Dynamically Adapting the Behavior of Serverless Functions in the Cloud-Edge Continuum

Carlo Puliafito*, Claudio Cicconetti†, Marco Conti†, Enzo Mingozzi*, Andrea Passarella†
* Department of Information Engineering, University of Pisa, Pisa, Italy
† Institute of Informatics and Telematics, National Research Council, Pisa, Italy

I. Introduction

Monolithic applications have shown limitations in terms of scalability and maintainability. To face such downsides, the Function as a Service (FaaS) model decomposes applications into small pieces of code called functions, each focusing on a specific aspect of the application. Functions are packaged within lightweight environments such as containers. With FaaS at its core, **serverless computing** builds modern cloud computing systems wherein developers only provide their functions code, and the cloud provider does the rest, i.e., deploying, scaling, and managing functions. Moreover, functions run following an event-based pattern, and final users are billed only for what they use with fine granularity.

In this context, consecutive function invocations from the same client can be independent from one another or, more often, can form a session. This has a state, which must be preserved across invocations for the whole session time. In cloud systems, where serverless was originally invented, functions typically have to retrieve this state remotely at each invocation, via an external service such as a database: we refer to these functions as **remote-state functions** or λ-functions. Different instances of the same λ-function are equivalent, as they do not retain any state locally. As a result, (i) different users can share the same function instance, (ii) consecutive invocations from the same user can be forwarded to different function instances, and (iii) resources allocated to inactive instances can be freed after a short period of idle time.

Despite serverless was introduced for cloud data centers, it is gaining momentum in **edge computing** as well. With edge computing, micro-data centers are pervasively deployed toward the network edge (close to or co-located with access networks), leading to a geo-distributed infrastructure where functions can run close to the users. This proximity provides low latency and high bandwidth, which are needed by modern Smart City applications having strict requirements, e.g., the Internet of Things, Augmented/Virtual Reality, and smart vehicles. Adopting serverless at the network edge is subject to strong investigation. The main obstacle comes from the cloud-oriented design of serverless, which does not always suit the characteristics of edge computing systems. Let us focus on session state management. λ-functions in the cloud degrade performance only slightly, as both functions and state are hosted in the same data center. Yet, for edge-hosted functions, remote-state access may cause significant service latency and network traffic, risking to nullify edge computing advantages.

To solve the above limitation, **local-state functions** (or simply μ-functions) are raising as an alternative in serverless computing. On the one hand, μ-functions avoid the overhead to retrieve the state from a remote source. Nonetheless, they are not as efficient and flexible as λ-functions, because each μ instance is dedicated to a specific user or application session, for which it provides data access in a private and persistent manner. Moreover, μ instances are not triggered on demand but are long-running to retain state across invocations.

Today, λ- and μ-functions are considered as two distinct alternatives in serverless systems. Moving one step further from this view, we advocate that such a dichotomy should not exist, but rather a serverless function should be able to **adapt dynamically**, i.e., switching its behavior from λ to μ and *vice versa*, based on both internal and external factors. This approach, which we introduce in [1] and further develop in [2] through the definition of a resource allocation problem, would relieve the function programmer from the risk of making an uninformed decision at development time. Moreover, it would let the serverless provider perform run-time optimizations based on the current conditions in the system, e.g., to increase resource efficiency. Furthermore, it would allow to reduce the cost of operation of serverless functions, as we briefly show in Sec. II. Finally, it would benefit applications having requirements that dynamically change over time, as we describe in Sec. III through a Smart Vehicle use case.

II. Cost Analysis

We analyze real serverless traces collected on Microsoft Azure Functions and made available publicly[1]. The dataset includes more than 44 millions of anonymized function invocations from 856 applications. Such applications are very heterogeneous, e.g., the number of daily invocations ranges from very few to millions. Read accesses are 77% out of the total. We consider that the cost of a μ-application is given only by the time window when it is assigned a dedicated container:

$$c_\mu = \Omega_\mu T_\mu, \tag{1}$$

where Ω_μ is the cost per time unit and T_μ is the time units the application spent as μ. On the other hand, for a λ-application we assume that its cost is given by the number of invocations and the type of state access, as follows:

[1] github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsBlobDataset2020.md

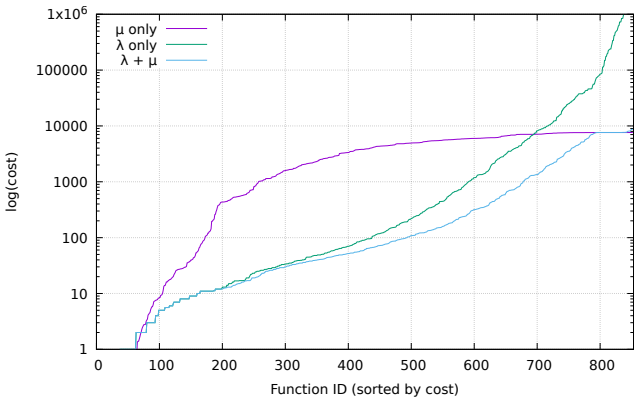

Fig. 1. Comparison of operational costs. All costs are in $\$10^{-6}$.

$$c_\lambda = \xi_\lambda \left(N_\lambda^R + N_\lambda^W\right) + \sigma_\lambda^R N_\lambda^R + \sigma_\lambda^W N_\lambda^W, \qquad (2)$$

where ξ_λ is the cost per function invocation, σ_λ^R (σ_λ^W) is the cost per read (write) access, and N_λ^R (N_λ^W) is the number of function invocations with read (write) accesses. Values used for the cost model are taken from publicly available prices of Amazon Lambda@Edge[2], where the invocation of 1 million functions costs \$0.6, the cost of a `GET` (`PUT`) operation to read (write) the state is about \$0.4 (\$5) for 1 million operations, and the cost per second of μ execution is $\$6.3 \cdot 10^{-6}$.

Fig. 1 depicts the costs obtained with the three policies. As shown, the μ-only and λ-only curves intersect: some applications are better served *always* as μ while others, representing the majority in the considered dataset, as λ. However, by using a $\lambda + \mu$ hybrid policy, the cost can be minimized for all functions, which confirms our intuition that all applications should be able to alternate between μ and λ in their lifetime.

III. A Smart Vehicle Use Case

Our use case is depicted in Fig. 2. We assume that a driver takes her green car along the office-home path, and that the car exploits a serverless function to assist the driver. On the way home, the driving assistance logic is limited to speed control and steering of the car. Latency and throughput requirements are loose – 1000 ms and 0.2 Mb*p*s [3] – and the function runs as λ. As shown on the left side of Fig. 2, the first function invocation is dispatched to a container on edge node 1 (step 1). The invocation is queued as that container has been previously invoked by the yellow car (step 2). Once the green car is served, the λ-container fetches the session state remotely (steps 3 and 4), computes the response (step 5), sends it to the green car (step 6), and pushes the new session state remotely (step 7). In step 8, a new function invocation is performed. Now, the invocation goes to a container on edge node 2. As shown, the λ-container accesses the remote storage again to read and write the session state. When the user reaches home, the function enters autonomous parking mode (right side of Fig.2), with stricter requirements – 10 ms and 100 Mb*p*s [3]. Hence, the function changes to μ: the logic is invoked in step 1, session state is retrieved remotely to instantiate the function as a μ-container, and the next invocations are dispatched to that dedicated, local-state instance (steps 6-8).

References

[1] C. Puliafito, C. Cicconetti, M. Conti, E. Mingozzi, and A. Passarella, "Stateful function as a service at the edge," *Computer*, 2022.

[2] ——, "Balancing local vs. remote state allocation for micro-services in the cloud-edge continuum," *Perv. and Mob. Comp.*, 2023.

[3] Huawei, "5G – Opening up new business opportunities," Tech. Rep., 2016.

[2] `aws.amazon.com/lambda/pricing/?nc1=h_ls`

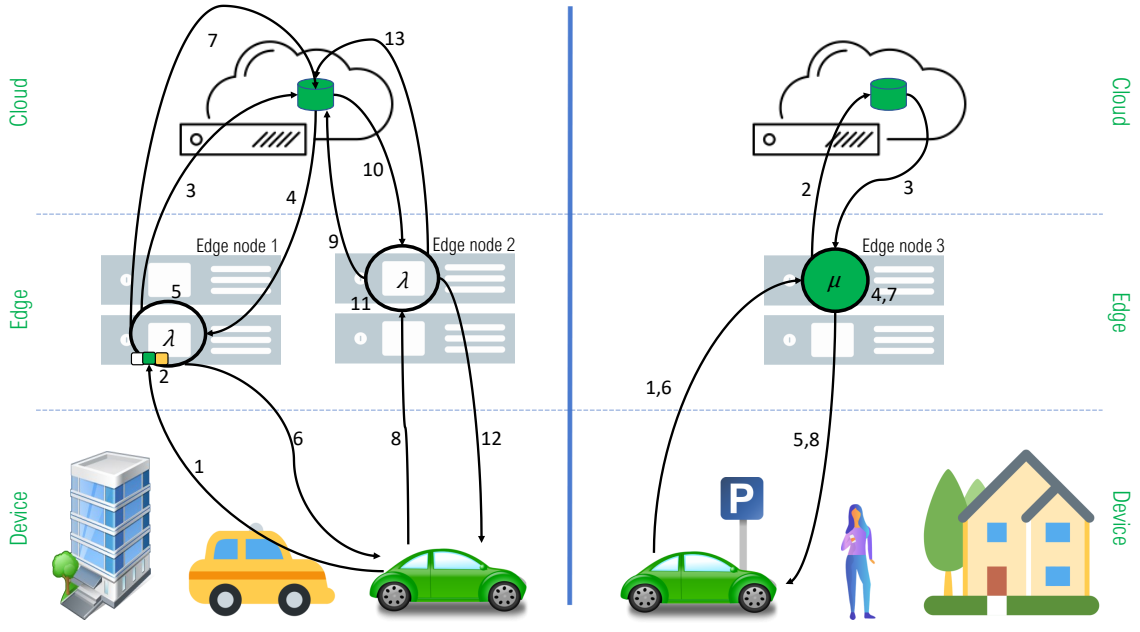

Fig. 2. On the left, λ-containers run a regular driving logic; on the right, a μ-container runs an autonomous parking logic.