veiga, A function-as-a-service middleware for decentralized collaborative edge computing, Future Gener. Comput. Syst. (2025) 108069.

[40] E. Carlini, P. Dazzi, L. Ferrucci, J. Massa, M. Mordacchini, Dynamic workload balancing in decentralized edge systems: A marginal cost approach, Future Gener. Comput. Syst. (2025) 108167.

[41] X. Chen, M.P. Paidiparthy, D. Da Silva, L. Hu, Ekko: Fully decentralized scheduling for serverless edge computing, in: 2025 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2025, pp. 15–27.

[42] A.K. Singh, S. Kumar, S. Jain, A multi-agent deep reinforcement learning approach for optimal resource management in serverless computing, Clust. Comput. 28 (2) (2025) 102.

[43] S.H. Mahdizadeh, S. Abrishami, An assignment mechanism for workflow scheduling in function as a service edge environment, Future Gener. Comput. Syst. 157 (2024) 543–557.

[44] R. Farahani, N. Mehran, S. Ristov, R. Prodan, Heftless: A bi-objective serverless workflow batch orchestration on the computing continuum, in: 2024 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2024, pp. 286–296.

[45] R. Farahani, R. Prodan, EnergyLess: An energy-aware serverless workflow batch orchestration on the computing continuum, in: 2025 IEEE 18th International Conference on Cloud Computing, CLOUD, IEEE, 2025, pp. 243–254.

[46] Y. Peng, M. Zhang, Z. Zhou, H. Huang, Joint container orchestrating and request routing for serverless edge computing-based simulation applications, J. Netw. Comput. Appl. 243 (2025) 104284, http://dx.doi.org/10.1016/j.jnca.2025.104284.

[47] L. Liu, H. Tan, S.H.-C. Jiang, Z. Han, X.-Y. Li, H. Huang, Dependent task placement and scheduling with function configuration in edge computing, in: Proceedings of the International Symposium on Quality of Service, 2019, pp. 1–10.

[48] A. Peri, M. Tsenos, V. Kalogeraki, Orchestrating the execution of serverless functions in hybrid clouds, in: 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS, IEEE, 2023, pp. 139–144.

[49] P. Raith, T. Rausch, S. Dustdar, F. Rossi, V. Cardellini, R. Ranjan, Mobility-aware serverless function adaptations across the edge-cloud continuum, in: 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing, UCC, IEEE, 2022, pp. 123–132.

[50] G. Russo Russo, D. Ferrarelli, D. Pasquali, V. Cardellini, F. Lo Presti, QoS-aware offloading policies for serverless functions in the Cloud-to-Edge continuum, Future Gener. Comput. Syst. 156 (2024) 1–15, http://dx.doi.org/10.1016/j.future.2024.02.019.

[51] G. Russo Russo, P. Spaziani, V. Cardellini, Towards QoS-aware serverless function offloading in the edge-cloud continuum through reinforcement learning, in: 2025 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, 2025, pp. 1073–1080.

[52] P. Daniëlse, H.-L. Tai, S. Ilager, Z. Zhao, Investigating efficient edge offloading architectures for serverless systems, in: 2025 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, 2025, pp. 1035–1038.

[53] E. Petriglia, F. Filippini, G. Pracucci, M. Savi, M. Ciavotta, Comparing actor-critic and neuroevolution approaches for traffic offloading in FaaS-powered edge systems, in: Proceedings of the 1st Workshop on Serverless at the Edge, 2024, pp. 17–24.

[54] E. Petriglia, F. Filippini, M. Ciavotta, M. Savi, Multi-agent reinforcement learning for workload distribution in faas-edge computing systems, in: 2025 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, 2025, pp. 1128–1131.

[55] S. Giordani, M. Lujak, F. Martinelli, A distributed multi-agent production planning and scheduling framework for mobile robots, Comput. Ind. Eng. 64 (1) (2013) 19–30, http://dx.doi.org/10.1016/j.cie.2012.09.004.

[56] R. Sala, B. Guindani, E. Galimberti, F. Filippini, H. Sedghani, D. Ardagna, S. Risco, G. Moltó, M. Caballer, OSCAR-P and aMLLibrary: Profiling and predicting the performance of FaaS-based applications in computing continua, J. Syst. Softw. 221 (2025) 112282, http://dx.doi.org/10.1016/j.jss.2024.112282.

[57] A. Tundo, F. Filippini, F. Regonesi, M. Ciavotta, M. Savi, Decentralized edge workload forecasting with gossip learning, IEEE Trans. Netw. Serv. Manag. 22 (4) (2025) 3016–3031, http://dx.doi.org/10.1109/TNSM.2025.3570450.

[58] D. Ardagna, S. Casolari, M. Colajanni, B. Panicucci, Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems, J. Parallel Distrib. Comput. 72 (6) (2012) 796–808, http://dx.doi.org/10.1016/j.jpdc.2012.02.014.

[59] M. Shahrad, R. Fonseca, Í.n. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, R. Bianchini, Serverless in the wild: characterizing and optimizing the serverless workload at a large cloud provider, in: Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference, in: USENIX ATC'20, USENIX Association, USA, 2020, pp. 204–218.

[60] A.W. Kambale, H. Sedghani, F. Filippini, G. Verticale, D. Ardagna, Tabular reinforcement learning methods for artificial intelligence tasks offloading in smart eye-wears, ACM Trans. Auton. Adapt. Syst. (2025) http://dx.doi.org/10.1145/3771092, Just Accepted.

[61] H. Sedghani, F. Filippini, D. Ardagna, SPACE4AI-D: A design-time tool for AI applications resource selection in computing continua, IEEE Trans. Serv. Comput. 17 (6) (2024) 4324–4339, http://dx.doi.org/10.1109/TSC.2024.3479935.

[62] F. Filippini, N. Calmi, L. Cavenaghi, E. Petriglia, M. Savi, M. Ciavotta, Analysis and evaluation of load management strategies in a decentralized faas environment: A simulation-based framework, in: Proceedings of the 1st Workshop on Serverless at the Edge, SEATED '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 1–8, http://dx.doi.org/10.1145/3660319.3660329.

Federica Filippini received her M.Sc. degree in Mathematical Engineering and her Ph.D. degree in Information Technology at Politecnico di Milano, and she is now a post-doc researcher at the Department of Informatics, Systems and Communication, University of Milano-Bicocca. Her research interests include optimization problems applied to resource selection and scheduling in Cloud and distributed environments.

Marin Lujak is an Associate Professor of Computational Science and Artificial Intelligence at Rey Juan Carlos University, Madrid, Spain. He received his Ph.D. in Management Engineering from the University of Rome Tor Vergata, Italy, in 2010. His research interests include distributed artificial intelligence, multi-agent coordination, and decentralized decision-making in cyber–physical systems with quality of solution guarantees.

Michele Ciavotta received the Ph.D. degree in automation and computer science from Università degli Studi Roma Tre, Italy, in 2008. He is currently an Associate in Computer Science at the Department of Informatics, Systems and Communication, University of Milano-Bicocca. His research interests include the modeling and optimization of highly constrained combinatorial problems, with particular emphasis on scheduling, resource management in distributed and cloud systems.