Analysis and Evaluation of Load Management Strategies in a Decentralized FaaS Environment: A Simulation-Based Framework

Federica Filippini
federica.filippini@unimib.it
University of Milano-Bicocca
Milan, Italy

Nicolas Calmi
n.calmi@campus.unimib.it
University of Milano-Bicocca
Milan, Italy

Luca Cavenaghi
l.cavenaghi@campus.unimib.it
University of Milano-Bicocca
Milan, Italy

Emanuele Petriglia
e.petriglia@campus.unimib.it
University of Milano-Bicocca
Milan, Italy

Marco Savi
marco.savi@unimib.it
University of Milano-Bicocca
Milan, Italy

Michele Ciavotta
michele.ciavotta@unimib.it
University of Milano-Bicocca
Milan, Italy

ABSTRACT

The Edge Computing paradigm has emerged to address new requirements in data processing. This approach enables the decentralization of computation by bringing computing capabilities to the edge of the network, i.e., close to the data sources, offering various benefits such as reduced latency and minimal network bandwidth consumption. In this context, Function as a Service (FaaS) emerges as a versatile and efficient solution, representing a specific instantiation of the Serverless Computing model. FaaS provides a scalable and reactive infrastructure that can be effectively applied to Edge Computing. Given the limited resource capacity of Edge nodes, appropriate load balancing is crucial. However, conducting on-field testing of any designed solution for this purpose can be arduous and time-consuming. This work addresses this challenge by proposing a simulation-based framework to design and evaluate load-management policies in decentralized FaaS environments. Additionally, we validate and compare four different load-balancing strategies, each characterized by varying degrees of complexity. Our experimental campaign demonstrates the effectiveness of the framework and our load-management methods across different operational scenarios.

CCS CONCEPTS

• **Computing methodologies → Simulation evaluation**; **Machine learning**; • **Computer systems organization** → *Peer-to-peer architectures.*

KEYWORDS

Function as a Service, Edge Computing, Load Balancing, P2P

ACM Reference Format:
Federica Filippini, Nicolas Calmi, Luca Cavenaghi, Emanuele Petriglia, Marco Savi, and Michele Ciavotta. 2024. Analysis and Evaluation of Load Management Strategies in a Decentralized FaaS Environment: A Simulation-Based Framework. In *Workshop on Serverless at the Edge (SEATED '24), June 3–7, 2024, Pisa, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3660319.3660329

1 INTRODUCTION

The Function as a Service (FaaS) model and Serverless Computing gained remarkable popularity in recent years, offering the most apparent advantages to compute-intensive and bursty workloads [3]. Indeed, reusable services deployed as stateless event-driven containers offer great flexibility to complex application workflows, since their execution can be triggered on demand and scaled to zero to limit overspending. Albeit developed as a Cloud service model, FaaS can be effectively extended to Edge Computing, enhancing the cost-effectiveness and the responsiveness of applications [4], and increasing resource utilization [1]. However, Edge systems are characterized by distributed resources with widely heterogeneous capacities in terms of computational and storage power. Exploiting FaaS in this context is challenging [7, 11], due to dynamic traffic patterns, limited resources availability, network-partition failures, locality and performance requirements.

The Decentralized FaaS (DFaaS) architecture, as initially proposed by Ciavotta et al. [6], partially addresses these needs by autonomously balancing the traffic load across Edge nodes within federated Edge FaaS ecosystems. Utilizing a peer-to-peer (P2P) overlay network, DFaaS facilitates information sharing among neighboring nodes, focusing on network topology, expected latency of running functions, and load states to determine the optimal redirection of a portion of ingress FaaS execution requests to less loaded peers. Developing efficient and effective load management strategies is crucial in this context. However, implementing and comparing alternative strategies across various scenarios is typically challenging. Hence, we propose an integrated framework that facilitates simulation-based analysis and evaluation of load management algorithms for decentralized FaaS environments. Note that, albeit initially designed and developed to support DFaaS, the framework we describe in this work is fundamentally agnostic to the specific platform we consider. It can therefore be applied to simulate and evaluate load-management algorithms implemented in any decentralized FaaS environment.

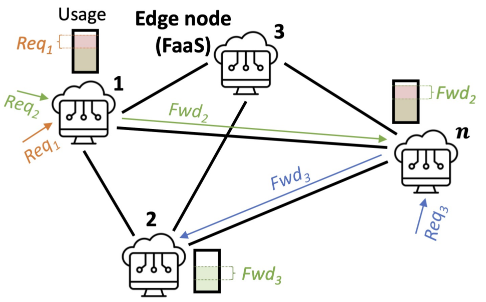
The rest of the paper is organized as follows: Section 2 frames the problem we tackle in this work and overviews the DFaaS system; Section 3 describes the framework we propose and its main contributions; Section 4 presents the four load-balancing algorithms we implemented; Section 5 includes the experimental validation of our framework and of the proposed strategies; Section 6 overviews the state of the art, and conclusions are finally drawn in Section 7.

2 PROBLEM STATEMENT

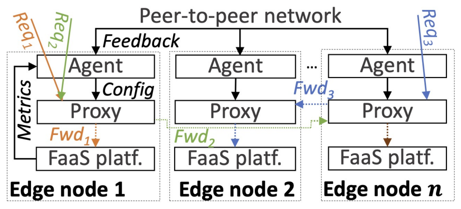
Applying the FaaS paradigm to Edge Computing holds promise as it facilitates application deployment and ensures improved reactivity, thereby enhancing response times. Sharing functions across multiple applications enables finer and more precise management of Service Level Agreements (SLAs), load balancing, and resource utilization in the Edge nodes of the network [5]. However, the traditional FaaS model is not seamlessly applicable in an intrinsically distributed and heterogeneous environment due to highly dynamic traffic, often characterized by geographic components, variable network topology, and limited resources at the Edge. Nodes need to be federated into a larger network to effectively leverage intelligent load-balancing strategies and benefit from shared computational resources [7]. In the following, we recall the considered scenario and related DFaaS platform architecture, which are our references for the framework proposed in this paper.

High-level representations of the reference scenario and DFaaS architecture are reported in Figure 1. A federation of geographically-distributed FaaS-enabled Edge nodes is considered, denoted by the set $\mathcal{N}$, each devoted to the execution of serverless functions by means of OpenFaaS [14]. Users send HTTP requests to the access point they are connected to, and these are processed locally or offloaded to neighboring nodes according to the current state and the decisions taken by suitable load managers. As an example, node 1 in Figure 1a receives sequentially two requests, denoted as Req_1 and Req_2, respectively; the first one is processed locally, but, as soon as it is enqueued, the node terminates its available capacity, so that Req_2 has to be forwarded to a different node n. Since also node n becomes overloaded after receiving the forwarded request Fwd_2, an incoming Req_3 is offloaded to node 2 by the load balancing algorithm running in node n.

The DFaaS architecture (see Figure 1b) comprises three main components: the *proxy*, implemented using HAProxy [9] and executed on each node to manage the arrival and distribution of HTTP requests; the *FaaS platform*, based on OpenFaaS and responsible for creating, executing, and managing functions; and identical, independent *agents* running on each node. These agents implement distributed and federated control functionalities, monitoring function- and node-level metrics (e.g., resource utilization), predicting the incoming load and resource demand for different classes of functions, and negotiating resources with neighboring nodes. In particular, to facilitate advanced load distribution strategies, the platform incorporates a mechanism to assess the impact of incoming traffic on the resources utilized and the Quality of Service (QoS) of deployed functions. This mechanism is achieved through performance models based on Machine Learning (ML) and trained using preliminary profiling of functions and nodes. Estimated metrics encompass CPU and RAM utilization, power consumption, and node state (i.e., overloaded or not). Presently deployed models also estimate the 5th and 95th percentiles for the considered metrics.



(a) Reference scenario



(b) High-level architecture

Figure 1: DFaaS [6] platform

In this work, we assume that the Edge nodes controlled by DFaaS can be categorized into three types characterized by different computing capacities, which we denote as *Light* ($\mathcal{N}_L$), *Mid* ($\mathcal{N}_M$), and *High* ($\mathcal{N}_H$) respectively, so that we can write $\mathcal{N}_L \cup \mathcal{N}_M \cup \mathcal{N}_H = \mathcal{N}$.

Functions are profiled and grouped according to observed resource utilization and Quality of Service (QoS); for the purposes of this study, we consider functions as members of three distinct classes denoted as *low-usage* (lu), *mid-usage* (mu), and *high-usage* (hu), respectively. Hence, each node $n \in \mathcal{N}$ is characterized by a set $\mathcal{F}^n$ of available functions, and each function belongs to a specific group in $\mathcal{G}^n \subseteq \{lu, mu, hu\}$. Identifying the set $\mathcal{F}^n$ for a given node n is crucial for advanced load-balancing algorithms, which can offload requests only to neighbors that share at least one function. We denote these potential *targets* for request offloading, defining their set as:

$$\mathcal{T}^n = \{q \in \mathcal{N} : \exists(n, q) \in \mathcal{E} \land \mathcal{F}^q \cap \mathcal{F}^n \neq \emptyset\}, \quad (1)$$

where $\mathcal{E}$ is the set of edges connecting nodes in the P2P network.

3 PROPOSED FRAMEWORK

This section describes the framework proposed to support the analysis and evaluation of load-management algorithms in decentralized FaaS environments, such as the one described in Section 2. The open-source code for this framework is available on GitHub[1]. A high-level component diagram is presented in Figure 2. Three main components are identifiable, each managing a distinct phase of the envisioned workflow for the framework. These components are:

- *Instance generator:* This component generates an instance of the problem (network topology, node types, incoming load per node) according to a provided configuration.

[1]https://github.com/UNIMIBInside/dfaas/tree/main/framework

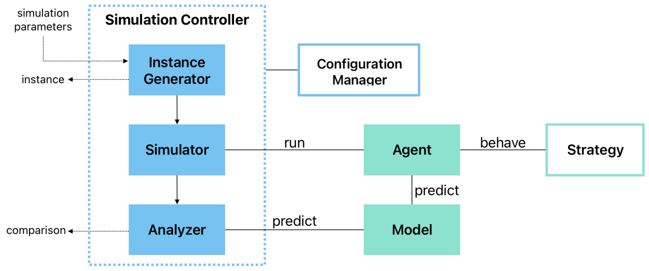


Figure 2: High-level diagram of the framework

- *Simulator:* It provides the core functionality for creating and executing distributed load management strategies. For each simulated time instant, it executes the considered strategy for each node and returns information about the per-function load to be sent to the neighbors (see Section 4).
- *Analyzer:* For each tested strategy, this component computes aggregate metrics such as the average success rate and the percentage of rejected requests, enabling the evaluation and comparison of the strategies' performance.

Instance Generator, Simulator and Analyzer are supported by some ancillary components, such as *Configuration Manager* and *Model.* The former facilitates the execution of the experimental campaign by automating the generation, execution, and assessment of several simulation scenarios according to user-specified parameters, including the number of simulation steps to be considered, the available nodes $\mathcal{N}$ per instance and the corresponding types, and the list of functions $\mathcal{F}^n$ and function classes $\mathcal{G}^n$ for each node. The latter provides an access point to retrieve the node-level metric estimates (see Section 2) needed for advanced strategies.

Details about the main framework components are provided in the following sections.

3.1 Instance Generator

This module supports the generation of a pseudo-random problem instance based on a high-level description provided by the user, including the number of Edge nodes to consider, the probability of connection between two nodes, and the maximum percentage of overloaded nodes. These three aspects significantly impact the simulator's performance. Instances characterized by a large number of nodes require more processing time, while strongly connected networks tend to favor intelligent load-management strategies, benefiting from a larger number of neighboring nodes. Additionally, if the percentage of overloaded nodes is too large, all algorithms perform equally poorly because the load cannot be effectively distributed. A problem instance is generated by randomly creating: (i) the set of Edge nodes $\mathcal{N}$, (ii) the list of function classes $\mathcal{G}^n$, and (iii) the list of functions $\mathcal{F}^n$ that can be executed on each node n. It is assumed that $\mathcal{G}^n$ and $\mathcal{F}^n$ remain fixed throughout the simulation.

Once these three steps are completed, the module generates the incoming load $\lambda_g^n(\tau)$ for each $n \in \mathcal{N}$, for each function class $g \in \mathcal{G}^n$ and for each simulation step τ starting from a user-specified maximum rate dictionary Λ. Each element $\{\Lambda_g^v\}_{v \equiv \mathcal{N}_L, \mathcal{N}_M, \mathcal{N}_H}$ corresponds to the maximum number of requests per second that can be reached by functions belonging to the class g when executed on the various node types. This component is designed to be extensible and accommodate various load patterns; however, the current version implements a single triangular pattern $\lambda_g^n(\tau)$, where values close to the maximum rate are reached at a time instant $\hat{\tau}$, referred to as the *peak time*, occurring around the midpoint of the simulation. For a specific node type v, the load profile $\lambda_g^n(\tau)$ is defined as follows:

$$\lambda_g^n(\tau) = \begin{cases} \Lambda_g^v \cdot random\left(\frac{\tau}{\hat{\tau}}, \frac{\tau+1}{\hat{\tau}}\right) \\ \Lambda_g^v \cdot random\left(\frac{T-\tau+1}{\hat{\tau}}, \frac{T-\tau}{\hat{\tau}}\right), \end{cases} \tag{2}$$

where T is the overall length of the simulation.

Note that directly using the values in Λ to determine the load at peak time generates a worst-case scenario where all nodes are simultaneously overloaded. Since no load-management strategy can effectively cope with this situation, the component introduces a preprocessing stage where the peak load values are tuned according to the user-specified maximum number of overloaded nodes in each instance. Finally, the $\lambda_g^n(\tau)$ profiles are used to determine the load values for each function in class g, and the process terminates by generating the graph representing the network topology.

3.2 Simulator

This component implements the main simulation loop. Specifically, given an instance of the problem, this module is responsible for executing a load management *strategy* by iterating over the simulation time τ and the node set $\mathcal{N}$, and retrieving information on load-balancing decisions.

To execute a strategy, the simulator must interact with the *agent* component. As detailed in Section 2, each agent controls a single Edge node and, based on the implemented strategy, calculates the percentage of requests to be served locally or offloaded to suitable neighboring nodes. A strategy leverages the features offered by the framework (models, predictions, information provided by neighbors, etc.). For ease of comparison and analysis, the framework supports the definition of multiple strategies that can be dynamically loaded and executed by the agent. Additionally, strategies may utilize performance models to make decisions; the framework supports several models simultaneously. For example, the current version of the simulator exposes endpoints to estimate the 5th, 50th, and 95th percentiles (0.05, 0.5 and 0.95 quantiles) for various metrics. The user can configure the simulator to use a specific strategy and model type. Further details on the implemented strategies can be found in Section 4.

Finally, collaboration with other components of the framework enables the automated execution of large experimental campaigns to compare different strategies on different instances, collecting data and metrics for later analysis by the Analyzer.

3.3 Analyzer

This component takes as input the problem definition and the output generated by the Simulator for one or more load-management strategies and facilitates comparison by calculating a set of indicators. Such an architectural choice thus enables the decoupling of the strategy execution phase from the evaluation phase, resulting in a more flexible and independent overall definition and evaluation process. Additionally, the development of these two components can proceed independently. At the time of writing, the Analyzer enables the calculation of the following metrics (although, due to its modular structure, it can be easily extended to include new ones):

- *Success Rate (SR):* It represents the percentage of requests successfully processed by the system over the considered time horizon.
- *Success Rate in the Stress Period (SR_{stress}):* This corresponds to *SR* when considering only the middle section of the simulation, during higher loads.
- *Mean Node Power Consumption (NPC_{avg}):* It denotes the average power consumption of each node in the network.
- *Max Node Power Consumption (NPC_{max}):* This indicates the maximum power consumption of each node in the network.
- *Rejected Requests (ρ):* It represents the total number of rejected requests over the considered time horizon. This includes requests forwarded to a node that does not host the required function, requests forwarded to an overloaded node, requests that would cause the threshold on the node power consumption to be exceeded (if any), and requests rejected by overloaded nodes until they return to the operational state.

4 LOAD MANAGEMENT ALGORITHMS

This section outlines four load-balancing algorithms that are implemented and distributed with the current version of the simulation framework. Two of these algorithms, as discussed in Section 4.1, are classified as *baselines*. These approaches are characterized as local or unintelligent, primarily serving the purpose of comparison. In contrast, the *node-margin* and *power-saving* strategies (detailed in Sections 4.2 and 4.3) are intelligent methods aimed at improving load balancing among neighbors and mitigating node power consumption, respectively.

Each strategy aims to generate a table of *forwarding weights* $\mathbf{W}$, where w_{ij}^{f} represents the proportion of requests to be offloaded from node $i \in \mathcal{N}$ to node j for all functions $f \in \mathcal{F}^i$. Correspondingly, w_{ii}^{f} denotes the requests served locally by node i. With the exception of the baseline strategy, all implemented methods utilize node-level metrics to determine $\mathbf{W}$, leveraging the ML-based performance models introduced in Section 2. These methods implement various choices based on the type of adopted model. For example, 0.95 quantile models generally result in more conservative strategies, as the associated node metrics are typically overestimated.

4.1 Baseline Strategies

We consider two baseline strategies, called *Base Strategy* and *Equal Strategy*.

In the *Base Strategy* the table $\mathbf{W}$ is generated under the assumption that no load balancing is performed. Therefore, $w_{ii}^{f} = 1$, and $w_{ij}^{f} = 0$ for all $j \neq i$. If the nodes become overloaded, requests will be rejected.

Another straightforward load-balancing mechanism is implemented for each node i by evenly distributing the estimated excess requests among its neighbors: this is what we called *Equal Strategy*. In particular, the matrix $\mathbf{W}$ is defined such that $w_{ii}^{f} = 1$ and $w_{ij}^{f} = 0$ for all neighbors j if the node is in an underloaded state (i.e., no excess requests are expected). Otherwise, the strategy involves an iterative approach in which, during each iteration, 1% of incoming requests for each function $f \in \mathcal{F}^i$ is equally redistributed among neighbors until node i returns to the operational state.

Therefore, the weights in matrix $\mathbf{W}$ can be derived as follows:

$$w_{ij}^{f} = \frac{1}{R^f} \cdot \begin{cases} R^f - r_{\text{fwd}}^{f} & \text{if } i = j \\ \frac{r_{\text{fwd}}^{f}}{v^i} & \text{otherwise,} \end{cases} \tag{3}$$

where R^f is the total number of incoming requests directed to function f, r_{fwd}^{f} is the number of requests directed to f to be offloaded ($r_{\text{fwd}}^{f} = 0$ if node i is expected to be in an underloaded state), and v^i is the number of neighboring nodes defined as nodes $j \in \mathcal{N}$ that are connected to i.

4.2 Node-Margin Strategy

The forwarding weights are computed by taking into account the network topology, the functions hosted by the neighboring nodes, and the estimated node-level metrics. In particular, each node $i \in \mathcal{N}$: (i) determines its *margin*, i.e., the percentage of resources it can offer to neighbors to process their excess of requests; (ii) exchanges information with the neighboring nodes, including the margin and the expected load for each function class in $\mathcal{G}^i$; (iii) computes the weights according to the node state and the neighbors' margin. Details on the three phases are provided in the following.

Margin Computation. The margin μ^i of node $i \in \mathcal{N}$ represents the portion of resources it can offer to neighboring nodes to host requests they cannot process. If the predicted node state for the next simulation step is *overloaded*, then $\mu^i = 0$. Otherwise, the node counts the number v^i of neighboring nodes $j \in \mathcal{T}^i$ (see Equation (1)). If v^i is zero, i.e., no function is in common with any of the neighboring nodes, then $\mu^i = 0$ since forwarded requests could not be processed. Otherwise, the node obtains predicted values for incoming load, CPU and RAM usage, and power consumption. For any metric m, a utilization threshold $\hat{u}_m^i$ is set, and the margin is computed to prevent the node from reaching an overloaded state. Specifically, if any predicted metric value $\tilde{u}_m^i$ exceeds the corresponding threshold, then $\mu^i = 0$. Otherwise, the margin can be calculated considering the difference between the expected values and their thresholds.

Information Exchange. The primary objective of this phase is to share information with neighboring nodes concerning the margin μ^i. Additionally, if a node expects to find itself in the overloaded state and intends to offload requests to other nodes, it must also communicate the predicted load value λ_g^i for each function class $g \in \mathcal{G}^i$ (see Section 3.1). Each node is tasked with assessing the *impact* that its forwarded requests will have on neighboring nodes, ensuring that the margin is not exceeded. Calculating this impact is typically challenging due to its dependence on factors such as the current number of requests being handled by neighbors, influenced by different runtimes in warm and cold states [8, 10, 12].

Forwarding Weights Computation. As with the other strategies, requests are forwarded by node i only if it is expected to be in an overloaded state; if not, $w_{ii}^f = 1$ and $w_{ij}^f = 0$ for all $j \neq i$, where $f \in \mathcal{F}^i$.

The selection of target neighbors capable of receiving the forwarded requests is determined by the margin value μ^j provided during the Information Exchange phase by all neighboring nodes $j \in \mathcal{T}^i$. Thus, node i proceeds as follows:

(1) It randomly selects a candidate $j \in \mathcal{T}^i$ and function $f \in \mathcal{F}^i \cap \mathcal{F}^j$, obtaining its margin μ^j.
(2) It attempts to assign 1% of requests for function f to node j, calculating the offloading impact on the neighbor's marginal resources.
(3) If the impact on resources does not exceed the provided margin, the request offloading is confirmed, and the node state estimation model is invoked to check whether i is still overloaded. Otherwise, the transfer is rejected, and function f is removed from the set of candidate functions. Note that if function f is the only one in $\mathcal{F}^i \cap \mathcal{F}^j$, node j is removed from $\mathcal{T}^i$.

These steps are repeated until the node exits the overloaded state or no feasible targets can be found. The forwarding weights are defined as in Equation (3), computing r_{fwd}^f for each neighbor and each function.

4.3 Power-Saving Strategy

This is the only implemented strategy that allows limiting node power consumption. It is important to note that a node bound to a particular consumption threshold can handle a lower number of requests than a node not subject to such limits, leading to a higher number of rejections in case of heavy traffic. Therefore, instead of replacing other methods that may be more efficient from the success-rate perspective, this strategy should be used when lower traffic is expected, thus guaranteeing good service quality and energy savings.

The method proceeds as described for the node-margin strategy in Section 4.2, except that only the power consumption metric is considered when evaluating the margin. Moreover, the algorithm terminates not only when node i exits the overloaded state or no available neighbors are found, but also if the expected power consumption falls below the prescribed threshold.

5 EXPERIMENTAL ANALYSIS

The proposed framework has been validated to assess its performance and compare the four built-in load-management strategies, considering the metrics listed in Section 3.3. The following sections describe the experimental setup (Section 5.1) and results, in terms of strategies comparison (Section 5.2) and scalability of the framework (Section 5.3). The results prove the effectiveness of the node-margin strategy in terms of success rate and highlight the promising outcomes of the power-saving strategy. The execution times of the simulator are also discussed.

5.1 Experimental Setup and Methodology

We conducted simulations over $T = 7$ time steps (with a step size of $\tau = 1$ minute) using networks which represent common Edge architectures [15]: 50% of the nodes belong to $\mathcal{N}_L$, characterized by 2 CPUs and 8GB of RAM; 30% are in $\mathcal{N}_M$ (4 CPUs and 16GB of RAM), and 20% in $\mathcal{N}_H$ (6 CPUs and 24GB of RAM). In the first series of experiments, whose results will be discussed in Section 5.2, we considered random networks of 10 nodes, with a probability of connection between two nodes equal to 0.4, and each problem instance was simulated with maximum percentages of overloaded nodes (referred to as MPO) set at 30%, 60%, and 90%, respectively, which can be controlled by the simulator. In the scalability analysis (see Section 5.3), we varied the number of nodes between 5 and 35 (with step 10), the probability of creating connections between nodes from 0.2 to 0.8 (with step 0.2), and considered MPO equal to 30%, 60% and 90%. Each experiment was replicated using three model types: linear regression, and quantile regressions at 0.05 and 0.95, resulting in a total of over 5000 simulations.

Linux Virtual Machines were created using KVM and Qemu on an Intel® NUC Kit NUC6i7KYK computer and profiled to model the performance of nodes and functions. The metric thresholds for the node-margin and power-saving strategies are listed in Table 1; as mentioned in Section 4.2, they are used to define the margin a node can offer to the neighbors. The values have been chosen empirically according to the results obtained from some exploratory experiments, with the rationale of choosing values below which it is very rare that a node is overloaded.

	node-margin			**power-saving**
Node type	CPU usage	RAM usage	power	power
$\mathcal{N}_L$	150%	4.1 GB	0.7 W	0.49 W
$\mathcal{N}_M$	290%	5.5 GB	2.1 W	1.54 W
$\mathcal{N}_H$	460%	6.0 GB	3.5 W	2.45 W

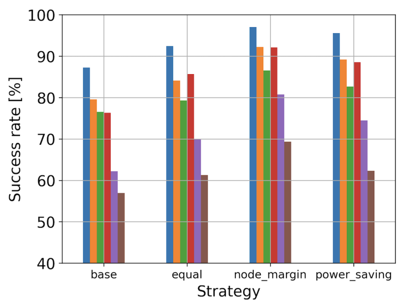
Table 1: Metric thresholds

As for the methodology, in the initial part of Section 5.2 we will assess the impact of MPO on both the success rate and power consumption achieved by the node-margin and the power-saving strategies. Subsequently, we will examine the average values of the five indicators discussed in Section 3.3 across various problem instances with different MPO levels and compare the performance of the four strategies, including the baseline ones. To facilitate discussion, instead of presenting the absolute values of the indicators, we will consider the *average percentage gain* (*APG*) obtained by the equal, node-margin, and power-saving strategies in comparison to the base strategy. This is defined as:

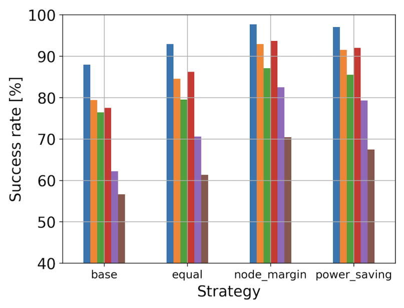
$$APG = \frac{\Delta I^s}{I^{base}} \cdot 100, \tag{4}$$

where I^{base} denotes the value of index I (which may be SR, SR_{stress}, NPC_{avg}, NPC_{max} or ρ) obtained by the base strategy, and ΔI^s denotes the difference between any strategy s and the base one. Since for the success rate we can say that a strategy s improves over the base one if the indicator's value is higher than the base's one, while for the other metrics we observe an improvement when the value is lower, we define ΔI^s as:

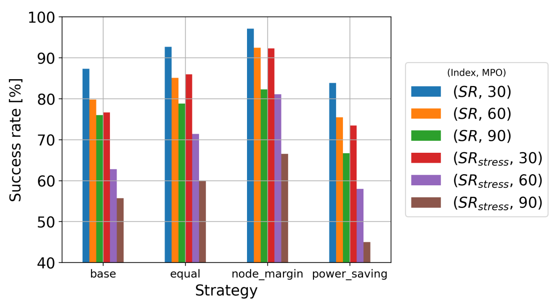
$$\Delta I^s = \begin{cases} I^s - I^{base} & \text{if } I \text{ is } SR \text{ or } SR_{stress} \\ I^{base} - I^s & \text{otherwise.} \end{cases} \tag{5}$$



(a) Linear regression

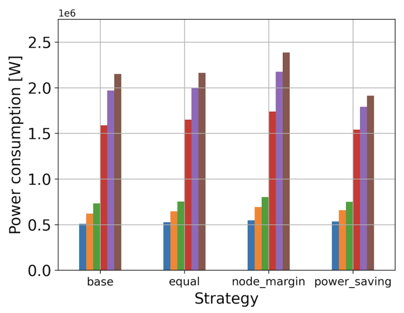


(b) 0.05 quantile regression

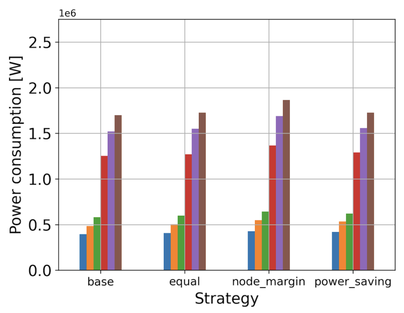


(c) 0.95 quantile regression

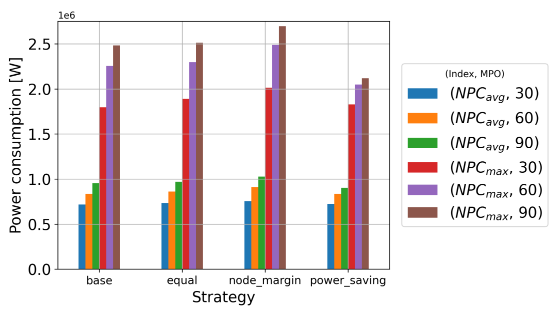
Figure 3: SR and SR_{stress} with the node-margin and power-saving strategy, varying MPO and model type



(a) Linear regression

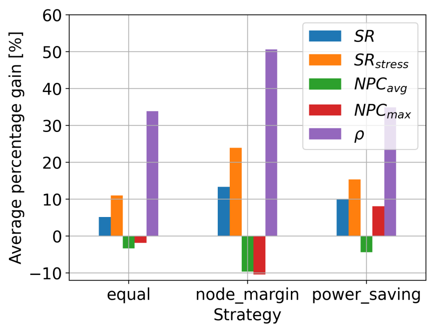


(b) 0.05 quantile regression

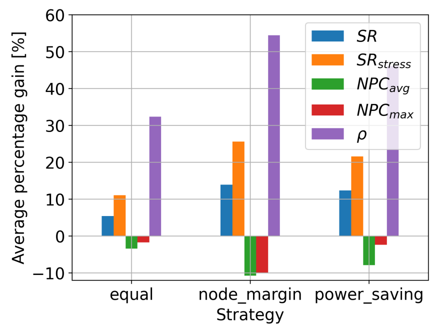


(c) 0.95 quantile regression

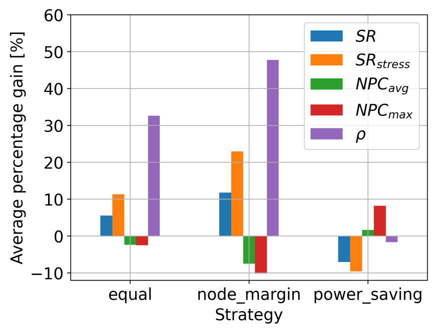
Figure 4: NPC_{avg} and NPC_{max} with the node-margin and power-saving strategy, varying MPO and model type



(a) Linear regression



(b) 0.05 quantile regression



(c) 0.95 quantile regression

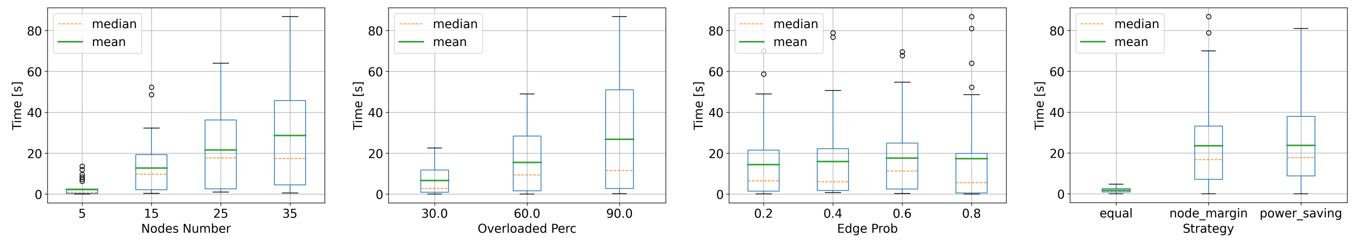
Figure 5: APG of all strategies with respect to *base*, considering SR, NPC and ρ and different model types

For the scalability analysis (see Section 5.3), we evaluated the average execution time of each simulation step. The tests were executed on a PC with Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz and 16GB of RAM.

5.2 Experimental Results

The impact of varying MPO was evaluated on the two most advanced strategies, namely node-margin and power-saving, to analyze their performance in increasingly challenging scenarios.

Figure 3 presents the results in terms of success rate, considering the three available performance models for node-level metrics. It is evident that SR progressively decreases for higher values of MPO since, as expected, achieving good load balancing becomes more challenging when a plurality of nodes are overloaded (i.e., the overall load in the network is higher). However, it is noteworthy that both node-margin and power-saving strategies maintain SR and SR_{stress} above 80% and 60%, respectively, even in the most challenging scenario.



(a) Varying number of nodes **(b) Varying MPO** **(c) Varying edge probability** **(d) Varying strategy**

Figure 6: Execution times distribution of each simulation step τ

A similar comparison regarding power consumption is depicted in Figure 4. Here, we observe that the node power consumption (NPC_{avg} and NPC_{max}) increases with higher MPO values, as efficient solutions tend to consume more energy in a complex setting. Additionally, it is evident that, as will be discussed further, the power-saving strategy typically limits energy consumption compared to the node-margin strategy. The only exception is illustrated in Figure 4b, where the 0.05 quantile regression model is considered: the specified threshold fails to restrict power consumption adequately, resulting in performance similar to that of the node-margin strategy.

Finally, Figure 5 compares the built-in strategies by reporting the achieved Average Performance Gain (*APG*), computed as discussed in Section 5.1, after averaging all available indicators over the three MPO values considered. We remind the reader that, according to the definition in Equation (5), for each plot in Figure 5, a strategy *s* outperforms the base strategy if *APG* is greater than 0 (the higher, the better). Observations from the figure are as follows:

- The performance of the equal strategy remains unaffected by the performance model type under consideration, as it determines forwarding weights by equally distributing excess requests regardless of node utilization metrics. Generally, it yields a higher success rate and a lower number of rejected requests compared to the base strategy, albeit at a slightly higher power consumption.
- The node-margin strategy is the best-performing in terms of success rate and ρ, achieving up to a 25% higher *APC* for the former and 50% for the latter with any model type. However, this method is the most energy-intensive, with an average Node Power Consumption (NPC_{avg}) between 7.5% and 10% higher than the base strategy.
- Lastly, the power-saving strategy demonstrates good performance in terms of success rate and number of rejected requests when considering linear regression and 0.05 quantile regression as performance models. However, as it bases decisions on a single metric, it is highly sensitive to variations in predictions, as evidenced by significant performance degradation when considering 0.95 quantile regression models. Consequently, accurate tuning of the optimal threshold value for power consumption is necessary. Notably, this strategy is the only one that manages to reduce maximum power consumption compared to the *base* strategy (see Figures 5a and 5c), with an *APG* around 7.5%.

Overall, these results highlight the promising performance of node-margin and power-saving in designing a load-balancing schema for networks of decentralized FaaS-enabled Edge nodes.

5.3 Scalability Analysis

We validated the impact of varying several instance parameters on the execution time of each simulation step τ. In particular, as mentioned in Section 5.1 and reported in Figure 6, we considered the instance dimension, represented by the size of the set of nodes $\mathcal{N}$, the probability of creating an edge connecting two nodes, the maximum percentage of overloaded nodes (MPO), and the impact of adopting different load-balancing strategies.

As expected, incrementally increasing the number of nodes (refer to Figure 6a) and MPO (refer to Figure 6b) results in a substantial rise in the average execution time. Indeed, both the node-margin and power-saving strategies require the identification of suitable target neighbors to offload part of the computation, and this operation becomes more time-consuming when a larger number of nodes is available (or when most of them are overloaded).

Conversely, varying the probability of creating edges between nodes does not exert a significant impact on the execution time; instead, it is marginally higher when increasing the probability between 0.2 and 0.6, but then slightly decreases when moving to 0.8 (refer to Figure 6c). While seemingly counter-intuitive, this result aligns with the observation that, on the one hand, addressing the load-management problem becomes more challenging when the network is more connected, but on the other hand, finding a suitable target for requests offloading becomes achievable when the average number of neighbors for each node increases.

Finally, as illustrated in Figure 6d, it is evident that the equal strategy is significantly faster, as it involves less complex operations to determine the forwarding weights. Nonetheless, the average execution time with both the node-margin and power-saving strategies remains below 30 seconds, making them suitable for practical applications.

6 RELATED WORK

The growing demand for real-time data processing and the simplification of applications development and deployment opened new research directions aimed at reconciling the Serverless service model and the Edge Computing paradigm. While this convergence promises to combine the advantages of both, enabling greater efficiency and agility in real-time data processing, it also comes with great challenges. Indeed, deploying Serverless functions to Edge devices requires optimizing resources and performance to accommodate hardware limitations and variable network conditions. At the same time, it is crucial to ensure the consistency and reliability of distributed operations across a heterogeneous network of these devices. This section overviews literature proposals concerning the application of the Serverless paradigm and FaaS model to Edge Computing, with primary focus on load management.

RACER [5] exploits Reinforcement Learning for the load management of FaaS-based applications in an Cloud-Edge continuum. In particular, nodes are organized in a hierarchical structure including three tiers: base-station nodes with limited computational capacity, intermediate hosts, and Cloud datacenters with huge capacity but large latency overheads. Based on the characteristics (e.g., quality of service requirements, execution cost, input data, etc.) of the tasks to be performed, the agent selects the tier where the function should be executed. Also λContSim [13] considers as a target the Cloud-Edge continuum. As our work, it allows to evaluate custom strategies through simulation. However, it focuses on positioning strategies (i.e., functions-to-nodes mapping methodologies designed to optimize desired quantities such as costs, energy consumption and availability), while the load balancing is not discussed. Finally, a FaaS platform to adaptively exploit the Cloud-Edge continuum, called Serverledge, is presented in [16]. It adopts a decentralized approach where Edge nodes can serve all the incoming requests or leverage vertical offloading to deal with workload peaks. As in our proposal, nodes exchange information with the neighbors to identify the best candidates to forward requests when needed. However, there is no specific focus on different offloading strategies.

Among proposals that consider only Edge resources (as ours), Hedgi [2] implements a centralized manager to coordinate the load balancing among highly heterogeneous nodes. This significantly differs from our approach, where a load-management agent acts independently on each node. The closest to our work is [7], whose authors propose: (i) an architectural framework to route to appropriate Edge devices the requests coming to stateless functions, and (ii) an emulation environment to test the framework supporting different load management strategies. However, they do not consider multiple classes of functions, while we take this aspect into account to evaluate the nodes compatibility in forwarding requests.

7 CONCLUSION AND FUTURE WORK

This paper proposes a simulation-based framework for evaluating load-balancing algorithms in decentralized FaaS environments. The framework automates the generation of problem instances characterized by different Edge networks and load distributions, facilitating the seamless execution and comparison of different load-management strategies based on metrics such as success rate, power consumption, and the number of rejected requests. We tested and evaluated the framework within the context of the Decentralized FaaS (DFaaS) platform [6], implementing and comparing sample strategies leveraging knowledge on nodes utilization and power consumption. Our findings highlight their promising performance in terms of the analyzed metrics.

Although the implemented framework represents an effective simulation environment for comparing load-management algorithms, it has some limitations, particularly related to the fact that it provides a limited view of the P2P network, focusing only on the closest neighbors. Future work will focus on extending the framework by considering robust techniques to enable message passing among Edge nodes, thereby enabling synchronization and effective management of malfunctions (e.g., sudden node disappearance due to unpredictable failures). Furthermore, we will consider developing load prediction models, opening new research paths aimed at outlining a strategy to alternate load-balancing algorithms based on incoming requests.

ACKNOWLEDGMENTS

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART")

REFERENCES

[1] Gojko Adzic and Robert Chatley. 2017. Serverless computing: economic and architectural impact. In *ACM ESEC/FSE Proceedings*, 884–889. ISBN: 9781450351058. DOI: 10.1145/3106237.3117767.

[2] Mohammad Sadegh Aslanpour, Adel N. Toosi, Muhammad Aamir Cheema, Mohan Baruwal Chhetri, and Mohsen Amini Salehi. 2024. Load balancing for heterogeneous serverless edge computing: A performance-driven and empirical approach. *Future Generation Computer Systems*, 154, 266–280. DOI: https://doi.org/10.1016/j.future.2024.01.020.

[3] Sanjay Chaudhary, Gaurav Somani, and Rajkumar Buyya, (Eds.) 2017. *Serverless Computing: Current Trends and Open Problems. Research Advances in Cloud Computing.* Springer Singapore, Singapore, 1–20. ISBN: 978-981-10-5026-8. DOI: 10.1007/978-981-10-5026-8_1.

[4] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The rise of serverless computing. *Commun. ACM*, 62, 12, 44–54. DOI: 10.1145/3368454.

[5] Chunglae Cho, Seungjae Shin, Hongseok Jeon, and Seunghyun Yoon. 2020. QoS-Aware Workload Distribution in Hierarchical Edge Clouds: A Reinforcement Learning Approach. *IEEE Access*, 8, 193297–193313. DOI: 10.1109/ACCESS.2020.3033421.

[6] Michele Ciavotta, Davide Motterlini, Marco Savi, and Alessandro Tundo. 2021. DFaaS: Decentralized Function-as-a-Service for Federated Edge Computing. In *IEEE CloudNet*, 1–4. DOI: 10.1109/CloudNet53349.2021.9657141.

[7] Claudio Cicconetti, Marco Conti, and Andrea Passarella. 2018. An architectural framework for serverless edge computing: design and emulation tools. In *IEEE CloudCom*, 48–55. DOI: 10.1109/CloudCom2018.2018.00024.

[8] Alexander Fuerst and Prateek Sharma. 2022. Locality-aware Load-Balancing For Serverless Clusters. In *ACM HPDC Proceedings*, 227–239. ISBN: 9781450391993. DOI: 10.1145/3502181.3531459.

[9] HAProxy. 2023. HAProxy. The Reliable, High Performance TCP/HTTP Load Balancer. http://www.haproxy.org. Accessed: (04/03/2024). (2023).

[10] Jeongchul Kim and Kyungyong Lee. 2019. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In *IEEE CLOUD*, 502–504. DOI: 10.1109/CLOUD.2019.00091.

[11] George Kousiouris and Dimosthenis Kyriazis. 2021. Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay. In *CLOSER*. https://api.semanticscholar.org/CorpusID:235234002.

[12] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. 2018. Cold Start Influencing Factors in Function as a Service. In *IEEE/ACM UCC Companion*, 181–188. DOI: 10.1109/UCC-Companion.2018.00054.

[13] Alessio Matricardi, Alessandro Bocci, Stefano Forti, and Antonio Brogi. 2023. Simulating faas orchestrations in the cloud-edge continuum. In *ACM FRAME Proceedings*, 19–26. DOI: 10.1145/3589010.3594893.

[14] OpenFaaS. 2023. OpenFaaS. Serverless Functions, Made Simple. https://www.openfaas.com. Accessed: (04/03/2024). (2023).

[15] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. 2018. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*, 5, 2, 1275–1284. DOI: 10.1109/JIOT.2018.2805263.

[16] Gabriele Russo Russo, Tiziana Mannucci, Valeria Cardellini, and Francesco Lo Presti. 2023. Serverledge: decentralized function-as-a-service for the edge-cloud continuum. In *IEEE PerCom*, 131–140. DOI: 10.1109/PERCOM56429.2023.10099372.