GIRAFF: Reverse Auction-based Placement for Fog Functions

Volodia Parol-Guarino, Nikos Parlavantzas

GIRAFF: Reverse Auction-based Placement for Fog Functions

Volodia PAROL-GUARINO
volodia.parol-guarino@inria.fr
Inria, Univ. Rennes
Rennes, France

Nikos PARLAVANTZAS
nikos.parlavantzas@irisa.fr
INSA Rennes, IRISA
Rennes, France

ABSTRACT

Function-as-a-Service (FaaS) is a compelling new programming model for developing applications running on fog infrastructures. FaaS applications are composed of short-lived, event-triggered units, called functions. Functions can be flexibly deployed on demand along the cloud-to-thing continuum. However, deciding how to place those functions in the fog presents multiple challenges. The fog contains diverse, geo-distributed, and potentially resource-constrained nodes that should be efficiently shared between applications with latency requirements. Importantly, fog nodes are owned by different entities that should be incentivized to share their resources. Most function placement approaches ignore latency requirements or the presence of multiple owners in fog infrastructures.

To address these challenges, this article proposes a market-based approach to place FaaS applications in the fog. Clients submit function placement requests associated with SLAs that specify expected guarantees over network latency and allocated resources. The approach then organizes an auction among fog node providers to determine the nodes that host each function and the revenue of the provider. The article presents an open-source implementation of the approach evaluated on the Grid'5000 testbed. Experiments demonstrate that our approach can reduce client spending by up to three times while delivering service quality that matches or exceeds that of baseline methods.

CCS CONCEPTS

• **Computer systems organization → Distributed architectures**.

KEYWORDS

fog, FaaS, function-as-a-service, serverless, auction, SLA, service-level agreement, QoS, quality of service, testbed, Grid'5000

ACM Reference Format:
Volodia PAROL-GUARINO and Nikos PARLAVANTZAS. 2023. GIRAFF: Reverse Auction-based Placement for Fog Functions. In Proceedings of 9th International Workshop on Serverless Computing (WoSC'23). ACM, New York, NY, USA, 6 pages.

DOI: 10.1145/3631295.3631402

WoSC'23, December 11–15, 2023, Bologna, Italy

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

DOI: 10.1145/3631295.3631402

1 INTRODUCTION

The rapid growth of data exchange creates significant scalability challenges, prompting the expansion of cloud computing into the fog computing domain [10]. Fog brings computing power closer to the border of the network where data is generated and consumed. This proximity alleviates bandwidth pressures on the cloud and enables low-latency application responses. A compelling programming model for developing fog applications is serverless computing [16] using the FaaS model. FaaS applications are made up of stateless event-triggered functions that can be flexibly deployed along the continuum from the cloud to the Internet network border. Functions are dynamically provisioned and unprovisioned, ensuring that resources are allocated only when needed, thus promoting efficient utilization of infrastructure resources.

The delivery of FaaS applications that can optimally exploit fog infrastructures poses significant challenges. First, fog-based applications often have stringent Quality-of-Service (QoS) requirements, especially in terms of latency. For example, constrained-time use cases, such as augmented reality (AR) workloads [13], require controlled latency to ensure reliable and expected performance. Achieving such low-latency responses in the fog is difficult because fog resources are shared among highly dynamic workloads with ever-changing resource demands. Second, fog resources are owned and operated by multiple independent infrastructure providers, such as individuals, companies, communities, and established cloud and edge providers. These various providers may have different resource management strategies and should be incentivized to contribute their computational resources for hosting application functions.

Although initial research has explored the use of FaaS in fog environments [2, 4], currently no system effectively addresses the challenges mentioned above. The broader issue of placing computational tasks in the fog infrastructure has received significant research attention [19]. However, most of that research assumes resources are owned and managed by a single provider and ignores the realistic scenario of multi-provider infrastructures. Some research has addressed the multi-provider requirement by using market-based approaches [12]. However, these systems primarily focus on economic aspects, are evaluated through simulations, and lack practical implementations that can be readily used in close-to-real-world fog deployments at scale.

This article introduces a novel approach, named Global Integration of Reverse Auctions and Fog Functions (GIRAFF), to optimize the placement of FaaS applications in the fog

taking into account economic concerns. The approach satisfies QoS requirements by exclusively reserving resources for functions for a specified duration while incentivizing resource providers through a market-based mechanism. In our proposed solution, clients submit function placement requests to a central marketplace. Function placement requests characterize functions through Service-Level Agreements (SLAs), which specify guarantees for network latency and required resources. The marketplace then organizes an auction in which fog nodes, owned by independent providers, compete for hosting functions by making bids. After the auction selects a fog node, the node allocates resources for a specified time to the client's function and receives payment from the client. The approach is evaluated in a close-to-real-world testbed by deploying a reproducible fog network on the Grid'5000 scientific infrastructure at scale. The article makes two main contributions:

- we propose an auction-based approach for placing FaaS functions in a multi-provider fog infrastructure;
- we assess the effectiveness of our approach[1] through reproducible experiments on the Grid'5000 testbed[2], comparing it with baseline algorithms, at scale.

The article continues with a review of related work (section 2) and the design section (section 3), which discusses the system model, our market-based algorithm, and its implementation. The article then presents a comparison of the algorithm with two baselines in the evaluation section (section 4). Finally, section 5 concludes the article.

2 RELATED WORK

There is extensive research on the placement of computational entities in emerging fog infrastructures [19]. This research assumes that resources are issued and managed by a single entity. Some research targets multi-provider fog environments using auction-based approaches [12]. However, these systems focus primarily on economic considerations, rely on simulations for evaluation, and lack practical implementations that can be readily deployed in fog environments. In the following, we only consider work related to the placement of FaaS applications in fog environments.

Bocci et al. [5] present a declarative approach for placing FaaS applications in a fog infrastructure using a Prolog engine. The approach considers hardware and software requirements, latency constraints, security constraints, and trust relationships between stakeholders. However, the economic goals of the stakeholders are not considered.

Rausch et al. [14] present a container scheduling system for placing FaaS functions on a fog cluster. The scheduler uses multi-objective optimization techniques, considering node capabilities, network topology, and data transfer requirements of functions. Similarly to our work, the prototype relies on OpenFaaS and Kubernetes. Unlike our approach, this scheduling system does not consider multiple clusters owned by different entities.

Bermbach et al. [4] present an auction-based function placement approach for FaaS applications running on fog platforms. In this approach, application developers specify bids on storage and processing resources, attached to function placement requests. Bids are processed in batches. Fog nodes then decide locally whether to serve the requests or offload the requests to the next node on the path towards the cloud. Unlike our approach, this work does not provide latency guarantees to clients.

Ascigil et al. [1] examine FaaS resource provisioning over computation spots distributed along the path to the cloud. This work uses simulations to evaluate various centralized and decentralized strategies for resource provisioning. Similarly to our work, FaaS functions have latency requirements. However, unlike our work, the contribution assumes that the computing infrastructure is owned by a single organization.

Baresi et al. [3] propose NEPTUNE, a framework for managing serverless applications with response-time requirements on fog topologies. The framework supports function placement, request routing, function scaling, and GPU management using Mixed Integer Programming and control-theoretic techniques. NEPTUNE divides the fog topology into independent communities to simplify management but does not consider competition between these communities.

Finally, Smolka and Mann's [18] survey analyzes 99 articles in the fog placement literature. In this survey, 88% of the articles use a simulator. Only 18% of analyzed articles tested their techniques on more than 100 fog nodes. Usually, the articles surveyed use a simulator for scaling, and they validate their results on a small testbed. One of the survey's conclusions is that most experiments are conducted on unrealistically small problem instances. In this article, we deploy up to 119 nodes. To the best of our knowledge, we are the first to experiment on a scaled-up, reproducible testbed.

3 GIRAFF DESIGN

This section presents our system model, the SLA contents, our auction-based placement method, the baseline methods, and the system implementation.

3.1 System Model

The actors in the system are the fog nodes, the clients, the end users, and the marketplace. *Fog nodes* contain resources and are owned by providers. *Clients* create and manage applications that use fog node resources. In our case, these applications are stateless functions that follow the FaaS paradigm. *End users* interact with client functions. The marketplace serves as a trusted intermediary between clients and fog nodes. It enables monetary and placement decisions using a reverse auction. In this auction, fog nodes are resource vendors, while clients are purchasers. A second-price auction [7] is held, in which the fog nodes submit bids, and the winner

[1] https://github.com/VolodiaPG/giraff; version used here is tagged 1.0.0 and is also available in the release section along with the raw data used in section 4.
[2] www.grid5000.fr

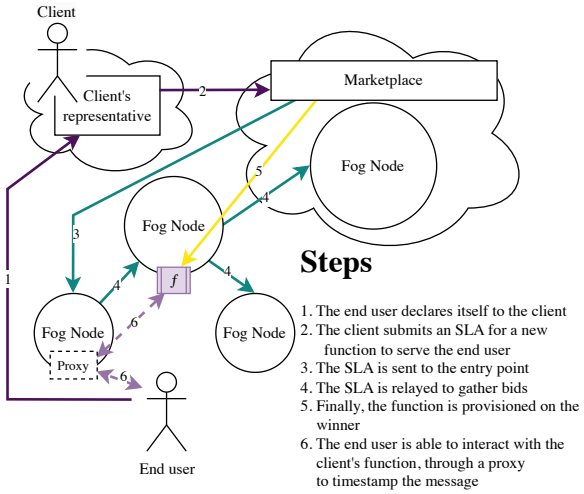

Figure 1: Provisioning function f

is the lowest bidder, yet gets paid the value of the second-lowest bid. This type of auction encourages fog nodes to bid their true resource cost, thus promoting economic efficiency. The marketplace is aware of the fog topology and centralizes payment and placement mechanisms. This design choice is in line with existing security and authentication proposals [11, 17] that aim to ensure the security of application data [6]. Requests from end users to functions do not go through a central gateway [15].

3.2 Service Level Agreement

The SLA specifies the constraints that must be met for a function to run as intended by the client. SLA constraints include the following: $SLA_{entrypoint}$ is the entry point, that is, the node where the end user is connected (this is the ideal node to provision the function). SLA_{max_lat} is the maximum latency allowed for the function; the latency is defined as the delay between the request of an end user entering $SLA_{entrypoint}$ and its reception by a function (when the network packets are reconstructed and the function is ready to unmarshal the data; see section 3.4). SLA_{dur} is the duration of the reservation. SLA_{CPU} and SLA_{mem} denote the CPU and memory constraints.

3.3 Algorithms

This section presents GIRAFF and two baseline placement methods.

3.3.1 GIRAFF approach. The interactions between the marketplace and the fog nodes are illustrated in figure 1. First, the end user who desires to use a client application contacts the client representative service (running in the cloud) to authenticate and identify as a new user (Step 1, figure 1). The client representative then sends an SLA of the required function to the central marketplace (Step 2, figure 1). The marketplace then sends the SLA to $SLA_{entrypoint}$, the ideal node to serve the end user, on the network border (Step 3, figure 1). Fog nodes transmit the SLA to their neighbors as long as the constraint SLA_{max_lat} is respected (Step 4, figure 1). When receiving an SLA, each node responds with a bid estimating the cost of the function if they were to provision it. This approach encourages competition among providers, each acting selfishly and autonomously to achieve their own goals, while competition is kept under control by the use of second-price auctions. The algorithms run by the marketplace and fog nodes are described next.

GIRAFF marketplace accepts a client's SLA as input. It uses its knowledge of the fog network to find the IP address and send the SLA to the $SLA_{entrypoint}$. This node then returns a list of bids made by nodes that meet the constraints described in the SLA. The algorithm proceeds with the second-price auction and validates function provisioning on the winning node.

GIRAFF participant evaluates the cost of the SLA for the node (expressed as bid) and forwards the SLA to neighbors of the node for further evaluation. The process repeats itself. Additionally, the node increments a variable that measures the accumulated latency between the first SLA reception by a fog node and the current node. This variable is included in the request to ensure adherence to the SLA_{max_lat} constraint. If this constraint cannot be met, the SLA is not sent to the neighbor.

3.3.2 Baseline placement methods. This subsection introduces the two baseline placement methods with which we compare our GIRAFF method; that is, the edge-furthest and edge-ward methods. These methods follow a collective approach in which the fog nodes collaboratively manage their resources, disregarding their interests.

Edge-furthest seeks to anticipate future placements. The SLA is accepted by a single node, the furthest possible, while still adhering to the SLA. The motivation is to keep available resources closer to entry points, allowing more constrained functions to be accepted as they arrive. This algorithm was chosen as a straightforward approach to increase the utilization of the fog platform while respecting SLAs.

Edge-ward is an algorithm implemented in iFogSim [9]. It establishes a path between the $SLA_{entrypoint}$ and the cloud. It attempts to greedily allocate the SLA at each layer, stopping at the first positive response. The algorithm ignores the latency constraint in the SLA.

3.4 Implementation

As described in section 3.3.1, our system is divided into the marketplace and fog node software. The first acts as the fog network's gateway for clients and coordinates function provisioning. The second runs on each fog node that participates in the fog network and is implemented on top of OpenFaaS[3] running on K3S[4]. The entire project is managed with Nix[5].

[3] www.openfaas.com
[4] k3s.io is a lightweight distribution of Kubernetes
[5] nixos.org is a package manager for reproducibility and determinism

Nix is used to generate deterministic and reproducible virtual machines (VMs) for our experiments.

4 EVALUATION

This section discusses experimental settings, the adopted cost model, and comparisons between GIRAFF and the two baselines concerning placement quality, client spending, and deployment time.

4.1 Experimental settings

The experiments were run on Grid'5000. The servers use Intel Xeons Gold 5220 from 2019, each of which has 36 SMT threads and 96 GiB of RAM. Each server runs multiple VMs that act as fog nodes. Depending on its location in our 4-layer deep fog network, each VM can use from 2 vCPUs and 4 GiB of RAM up to 16 vCPUs and 46 GiB of RAM when the node is located closer to the cloud. In our experiments, we used up to 20 servers for 119 fog nodes. The number of nodes and the network's topology are randomly generated to fit onto a certain number of servers.A fog network requires that latency be set between all pairs of VMs. To this end, we employ Netem[6], wrapped by Enoslib[7] (the deployment library that we use for Grid'5000). Then, a dedicated VM acting as all end users (cf. figure 1) generates a load for 300 seconds. This load is low (*low-load*; one request sent every 1000 ms) or high (*high-load*; one request sent every 8 ms). Furthermore, the submitted functions are divided in *low-latency* ($10ms \leq SLA_{max\_lat} \leq 16ms$) and *high-latency* ($80ms \leq SLA_{max\_lat} \leq 125ms$). SLA_{max_lat} is randomly set between the given intervals.

4.2 Cost model

In our experiments, fog nodes adopt a cost model similar to the pricing model used by most cloud offerings (e.g., a fixed cost per vCPU per second). Allocating a resource unit at a given fog node incurs a fixed cost per second; we call this the unit cost. The unit cost typically decreases as the nodes are closer to the cloud [8]. Thus, a fog node i can estimate the cost of running a client's function for SLA_{dur} seconds using:
$cost_i(SLA) = SLA_{dur} \quad \times \sum_{res}^{[CPU,mem]} SLA_{res} \times UnitCost_{res,i}$

4.3 Placement quality

This section evaluates the quality of the function placement produced by the methods. We first examine the placed function ratio. This is defined as the number of provisioned functions over the number of submitted functions. Second, we look at the satisfaction rate, defined as the ratio of the number of requests that arrive under the $SLA_{max\_lat}+1ms$ threshold[8] over the total received number requests per function. Figure 2 shows the placed function ratio split into two buckets ($\approx$ 30 and $\approx$ 100 nodes). We use a two-way analysis of variance (ANOVA) test followed by a Tuckey honest significant difference test to reveal differences between the placement methods for the buckets. Differences are highlighted by the use of the compact letter display procedure, in which variables are statistically indistinguishable if and only if they share at least one letter. Edge-ward is the only method that provisions significantly fewer functions when the size of the network grows. Meanwhile, GIRAFF successfully places 80% of the functions on average, which is statistically indistinguishable from the other baselines, independently of the size of the network. Figure 3 displays the mean satisfaction rate. A one-way ANOVA followed by a Tuckey's test was used to analyze the data. It shows that in the case of both high-latency and low-latency functions, GIRAFF is indistinguishable from edge-furthest. In the case of high-latency functions, the GIRAFF satisfaction rate is 98.8% and edge-ward is 99.6%. However, in the case of low-latency functions, our method serves 94.7% of requests against 63.7% for edge-ward. In conclusion, GIRAFF reliably places as many functions as the other baselines, regardless of the size of the fog network. Furthermore, GIRAFF manages to satisfy 49% more requests directed to low-latency functions than edge-ward.

4.4 Client spending

This section examines economic performance from the point of view of the client. The economic performance of each placement method is given by the mean monetary spending to provision a function; it is displayed in figure 5. The figure features a one-way ANOVA test followed by a Tuckey's test for each function category. When considering high-latency functions, the client spends the least using GIRAFF. For the case of high-latency high-load functions, GIRAFF is on par with edge-ward. For high-latency low-load functions, edge-ward's client spending is 3 times that of GIRAFF. Low-latency low-load functions do not show any difference between the methods. Lastly, for low-latency high-load functions, our method is similar to edge-furthest, but results in 75% more spending than edge-ward. This is expected since edge-ward does not enforce the SLA_{max_lat} constraint. Since low-latency functions can be provisioned on fog nodes closer to the cloud, they have lower prices (cf. section 4.2), and edge-ward shows lower spending but fails to deliver proper SLA satisfaction, as shown in figure 3. The valid comparison for that category of functions is against edge-furthest, which is similar in client spending to our method.

4.5 Deployment time

This section examines the deployment time of functions. This is the time between the arrival of the client's SLA at the marketplace (step 2, figure 1) and the start of function provisioning on the selected node (step 5, figure 1). Figure 4 illustrates the mean deployment time for high-latency and low-latency functions. A one-way ANOVA test followed by a Tuckey's test reveals that the GIRAFF mean deployment time is 2 times that of edge-ward, with an average overhead of under 300ms in this experiment. However, this difference

[6] www.linux.org/docs/man8/tc-netem.html
[7] discovery.gitlabpages.inria.fr/enoslib/index.html
[8] $1ms$ is added to compensate for the lack of precision of the NTP clock and proxy overhead, cf. figure 1.

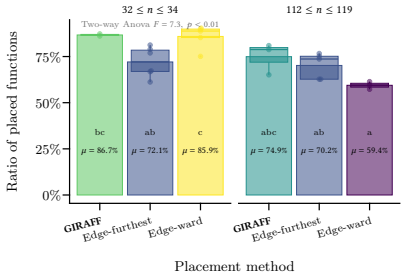

Figure 2: Ratio of placed functions for n-node fog networks

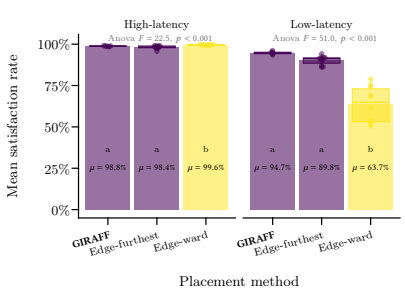

Figure 3: Mean satisfaction rate

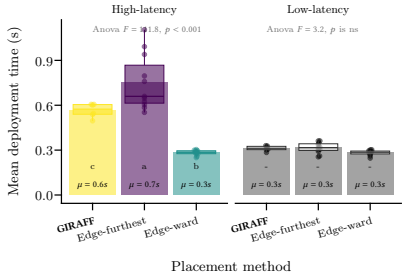

Figure 4: Mean deployment time of a function

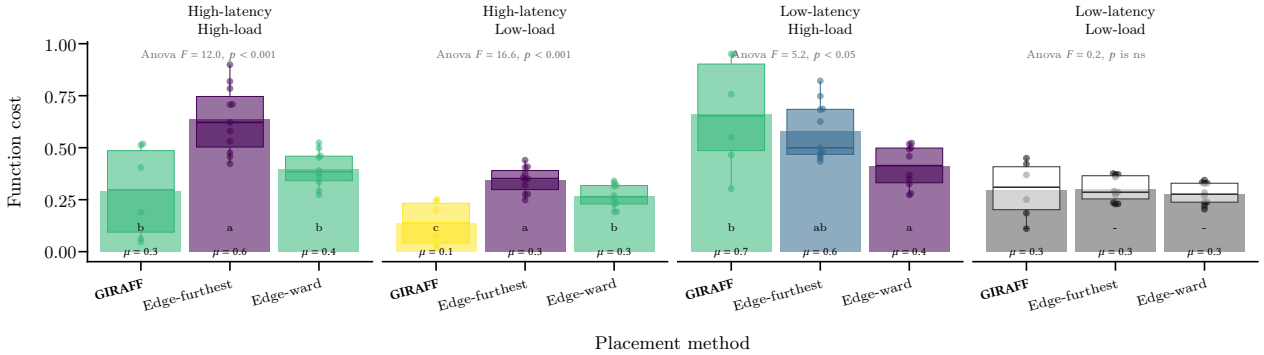

Figure 5: Mean cost of a function by type

is only present for high-latency functions whose SLAs travel deeper in the fog network.

5 CONCLUSION

This article proposes the GIRAFF approach to FaaS application placement in a multi-provider fog infrastructure. The approach uses an auction mechanism to distribute functions among independent fog nodes while ensuring that the latency and resource requirements of these functions are satisfied.

The approach was assessed using a reproducible open-source testbed running on Grid'5000. Within this environment, we run experiments with up to 119 fog nodes. In these experiments, we evaluated GIRAFF in comparison to two baseline methods, that is, edge-ward [9] and edge-furthest. We found that GIRAFF reduced by up to 3 times client spending on a function while maintaining the expected client's QoS. We also found that GIRAFF increased deployment time, specifically for functions with relaxed latency requirements. In conclusion, this study shows that the GIRAFF approach, featuring independent, competing fog providers, does not adversely affect performance compared to collaborative baselines. Our method makes the client spend less when possible, without being detrimental to the QoS.

The study has revealed numerous avenues for further research. We first intend to investigate the placement of directed acyclic graphs of functions. We also intend to allow for a more opportunistic approach to executing functions, in addition to always reserving resources. These additions will increase the versatility and applicability of our proposed system to more realistic fog computing scenarios.

ACKNOWLEDGMENTS

The experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations. GNU parallel [20] has been used to ease parallelization across the contribution.

We thank the reviewers whose valuable comments improved the paper.

REFERENCES

[1] Onur Ascigil, Argyrios G. Tasiopoulos, Truong Khoa Phan, Vasilis Sourlas, Ioannis Psaras, and George Pavlou. 2022. Resource provisioning and allocation in function-as-a-service edge-clouds. IEEE Transactions on Services Computing. DOI: 10.1109/TSC.2021.3052139.

[2] Mohammad S. Aslanpour et al. 2021. Serverless edge computing: vision and challenges. In Proceedings of the 2021 Australasian Computer Science Week Multiconference. DOI: 10.1145/3437378.3444367.

[3] Luciano Baresi, Davide Yi Xian Hu, Giovanni Quattrocchi, and Luca Terracciano. 2022. Neptune: network- and gpu-aware management of serverless functions at the edge. In Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems. DOI: 10.1145/3524844.3528051.

[4] David Bermbach, Jonathan Bader, Jonathan Hasenburg, Tobias Pfandzelter, and Lauritz Thamsen. 2022. AuctionWhisk: Using an auction-inspired approach for function placement in serverless fog platforms. Softw. Pract. Exp. DOI: 10.1002/spe.3058.

[5] Alessandro Bocci, Stefano Forti, Gian Luigi Ferrari, and Antonio Brogi. 2021. Placing faas in the fog, securely. In Proceedings ITASEC.

[6] Alessandro Bocci, Stefano Forti, Gian-Luigi Ferrari, and Antonio Brogi. 2023. Declarative Secure Placement of FaaS Orchestrations in the Cloud-Edge Continuum. Electronics. DOI: 10.3390/electronics12061332.

[7] Pak-Sing Choi and Felix Munoz-Garcia. 2021. Second-Price Auctions. In Auction Theory: Introductory Exercises with Answer Keys. DOI: 10.1007/978-3-030-69575-0_1.

[8] Pedram Farzin, Sadoon Azizi, Mohammad Shojafar, Omer Rana, and Mukesh Singhal. 2022. FLEX: A Platform for Scalable Service Placement in Multi-Fog and Multi-Cloud Environments. In Proc. 2022 Australas. Comput. Sci. Week. DOI: 10.1145/3511616.3513105.

[9] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2017. Ifogsim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Software: Practice and Experience. DOI: 10.1002/spe.2509.

[10] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Fog computing: a taxonomy, survey and future directions. In Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives. DOI: 10.1007/978-981-10-5861-5_5.

[11] Deepak Puthal, Mohammad S. Obaidat, Priyadarsi Nanda, Mukesh Prasad, Saraju P. Mohanty, and Albert Y. Zomaya. 2018. Secure and Sustainable Load Balancing of Edge Data Centers in Fog Computing. IEEE Commun. Mag. DOI: 10.1109/MCOM.2018.1700795.

[12] Houming Qiu, Kun Zhu, Nguyen Cong Luong, Changyan Yi, Dusit Niyato, and Dong In Kim. 2022. Applications of auction and mechanism design in edge computing: a survey. IEEE Transactions on Cognitive Communications and Networking. DOI: 10.1109/TCCN.2022.3147196.

[13] Thomas Rausch, Waldemar Hummer, Christian Stippel, Silvio Vasiljevic, Carmine Elvezio, Schahram Dustdar, and Katharina Krösl. 2021. Towards a platform for smart city-scale cognitive assistance applications. In 2021 IEEE VRW. DOI: 10.1109/VRW52623.2021.00066.

[14] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. 2021. Optimized container scheduling for data-intensive serverless edge computing. Future Generation Computer Systems. DOI: 10.1016/j.future.2020.07.017.

[15] Gabriele Russo Russo, Tiziana Mannucci, Valeria Cardellini, and Francesco Lo Presti. 2023. Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum. In 2023 IEEE Int. Conf. Pervasive Comput. Commun. PerCom. DOI: 10.1109/PERCOM56429.2023.10099372.

[16] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson. 2021. What serverless computing is and should become: the next phase of cloud computing. Commun. ACM. DOI: 10.1145/3406011.

[17] Mohammad Shojafar and Mehdi Sookhak. 2020. Internet of everything, networks, applications, and computing systems (IoENACS). Int. J. Comput. Appl. DOI: 10.1080/1206212X.2019.1575621.

[18] Sven Smolka and Zoltán Ádám Mann. 2022. Evaluation of fog application placement algorithms: a survey. Computing. DOI: 10.1007/s00607-021-01031-8.

[19] Balázs Sonkoly, János Czentye, Márk Szalay, Balázs Németh, and László Toka. 2021. Survey on placement methods in the edge and beyond. IEEE Communications Surveys and Tutorials. DOI: 10.1109/COMST.2021.3101460.

[20] [SW] Ole Tange, GNU Parallel 20230622 ('Nova Kakhovka') 2023. doi: 10.5281/zenodo.8051271.