DFaaS: Decentralized Function-as-a-Service for Federated Edge Computing

Michele Ciavotta, Davide Motterlini, Marco Savi, Alessandro Tundo
Department of Informatics, Systems and Communication, University of Milano - Bicocca, Milano, Italy
{michele.ciavotta, marco.savi, alessandro.tundo}@unimib.it, d.motterlini@campus.unimib.it

***Abstract*—Edge Computing pushes cloud capabilities to the edge of the network, closer to the users, to address stringent Quality-of-Service requirements and ensure more efficient bandwidth usage. Function-as-a-Service appears to be the most natural service model solution to enhance Edge Computing applications' deployment and responsiveness. Unfortunately, the conventional FaaS model does not fit well in distributed and heterogeneous edge environments, where traffic demands arrive to (and are served by) edge nodes that may get overloaded under certain traffic conditions or where the access points of the network might frequently change, as for mobile applications. This short paper tries to fill this gap by proposing DFaaS, a novel decentralized FaaS-based architecture designed to autonomously balance the traffic load across edge nodes belonging to federated Edge Computing ecosystems. DFaaS implementation relies on an overlay peer-to-peer network and a distributed control plane that takes decisions on load redistribution. Although preliminary, results confirm the feasibility of the approach, showing that the system can transparently redistribute the load across edge nodes when they become overloaded.**

***Index Terms*—Function-as-a-Service, Edge Computing, Edge Federation, Load Balancing, Peer-to-Peer Networking**

I. Introduction

Edge Computing brings cloud capabilities to the edge of the network, in proximity to the users, making it possible to simultaneously address stringent Quality-of-Service (QoS) requirements and ensure more efficient bandwidth usage [1], [2]. Nowadays, examples of Edge Computing solutions are plenty and range from Smart City to Industry 4.0 applications, introducing significant challenges and many open problems [2], [3]. *Function-as-a-Service* (FaaS) is a recent cloud service model that can be successfully applied in Edge Computing as it reduces time-to-market, simplifies deployment, and ensures better application responsiveness [4]. *Serverless functions* can be shared across multiple applications, and their fine-grained nature allows for more accurate Service Level Agreement (SLA) management, load balancing, and resource planning. Finally, FaaS can help in optimizing the edge operating costs by increasing resource utilization [5].

However, conventional FaaS is not applicable in a distributed edge environment due to dynamic traffic demands and limited resources [6]. To keep the advantages of this approach, it is necessary to federate execution environments and distribute the load among the network edge nodes, which may or may not belong to different tenants. This is especially important in a dynamic environment where the workload shifts from one edge node to another, as in mobile applications scenarios [7]. In cases like those, it is not economically viable to provision every node to handle peak loads. Still, one can consider leveraging the unused edge network capacity to support struggling nodes [8]. Ultimately, a federated and decentralized approach appears to be the most suitable design choice to ensure the extensibility and resilience of the edge network in the face of reduced human intervention [9].

In this work we propose *DFaaS*, a novel decentralized FaaS-based architecture designed to autonomously balance the traffic load across edge nodes belonging to federated edge FaaS ecosystems. DFaaS relies on an overlay peer-to-peer network to share information about the network topology, latency, and node load states. These pieces of information are exploited by overloaded nodes to redirect part of the ingress FaaS execution requests to unloaded peers.

The rest of this paper is structured as follows. The relevant literature on the subject is reviewed in Section II. A reference scenario is described in Section III. Section IV introduces the main architectural elements of the proposed solution while Section V presents some preliminary results. Finally, future work is discussed and conclusions are drawn in Section VI.

II. Related Work

Although serverless computing and FaaS have been a hot research topic for a few years [3] and many commercial [10] and open source [11] platforms are currently available, little attention has been paid to distributed/decentralized solutions that could be adopted in edge federations. Pfandzelter *et al.* [12] propose a lightweight FaaS system that is specifically designed for edge environments. Their platform is designed to run on low-performance single-machine edge nodes, but their research does not focus on approaches for federating FaaS systems at the edge. Two works introduce novel strategies for resource allocation and load balancing in distributed FaaS platforms. HoseinyFarahabady *et al.* [13] propose an algorithm to be executed by a predictive centralized controller, while Cho *et al.* [14] propose a Reinforcement Learning (RL) approach, where a centralized policy trainer component trains the RL agent. Our research shares a similar goal, but our FaaS-based proposal is fully decentralized.

From the architectural viewpoint, Soltani *et al.* [15] report a peer-to-peer serverless architecture for multi-cloud deployments based on Kubernetes. The solution is platform-dependent and does not tackle load balancing aspects. Vasconcelos *et al.* [16] propose a distributed architecture for the deployment of FaaS platforms in federations of cloud

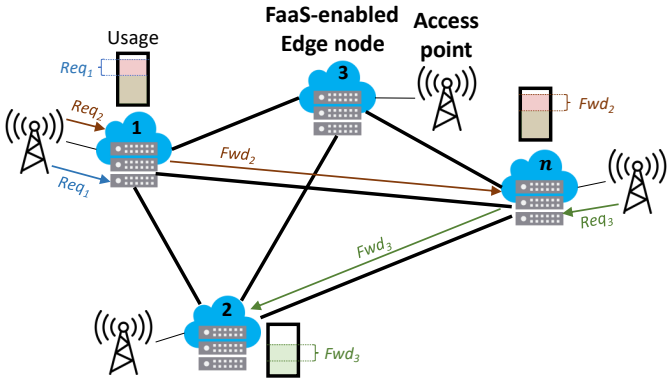

Fig. 1. Reference scenario: decentralized FaaS for federated Edge Computing.

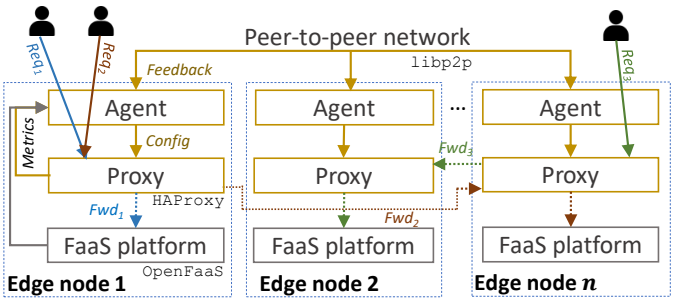

Fig. 2. DFaaS platform: proposed architecture.

providers. Our architecture is similar but with a substantial difference: while Vasconcelos *et al.* [16] envision a single FaaS platform that spans multiple domains and requires a centralized resource allocator, we conceive a federation of independent FaaS platforms, each autonomously managing an edge node (or a cloudlet) and executing a load balancing distributed algorithm. Finally, Cicconetti *et al.* [17] propose a framework where network nodes dispatch stateless tasks to federated executors (i.e., edge nodes) while ensuring fairness. While similar in scope, our work considers a slightly different scenario, where a specific edge node is always responsible for a task execution, but can transparently offload it to others.

III. Reference Scenario

Figure 1 depicts the considered network scenario. A set of geographically-distributed *FaaS-enabled edge nodes* (or simply *edge nodes*) is deployed at the edge of the access network. Each of these nodes deploys a *DFaaS platform* for the execution of *serverless functions*, and is connected to a wireless or wired *access point* (e.g., a base station, a broadband network gateway, or a WiFi access point).

The edge node can receive function execution *requests* (e.g. in the form of HTTP requests) generated by the *users* served by the access point. From an ownership perspective, an edge node (or a subset of edge nodes) is deployed and managed by an *edge provider*, which is usually a network operator, but it can also be a third-party entity offering computing services. Different edge nodes (owned by the same or different edge providers) are inter-connected by *network connections*, which may be best-effort or subject to SLAs.

This scenario recalls the one described by Faticanti *et al.* [18]. However, our main goal is to manage serverless function execution. In particular, a *federated edge computing ecosystem* is considered, where multiple edge providers, each one managing the DFaaS platform across a set of edge computing nodes, join the federation and can transparently rely on available computational resources from edge nodes managed by other providers if needed, e.g. each time their own nodes become overloaded due to traffic fluctuations.

A. Example of operation

Figure 1 reports a simple example where three requests, named $\texttt{Req}_{1,2,3}$, are sequentially delivered to two different edge nodes (i.e., $\texttt{Req}_{1,2}$ to edge node 1 and $\texttt{Req}_3$ to edge node n). When $\texttt{Req}_1$ arrives to node 1, the node has enough capacity to accommodate it. As a matter of example, we here refer to *capacity* as the marginal service rate per function class, but other definitions of capacity can be considered (e.g., based on CPU utilization or service queue fill level). We assume that, as $\texttt{Req}_1$ is enqueued, node 1 becomes overloaded and cannot accommodate other requests. Thus, $\texttt{Req}_2$ is transparently forwarded ($\texttt{Fwd}_2$) to another node (i.e., node n) with enough capacity. After enqueuing this request, also node n becomes overloaded, and a subsequent $\texttt{Req}_3$ to node n needs to be forwarded ($\texttt{Fwd}_3$) to another node (i.e., node 2). How requests are balanced is defined by a *load balancing algorithm* running in the edge nodes and using information collected from others.

IV. System Architecture

Figure 2 represents the high-level architecture of the proposed platform (DFaaS). In particular, for the sake of modularity, maintainability, and evolvability, we built DFaaS using preexisting, production-tested, and actively maintained components (i.e., the *Proxy* and the *FaaS platform*) and implemented distributed and federated control functionalities from scratch within the *Agent* component. The code is released open source.[1] A short description of the main components follows.

Agents are identical, independent, and deployed on individual edge nodes (or cloudlets); they interact through a peer-to-peer (P2P) overlay network. We implemented the P2P network using *libp2p*[2], which relies on the InterPlanetary File System (IPFS) protocol and on Distributed Hash Tables (DHTs). Each agent is responsible for:

1) Creating and managing the P2P network, that is, discovering other agents, noticing their disappearance, and communicating topology updates to its circle of connections;
2) Monitoring different *metrics* to disclose the state of the FaaS platform in terms of resource utilization and QoS for different classes of serverless functions;
3) Predicting the incoming load in the next time slot, calculating the resource demand for individual classes of functions;
4) Negotiating with neighboring nodes the resources to support the predicted load in the next time slot (requesting or providing resources) by means of *feedback* messages;

[1] https://unimibinside.github.io/dfaas/
[2] https://libp2p.io/

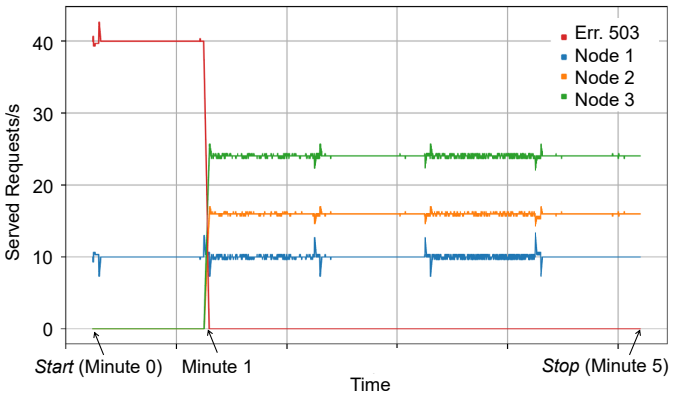

Fig. 3. Per node service rate (req/s).

5) Configuring the Proxy so that it redirects traffic to neighboring nodes according to the agreements made.

The **Proxy** (viz. *HAProxy* [19]) is responsible for receiving requests comprising the incoming load at the node and the actual load distribution among the edge network peers visible to the *Agent*. Currently, the Proxy only supports HTTP requests (also forwarded by other Proxies), but can be easily extended to support other protocols. Then, for each class of serverless functions, it partitions the load among the local FaaS platform and those running on the other nodes in the network.

Requests are finally served by functions managed by the **FaaS platform** (implemented in this work using *OpenFaaS* [11], but other platforms could be adopted), which embodies a set of services able to serve requests by instantiating and executing functions. The *FaaS platform* can be deployed locally (on the same node as the *Agent* and *Proxy*) or run externally on one or more computational nodes (if the edge node in question is a cloudlet).

V. Preliminary Results

This section is dedicated to present some preliminary experiments performed on the DFaaS platform. Please notice that the proposed platform is still under intensive development; for this reason, some features such as load forecasting have not been completed yet, while others are available in a preliminary and simplified form (such as the distributed algorithm for load balancing). Therefore, such experiments are only meant to validate the correct functioning of the components of the platform (Agent, Proxy, FaaS platform) and their interactions.

On these grounds, we deployed three virtual edge nodes (Node 1, Node 2, and Node 3) on a physical machine (Intel Core i7, 16 GB RAM). The Agent and Proxy components are executed as Docker containers within each node. The FaaS Platform (OpenFaaS) also runs on each virtualized node and is configured to only execute a single serverless function.

Each Agent i executes a load balancing algorithm in two phases; in the first one, it periodically calculates the slack capacity available for external requests (r^i_{ext}) and advertises it among its peers (along its state, normal/overloaded). In the second phase, a fraction of the ingress requests is forwarded to other nodes if the service rate r^i_{loc} overcomes a threshold r^i_{max}. If this condition occurs, the node i enters an *overloaded* state and starts forwarding excess requests to other nodes. The excess traffic $r^i_{fwd} = r^i_{loc} - r^i_{max}$ is proportionally balanced among the other nodes (not in *overloaded* state) according to their available slack capacity.

In our test, we set $r^{\{1,2,3\}}_{max} = \{10, 20, 30\}$ req/s. We then used the Vegeta HTTP load testing tool[3] to generate $r^1_{loc} = 50$ req/s towards Node 1. Nodes 2 and 3 did not receive any local request, meaning that $r^{\{2,3\}}_{loc} = \{0, 0\}$ req/s and $r^{\{2,3\}}_{ext} = \{20, 30\}$ req/s. Node 1 should thus enter the *overloaded* state, being $r^1_{fwd} = 40$ req/s. r^1_{fwd} should then be proportionally balanced between Nodes 2 and 3 according to r^i_{ext}, i.e., $w_2 = r^2_{ext}/(r^2_{ext} + r^3_{ext}) = 0.4$ and $w_3 = r^3_{ext}/(r^2_{ext} + r^3_{ext}) = 0.6$, leading to $r^1_{fwd}(2) = w_2 \cdot r^1_{fwd} = 16$ req/s and $r^1_{fwd}(3) = w_3 \cdot r^1_{fwd} = 24$ req/s.

The test lasted 5 minutes (15000 generated requests). We set 1 minute as *update interval* of r^i_{ext} and of any state change check. Figure 3 confirms the ability of DFaaS to distribute the excess traffic among nodes, as expected, starting from minute 1. It can be seen that during the first minute only 10 req/s were served by Node 1, and that the remaining 40 req/s obtained a `503 Service Unavailable` response. This happened because of the long update interval: as soon as the test started and Node 1 became overloaded, still one minute had to pass before the agent could notify its state to the neighbours and start redistribute the excess traffic. Once load balancing occurs, no more `503` responses are recorded. Clearly, with a higher update frequency, fewer `503` responses would have been generated, but a higher communication overhead would have been put on the P2P network. Concerning latency, the minimum experienced was 301 μs, the maximum 91.1 ms and the 99th percentile 11.4 ms. These values, given the experimental setup, can be considered acceptable.

VI. Conclusion and Future Work

In this short paper we proposed DFaaS, a decentralized FaaS-based architecture for federated edge computing environments. DFaaS relies on an overlay peer-to-peer network to exchange information on node state and on a distributed balancing algorithm that makes it possible to distribute incoming function execution requests across edge computing nodes.

Despite a fully working prototype of DFaaS is available for the scientific community to test, it needs to be improved and extended in different directions. Most importantly, the load forecasting mechanism has to be completed and the load distribution logics massively improved. We plan to adopt distributed Reinforcement Learning to enable QoS-aware proactive load balancing, as current reactive decision making approach may be ineffective in the case of frequent and noticeable load variations. We also plan to perform more exhaustive non-functional tests on throughput and latency, to investigate security implications of the proposed solution, and to extend it to support other protocols than HTTP.

[3] https://github.com/tsenart/vegeta

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] I. Baldini, P. Castro, K. Chang *et al.*, "Serverless Computing: Current Trends and Open Problems," *Research Advances in Cloud Computing*, pp. 1–20, 2017.

[4] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The Rise of Serverless Computing," *Commun. ACM*, vol. 62, no. 12, p. 44–54, 2019.

[5] G. Adzic and R. Chatley, "Serverless Computing: Economic and Architectural Impact," in *Joint Meeting on Foundations of Software Engineering*, 2017.

[6] G. Kousiouris and D. Kyriazis, "Functionalities, Challenges and Enablers for a Generalized FaaS based Architecture as the Realizer of Cloud/Edge Continuum Interplay," in *CLOSER*, 2021.

[7] C. Cicconetti, M. Conti, A. Passarella, and D. Sabella, "Toward Distributed Computing Environments with Serverless Solutions in Edge Systems," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 40–46, 2020.

[8] X. Cao, G. Tang, D. Guo, Y. Li, and W. Zhang, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1116–1129, 2020.

[9] R. F. Hussain, M. A. Salehi, A. Kovalenko, Y. Feng, and O. Semiari, "Federated Edge Computing for Disaster Management in Remote Smart Oil Fields," in *IEEE HPCC/SmartCity/DSS Conferences*, 2019.

[10] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms," in *IEEE CloudCom Conference*, 2017.

[11] S. K. Mohanty, G. Premsankar, and M. di Francesco, "An Evaluation of Open Source Serverless Computing Frameworks," in *IEEE CloudCom Conference*, 2018.

[12] T. Pfandzelter and D. Bermbach, "tinyFaaS: A Lightweight FaaS Platform for Edge Environments," in *IEEE ICFC Conference*, 2020.

[13] M. HoseinyFarahabady, Y. C. Lee, A. Y. Zomaya, and Z. Tari, "A QoS-Aware Resource Allocation Controller for Function as a Service (FaaS) Platform," in *Conference on Service-Oriented Computing*, 2017.

[14] C. Cho, S. Shin, H. Jeon, and S. Yoon, "QoS-Aware Workload Distribution in Hierarchical Edge Clouds: A Reinforcement Learning Approach," *IEEE Access*, vol. 8, pp. 193 297–193 313, 2020.

[15] B. Soltani, A. Ghenai, and N. Zeghib, "Towards Distributed Containerized Serverless Architecture in Multi Cloud Environment," *Procedia Computer Science*, vol. 134, pp. 121–128, 2018.

[16] A. Vasconcelos, L. Vieira, Í. Batista *et al.*, "DistributedFaaS: Execution of Containerized Serverless Applications in Multi-Cloud Infrastructures," in *CLOSER*, 2019.

[17] C. Cicconetti, M. Conti, and A. Passarella, "A Decentralized Framework for Serverless Edge Computing in the Internet of Things," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2166–2180, 2021.

[18] F. Faticanti, M. Savi, F. Pellegrini *et al.*, "Deployment of Application Microservices in Multi-Domain Federated Fog Environments," in *COINS Conference*, 2020.

[19] A. B. Prasetijo, E. D. Widianto, and E. T. Hidayatullah, "Performance Comparisons of Web Server Load Balancing Algorithms on HAProxy and Heartbeat," in *ICITACEE Conference*, 2016.