A novel fog-cloud architecture to process serverless functions with adaptive timeout

Gustavo André Setti Cassel[1], **Rodrigo da Rosa Righi**[1], **Vinicius Facco Rodrigues**[1]

[1]Applied Computing Graduate Program
University of Vale do Rio dos Sinos (UNISINOS) – São Leopoldo/RS

gustavoasc@edu.unisinos.br, {rrrighi,vfrodrigues}@unisinos.br

Abstract. *This paper presents a novel architecture to handle serverless functions with adaptive timeout, leveraging prediction to foresee how long the incoming request will take to finish based on historical data. This decision-making process aims to ensure that no request will be discarded, while maximizing execution throughput and offloading requests from the fog to the cloud when needed.*

1. Introduction

Serverless computing, also known as Function as a Service (FaaS), represents an emerging technology where business logic is written as a composition of stateless functions. Developers write pieces of code that are separately deployed to a FaaS platform, which in turn is responsible for spawning multiple instances of a given function according to the amount of requests. This behavior helps to achieve the desired level of scalability. Therefore, platforms are responsible for handling workloads in a parallel fashion [Chowhan 2018].

This kind of computing also imposes limitations, though, such as maximum timeout for running functions [Chowhan 2018]. Requests that do not finish within the time limit are abruptly finished by the platform. In fact, it is such a challenge to fit complex functions into a FaaS platform, as requests can last a long time because of different reasons: *i)* dependency on external entities taking longer than normal to perform operations, *ii)* algorithms with high runtime complexity that last longer as input grows, *iii)* legacy code that is difficult to optimize or refactor into different functions, *iv)* hardware concurrency caused by functions performing intense I/O operations, among others [Gorbenko et al. 2019, Rauback Aubin et al. 2021, Szwarcfiter and Markenzon 1994].

On the other hand, timeout is important because it prevents long requests from monopolizing resources and reducing execution throughput. There is a need for a solution that handles both short and long-term requests with adaptive timeout, in a way that no request is discarded even when finished because of timeout, while giving higher priority to short requests in order to maximize throughput. We fill this gap by proposing a novel architecture that integrates fog and cloud with the goal of ensuring execution of requests, no matter how long they take to finish, as well as maximizing execution throughput, financial savings, elasticity, reliability, and availability of resources.

2. Model

Figure 1 presents our architecture with its main components. Requests are received by a load balancer and forwarded to a Request Manager instance (1), which in turn dispatches incoming requests to the most appropriate environment according to prediction based on

weighted moving average. This component tries to foresee whether the incoming request will take a short or a long time to finish, delegating the execution to a FaaS platform (2) on the fog in case it is a short task, or to a container pool (3) also located on the fog in case it is a medium or long task. In case execution is finished by timeout on the FaaS platform (2), request is re-executed on the container pool (3), which does not have time limit.

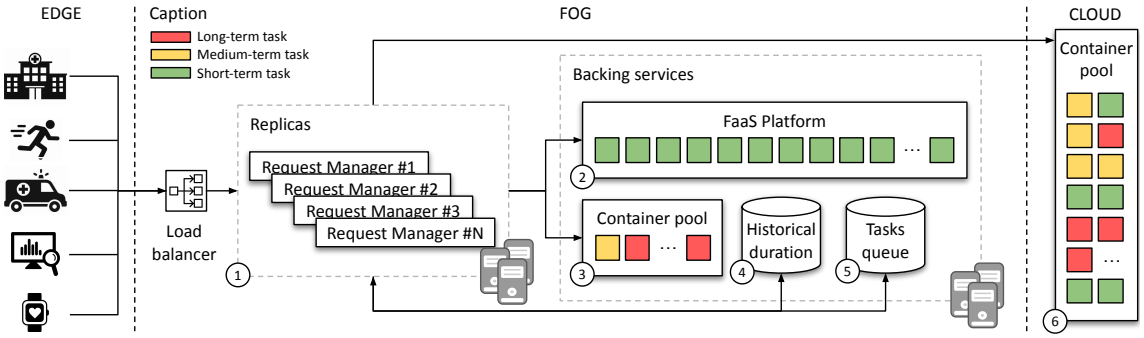

Figure 1. Architecture details highlighting correlation between components.

After executing the request, its duration is stored in a database (4) located on the fog, so the Request Manager can use this information to predict how long future executions will take to finish. Recent durations have higher impact on predictions. Request parameters and inputs should also be stored in the database, as duration may vary depending on the given input. In case local resources are overloaded and Internet connection is unavailable, incoming requests are added to a queue (5) to be processed in a later moment. This ensures that no request will be discarded. In case local resources become available again, queued requests are executed locally. When Internet is back but local resources are overloaded, requests are vertically offloaded to containers on the cloud (6) without time limit, executing with unlimited processing capabilities provided by cloud vendors.

This architecture is subject to changes, as it is a first proposal. We are currently building the first Request Manager prototype. Further steps include proposing algorithms to improve distribution of requests between physically close fog nodes, building prototypes to merge small requests into a single execution on the FaaS platform in order to reduce cold start, and analyzing how to prevent starvation when giving higher priority to short tasks. Finally, we will assess benefits of using FaaS platforms and container pools on the cloud instead of using a single container pool, and compare results of building the Request Manager itself as a microservice or as functions.

References

Chowhan, K. (2018). *Hands-on serverless computing*. Packt Publishing, 1st edition.

Gorbenko, A., Romanovsky, A., and Tarasyuk, O. (2019). Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency. *Journal of Network and Computer Applications*, 146:102412.

Rauback Aubin, M., da Rosa Righi, R., Valiati, V. H., et al. Helastic: On combining threshold-based and serverless elasticity approaches for optimizing the execution of bioinformatics applications. *Journal of Computational Science*, 53:101407.

Szwarcfiter, J. L. and Markenzon, L. (1994). *Estruturas de Dados e seus Algoritmos*, volume 2. Livros Técnicos e Científicos.