Network Measurements with Function-as-a-Service for Distributed Low-latency Edge Applications

Emanuele Carlini
CNR-ISTI,
National Research Council
Pisa, Italy
emanuele.carlini@isti.cnr.it

Hanna Kavalionak
CNR-ISTI,
National Research Council
Pisa, Italy
hanna.kavalionak@isti.cnr.it

Patrizio Dazzi
University of Pisa,
Department of Computer Science
Pisa, Italy
patrizio.dazzi@unipi.it

Luca Ferrucci
CNR-ISTI,
National Research Council
Pisa, Italy
luca.ferrucci@isti.cnr.it

Massimo Coppola
CNR-ISTI,
National Research Council
Pisa, Italy
massimo.coppola@isti.cnr.it

Matteo Mordacchini
CNR-IIT,
National Research Council
Pisa, Italy
matteo.mordacchini@iit.cnr.it

ABSTRACT

Edge computing promises to bring computation and storage close to end-users, opening exciting new areas of improvement for applications with a high level of interactivity and requiring low latency. However, these improvements require careful scheduling of applications in the correct Edge resource. This decision is generally taken by considering multiple parameters, including the network capabilities. In this paper, we discuss an approach that measures latency and bandwidth between multiple clients and Edge servers. The approach is based on recent Serverless computing technologies, and it is meant as a support to take timely and correct scheduling decisions in the Edge. We also provide the description of a proof of concept implementation of the said approach.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Networks → Network performance analysis**.

KEYWORDS

Serverless computing, Edge Computing, Latency measurement

ACM Reference Format:
Emanuele Carlini, Hanna Kavalionak, Patrizio Dazzi, Luca Ferrucci, Massimo Coppola, and Matteo Mordacchini. 2022. Network Measurements with Function-as-a-Service for Distributed Low-latency Edge Applications. In *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge (FRAME '22), July 1, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3526059.3533622

FRAME '22, July 1, 2022, Minneapolis, MN, USA

ACM ISBN 978-1-4503-9310-2/22/07…$15.00
https://doi.org/10.1145/3526059.3533622

1 INTRODUCTION

In the current technological landscape, Cloud Computing represents a popular landing space for a very large number of applications. Two core motivations drive cloud adoption. Firstly, the cost-effectiveness for application providers that avoid buying and operating hardware infrastructures sized for peak loads. Secondly, the end-users enjoy cheaper and always-on services with lightweight clients. However, several classes of applications are hard to migrate to Cloud straightforwardly. For example, applications that require high interactivity still resist the migration en-masse to the Cloud due to the potential high latency from the clients to the remote Cloud servers.

In this context, the Edge computing paradigm represents a concrete attempt to bring the Utility Computing paradigm [1] (i.e., the idea of using and renting computing capacity as any other utility) to a fully distributed, and often decentralised, dimension. Ideally, end-users and application providers can take advantage of a pervasive and geographically sparse computing presence to offload or run computing tasks. The realisation of this vision is currently being approached by the academia with various research efforts [2, 12, 17, 22]. A common trait of this effort is the necessity to

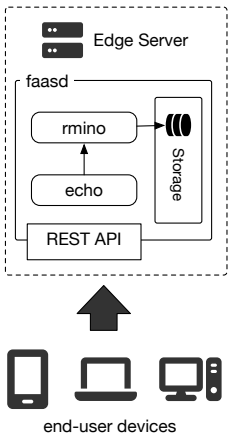

Figure 1: Overview of our solution

have solid scheduling strategies to enable a computing continuum that virtually ”follows” the users leaving a perception of consistent performance and high Quality of Experience (QoE). Scheduling decisions must be dynamic and adapt to the changes that stem from the applications (e.g., the sudden execution of a specific function that affects the QoE requirements) and the end-users (e.g., more users from various locations access the application). In this context, a scheduling decision has the duty to decide on which edge resource to deploy a particular application service in order to, for example, minimize the interaction latency with the end-user [6, 14, 15]. In this case, interactive and low-latency applications, such as video games, mixed and virtual reality, and voice-activated applications, can arguably extract the most advantage from this approach.

However, scheduling strategies are usually as good as the data which they are fed with. Besides the static computational characteristics of the various edge nodes (i.e. RAM, CPU, storage), dynamic values such as occupation and network performances are valuable information for the edge scheduler to take correct and timely scheduling decision. Therefore, there is the need to properly monitor and characterise computational edge resources, also in terms of network capabilities.

In this context, we envision a lightweight approach based on the Function-as-a-Service (FaaS) paradigm to measure and profile the network status between an edge resource and potential end-user devices (i.e., clients) The FaaS paradigm provides an on-demand computing architecture where functions are the main actor [18]. FaaS approaches show key advantages for this task. From a client perspective, it does not require the installation of complex or custom additional services. A simple web access and basic functionalities is all that is required. From the server perspective, it does not require to setup a dedicated infrastructure, the performance footprint is low, and the service can potentially scale with the number of clients.

In this paper, we provide a description of a service for the measurement of latency and bandwidth between a FaaS server and multiple client devices. The idea is to have a lightweight procedure that is able to run even on edge servers of limited capacity (e.g., ARM-based processors), typical of far-edge architectures [3]. We also show some initial observation from a proof of concept implementation, which is composed by two clients and a server.

2 RELATED WORK

Placing applications and services near the end-user to increase responsiveness is one of the most advertised features of edge computing [13]. Recent estimations report a reduction in latency in the order of 30% for edge computing infrastructures when compared to classical cloud ones [5] and that the number of users having very low latency is essentially doubled when using edge infrastructures [4]. Typically, placing applications near users has a geographical connotation: bringing services in the geographical proximity of the users to reduce the network latency between services and end-user devices. However, as stated in [11], "Physical distance alone does not always represent a good approximation of the propagation delay, given the possible effects of routing inefficiencies." The same work also reports that moving datacenters and applications close to users is only a part of the solution, which also requires "continuous monitoring [..] to quickly detect modifications and, when possible, put in place countermeasures".

Therefore, tools and mechanisms that provide network monitoring are often critical to driving cloud-edge application schedulers. Several approaches have been proposed over the years. Typically, prediction or interpolation mechanisms help avoid measuring all point-to-point connections between all possible clients and servers. Huang et al. [10] provides a general overview and identifies three types of prediction solutions: (i) coordinate-based approaches, (ii) path fitting approaches, and (iii) data-driven approaches. Our approach does not fit this classification, but it is a lightweight method that can assist as an alternative for point-to-point measurements in more complete solutions.

Among the most popular solutions, Gummadi et al. [8] propose King, a tool that uses recursive DNS queries to estimate latency between arbitrary hosts. Sharma et al. [19] present Netvigator, a tool to estimate network proximity between clients and servers. It uses a set of landmark nodes to avoid requiring to know the IPs of the servers. Song et al. [20] describe a system to monitor network capabilities that adapt the number of measurements according to the server’s load. Their approach also measures only a small part of the network to infer the entire network properties. Fu et al. [7] suggest a distributed monitoring method for network properties. They organize the nodes with a system of coordinates maintained in a decentralized fashion.

More recent approaches use statistical and data-driven techniques to measure latency and network properties indirectly. Zhang et al. [21] design an indirect method that estimates the latency between two arbitrary nodes using the measurements from selected clients near end nodes. Huang et al. [9] propose a data-driven prediction approach by leveraging a tensor-based network to predict distance with confidence intervals.

3 SERVERLESS SOLUTION

The conceptual overview of our solution is depicted in Figure 1. We envision edge servers installed with a FaaS solution. Clients call a specific REST API exposed by the FaaS service to trigger the profiling of the network, which for the bandwidth consists of the exchange of files of various dimensions. The profiling is repeated periodically to provide a historical overview of the main network characteristics. The observed network characteristics are both stored locally in the server (which may make them available to any scheduler with additional interfaces not covered here), and stored to end-user client devices. Clients requirements are minimal: network connectivity, the ability to call a REST interface, and some (minimal) storage to keep the files to be exchanged and the measurement results.

3.1 Selection of the FaaS framework

For our network measurement service we were looking for several specific requirements:

- *Ease of installation.* The service is supposed to be installed in edge nodes of different kind and nature, possibly in cohabitation with other services. Therefore a fast and easy installation is a core requirement;

Input: End point of the FaaS server
Input: *sizes*: an array of different file dimensions (must start with 0)
Input: *waitingTime*: time interval before subsequent invocations

```
1 i = 0;
2 while true do
3     Upload a file of size: sizes[i % sizes.length] and (b_UP, b_DOWN)^(i-1);
4     Download a file of size: sizes[i % sizes.length];
5     Save locally (b_UP, b_DOWN)^i;
6     i = i + 1;
7     Wait for waitingTime;
8 end
```

Algorithm 1: Client's pseudocode

	Server	Client 1	Client 2
Location	Pisa, Italy	Pisa, Italy	Catania, Italy
Dist. from server	N/A	0	800 km
Architecture	x86	VM	x86
Operating System	Ubuntu 20.04	Ubuntu 20.04	Windows 10
Network	N/A	Same server network	Residential Fiber-optic

Table 1: Machines used in the proof of concept implementation

- *Lightweight execution footprint.* Our system is also supposed to run on machine with limited capacity, such as ARM-equipped system-on-chips and mobile devices. In fact, we were not necessarily searching for support to complete orchestration frameworks, such as Kubernetes.
- *Monitoring facilities.* In particular, monitoring facilities that are already present in the framework. These facilities can help in monitoring further aspects that can be taken into account by the edge scheduler during the decision making process.
- *Support for dockerized functions.* Containerized functions are a de facto standard way to decouple the execution of some services from a particular environment. In the context of our services, this allow us to implement the measurements functions without sticking to a particular technology or programming language.

Many frameworks are currently available to deploy a FaaS Serverless service. Among those, we performed a selection by analyzing the literature to check what fit our requirements. Palade et al. [16] performed a qualitative and quantitative analysis of four different Serverless frameworks, namely Apache OpenWhisk[1], OpenFaaS [2], Kubeless [3], and Knative [4]. According to our requirements, Knative has been discarded due to some difficulties in the installation and a limited support for several programming languages, and Openwhisk due to the lacking of monitoring subsystems. Finally, since we planned to work with Docker images, OpenFaas looked like a good candidate. To further reduce the load footprint, we have chosen faasd [5], a lightweight OpenFaaS implementation that does not require Kubernets to work.

3.2 Client operations

The network characteristics measured between the server and a client are the round trip time (RTT) and the bandwidth. We assume that the client knows the end point of the server.

The measurement process (i.e. invocation of the *echo* function on the server) is initiated by the client (see Algorithm 1). Firstly, it initiates the dialog with the server by uploading a file. Once the transfer is completed, the same file is sent back (still via the echo function) from the server to the client. The client locally registers the total transfer time, for both the upload and the download phases. From the perspective of the server, the *echo* function implements both the upload and download of the file.

This process is iterative. Each iteration is repeated after a fixed time interval. At each iteration, the client sends to the server the upload and download transfer time observed in the previous iteration (b_{UP} and b_{DOWN} in the pseudocode). The server stores these values in the local storage (see Section 3.3). Moreover, at each iteration the size of the file is selected among a set of predefined sizes. When the file size is 0, the RRT is measured.

3.3 Server Storage

FaaS is at its core a stateless paradigm. A storage module had to be attached to the regular FaaS service for the network measurements to be saved and potentially accessed at a later time by the edge scheduling system. We implemented the storage as a MinIO service [6]. MinIO is an object storage service that operates with the *bucket* abstraction: a bucket is a data container that helps to categorize and organize the object stored. For our purposes, we have created a bucket for each unique client to store the information about bandwidth and latency. The idea is for the server to have timeseries of measurements that can be later used to perform statistics on the behaviour of the network. Once a session of network measurements has been performed (i.e. one invocation of the echo function), these are saved in the corresponding client's bucket in the JSON format.

The communication with the MinIO is realised through an additional Faas function that invokes the MinIO's REST endpoint. This allows for the writing of the data to proceed asynchronously with respect to the main network measurement function.

3.4 Proof of concept implementation

We setup a proof of concept implementation of the approach using the machines in Table 1. We have considered the following file sizes: 0, 5, 10, and 30 MBs. The invocation delay is set to 1 minute. We run the client for 24 hours.

Results for bandwidth of Client 2 (file size > 0) are shown in Figure 2. The results are pretty straightforward for a residential fiber-optic network. The upload bandwidth is quite stable around

[1] https://openwhisk.apache.org/
[2] https://www.openfaas.com/
[3] https://github.com/vmware-archive/kubeless
[4] https://knative.dev/docs/
[5] https://github.com/openfaas/faasd
[6] https://min.io/

Figure 2: Download (above) and upload (below) bandwidth observation from Client 2

a mean of about $1MB/s$, a typical value for Italy. The download bandwidth is more variable, with a mean of about $36MB/s$ and some occasional valleys down to $8MB/s$.

4 CONCLUSION

Having an updated profiling of the network characteristics between clients and servers is a core requirement for the precise scheduling of application in the edge computing continuum. In this paper we presented the design of a client-initiated approach for the profiling of the network capabilities between a FaaS server and a client. The approach has been designed to be as lightweight as possible, to support low-power architectures typical of the far edge. Future work includes the extensions of the testbed with more geographically sparse nodes, and the development of specific interfaces for the integration with start-of-the-art edge scheduler.

ACKNOWLEDGMENTS

This work has been partially supported by the European Union's Horizon 2020 Research and Innovation program, under the project ACCORDION (Grant agreement ID: 871793).

REFERENCES

[1] James Broberg, Srikumar Venugopal, and Rajkumar Buyya. 2008. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing* 6, 3 (2008), 255–276.

[2] Hung Cao, Monica Wachowicz, Chiara Renso, and Emanuele Carlini. 2019. Analytics everywhere: generating insights from the internet of things. *Ieee Access* 7 (2019), 71749–71769.

[3] Maurantonio Caprolu, Roberto Di Pietro, Flavio Lombardi, and Simone Raponi. 2019. Edge computing perspectives: architectures, technologies, and open security issues. In *2019 IEEE Intern. Conference on Edge Computing (EDGE)*. IEEE, 116–123.

[4] Batyr Charyyev and Mehmet Hadi Gunes. 2020. Latency Characteristics of Edge and Cloud. In *Network Traffic Measurement and Analysis Conference (TMA 2020) Lightning Talks Session, Dublin Ireland.*

[5] Lorenzo Corneo, Nitinder Mohan, Aleksandr Zavodovski, Walter Wong, Christian Rohner, Per Gunningberg, and Jussi Kangasharju. 2021. (How Much) Can Edge Computing Change Network Latency?. In *2021 IFIP Networking Conference (IFIP Networking)*. IEEE, 1–9.

[6] Luca Ferrucci, Matteo Mordacchini, Massimo Coppola, Emanuele Carlini, Hanna Kavalionak, and Patrizio Dazzi. 2020. Latency preserving self-optimizing placement at the edge. In *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*. 3–8.

[7] Yongquan Fu, Yijie Wang, and Ernst Biersack. 2013. A general scalable and accurate decentralized level monitoring method for large-scale dynamic service provision in hybrid clouds. *Future Generation Computer Systems* 29, 5 (2013), 1235–1253.

[8] Krishna P Gummadi, Stefan Saroiu, and Steven D Gribble. 2002. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. 5–18.

[9] Haojun Huang, Li Li, Geyong Min, Wang Miao, Yingying Zhu, and Yangming Zhao. 2021. TNDP: Tensor-based Network Distance Prediction with Confidence Intervals. *IEEE Transactions on Services Computing* (2021).

[10] Haojun Huang, Hao Yin, Geyong Min, Dapeng Oliver Wu, Yulei Wu, Tao Zuo, and Ke Li. 2015. Network distance prediction for enabling service-oriented applications over large-scale networks. *IEEE Communications Magazine* 53, 8 (2015), 166–174.

[11] Marco Iorio, Fulvio Risso, and Claudio Casetti. 2021. When latency matters: measurements and lessons learned. *ACM SIGCOMM Computer Communication Review* 51, 4 (2021), 2–13.

[12] Ioannis Korontanis, Konstantinos Tserpes, Maria Pateraki, Lorenzo Blasi, John Violos, Ferran Diego, Eduard Marin, Nicolas Kourtellis, Massimo Coppola, Emanuele Carlini, et al. 2020. Inter-operability and Orchestration in Heterogeneous Cloud/Edge Resources: The ACCORDION Vision. In *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*. 9–14.

[13] Minseok Kwon, Zuochao Dou, Wendi Heinzelman, Tolga Soyata, He Ba, and Jiye Shi. 2014. Use of network latency profiling and redundancy for cloud server selection. In *2014 IEEE 7th Intern. Conference on Cloud Computing*. IEEE, 826–832.

[14] Antonios Makris, Abderrahmane Boudi, Massimo Coppola, Luís Cordeiro, Massimiliano Corsini, Patrizio Dazzi, Ferran Diego Andilla, Yago González Rozas, Manos Kamarianakis, Maria Pateraki, et al. 2021. Cloud for Holography and Augmented Reality. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. IEEE, 118–126.

[15] Matteo Mordacchini, Luca Ferrucci, Emanuele Carlini, Hanna Kavalionak, Massimo Coppola, and Patrizio Dazzi. 2021. Self-organizing Energy-Minimization Placement of QoE-Constrained Services at the Edge. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer, 133–142.

[16] Andrei Palade, Aqeel Kazmi, and Siobhán Clarke. 2019. An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge. In *2019 IEEE World Congress on Services (SERVICES)*, Vol. 2642-939X. 206–211. https://doi.org/10.1109/SERVICES.2019.00057

[17] 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 (2018), 1275–1284.

[18] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. 2019. Architectural implications of function-as-a-service computing. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 1063–1075.

[19] Puneet Sharma, Zhichen Xu, Sujata Banerjee, and Sung-Ju Lee. 2006. Estimating network proximity and latency. *ACM SIGCOMM Computer Communication Review* 36, 3 (2006), 39–50.

[20] Han Hee Song and Praveen Yalagandula. 2007. Real-time end-to-end network monitoring in large distributed systems. In *2007 2nd International Conference on Communication Systems Software and Middleware*. IEEE, 1–10.

[21] Xu Zhang, Hao Yin, Dapeng Oliver Wu, Haojun Huang, Geyong Min, and Ying Zhang. 2017. SSL: A surrogate-based method for large-scale statistical latency measurement. *IEEE Transactions on Services Computing* 13, 5 (2017), 958–968.

[22] Zhao Ziming, Liu Fang, Cai Zhiping, and Xiao Nong. 2018. Edge computing: platforms, applications and challenges. *Journal of computer research and development* 55, 2 (2018), 327.