In-Network Computing with FaaS at the Edge

Claudio Cicconetti, Marco Conti, and Andrea Passarella

Abstract—**Offloading computation from user devices to nodes with processing capabilities at the edge of the network is a major trend in today's network/service architectures. At the same time, serverless computing has gained a huge traction among the cloud computing technologies and has, thus, promoted the adoption of Function-as-a-Service (FaaS). The latter has some characteristics that make it generally suitable to edge applications, except for its cumbersome support of stateful applications. This work is set to provide a broad view on the options available for supporting stateful FaaS, which are distilled into four reference execution models that differ on where the state resides. While further investigation is needed to advance our understanding of the opportunities offered by in-network computing through stateful FaaS, initial insights are provided by means of a qualitative analysis of the four alternatives and their quantitative comparison in a simulator.**

Index Terms—**Edge computing, Serverless, Function-as-a-Service, Distributed computing, In-network intelligence**

I. Introduction

Today we are witnessing the transition of service provisioning from cloud-centric to edge-centric [1]: centralizing storage and processing in data centers is now showing its limitations for a wide class of scenarios where a significant amount of traffic is generated/consumed at the edge of the network, or which include applications that cannot tolerate the latencies associated to reaching the cloud [2]. All these applications require offloading (part of) their computation tasks to external elements with processing capabilities, e.g., because their local resources are CPU- or memory-limited or to save power since they have constrained energy availability [3]. Furthermore, they do not always need long-term storage of their transactions. Thus, in-network processing is a much better option than centralizing computation in a cloud platform, since this keeps processing close to where data are actually used, which can be also an additional benefit in terms of privacy [4].

Serverless computing is thriving among cloud providers as the next step of evolution from Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). With serverless, the service providers are offered an extremely light abstraction of the physical computation resources: they simply need to upload a container image, while orchestration, monitoring, and scaling are handled by the platform [5]. All serverless platforms support a programming model called *Function as a Service (FaaS)*, where the basic unit of computation is the *function*: developers can focus on the implementation of elementary units with well-defined input vs. output, while providers compose such functions into chains of invocations to offer complex services to the end users [6]. In principle, due to its programming simplicity, FaaS is a good candidate for in-network processing at the edge of the network, and some studies show promising results in this direction [7]. However, many practical applications are made of *stateful* tasks. In [**?**] the authors discuss the issues they had to face in the migration of some micro-service-based applications to serverless, which required defining *ad hoc* solutions for state management. Examples of applications include a real-time collaborative LaTeX editor, where the state is the document being edited, and an e-commerce application, where the state is the list of items in the basket. In Internet of Things (IoT) applications, which are even more relevant at the edge, commonly the tasks have a strong dependency from one another as they compose Machine Learning (ML) pipelines, where the state are the models or windows of observations [8]. To manage the state, cloud serverless platforms include backend services for state synchronization and data exchanges between function execution [10]: however, the use of these services incurs additional costs, increases vendor lock-in, and violates the underlying pure functional paradigm. Indeed, efficient state management has been identified as a critical open issue in the position paper [11]. At the edge, the above issues are exacerbated by the decentralized nature of computation elements over geographical distances, the limited capabilities of edge nodes (compared to high-end servers in a data center), and the possible presence of multiple competing platform providers in the same area [12].

In this paper we first very briefly survey serverless works in the literature that are particularly relevant to aspects related to edge computing systems (Sec. II). We then focus on the execution of stateful functions at the edge, for which we propose four models in Sec. III, which are compared via simulation in Sec. IV. This work sets the scene for further research activities on the emerging topic of in-network computing at the edge through serverless/FaaS, as discussed in Sec. V.

II. Related work

Serverless computing only recently began expanding towards edge deployments (see [12]). In addition to a more efficient execution of functions on edge devices, which have more limited capabilities than their cloud counterparts, deploying at the edge requires modifications to the system architecture. For instance, in [13] the authors investigate the possibility to decentralize control to local edge networks, as opposed to manage resources in a fully centralized manner. Also the problem of allocating applications, consisting of multiple interrelated functions, to edge nodes has been studied in some works, for instance, in [14] the authors propose a mathematical formulation of the problem taking into account different

All the authors are with the Institute of Informatics and Telematics (IIT) of the National Research Council (CNR), Pisa, Italy.

cost categories (activation, placement, proximity, sharing), for which they put forward offline and online heuristic algorithms.

Another key research challenge is how to support data-intensive applications: in [8] the authors focus on selecting the best edge nodes for the execution of functions that match the application requirements and data locations. In this work, instead, we explore the concept of stateful FaaS in an open manner, and envision the generic mechanisms to support stateful serverless applications at the edge. We have taken inspiration from [15], where we have proposed to dispatch stateless functions using lightweight brokers installed at each edge node, and [16], which defines an architecture where stateless micro-services can be deployed anywhere in the device-to-cloud continuum, while stateful components operate in the device or the cloud only.

III. Execution models of stateful FaaS

In this section, we introduce some terminology and fundamental concepts of cloud serverless. Afterwards, we explain in Sec. III-B how the fundamental design principles of cloud serverless are violated at the edge, and introduce four execution models of stateful FaaS (Sec. III-C), with a focus on execution of Directed Acyclic Graphs (DAGs) applications, see Sec. III-D.

A. Serverless in the cloud

The typical execution model of FaaS in cloud serverless is illustrated in Fig. 1. The clients invoke the execution of functions (e.g., $f(\cdot)$ and $g(\cdot)$ in the figure) by providing their input x_i and expecting a return value y_i. All the invocations are directed towards a logically centralized entry point, which performs load balancing towards one of the *workers*, i.e., instances of containers of matching type in a virtualized infrastructure, managed by an orchestrator.

A function may require accessing the state of an application: this is done through an external service providing persistence, such as a database; in our figure this is represented by the execution of $f(x_i, s_i)$ and $g(x_j, s_j)$, where s_i and s_j are the states bound to the application in execution at clients i and j, respectively. Commercial cloud service providers encourage using services in the same ecosystem to disguise stateful operations as stateless, since this increases billing and vendor lock-in. This model highlights the following implicit assumptions of serverless/FaaS:

1. *Access to the state is cheap:* in commercial systems the state repository in Fig. 1 is located in the same data center as the serverless platform, and the state usually consists of a small amount of data that can be retrieved/updated with a single query.
2. *Location of workers is irrelevant:* since all the workers run in containers managed by the same orchestration platform, the physical location of a worker in general can be assumed to have a negligible impact on the performance: in [17] the authors have found that cold-start effects are dominated by the creation and initialization of containers' namespaces, rather than other location-dependent effects.

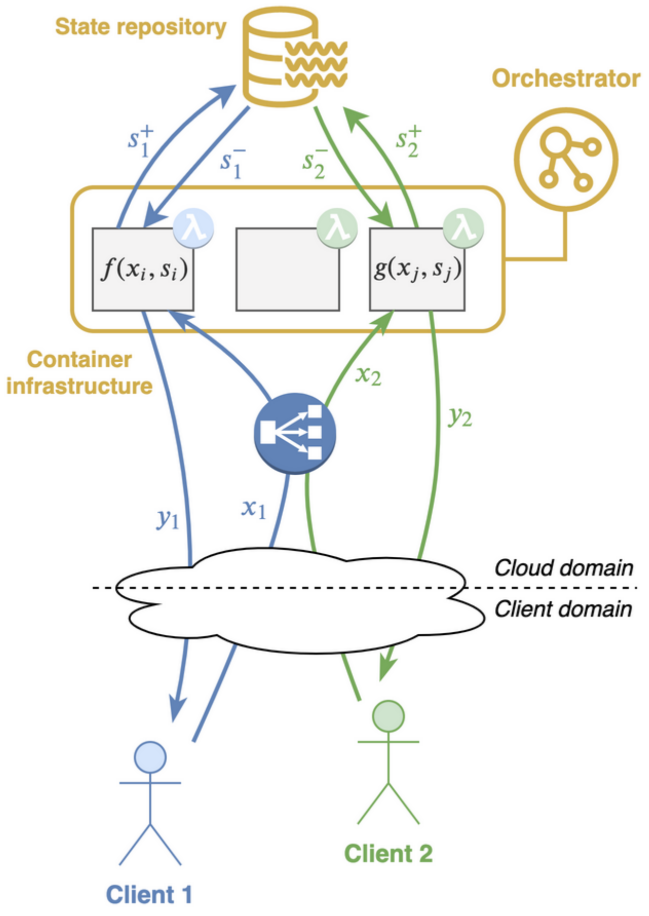

Fig. 1. FaaS execution model in serverless cloud.

3. *Location of clients is irrelevant:* as shown in Fig. 1, there is a full separation between the clients (in the Internet) and the serverless platform services (in the data center), their only point of contact being the load balancer, which is logically centralized and located within the cloud domain.

B. Serverless at the edge

A reference scenario of edge computing is illustrated in the top left part of Fig. 2: *edge nodes* have compute capabilities that can be used to provide services to *clients*, which reach them via access networks created by *access points* and internal connectivity provided by *edge network devices*. The scenario may represent a smart city with sensing nodes or an industrial IoT deployment with embedded devices with constrained resources.

Some of the edge elements may have an Internet connection, but, in general, reaching the cloud cannot be considered to be *cheap* like for cloud serverless. In fact, one of the driving motivations of edge computing is that it reduces the application latency since computation happens close to the clients: if the edge nodes have to reach the cloud to retrieve/update the application state as part of the function invocation procedure,

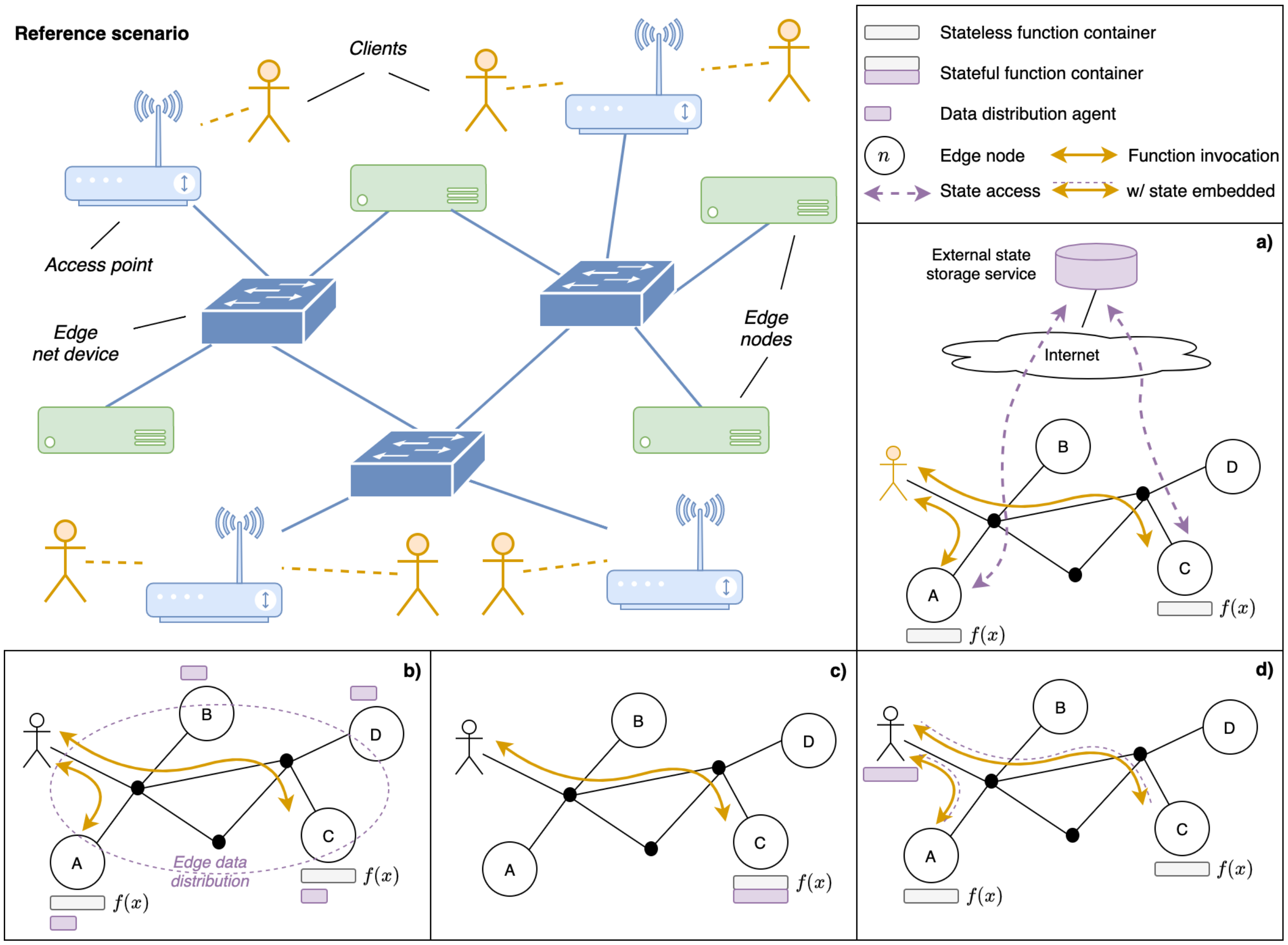

Fig. 2. Four stateful FaaS execution models in serverless edge: a) external; b) in-edge; c) in-function; d) in-client.

then the edge advantage may simply vanish (*violation of assumption 1.*).

Furthermore, edge networks may be very heterogeneous, in terms of both the compute capabilities of the nodes (ranging from single-board computers to full-fledged servers) and the communication links: therefore, the function invocation performance depends very much on the location of the worker, unlike what happens in a cloud deployment (*violation of assumption 2.*).

Finally, in edge networks there is no clear separation between clients and processing nodes: the users are interspersed in the same network connecting the edge nodes to one another. As a result, the location of the clients matters a lot in determining the overall quality of the performance they perceive, because the cost to reach a node is not uniform across all the clients (*violation of assumption 3.*).

In summary, *none of the design assumptions of cloud serverless (Sec. III-A) hold at the edge*: this is particularly problematic for stateful functions, since they are less flexible regarding their lifecycle and incur an additional overhead to access the state.

C. Stateful FaaS execution models

In the following we illustrate four models for the execution of stateful FaaS in an edge network, with different characteristics. This analysis is a step forward towards the definition of better solutions to support (stateful) FaaS at the edge under different application constraints and deployments: this is an open area of investigation, whose importance is steadily increasing thanks to the widespread diffusion of edge computing and the growing appeal of serverless for not only mobile but also IoT and data analytics applications.

The four models are shown with the help of the example in Fig. 2. The first model (Fig. 2a) is a straightforward transposition of cloud serverless to an edge network: the workers remain in fact stateless, as they access the state on demand via external services in the cloud. We thus call this model **external**. It has the advantage that a function developed for the cloud can be deployed at the edge without any modifications. However, there is a cost for reaching the state repository in the cloud, which has to be paid in terms of application latency and outbound traffic.

Our second model, called **in-edge** (see Fig. 2b), is similar to the previous one, but with an important difference: the functions remain stateless, but the state-associated data are kept by the edge nodes themselves. This removes the economic/performance burden of accessing external services, but requires *data distribution agents* to be deployed on heterogeneous and resource-constrained nodes, possibly with limited connectivity. An in-edge model is proposed, for instance, by Cloudstate (`https://cloudstate.io/`), which relies on a commercial system (Akka) to manage the applications' state in a distributed manner. However, to the best of our knowledge, a general-purpose definitive solution has not yet been found.

A different approach is shown in Fig. 2c, where the state of a function for a given application instance is kept within the worker itself, called **in-function**. In FaaS, the developer is not allowed to use local resources to keep persistent data, whereas with this model such a feature must be provided by the serverless platform and used by the programmer. In this model there is a mapping between a given application context and its worker, which outlasts a single invocation: we represent this in the figure with a single container being present in the edge network, towards which all functions invocations of that client must be directed. The in-function model is expected to perform better than the others, because it does not require any transfer of information for the function to retrieve/update the state and it allows exploiting data locality (e.g., hot caches). The in-function model can be realized via the integration of a system-wide serverless framework with an underlying Kubernetes (K8s), as in [8], which proposes to influence the scheduling of K8s by adjusting its internal weights based on metadata specified by the application developers.

Finally, in Fig. 2d we illustrate **in-client**, where the state is embedded in the function arguments (so that it can be read by the container executing the function) and in the return value (so that a modified state can be returned to the client). The client remains the sole owner of the state, whose ownership is delegated only temporarily for the execution of a single function invocation. This way, the containers are in fact stateless, as in external and in-edge, and consecutive invocations of the same function from the same client may happen on different edge nodes, as shown in the example. The traffic never leaves the edge network, but in-client may increase the amount of internal traffic generated because the state has to be piggybacked on every function invocation exchange, which is instead unnecessary with the other models. Furthermore, the serverless framework has to provide a means for the client to embed the state in the arguments and return value.

To summarize:

- All the models identified, except in-function, can be used jointly with scheduling policies to dispatch the function invocation requests towards any of the currently available containers that can serve requests of matching types: this can be exploited to follow short-scale variable load/network conditions as in [15].
- In-function, instead, uses stateful containers. Therefore, it must rely on container migration to optimize the performance by adapting to the changing environment; migration may incur a non-negligible cost in terms of unavailability, negatively impacting the quality perceived by the application [18], and hence cannot be done too frequently.
- In-client is the only model offering a straightforward opportunity to enable multi-tenant applications: since the client owns its data, it is possible to use multiple service providers at the same time, which might coexist in the same geographical area or even in the same virtualized infrastructure, thus contributing to the removal of dreaded vendor lock-ins.

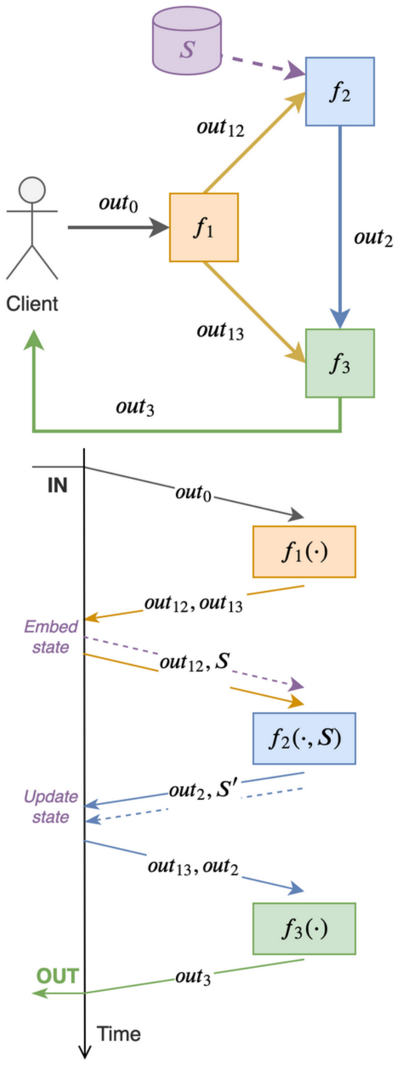

Fig. 3. Example of in-client FaaS execution of an application composed of three functions in a DAG, where only f_2 requires access to the application's state.

D. Execution of DAGs

One of the most appealing features of FaaS is that it allows the creation of complex workflows by combining functions, so that the input of a function is the output of one or more preceding functions arranged in a DAG. An example is shown in the top part of Fig. 3, where the final output to the user is given by $f_3(\cdot)$, which takes as input the output of $f_1(\cdot)$ and

$f_2(\cdot)$, the latter also depending on the persistent state S bound to the specific application instance.

All the models in Sec. III-B, illustrated in Fig. 2, can be adopted in a straightforward manner by a serverless platform that allows the composition of functions in DAGs. For the in-client model, we show in the bottom part of Fig. 3 the sequence of function invocation exchanges to realize the example DAG at the top of the figure. As can be seen, when the client invokes f_2 it embeds the state S in the function arguments together with out_{12}, i.e., the output of f_1 intended for f_2.

IV. Performance evaluation

We have carried out a high-level performance evaluation of the execution models described in Sec. III. Due to the absence of publicly available data from real edge deployments, we have created our dataset for the simulation using the technique of workload composition:

– for the network we have used "ether: Edge Topology Synthesizer", with a pre-configured topology model for an Industrial Internet of Things (IIoT) scenario [8], including a mix of hardware architectures and communications links;
– for the workload we have started from DAG traces generated by "Spår: Cluster Trace Generator" [19], which we have transformed into a linear chain of stateful function invocations, with CPU/memory requirements and hardware architecture affinity.

Our methodology follows a Monte Carlo approach: for a given set of configuration parameters we extract 200 random subsets of *jobs*, each made of a chain of heterogeneous function invocations with different length, and perform an assignment of function invocations to the edge nodes with processing capabilities. The assignment is done using a greedy online algorithm that selects for every function of every job (sorted randomly) the edge nodes that minimizes the response time, according to one of two *assignment policies*: (i) processing time only (`Load`), vs. (ii) processing time + transfer time for input arguments, return values, and states (`Load+net`). Such assignment assumes global knowledge of the network and task execution times. We then measure the response time, due to processing and network transfer, of each chain of function invocations, with the four execution models in Sec. III-B. For the *external* model we assume that access to the cloud has an RTT of 120 ms and a bandwidth of 50 Mb/s [20]; for the *in-edge* model, for simplicity we did not simulate a distributed storage system, but rather we assumed that the state repository is located in the central node of the topology. The simulator used has been released as *open source* and the instructions to fully reproduce the results are available on GitHub (`ccicconetti/serverlessonedge`, tag `v1.1.1`, simulations `002`).

In Fig. 4 we show the traffic generated per function chain with the four models as the load increases from 100 to 500 jobs, with the two function assignment policies. With `Load`, *in-client* execution incurs greatest network overhead, because the functions tend to be allocated always towards the center of the network, which has most powerful servers. However, with

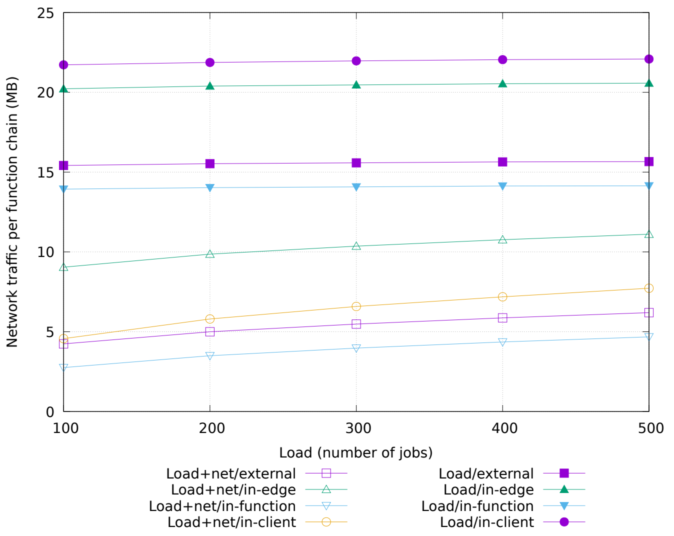

Fig. 4. Network traffic generated per function chain, with both allocation strategies and all execution models, and increasing load.

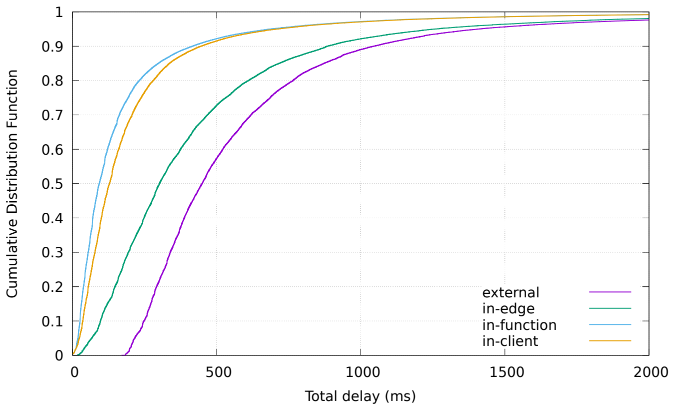

Fig. 5. Total latency (processing and network), with Load+net allocation strategy, all execution models, and 500 jobs.

`Load` the traffic is much higher than that with `Load+net` for all execution models, which may jeopardize the edge network resources and also negatively impact the quality experienced by the applications. Hence, in the following we focus on the `Load+net` allocation strategy only. With the latter, the traffic increases slightly with the load, and *in-edge* exhibits the worst performance, while all the other execution models have similar performance.

For the representative case of `Load+net` and 500 jobs, we report in Fig. 5 the total latency of all the jobs across all the random subsets, sorted and normalized in $[0, 1]$. As can be seen, *external* and *in-edge* yield a much higher latency than the others, with *in-function* performing only slightly better than *in-client*. The results show that *in-function* achieves best performance, which however comes at the cost of adding protocol complexity and limiting the opportunities for platform optimization, as explained in Sec. III-B. On the other hand, even though *in-client* incurs additional overhead due to the

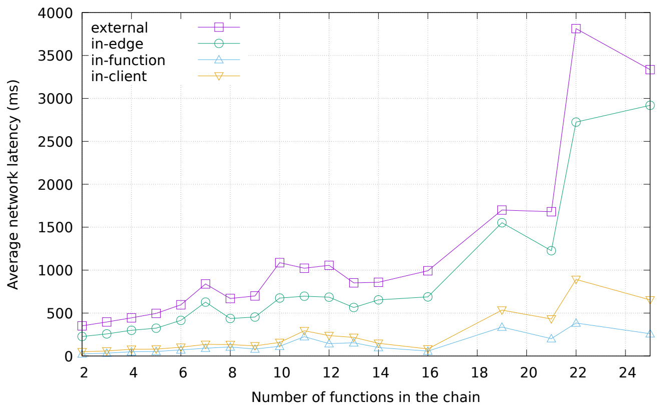

Fig. 6. Network latency, with Load+net allocation strategy, all execution models, and 500 jobs, per number of functions in the chain of invocations.

need of piggybacking the required state of a function on the input arguments and return value, in our scenario the overall function response times are smaller than with *external* and *in-edge*.

We conclude with a deeper insight on the latency due to the network transfers only in Fig. 6, combining together jobs with the same chain length. The sharp increase of all curves for longer function chains is because they tend to be more heavyweight in the Spår-generated traces, in accordance with the real ones from Alibaba. With all the execution models the average latency increases with the chain length; however, such an increase is more evident with the *external* and *in-edge* models. Finally, the *in-client* strategy deviates in a noticeable manner from *in-function* only for the jobs with longest chains.

V. Conclusions and Outlook

In this paper we have provided an overview of the nascent topic of stateful FaaS to support in-network computing. We have identified four reference execution models: *external*, where persistence is provided by an external service in the cloud (current standard); *in-edge*, which envisions distributing the data within the edge nodes themselves; *in-function*, where the state remains into a stateful instance of the function's container; and, *in-client*, where the client's device is the sole owner of the application's state. The four models have different implications on the architecture and protocols, and we envisage that further research is required to determine which one is best for a given deployment or set of edge applications. As an initial step in this direction, we have built a trace-driven simulator to identify the traffic overhead and latency. Our results have shown that *in-client* is very promising since it can surpass *external* and *in-edge*, while providing users with almost same performance as *in-function*, despite being simpler and more amenable to system-wide optimization.

Acknowledgment

This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337, project MARVEL.

References

[1] Mark Campbell. Smart Edge : The Center of Data Gravity Out of the Cloud. *Computer*, 52(December):99–102, 2019.

[2] Umakishore Ramachandran, Harshit Gupta, Adam Hall, Enrique Saurez, and Zhuangdi Xu. A Case for Elevating the Edge to be a Peer of the Cloud. *GetMobile: Mobile Computing and Communications*, 24(3):14–19, 2021.

[3] Yaqiong Liu, Mugen Peng, Guochu Shou, Yudong Chen, and Siyu Chen. Toward Edge Intelligence: Multiaccess Edge Computing for 5G and Internet of Things. *IEEE Internet of Things Journal*, 7(8):6722–6747, 2020.

[4] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-Network Computation is a Dumb Idea Whose Time Has Come. *Proc. HotNets*, 2017.

[5] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The Rise of Serverless Computing. *Commun. ACM*, 62(12):44–54, nov 2019.

[6] Ioana Baldini, Perry Cheng, Stephen J. Fink, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Philippe Suter, and Olivier Tardieu. The serverless trilemma: Function composition for serverless computing. *Proc. Onward!*, 2017.

[7] Claudio Cicconetti, Marco Conti, Andrea Passarella, and Dario Sabella. Toward Distributed Computing Environments with Serverless Solutions in Edge Systems. *IEEE Communications Magazine*, 58(3):40–46, mar 2020.

[8] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems*, 114:259–271, 2021.

[9] Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back. *Proc. CIDR*, 2019.

[10] Djob Mvondo, E N S Lyon, Kevin Nguetchouang, Lucien Ngale, E N S Lyon, Renaud Lachaize, Tim Wood, Daniel Hagimont, and Alain Tchana. OFC : An Opportunistic Caching System for FaaS Platforms. *Proc. ACM EuroSys*, 2021.

[11] Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca A D A Popa, Joseph E Gonzalez, I O N Stoica, and David A Patterson. What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing. *Communications of the ACM*, 64(5):76–84, 2021.

[12] Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj K Gaire, and Schahram Dustdar. Serverless Edge Computing: Vision and Challenges. *Proc. AusPDC*, 2021.

[13] Onur Ascigil, Argyrios Tasiopoulos, Truong Khoa Phan, Vasilis Sourlas, Ioannis Psaras, and George Pavlou. Resource Provisioning and Allocation in Function-as-a-Service Edge-Clouds. *IEEE Transactions on Services Computing*, 1374(c):1–14, 2021.

[14] Lin Wang, Lei Jiao, Ting He, Jun Li, and Henri Bal. Service Placement for Collaborative Edge Applications. *IEEE/ACM Transactions on Networking*, pages 1–14, 2020.

[15] Claudio Cicconetti, Marco Conti, and Andrea Passarella. A Decentralized Framework for Serverless Edge Computing in the Internet of Things. *IEEE Transactions on Network and Service Management*, pages 1–1, 2020.

[16] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi. A unified model for the mobile-edge-cloud continuum. *ACM Transactions on Internet Technology*, 19(2), 2019.

[17] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. Agile cold starts for scalable serverless. *Proc. HotCloud*, 2019.

[18] Carlo Puliafito, Antonio Virdis, and Enzo Mingozzi. The Impact of Container Migration on Fog Services as Perceived by Mobile Things. *Proc. SMARTCOMP*, 2020.

[19] Huangshi Tian, Yunchuan Zheng, and Wei Wang. Characterizing and Synthesizing Task Dependencies of Data-Parallel Jobs in Alibaba Cloud. *Proc. SoCC*, 2019.

[20] Valerio Persico, Alessio Botta, Pietro Marchetta, Antonio Montieri, and Antonio Pescapé. On the performance of the wide-area networks interconnecting public-cloud datacenters around the globe. *Computer Networks*, 112:67–83, jan 2017.