Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Benchmark de FaaS baseado na arquitetura distribuída do framework Orama

Bruno Abreu Kamienski

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Profa. Dra. Aletéia Patrícia Favacho de Araújo

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Benchmark de FaaS baseado na arquitetura distribuída do framework Orama

Bruno Abreu Kamienski

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Brasília, 8 de Dezembro de 2023

Dedicatória

Dedico esse trabalho aos meus familiares, que me deram todo suporte por essa árdua jornada, como manifestado pela expressão latina "ad astra per aspera".

Agradecimentos

Em primeiro lugar e, acima de todas as coisas, agradeço a Deus. De maneira mais que especial, agradeço à minha orientadora por ser um farol em meio a mares revoltos.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

À medida que soluções orientadas a *Function as a Service (FaaS)* são adotadas de maneira generalizada, aumenta o interesse em ferramentas que possibilitam a execução de *benchmarks* nesses ambientes. O *framework* Orama se destaca neste contexto por oferecer uma solução altamente configurável e escalável para auxiliar no provisionamento e na avaliação de desempenho, além de análises comparativas e estatísticas dos resultados aferidos. Neste trabalho, foi proposta e desenvolvida uma arquitetura distribuída para o *framework* Orama, por meio da qual é possível executar *benchmarks* com altos níveis de concorrência, bem como disparar requisições geograficamente dispersas, aproximando-se do ambiente de execução real. Os resultados do experimento mostraram que a arquitetura proposta foi capaz de dividir a carga entre os *workers* distribuídos, e conseguiu consolidar adequadamente os resultados de retorno. Além disso, foi possível observar características específicas dos provedores envolvidos no experimento, como o excelente desempenho do serviço do tipo FaaS da Alibaba, cujo tempo médio de execução foi o menor dentre todos os testes realizados e livre de falhas. Google Cloud Function e AWS Lambda registraram resultados intermediários para tempo de execução, além de registrarem falhas. Por fim, o Azure Function teve os piores resultados em tempo médio de execução e de inicialização a frio.

Palavras-chave: *Benchmark* distribuído, Função como um serviço, *framework* Orama

Abstract

As the adoption of Function-as-a-Service-oriented solutions grows, interest in tools that enable the execution of benchmarks on these environments increases. The Orama framework stands out in this context by offering a highly configurable and scalable solution to aid in provisioning, running benchmarks and comparative and statistical analysis of results. In this work, a distributed architecture for the Orama framework was proposed and implemented, through which it is possible to run benchmarks with high levels of concurrency, as well as with bursts of geographically dispersed requests, just as in real environments. The results of the experiment showed that the proposed architecture was able to divide the workload between the distributed workers and able to consolidate properly in the return of the results. In addition, it was possible to observe specific characteristics of the providers involved in the experiment, such as the excellent performance of Alibaba's FaaS, whose average execution time was the lowest of the tests and free of failures. Google Cloud Function and AWS Lambda recorded intermediate results for average execution time and recorded failures. Finally, Azure Function had the worst results in average execution time and cold start.

Keywords: *Distributed Benchmark, Function-as-a-Service, Orama framework*

Sumário

Lista de Abreviaturas e Siglas

AFC	Alibaba Function Compute.
AWS	Amazon WebServices Lambda.
AZF	Azure Functions.
DBaaS	Database as a Service.
FaaS	Function as a Service.
GCF	Google Cloud Functions.

Capítulo 1

Introdução

A computação em nuvem é hoje utilizada de maneira transparente no cotidiano de grande parte da população. De forma análoga, os desenvolvedores de software e operadores de infraestrutura têm buscado se utilizar da computação em nuvem de forma igualmente transparente. Desta forma, a computação sem servidor [1], como o paradigma de programação em nuvem, tem ganhado mais importância nas soluções de software. *Function as a Service (FaaS)* [2] permite que os usuários publiquem funções escritas em alguma linguagem de programação que, quando acionadas, devem ter seu devido processamento garantido pelo provedor.

Em virtude da demanda de alguns serviços se distribuir por diferentes regiões do planeta, surge a necessidade dos provedores estarem o mais próximo possível do usuário final e, por consequência, é adotada a estratégia de implantar infraestruturas geograficamente distribuídas ao redor do mundo. Por outro lado, é comum que diferentes serviços de nuvem sejam combinados para compor uma solução, e entre os serviços de armazenamento de dados, se destaca o *Database as a Service (DBaaS)* [3] em que os provedores entregam ambientes de banco de dados totalmente gerenciados por eles.

Considerando a possibilidade de diferentes implementações de FaaS entre regiões impactarem o desempenho das aplicações que as utilizam, foram avaliados os principais FaaS combinados com serviços DBaaS. Para isso, foram selecionadas cinco regiões do planeta que possuem infraestruturas implantadas nas quais as funções Amazon WebServices Lambda (AWS) da Amazon, Google Cloud Functions (GCF) da Google, Azure Functions (AZF) da Microsoft e Alibaba Function Compute (AFC) da Alibaba. O framework usado para conduzir os testes foi o Orama [4], que será apresentado na capítulo 2. Os resultados indicam que o Alibaba aparentemente implementa uma estratégia de gestão mais eficiente para sua plataforma FaaS em todas as regiões avaliadas, pois seu tempo médio de execução foi o menor entre os provedores, assim como sua taxa de falhas. AWS e GCF obtiveram resultados intermediários e muito próximos. Já o AFC registrou os piores resultados, tanto em tempo médio de execução quanto em taxas de falhas.

1.1 Problema de Pesquisa

1.1.1 Justificativa

Os provedores de computação em nuvem pública têm expandido a cobertura de seus serviços baseados no paradigma de computação sem servidor, que tem previsão de crescimento de cerca de 20% em 2023. Desta forma, faz-se necessária a criação de mais ferramentas para estudar o comportamento dos *Function as a Service (FaaS)* em diferentes provedores.

1.1.2 Objetivos

O objetivo principal deste trabalho foi avaliar o comportamento das FaaS dos provedores de nuvem pública Alibaba, Amazon, Google e Microsoft diante de altos níveis de concorrência e por meio da avaliação do tempo de resposta médio resultante das execuções de funções simples e funções atreladas aos serviços de persistência principais destes provedores. Outra característica interessante analisada foi o tempo de execução da primeira requisição feita ao FaaS após o provisionamento, chamado de "*warmup*".

1.2 Metodologia

Levando em consideração a perspectiva de crescimento da adoção de FaaS, bem como a possibilidade de diferentes implementações entre regiões impactarem o desempenho de aplicações operando em ambientes dessa natureza, foram avaliados os principais FaaS em diferentes regiões. Cinco importantes regiões do planeta onde Amazon WebServices (AWS), Google Cloud Platform (GCF), Azure (AZF) e Alibaba (AFC) possuem infraestruturas implantadas foram escolhidas para receber um dos casos de uso disponíveis do framework Orama [4]. Utilizando FaaS integrado com o respectivo DBaaS, foram executadas diversas baterias de testes simulando acessos simultâneos a serviços desde 1 requisição simultânea até 4096 acessos paralelos. Os processos de provisionamento de ambientes FaaS, execução de testes, análise de resultados e desprovisionamento de ambientes foram realizados utilizando o framework Orama. O resultado deste trabalho foi publicado em artigos científicos, os quais serão apresentados ao longo dos próximos capítulos, com a co-autoria de Leonardo Rebouças de Carvalho, criador da versão *standalone* do *framework* Orama, e tendo o autor desta monografia trabalhado principalmente na expansão da *framework* para a versão distribuída com o provedor Alibaba.

1.3 Estrutura da Monografia

Este trabalho está dividido em cinco capítulos, sendo o primeiro esta introdução com conceitos básicos e a estrutura do trabalho. Na sequencia, são apresentados três capítulos com artigos publicados detalhando o trabalho realizado. O Capítulo 2 apresenta o primeiro artigo, detalhando a dinâmica de construção da versão distribuída do framework Orama, a fim de analisar uma carga de requisições maior que a sua versão concentrada. O Capítulo 3 apresenta um artigo com a aplicação da versão distribuída em um cenário de persistência de dados, contexto amplamente utilizado no mercado corporativo e em diversos sistemas comerciais. O Capítulo 4 apresenta um artigo com uma versão do cenário de persistência e, ao final, o Capítulo 5 apresenta a discussão das conclusões alcançadas e a indicação de trabalhos futuros.

Capítulo 2

Benchmark de FaaS baseado na arquitetura distribuída do *framework* Orama

FaaS Benchmarking over Orama Framework's Distributed Architecture

Leonardo Rebouças de Carvalho[1][a], Bruno Kamienski[1][b] and Aleteia Araujo[1][c]
[1] *Computer Science Department, University of Brasilia, Campus Darcy Ribeiro, Brasilia, Brazil*
{*leouesb, brunosabreu*}@*gmail.com, aleteia@unb.br*

Keywords: Distributed Benchmark, Function-as-a-Service, Orama framework.

Abstract: As the adoption of Function-as-a-Service-oriented solutions grows, interest in tools that enable the execution of benchmarks on these environments increases. The Orama framework stands out in this context by offering a highly configurable and scalable solution to aid in provisioning, running benchmarks and comparative and statistical analysis of results. In this work, a distributed architecture for the Orama framework is presented, through which it is possible to run benchmarks with high levels of concurrency, as well as with bursts of geographically dispersed requests, just as in real environments. The results of the experiment showed that the proposed architecture was able to divide the loads between the distributed workers and able to consolidate properly in the return of the results. In addition, it was possible to observe specific characteristics of the providers involved in the experiment, such as the excellent performance of Alibaba Function, whose average execution time was the lowest of the tests and free of failures. Google Cloud Function and AWS Lambda recorded intermediate results for average execution time and recorded failures. Finally, Azure Function had the worst results in average execution time and cold start.

1 INTRODUCTION

Worldwide end-user spending on public cloud services is forecast to grow 20.7% to total $591.8 billion in 2023, up from $490.3 billion in 2022 (Gartner, 2022). All this perspective of growth shows the solidity of this area of computing. Although the main cloud models, such as Infrastructure-as-a-Service (IaaS) (MELL and Grance, 2011) and Platform-as-a-Service (PaaS) still lead the focus of investment, other models that integrate the archetype of Everything-as-a-Service (XaaS) (DUAN et al., 2015) also indicate growth. In this context, serverless-based cloud computing models such as Function-as-a-Service (FaaS) (Schleier-Smith et al., 2021) have been predicted as the main programming paradigms of the next generation of the cloud.

In this scenario, there has been a growing interest in evaluations and benchmark tools that analyze the deliveries of providers, such as Sebs (Copik et al., 2021), PanOpticon (Somu et al., 2020), FaaSDom (Maissen et al., 2020), BeFaaS (Grambow et al., 2021) and Orama framework (Carvalho and Araujo, 2022). However, given the wide range of possibilities for adoptable strategies by providers, as well as the different regions in which these services can be implemented, this type of study needs to be increasingly dynamic and adjustable to real-world situations. Therefore, this work presents a distributed architecture for the Orama framework (Carvalho and Araujo, 2022) in which it is possible to run benchmarks in FaaS environments, exploring a wider range of scenarios than was possible in the standalone version. Through the master/workers architecture, it is possible to divide workloads between several instances, including geographically dispersed ones, allowing both the expansion of assessable levels of concurrency, as well as expanding the capacity to represent a reality of distributed demand.

In experiments, it was verified that the distributed architecture allows high levels of requests in comparison to the standalone version. In addition, it was found that the performance calculated between the scenario whose workers were in the same region and the scenario in which the workers were geographically spread maintained equivalent results, demonstrating that the separation of workers, in the proposed architecture, does not affect the analysis of the results. In addition, it was possible to observe behaviors intrinsic to the services, such as the superior performance demonstrated by Alibaba Cloud Func-

[a] https://orcid.org/0000-0001-7459-281X
[b] https://orcid.org/0000-0001-5817-8694
[c] https://orcid.org/0000-0003-4645-6700

tions. AWS Lambda and Google Cloud Function both had intermediate results. Azure services, on the other hand, obtained the worst results, presenting high average execution times and cold start.

This article is divided into six parts, the first being this introduction. Section 2 presents the theoretical foundation that supports the work. Section 3 describes the distributed implementation of the Orama framework. Section 4 presents the related works. Section 5 shows the results obtained and finally, Section 6 presents the conclusions and future work.

2 BACKGROUND

Thenceforward NIST (MELL and Grance, 2011) defined the traditional cloud models in 2011 as IaaS, PaaS and SaaS, the main public cloud providers began to name their services with another set of acronyms, which culminated in the emergence of the term "XaaS" to define Everything-as-a-Service (DUAN et al., 2015). Amidst this profusion of cloud services, in 2014 AWS launched Lambda, which then inaugurated the Function-as-a-Service (FaaS) model (Motta. et al., 2022). In the FaaS paradigm, the customer delivers a piece of code of interest to the provider, generally written in some language supported by the provider. In addition, the client must configure a trigger to activate the function and this trigger can happen from other services that make up the provider's platform or through a REST API.

Since FaaS is designed to run state-independently, some restrictions are imposed on customers, such as limits on allocable vCPUs, RAM and maximum execution time. Another aspect that significantly distinguishes FaaS from other cloud service models is the billing method. Instead of being charged based on the execution time of instances, as in the IaaS model, in FaaS the customer will only pay based on the activation of the service and its respective duration.

Major public cloud providers have FaaS offerings. In addition to AWS Lambda, which is the forerunner of this concept, it is possible to find solutions from Azure, GCP, Alibaba Cloud, among others. Azure Function (AZF) is Azure's entry into this slice of the cloud market. Google Cloud Function (GCF) is the name of GCP's FaaS. Alibaba Cloud offers Alibaba Function Compute (AFC) as its FaaS. Other providers such as Oracle and IBM also offer FaaS options.

Faced with so many FaaS options, it is essential to understand how the strategies adopted by providers deliver environments. Considering the large number of regions and availability zones in which the main providers are installed, it is possible that implementation differences between providers or between regions can meet different needs or may even be impeding, unfeasible or expensive. Since these strategies are the provider's big trade secrets, the best way to elicit these strategies is by running benchmarks. As these environments are highly dynamic and maintain constant evolution, it is important that the solutions that promote benchmarks on these platforms are configurable to the point of better representing the problems found in real environments. The development of solutions aimed at the current multicloud context should favor the incorporation of new providers and their services as they enter this market. Because of this, it is essential to use a cloud orchestration solution, such as Terraform (HashiCorp, 2021).

2.1 Infrastructure Tools

Terraform (HashiCorp, 2021) is a cloud orchestrator that helps solutions integrate with different virtualization and automation platforms, especially in cloud environments. Through Terraform it is possible to create lightweight and portable infrastructure definition artifacts that can be easily incorporated into other solutions. Other cloud orchestration solutions, such as Heat (Gupta et al., 2014) and CloudFormation (Wittig and Wittig, 2018) propose similar approaches to Terraform, however, as evaluated in (Carvalho and Araujo, 2020) Terraform presents better results.

Performance testing (Erinle, 2013) is a type of testing intended to determine the responsiveness, reliability, throughput, interoperability, and scalability of a system and/or application under a given workload. It could also be defined as a process of determining the speed or effectiveness of a computer, network, software application, or device. Testing can be conducted on software applications, system resources, targeted application components, databases, and a whole lot more. It normally involves an automated test suite, such as JMeter (Erinle, 2013), as this allows for easy, repeatable simulations of a variety of normal, peak, and exceptional load conditions. Such forms of testing help verify whether a system or application meets the specifications claimed by its vendor. Technology solutions currently need to deal with large and fast flows of information and any instability in the service can lead to loss of information. Because of this, it is important to use queuing mechanisms in order to guarantee the correct processing of requests.

Apache Kafka (Sax, 2018) is a scalable, fault-tolerant, and highly available distributed streaming platform that can be used to store and process data streams. The Kafka cluster stores data streams, which are sequences of messages/events continuously produced by applications and sequentially and incrementally consumed by other applications. The Connect API is used to ingest data into Kafka and export data streams to external systems such as distributed file systems, databases, and others. For data stream processing, the Streams API allows developers to specify sophisticated stream processing pipelines that read input streams from the Kafka cluster and write results back to Kafka. Apache Kafka is a solution that adheres to the microservices paradigm, that is, the development model in which the solution's complexity blocks are segmented into smaller processes. In order to manage this large amount of services, without the need to deal with several virtual machines, it is possible to use container-oriented environments.

Docker (Ibrahim et al., 2021) enables the containerization of a software package along with associated configuration and setup details. Such containers can be easily and rapidly deployed while avoiding compatibility issues. In fact, a recent study reports that Docker can speed up the deployment of software components by 10-15 folds. Much of today's applications are multi-component (i.e., multi-container) applications. For instance, a simple web application would require a web server and a database component. Docker Compose (Ibrahim et al., 2021), a natural progression of Docker, enables practitioners to compose such complex applications. Applications transcribe such compositions in a Docker Compose file, where components are specified by describing their Docker image and associated configuration as well as the relations between components. The various services of a solution supported by a container environment can generate large volumes of data that needed processing techniques to make any sense.

2.2 Statistical Analysis Tools

Statistical analysis of benchmark results allows the observation of several phenomena. The factorial design (Jain, 1991), for example, helps to identify the effect of mapped factors on the results. With the factorial design, it is possible to identify whether the variation of any factor in a given scenario causes (or does not) any statistically significant impact on the results. It is also possible to identify the existence of factors that were not initially mapped and that may have a significant influence on the results. Another important statistical analysis tool for understanding the results is the paired t-test. In this test, the difference between two results is evaluated in order to determine its statistical significance, as well as the respective degree of confidence. This test allows the user to establish whether the difference between two results can be considered relevant or insignificant, in the second case the results can be considered statistically equal.

With the purpose of implementing a specific benchmark solution for a FaaS environment that adheres to different scenarios as close to real ones as possible, this work uses tools such as Terraform, JMeter, Apache Kafka, Docker, factorial design, and t-test to propose a distributed architecture for Orama framework, whose detailed description will be presented in the next section.

3 DISTRIBUTED ORAMA FRAMEWORK

The Orama framework (Carvalho and Araujo, 2022) is a tool developed to conduct benchmarks on FaaS environments. Although it is possible to run benchmarks on other types of environment, its focus is directed towards the evaluation of cloud environments oriented to the FaaS paradigm. Through the Orama framework, it is possible to provision environments in an automated way, thanks to its integration with Terraform, which makes the incorporation of new providers as simple as building a Terraform infrastructure definition artifact. Once provisioned, the environment can also be de-provisioned using Terraform automations through the Orama framework. In addition, the entire process of running the benchmarks is conducted by the framework based on the settings entered in the system. It is possible to create several test scenarios varying the level of concurrency, the number of repetitions of a scenario, the establishment of intervals between tests and the execution of warm-up requests, whose objective is to observe the cold start phenomenon, which is very common, and impactful in FaaS environments.

The Orama framework comes with some pre-configured use cases that can be deployed to major providers without any user intervention. These use cases range from a simple calculator, whose objective is only to validate the implementation and correct operation of the service, as well as use cases that use other services from providers, such as object storage and databases. The use cases accept parameterizations that allow the provisioning of different configurations for the use cases, from the amount of allocable memory to the region of the provider where the use case must be implemented.

The standalone version of the framework presented at (Carvalho and Araujo, 2022) has all of its components implemented in a container environment, including the "benchmarker" which is the component responsible for triggering bursts of requests that sim-

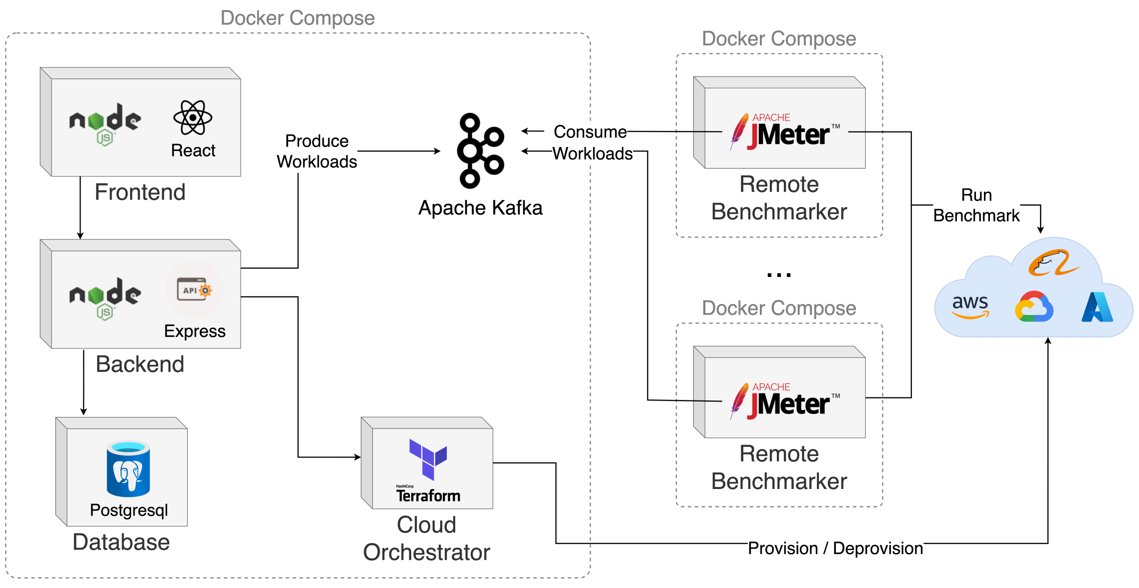


Figure 1: Distributed Orama framework architecture.

ulate real demands on the environments. It turns out that using this approach, the level of concurrency that the Orama framework can simulate on the environment is limited to the amount of resources available on the machine where it is installed. Furthermore, test requests compete with framework management traffic, as framework components also communicate via HTTP.

In this work, a distributed architecture for the Orama framework is presented, as shown in Figure 1. In this architecture, it is observed that the benchmarker component is deployed outside the main environment and therefore its installation can occur in another instance. Furthermore, the number of instances of the benchmarker is variable and can be adjusted to the characteristics of the test to be performed. It is possible, for example, to concentrate the benchmarker's workers in the same region or to distribute them among several regions. It is also possible to include a number of workers, whose load distribution of requests prevents the saturation of resources on the machine, preventing the occurrence of faults attributed to the test execution worker.

Communication between the remote benchmarkers (workers) and the main Orama framework environment (master) composed of frontend, backend, database, and orchestrator is done through Kafka, as can be seen in Figure 1. There are three types of "topics" that are managed by Kafka. The "Health check" topic allows remote workers to inform the master that they are able to receive triggers to run benchmarks. Once health is up to date, this worker will be considered in the distribution of loads of a respective bench-

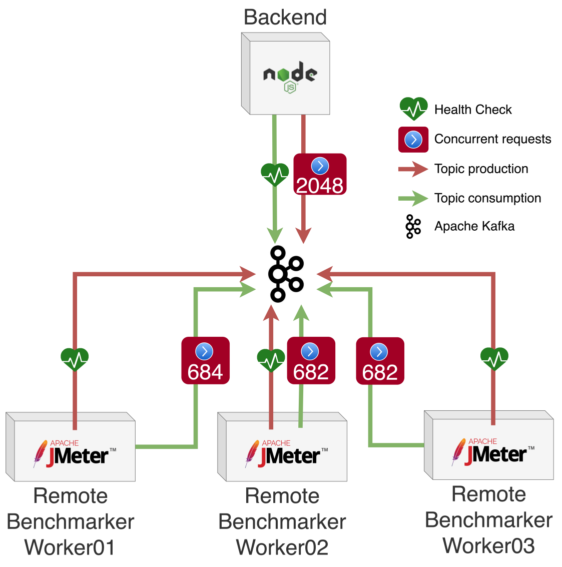


Figure 2: Kafka workflow in Orama framework.

mark and then a topic with the "uuid" of the respective worker will be included in Kafka by the backend of the Orama framework, to be consumed by the respective worker and executed, as can be seen in Figure 2. After running the benchmark, the remote worker records the partial result and inserts it into a third Apache Kafka topic, for consumption by the master's backend. It is worth noting that the Orama framework implements a balance between the number of requests requested for each worker to execute on the target environment.

Figure 2 shows a scenario with three workers and a request to run a benchmark with 2048 concurrent requests. It is possible to notice that all workers are asked for 682 requests, which adds up to 2046 requests. This leaves 2 requests that are assigned to one of the workers, in the case shown in Figure 2 the "Remote Benchmarker Worker 01". As soon as the backend perceives the receipt of all partial results, it promotes the consolidation of the results. If these results do not appear within a configured timeout, then the respective partial result is assigned a failure result.

Once the results are consolidated, it is possible to create factorial designs combining two benchmarks and two levels of concurrence. If the benchmarks involve two different providers, it will be possible to observe the influence of the providers on the results, as well as the difference from the concurrence. Using the factorial design as established by the Orama framework, it is possible to identify whether the strategies adopted by the providers impact the results and whether the level of concurrence is the factor whose influence prevails. It is still possible to identify the existence of some other factors apart from the provider and the level of concurrence influencing the results, in this case the portion of the influence related to the statistical error will be significantly high.

The distributed architecture of the Orama framework presented in this work opens up a wide range of possibilities for running benchmarks on FaaS environments. At this moment of leveraging this approach, it is essential to have a tool available that allows the evaluation of environments delivered by providers, including high levels of concurrence and distributed customers, whose characteristics are more similar to critical situations in the real world experienced by solutions supported by FaaS environments.

In the next section, the results of an experiment using the aforementioned architecture will be presented. The capabilities of the Orama framework using a distributed approach and the insights obtained from the results of the experiment will be illustrated.

4 RELATED WORKS

This article presents a distributed architecture for executing benchmarks in FaaS environments over the Orama framework. The related work is discussed from the perspective of benchmarking FaaS platforms in general as well as work on serverless benchmark frameworks. A comparison between related works is presented in Table 1.

In the paper (Back and Andrikopoulos, 2018) the authors used a microbenchmark in order to investigate two aspects of the FaaS: the differences in observable behavior with respect to the computer/memory relation of each FaaS implementation by the providers, and the complex pricing models currently in use. They used AWS, IBM, GCP, Azure, and OpenWhisk in their evaluation. However, the authors did not present an evaluation of the performance of their microbenchmark in different regions of the providers, nor with different levels of concurrence, as presented in this work.

The quality impacts of operational tasks in FaaS platforms as a foundation for a new generation of emerging serverless big data processing frameworks and platforms are evaluated in (Kuhlenkamp et al., 2019). The authors presented SIEM, a new evaluation method to understand and mitigate the quality impacts of operational tasks. They instantiated SIEM to evaluate deployment package and function configuration changes for four major FaaS providers (AWS, IBM, GCP, and Azure), but only in European regions for the same level of concurrence. In this work, on the other hand, several levels of concurrence are evaluated using two different worker approach (concentrated and distributed).

PanOpticon (Somu et al., 2020) provides a comprehensive set of features to deploy end-user business logic across platforms at different resource configurations for fast evaluation of their performance. The authors conducted a set of experiments testing separate features in isolation. An experiment comprising a chat server application was conducted to test the effectiveness of the tool in complex logic scenarios in AWS and GCP. Furthermore, in this work, the range of tests that the Orama framework can evaluate was extended beyond the execution of benchmarks on AWS and GCP, to include the execution of benchmarks on Azure and Alibaba, which are two other important players in this market.

FaaS-dom (Maissen et al., 2020) is a modular set of benchmarks for measuring serverless computing that covers the major FaaS providers and contains FaaS workloads (AWS, Azure, Google, and IBM). A strong element of FaaS-dom's functions is that they were created in a variety of languages and for a variety of providers. However, the operations that the FaaS-dom functions carry out can be viewed as basic, and they lack Orama's approach to integration with other cloud services.

BeFaaS (Grambow et al., 2021) offers a benchmark methodology for FaaS settings that is application centric and focuses on evaluating FaaS apps using real-world and prevalent use cases. It offers enhanced result analysis and federated benchmark testing, where the benchmark application is split across several providers. It does not, however, provide a superior approach to statistical analysis, such as the fac-

Table 1: Related works.

Paper	Providers	Configu-rable	Factorial Design	T-test	Distri-buted
(Back and Andrikopoulos, 2018)	AWS, IBM, GCP, Azure, and OpenWhisk	No	No	No	No
(Kuhlenkamp et al., 2019)	AWS, IBM, GCP, and Azure	No	No	No	No
(Somu et al., 2020)	AWS and GCP	No	No	No	No
(Grambow et al., 2021)	AWS , GCP, Azure, TinyFaaS, OpenFaaS, and OpenWhisk	Yes	No	No	No
(Barcelona-Pons and García-López, 2021)	AWS, IBM, GCP, and Azure	No	No	No	No
(Copik et al., 2021)	AWS, Azure, and GCP	No	No	No	No
(Wen et al., 2020)	AWS, Azure, GCP, and Alibaba	No	No	No	No
(Carvalho and Araujo, 2022)	AWS and GCP	Yes	Yes	Yes	No
This paper	**AWS, Azure, GCP, and Alibaba**	**Yes**	**Yes**	**Yes**	**Yes**

torial design or t-test that are covered by this study.

In paper (Barcelona-Pons and García-López, 2021) the authors analyzed the architectures of four major FaaS platforms: AWS Lambda, AZF, GCP, and IBM Cloud Functions. The research focused on the capabilities and limitations the services offer for highly parallel computations. The design of the platforms revealed two important traits influencing their performance: virtualization technology and scheduling approach. This work, on the other hand, focuses on investigating the differences in performance of the main providers with different levels of concurrence.

In the Serverless Benchmark Suite (Sebs) (Copik et al., 2021), typical workloads are chosen, the implementation is tracked, and the infrastructure is assessed. The benchmark's applicability to several commercial vendors, including AWS, Azure, and Google Cloud, is guaranteed by the abstract concept of a FaaS implementation. Based on the executed test cases, this work assesses variables including time, CPU, memory, I/O, code size, and cost. However, unlike the Orama framework used in this work, their solution can't work in distributed mode.

In (Wen et al., 2020), the authors run a test flow employing micro benchmarks (CPU, memory, I/O, and network) and macro benchmarks to evaluate FaaS services from AWS, Azure, GCP, and Alibaba in detail (multimedia, map-reduce and machine learning). The tests made use of specific Java, Node.js, and Python methods that investigated the benchmarking attributes to gauge resource usage efficiency and initialization delay. However, they do not present evaluations in different levels of concurrence and also do not perform a statistical analysis using factorial design and t-test as is done in this work.

Although the aforementioned work presents an ad-hoc evaluation of the providers, the scalability and manageability of this evaluation is limited to the parameters that were used in the work. On the other hand, this article uses a fully manageable solution prepared for the incorporation of new providers and use cases.

5 RESULTS

The Orama framework makes it possible to run benchmarks on a FaaS environment in different scenarios, especially with the introduction of the distributed architecture presented in this work. In order to explore some possibilities of running benchmarks, an experiment was designed to allow the behavior of the architecture to be observed, as well as obtaining some insights into FaaS environments. Details of how the experiment was conducted will be presented in Section 5.1, while an analysis of the results is provided in Section 5.2.

5.1 Methodology

Considering that the main objective of this experiment is to analyze the behavior of the distributed architecture of the Orama framework, a use case was selected which had the role objective of guaranteeing the correct execution of FaaS in the providers, without integration with other services, such as Database or Object Storage, which could introduce specific features of these services and divert the focus from the archi-

Table 2: Average execution times (in milliseconds).
(C) → concentrated workers — (D) → distributed workers.

Concurrence level	**Scenario**							
	AWS Lambda		**GCF**		**AZF**		**AFC**	
	(C)	**(D)**	**(C)**	**(D)**	**(C)**	**(D)**	**(C)**	**(D)**
1	581	793	592	684	643	954	**102**	281
2	595	749	605	757	612	910	**93**	360
4	663	776	699	769	715	1004	**91**	302
8	592	859	595	832	622	1149	**133**	511
16	637	897	638	817	695	1145	**100**	358
32	709	983	707	918	890	1414	**100**	367
64	911	1158	924	1097	1303	1732	**109**	378
128	1328	1569	1367	1548	2264	2390	**128**	393
256	2274	2417	2386	2466	4130	3749	**233**	565
512	4068	4418	4447	4489	7506	6648	**168**	426
1024	6852	7334	5181	5787	11514	11884	**116**	367
2048	6139	6072	3596	3270	18378	17234	**67**	338

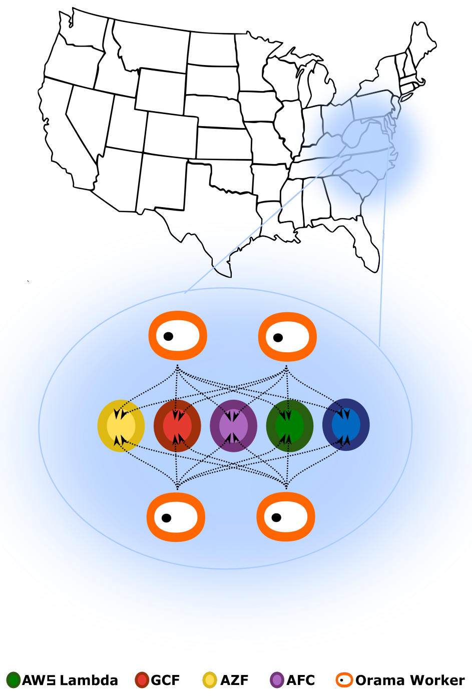


Figure 3: Concentrated workers scenario.

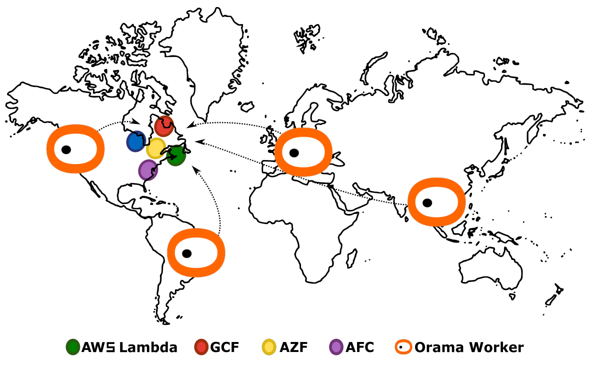


Figure 4: Distributed workers scenario.

tecture itself. Therefore, the use case that deploys a simple math calculator in each of the four main FaaS providers supported by the Orama framework was selected. The providers are AWS, Azure, Google and Alibaba.

The distributed architecture of the Orama framework allows workers in different regions to be deployed. Therefore, in this experiment two scenarios were elaborated. In the first scenario, shown in Figure 3, the workers were all deployed in the same region where the target FaaS environments were also provisioned (concentrated workers scenarios). In the second scenario, workers were distributed in geographically distant regions, as shown in Figure 4. With these two scenarios, it was expected to demonstrate the influence of latency on FaaS, however, their behavior should be equivalent and demonstrate the same strategy applied by providers.

Both scenarios were subjected to concurrent loads of requests from just one simultaneous request to up to 2048 requests with exponential growth. Thus, the FaaS of each provider were submitted individually to loads of 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048. It is noteworthy that the tests performed with the Orama standalone architecture framework were limited to up to 512 simultaneous requests, noting the limitations of opening a connection with FaaS services imposed by the instance in which the Orama framework was installed. As 2048 was the maximum level of this experiment, a quantity of 4 benchmarker workers was defined so that each one was responsible for carrying out a maximum of 512 concurrent requests. Furthermore, each battery of tests with different levels of concurrency was repeated 10 times in order to establish a statistical data mass for further analysis.

In the first scenario (concentrated workers), the Orama framework was deployed on an instance of GCP's Compute Engine in the US-East region with 16GB of RAM and 4 vCPUS, and each of the four workers was also deployed in the same GCP region, the servers had 4GB of RAM and 2 vCPUS each. For the second scenario, the master installation of Orama framework was kept on the same instance in the US-East GCP region as used in the first scenario. However, the workers were spread out in other GCP regions such as: asia-northeast1 (Tokyo, Japan), US -West (The Dalles, Oregon, United States), europe-west2 (London, England), and southamerica-east1 (Osasco, Brazil). All instances of GCP used in this experiment ran Debian 11 as the operating system. The FaaS that were the target of this experiment in both scenarios were configured to be deployed in the respective East American regions at each of the respective providers, namely AWS Lambda, AZF, GCF and AFC.

At the end of the execution of the repetitions of both test scenarios, factorial designs and t-tests were created in order to analyze the effects of the difference between the providers and the difference between the lowest level of concurrency (only one request) and the highest level, with 2048 concurrent requests. The analysis of the obtained results is presented in the next section.

5.2 Outputs Analysis

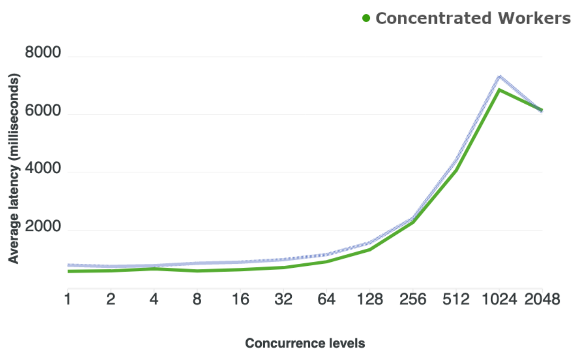
Table 2 presents the consolidated result of all eight test scenarios conducted in the experiment. It is possible to observe that the provider whose scenarios had the lowest average execution times was AFC, highlighted in bold. Next in very close range are the average execution times for AWS Lambda and GCP. Finally, the average execution times for AZF were the highest in this experiment. In addition, in Table 2, it is also possible to observe that the scenarios with distributed workers have generally higher averages than the averages found in the scenario with workers concentrated in the same region, this is the expected result, since the greater geographic distance between workers and target FaaS should raise averages by introducing higher latency to traverse the network.

Figure 5a presents a comparative result between the scenarios involving AWS Lambda. It may be seen that the average execution time of both scenarios in AWS Lambda follow the growth of the concurrency level the previous level to the maximum (2048 concurrent requests) when there is a small drop in the average. This decrease in the average execution time after 1024 indicates that, when faced with a growing demand, the provider reinforced the service infrastructure in order to maintain its quality in terms of execution time. Despite this effort by the provider, failures occurred from 256 concurrent requests, as shown in Figure 5b, which shows the percentage failure rates that occurred at each concurrency level. To corroborate the indication that the provider promoted an escalation before 2048 requests, as well as the average time, the failure rate also showed a reduction.

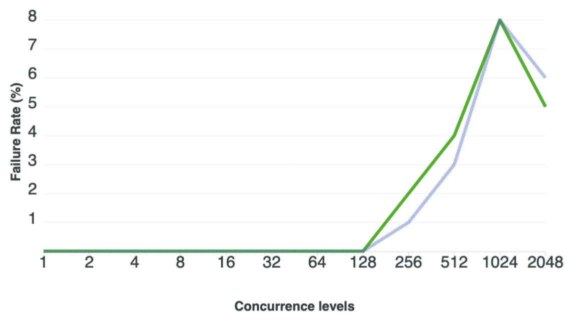
Figures 5c and 5d show the average execution times of the scenarios involving the GCF and their respective failure rates at each concurrency level, respectively. It is possible to observe that the graph of the average execution time of the GCF is similar to the same graph for AWS Lambda, in which the average time accompanies the growth of the concurrency level until the intervention of the provider causes a decrease in the average time. However, GCF's average time threshold is lower than AWS Lambda's. While in AWS Lambda the peak point is 7.3 seconds (at 1024 in the distributed scenario), in CGF the peak is recorded at 5.7 seconds (at 1024 in the distributed scenario). This proximity between AWS Lambda and GCF average times demonstrates a strategic equivalence between providers. Despite this, with regard to the failure rate, the occurrence of failures in GCF started only after 1024 requests, while AWS Lambda already had failures at 256. However, the level of failures recorded by GCF was higher compared to AWS Lambda, because while in AWS Lambda the failures peaked at 8% of requests, in GCF these failures reached around 20%. Another difference in the AWS Lambda and GCF results is the continuous growth of failures demonstrated in the GCF distributed scenario, which indicates that the provider's monitoring layer had greater difficulty in dealing with requests from different regions than those whose origin was the same.

Unlike AWS Lambda and GCF, which showed a point of reduction in the average execution time, AZF, as shown in Figure 5e, maintained a continuous growth in the average execution time, registering the highest average times in the experiment. This indicates that in the AZF environment there was no reinforcement of the infrastructure as demand grew, although this is a FaaS premise. Despite the high average time, AZF recorded a low failure rate, with failures occurring in only 3% of the 2048 concurrent requests in the distributed scenario, as can be seen in Figure 5f.

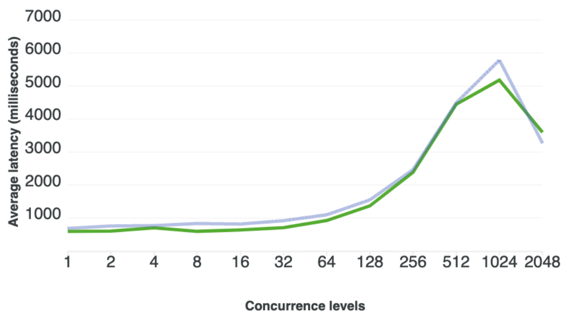
Figure 5g shows the average AFC execution



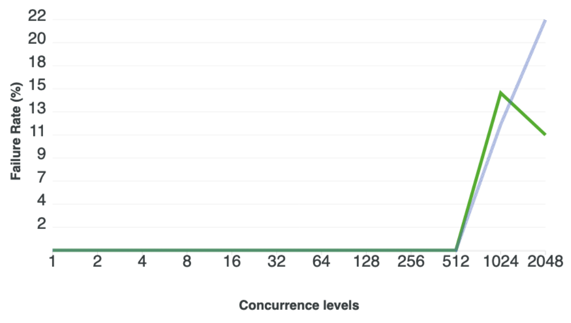
(a) AWS Lambda execution times.



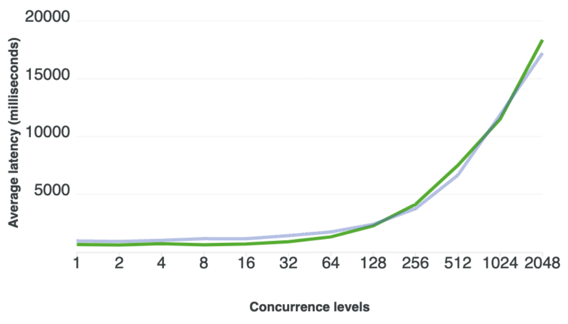
(b) AWS Lambda failure rates.



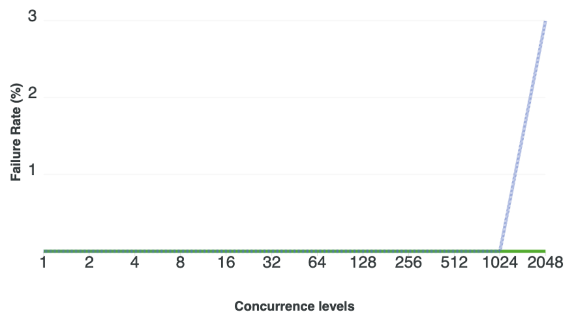
(c) GCF execution times.



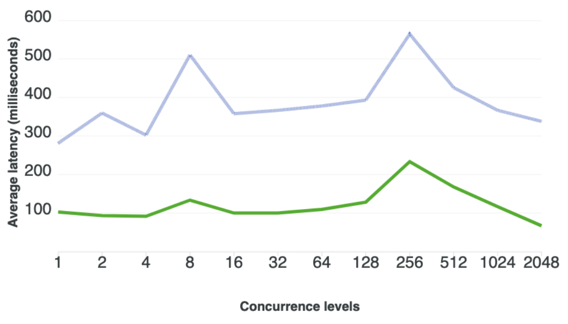
(d) GCF failure rates.



(e) AZF execution times.



(f) AZF failure rates.



(g) AFC execution times.

Figure 5: Comparative results.

times. It is possible to observe three points where the provider seems to have reinforced the infrastructure, such as from 2 to 4 requests, from 8 to 16, and from 256 to 512. This meant that the provider maintained the lowest average times of the experiment and did not present an increasing curve of the average time, as occurred with the other providers. In addition, there was no occurrence of failure during the ten repetitions of the test batteries. The AFC result also shows the expected result of the experiment, in which it is possible to observe the same behavior both in the scenario with workers concentrated in the same region, and with distributed workers, differing only in the average level.

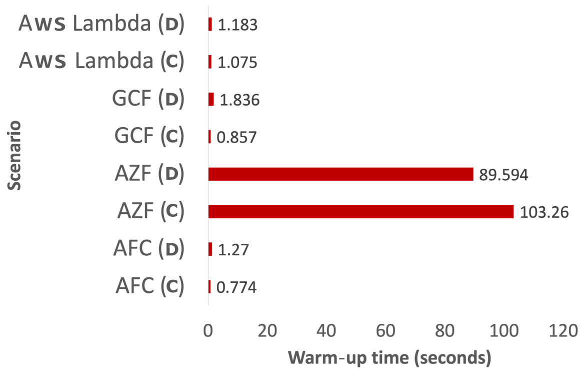


Figure 6: Warm-up times.
(C) → concentrated workers — (D) → distributed workers.

Another interesting aspect to be analyzed in this experiment is the cold start phenomenon, that is, the time it takes the provider to put its FaaS into operation for the first time or after a certain period of inactivity. Figure 6 provides the warm-up times for all eight scenarios. This is the first request made before the start of the battery of tests. The arcing times showed a big difference between AZF and the other providers. While AWS Lambda, GCF and AFC recorded warm-up times around 1s, AZF took 89 seconds in the distributed scenario and 103 seconds in the concentrated scenario. This difference shows that the AZF deployment strategy is considerably different from the others and this can significantly impact the performance of applications supported by this service, since after some downtime, AZF FaaS will have a much higher response time than usual and this will certainly negatively impact the user experience.

In order to understand the impact of the provider and concurrence factors on the results, six factorial designs were elaborated comparing the four providers with each other and the minimum and maximum concurrence levels (1 and 2048), as shown in Figure 7. In Figures 7a, 7b and 7d, it is possible to observe the predominance of the concurrence factor, to the detriment of the provider factor, which indicates that in these benchmark results, the providers' strategies influenced the result less than the differences between the concurrences. In Figures 7c, 7e and 7f, there is a predominance of the provider factor and this corroborates the results shown previously for average execution time and failure rate, since AFC participates in both factorial designs.

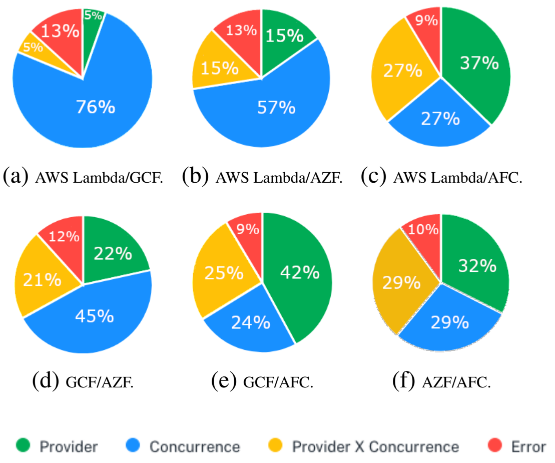


Figure 7: Factorial design results.

The result of the t-tests for the six comparisons between the providers is shown in Table 3. It is possible to notice that all the differences between the average results were considered statistically relevant with a 99.95% confidence level. From which it may be stated that these differences are not irrelevant and may be considered as defining the result of the benchmarks.

Table 3: T-test results.

Scenario	**Difference (ms)**	**Standard deviation**	**Confidence level (%)**
AWS x GCF	1,317.03	281.82	99.95
AWS x AZF	7,792.25	928.90	99.95
AWS x AFC	5,454.88	259.53	99.95
GCF x AZF	9,109.28	898.69	99.95
GCF x AFC	4,137.84	110.14	99.95
AZF x AFC	13,247.13	891.95	99.95

The results of this work can serve as input in decision-making processes according to the characteristics and requirements of the real use case. For example, if the use case requires high reliability of the FaaS service, AFC would be the best option among the evaluated providers, since it did not present failures. On the other hand, if the use case is very sensitive to cold start, then the Azure provider should be avoided, as it presented high values in this regard.

6 CONCLUSION

In this work, the distributed architecture of the Orama framework was presented, with which it is possible to perform benchmarks with high levels of concurrency on a FaaS environment, as well as configure the load to be triggered from different locations on the globe, considerably expanding the range of possibilities for benchmarks against state-of-the-art tools.

The Orama framework allows the visualization of different scenarios for the same use case, especially for concurrency levels above 512 concurrent requests, which is the amount observed as the safest maximum for management by an intermediate configuration instance. In addition, it was evidenced that the Orama framework in its distributed architecture is capable of visualizing the same behavior as the FaaS provider when subjected to a concentrated and distributed approach of bursts of requests, as well as eventual differences.

In the experiments, it was also evident that the AFC FaaS delivers greater consistency in terms of average execution time and occurrence of failures, followed by AWS Lambda and CGF, which registered close results, and finally the AZF results with high average execution times and cold start.

In future work, other providers will be integrated into the Orama framework, such as IBM and Oracle, in order to expand the coverage of the analyses presented in this work. Furthermore, even higher levels of concurrency can be evaluated by designing experiments that include a larger number of distributed workers. Furthermore, evolutions in the Orama framework will allow the analysis of benchmark results using percentiles.

REFERENCES

Back, T. and Andrikopoulos, V. (2018). Using a microbenchmark to compare function as a service solutions. In *ECSOCC*, pages 146–160. Springer.

Barcelona-Pons, D. and García-López, P. (2021). Benchmarking parallelism in faas platforms. *Future Generation Computer Systems*, 124:268–284.

Carvalho, L. and Araujo, A. (2022). Orama: A benchmark framework for function-as-a-service. In *Proceedings of the 12th CLOSER*, pages 313–322. INSTICC, SciTePress.

Carvalho, L. R. and Araujo, A. P. F. (2020). Performance comparison of terraform and cloudify as multicloud orchestrators. In *2020 20th IEEE/ACM CCGRID*, pages 380–389.

Copik, M., Kwasniewski, G., Besta, M., Podstawski, M., and Hoefler, T. (2021). Sebs: A serverless benchmark suite for function-as-a-service computing.

DUAN, Y., Fu, G., Zhou, N., Sun, X., Narendra, N. C., and Hu, B. (2015). Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. volume 00, pages 621–628.

Erinle, B. (2013). *Performance testing with JMeter 2.9*. Packt Publishing Ltd. ISBN: 9781782165842.

Gartner (2022). Gartner forecasts worldwide public cloud end-user spending to reach nearly $600 billion in 2023. [online; 01-Jan-2023; url: https://tinyurl.com/3z72zebw].

Grambow, M., Pfandzelter, T., Burchard, L., Schubert, C., Zhao, M., and Bermbach, D. (2021). Befaas: An application-centric benchmarking framework for faas platforms.

Gupta, P. R., Taneja, S., and Datt, A. (2014). Using heat and ceilometer for providing autoscaling in openstack. *JIMS8I-International Journal of Information Communication and Computing Technology*, 2(2):84–89.

HashiCorp (2021). Terraform: Write, plan, apply. [online; 11-Aug-2021; url: https://www.terraform.io].

Ibrahim, M. H., Sayagh, M., and Hassan, A. E. (2021). A study of how docker compose is used to compose multi-component systems. *Empirical Software Engineering*, 26(6):128.

Jain, R. (1991). The art of computer systems: Techniques for experimental design, measurement, simulation, and modeling. ISBN:13.978-0471503361.

Kuhlenkamp, J., Werner, S., Borges, M. C., El Tal, K., and Tai, S. (2019). An evaluation of faas platforms as a foundation for serverless big data processing. In *Proceedings of the 12th IEEE/ACM*, UCC'19, page 1–9, NY, USA. ACM.

Maissen, P., Felber, P., Kropf, P., and Schiavoni, V. (2020). Faasdom. *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*.

MELL, P. and Grance, T. (2011). The NIST definition of cloud computing. *National Institute of Standards and Tecnology*.

Motta., M. A. D. C., Reboucas De Carvalho., L., Rosa., M. J. F., and Favacho De Araujo., A. P. (2022). Comparison of faas platform performance in private clouds. In *Proceedings of the 12th CLOSER,*, pages 109–120. INSTICC, SciTePress.

Sax, M. J. (2018). *Apache Kafka*, pages 1–8. Springer International Publishing, Cham. ISBN: 978-3-319-63962-8.

Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N. J., Popa, R. A., Gonzalez, J. E., Stoica, I., and Patterson, D. A. (2021). What serverless computing is and should become: The next phase of cloud computing. *ACM*, 64(5):76–84.

Somu, N., Daw, N., Bellur, U., and Kulkarni, P. (2020). Panopticon: A comprehensive benchmarking tool for serverless applications. In *2020 COMSNETS*, pages 144–151.

Wen, J., Liu, Y., Chen, Z., Chen, J., and Ma, Y. (2020). Characterizing commodity serverless computing platforms. DOI:10.48550/ARXIV.2012.00992.

Wittig, M. and Wittig, A. (2018). *Amazon web services in action*. Simon and Schuster. ISBN: 978-1617295119.

Capítulo 3

Como FaaS integrados a DBaaS se comportam em diferentes regiões: uma avaliação por meio do *framework* Orama

How FaaS with DBaaS performs in different regions: an evaluation by the Orama Framework

Leonardo Rebouças de Carvalho[1], **Bruno Kamienski**[1], **Aleteia Araujo**[1]

[1]Department of Computer Science – University of Brasilia (UnB)
Campus Darcy Ribeiro – 70.910-900 – Brasilia – DF – Brazil

{leouesb,brunosabreu}@gmail.com, aleteia@unb.br

Abstract. *Studies indicate that cloud services based on the serverless paradigm, such as Function-as-a-Service (FaaS) should become the main mechanisms of the next generation of cloud computing. Given this perspective, public cloud providers have made efforts to expand the coverage of their services in order to meet this need. However, the effort needed to maintain equivalence between different regions highlights the importance of studying the behavior of FaaS environments in different regions of providers. This work presents a study aided by the Orama framework in order to evaluate the performance of the main FaaS integrated with Database-as-a-Service (DBaaS) services in five regions spread across the globe. The results indicate that the Alibaba provider was able to guarantee good equivalence between its regions, in addition to a lower average execution time. AWS and GCP had similar results, although the error rate on AWS was the highest on average. Azure, on the other hand, had the worst performance, with the highest average execution time, in addition to significant failure rates.*

1. Introduction

Serverless computing [Nupponen and Taibi 2020] as the default cloud programming paradigm have become an increasingly present idea in recent publications and this shows the importance it has gained for cloud computing. Function-as-a-Service (FaaS) [Schleier-Smith et al. 2021] allow users to publish functions written in some programming language supported by the provider and configure a trigger. When triggered, it is the role of the provider to ensure proper processing, whether in the face of low demand or when subjected to high levels of competition. The respective adjustment in the infrastructure takes place without any user intervention. This autascaling feature, combined with the billing model based on activating functions, explains the recent success of this service model. In this context and considering the Everything-as-a-Service (XaaS) concept, the main public cloud providers have been massively investing in serverless-oriented services, especially in Function-as-a-Service (FaaS) [Schleier-Smith et al. 2021].

Since the need for providers to be as close as possible to the end user, it is a common strategy used by cloud companies to deploy infrastructures geographically distributed around the world. For this strategy to be effective, it is important that the products marketed through the cloud are available in as many geographic locations as possible, and this introduces a major challenge in this context: maintaining equivalence for the same service across regions. In addition, in real solutions it is very common that different cloud services are combined to compose the solution, therefore, cloud providers offer various solutions for data storage, among which stand out Database-as-a-Service (DBaaS) [ZHENG 2018] in which providers deliver database environments fully managed by them.

Taking into consideration the growth perspective of FaaS adoption, as well as the possibility of different implementations across regions impacting the performance of applications operating in environments of this nature, this paper evaluates the main FaaS in different regions. Five important regions of the planet where AWS, GCP, Azure and Alibaba have deployed infrastructures were chosen to receive one of the available use cases of the Orama framework [Carvalho. and Araujo. 2022]. Using FaaS integrated with the respective DBaaS, several test batteries were executed simulating concurrent accesses to services from 1 simultaneous request to up to 4096 parallel accesses. The processes of provisioning FaaS environments, running tests, analyzing results and deprovisioning environments were carried out using the Orama framework. The results indicate that Alibaba apparently implements a more efficient management strategy for its FaaS platform in all evaluated regions, since its average execution time was the lowest among the providers, as well as its failure rate. AWS and GCP obtained intermediate and very close results. Azure, on the other hand, recorded the worst results, both in average execution time and in failure rates.

This article is divided into six parts, the first being this introduction. Section 2 presents the theoretical foundation that supports this work. Section 3 presents the related works. Section 4 describes the methodology used in the experiments carried out. Section 5 shows the results obtained, and finally Section 6 presents the conclusions and future work.

2. Background

In traditional cloud computing models, such as Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) [MELL and Grance 2011], in general the charge for the service is based on the operating time of the servers, even if they are idle, in addition, any increases in demand on the systems supported by these platforms should receive special attention from the client, either to configure elasticity strategies or even to implement them themselves. The FaaS model [Malawski et al. 2020], on the other hand, relieves the customer of responsibility for the elasticity of the environment, since this characteristic is generally intrinsic to the service. Furthermore, billing in the context of FaaS is based on the actual activation of the service rather than on the operation of the servers.

In addition to the billing model and automatic elasticity, FaaS also offers a simplification of the deployment process, since it is up to the provider to deploy the respective runtime to execute the functions, leaving the customer only to submit a snippet of source code and configure a trigger for the service to be ready for use [Nupponen and Taibi 2020].

Currently, the main public cloud providers have FaaS solutions. AWS, for example, offers Lambda [AWS 2021] in its 30 regions around the world. Azure, which currently has 60 regions, offers Azure Functions (AZF) [Microsoft 2021], while Google Cloud Function (GCF) [Google 2021] is available in all 35 GCP regions. Alibaba Cloud, on the other hand, has the Function Compute (AFC) [Cloud 2021] service in its 24 regions. Maintaining operational equivalence between all these regions is a huge challenge faced daily by providers and each adopted strategy can impact performance. These strategies include, but are not limited to hardware, software, and resource allocation approaches.

FaaS was originally designed to operate in an isolated and stateless manner [García López et al. 2018]. However, it is common for real-world solutions to involve other products within the cloud ecosystem, such as object storage, databases, among others. A common product to associate with FaaS is DBaaS. In this context FaaS and DBaaS act as a storage solution whose API is under the responsibility of FaaS, while data management is the responsibility of DBaaS. Considering that this type of association can involve different strategies adopted by providers, including different forms between their own regions, this work investigates how environments like these operate in the main clouds in different regions, and to assist in this evaluation the Orama framework [Carvalho. and Araujo. 2022] was used.

The Orama framework [Carvalho. and Araujo. 2022] is a tool whose objective is to aid in benchmark execution over FaaS environments. The framework enables some built-in use cases that can be provisioned and deprovisioned automatically. Besides this, the Orama framework coordinates the benchmark execution from these configurations. With Orama it is possible to provision a FaaS use case in seconds, perform different concurrence scenario tests, adjust configuration on the environment, execute the tests again and analyze the results with the comparative and statistical tools offered by the platform. The Orama framework can be configured to work standalone, however its ability to activate FaaS will be limited to the amount of resources available on the machine where it is installed. It is also possible to configure the Orama framework to act in a distributed way following the "master/workers" architecture in which the workers will be responsible for activating the FaaS and thus the concurrence load can be divided among the worker nodes configured in the framework environment, in this way by increasing the capacity of the platform's concurrence levels.

Considering that the results analysis phase is a crucial step for understanding the performance of FaaS environments, the Orama framework provides two statistical analysis tools. The factorial design [Jain 1991] helps in identifying factors that influence the results. In the Orama framework it is possible to build a 2^k factorial design, with 2 being the lower and upper levels of the factors and the k the number of factors. In Orama, two factors are considered: level of concurrence and provider, so the factorial design implemented is in 2^2 format. If the results are composed of more than one round (repetition), then it will be possible to analyze the statistical error of the factorial design. If the statistical error is high, this indicates the existence of another factor in addition to those initially mapped. Another statistical tool available in the Orama framework is the paired t-test. In this test, the statistical significance of the difference found in two juxtaposed results is analyzed. The higher the confidence level is, the more statistically significant a difference will be. On the other hand if this confidence level is very low or not observed, then the difference is negligible and the results can be considered statistically equal.

The Orama framework has some built-in use cases that can be used to quickly provision FaaS environments, which the framework can benchmark against. These use cases consist of automation artifacts configured to deploy environments of a simple calculator, a function for genetic sequence alignment, functions acting as API for object storage and DBaaS. The latter shown in Fig. 1, in which it is possible to observe the deployment of

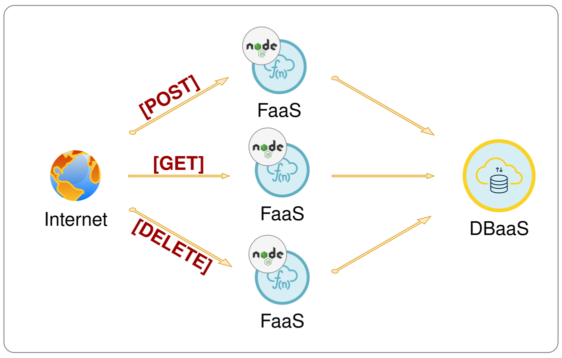


Figure 1. FaaS for DBaaS Orama framework built-in use case.

three FaaS to handle GET, POST and DELETE requests. These functions were written in Node.js considering the wide adoption of this language and are intended to act as an API for managing data saved in the data storage solution of the respective cloud. Therefore, these functions will be reflected in the respective DBaaS at the target provider, that is DynamoDB in AWS, Firebase in GCP, CosmosDB in Azure, and Tablestore in Alibaba Cloud.

The Orama Framework contains built-in functions for various purposes, from a simple calculator, whose purpose is only to validate the FaaS flow, to real functions for aligning genetic sequences. Other examples of built-in functions offered by Orama framework are APIs for storing data in Object Storage or in DBaaS. Considering that solutions involving databases are frequently adopted, in this work the Orama framework use case chosen was that which deploys FaaS integrated with the respective DBaaS to evaluate the performance of this type of environment in different regions. A detailed description of the methodology adopted will be provided in the Section 4.

3. Related Works

This paper addresses the experiment-driven evaluation of FaaS platforms in different regions under different concurrence levels using the Orama framework, including very high concurrence scenarios, such as 2048 and 4096 concurrent requests. The related works are discussed from the perspective of benchmarking FaaS platforms and are shown in Table 1.

In the paper [Back and Andrikopoulos 2018] the authors used a microbenchmark in order to investigate two aspects of the FaaS: the differences in observable behavior with respect to the computer/memory relation of each FaaS implementation by the providers, and the complex pricing models currently in use. They used AWS, IBM, GCP, Azure, and OpenWhisk in their evaluation. However, the authors did not present an evaluation of the performance of their microbenchmark in different regions of the providers, especially in the face of different levels of concurrence, as presented in this work.

The quality impacts of operational tasks in FaaS platforms as a foundation for a new generation of emerging serverless big data processing frameworks and platforms are evaluated in [Kuhlenkamp et al. 2019]. The authors presented SIEM, a new evaluation method to understand and mitigate the quality impacts of operational tasks. They instantiated SIEM to evaluate deployment package and function configuration changes for four

Table 1. Related Works.

	[Back and Andrikopoulos 2018]	[Kuhlenkamp et al. 2019]	[Barcelona-Pons and García-López 2021]	[Wen et al. 2021]	[Somu et al. 2020]	[Grambow et al. 2021]	[Motta et al. 2022]	**This paper**
Providers	AWS, IBM, GCP, Azure, and OpenWhisk	AWS, IBM, GCP, and Azure	AWS, IBM, GCP, and Azure	AWS, Azure, GCP, and Alibaba	AWS and GCP	AWS , GCP, Azure, TinyFaaS, OpenFaaS, and OpenWhisk	Fission, OpenFaaS and OpenWhisk	**AWS, Azure, GCP, and Alibaba**
Factorial Design	-	-	-	-	-	-	✓	✓
T-test	-	-	-	-	-	-	✓	✓
Distributed	-	-	-	-	-	-	-	✓

major FaaS providers (AWS, IBM, GCP, and Azure), but only in European regions for the same level of concurrence. In this work, on the other hand, several levels of concurrence are evaluated in five regions for each of the providers involved in the analysis, totaling 20 regions.

In paper [Barcelona-Pons and García-López 2021] the authors analyzed the architectures of four major FaaS platforms: AWS Lambda, AZF, GCP, and IBM Cloud Functions. The research focused on the capabilities and limitations the services offer for highly parallel computations. The design of the platforms revealed two important traits influencing their performance: virtualization technology and scheduling approach. This work, on the other hand, focuses on investigating the differences in performance of the main providers in their different regions, including in the face of different levels of concurrence.

In [Wen et al. 2021], the authors ran a test flow employing micro benchmarks (CPU, memory, I/O, and network) and macro benchmarks to evaluate FaaS from AWS, Azure, GCP, and Alibaba in detail (multimedia, map-reduce and machine learning). The tests made use of specific Java, Node.js, and Python methods that investigated the benchmarking attributes to gauge resource usage efficiency and initialization delay. However, they did not present evaluations in different regions, with different levels of concurrence.

PanOpticon [Somu et al. 2020] provides a comprehensive set of features to deploy end-user business logic across platforms at different resource configurations for fast evaluation of their performance. The authors conducted a set of experiments testing separate features in isolation. An experiment comprising a chat server application was conducted to test the effectiveness of the tool in complex logic scenarios in AWS and GCP. Furthermore, in this work, the range of tests that the Orama framework can evaluate was extended beyond the execution of benchmarks on AWS and GCP, to include the execution of benchmarks on Azure and Alibaba, which are two other important players in this market.

BeFaaS [Grambow et al. 2021] offers a benchmark methodology for FaaS settings that is application centric and focuses on evaluating FaaS apps using real-world and prevalent use cases. It offers enhanced result analysis and federated benchmark testing, where the benchmark application is split across several providers. It does not, however, provide a superior approach to statistical analysis, such as the factorial design or t-test that are covered by this study.

The main FaaS platforms for private cloud deployment are subjects of evaluation at [Motta et al. 2022]. Some FaaS-dom functions are subjected to different levels of concurrence in Fission, OpenFaaS and OpenWhisk. Based on the results, an analysis is performed using a factorial design. However, the configurability is limited and a t-test is not presented in order to validate the statistical significance of the differences, as is done in this work.

4. Methodology

Since this work addresses the comparative study of the performance of FaaS environments in different regions, the various infrastructure deployment positions of AWS, GCP, Azure and Alibaba providers were confronted in order to find macro-regions in which there was a presence of both providers so they can be compared against each other with minimal impact on network latency. Thus, five macro-regions were found where it is possible to verify concentrations of cloud supply in the East and West regions of the United States, in Europe, in the Asian Pacific region and in Oceania. It is noteworthy that in the regions of Europe and Oceania it was necessary to select regions that were not exactly in the same micro-region due to the lack of availability of both services involved in the experiment (FaaS and DBaaS), however the selected region was the closest operating the respective services together.

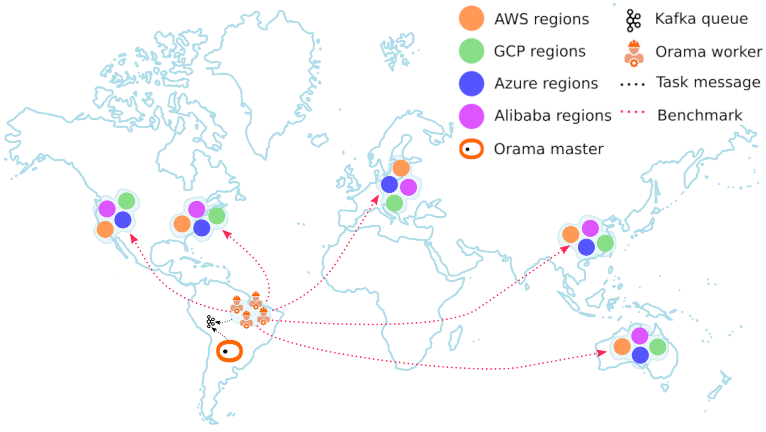


Figure 2. Overall architecture of the experiment.

Solutions involving the use of FaaS in collaboration with database services are common choices when solving real-world problems. Thus, in this work, the choice of the use case of the Orama framework that deploys FaaS in different providers integrated with DBaaS solutions in providers such as DynamoDB on AWS, Firestore on GCP, CosmosDB on Azure, and TableStore on Alibaba Cloud is justified.

With the purpose of submitting the FaaS environments to different levels of concurrence, 13 test scenarios were defined with degrees of concurrence starting at 1 and ending at 4096 with exponential growth, that is, scenarios with 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 concurrent requests to FaaS. Each test battery was configured in the Orama framework to be repeated 10 times in order to build an average of execution times.

Considering the highest levels of concurrence (2048 and 4096) it was decided to implement the Orama framework in master worker mode, since at the highest level each worker would be responsible for activating the FaaS with 1024 concurrent requests. Therefore, the Orama framework was deployed on a GCP Compute Engine instance in the São Paulo/Brazil region. The master node had 4 vCPUs and 16GB of RAM (e2-standard-4), while each of the 4 workers had 2 vCPUs and 4GB of RAM (e2-medium). All instances used the Debian 11 operating system.

As can be seen in Fig. 2, once the Orama framework had been deployed in a region in South America, the requests for activating the FaaS departed from there and traveled through the network until reaching the respective regions, where the target services were allocated. A difference in latency between the regions was expected, since their positions are not identical. However, as the focus of this work is to analyze the differences in implementations between regions, it was considered sufficient to just allocate close regions between the providers in order to mitigate the impacts of the latency difference.

Once the framework had been implemented, the deployments of the respective environments of the use case of FaaS with DBaaS were requested for each region of each provider, totaling 20 provisionings. Each provisioning was submitted to battery repetitions, so that the results could be analyzed. The analysis of the results obtained from the methodology described in this section will be dealt with in the next section.

5. Results

The analysis detailed in this section approaches the results from two points of view. First, a comparison between the performances found in all regions of the same provider, in order to assess whether the adopted strategies are equivalent between the regions of the same provider. Next, an analysis is carried out from the point of view of the region, comparing the performance of different providers in the same region, allowing the assessment of the difference between providers in the respective region.

Figs. 3a, 3b, 3d, and 3d show the average execution times achived in each level of concurrence by each provider in its five evaluated regions. As may be observed in Fig. 3a and 3b, the average times of Lambda and GCF services show levels close to around 2k ms, with Lambda peaking at close to 8k ms, while GCF peaked at around 6k ms. It is important to observe the performance of these providers under 1024 concurrent requests, in both cases there are sharp drops in the average execution time for higher levels of concurrence. This phenomenon can be explained by the characteristic of automatic elasticity intrinsic to the FaaS model. It is also possible to observe in Fig. 3a and 3b behavior equivalent to AWS and GCP regions.

Fig. 3c shows the average execution times calculated for the AZF service. First, it is important to note that the average time threshold in AZF is significantly higher compared to the threshold recorded by AWS and GCP. While AWS and GCP recorded average times of around 2k ms, AZF recorded average times of around 20k ms, and peaks of 50k

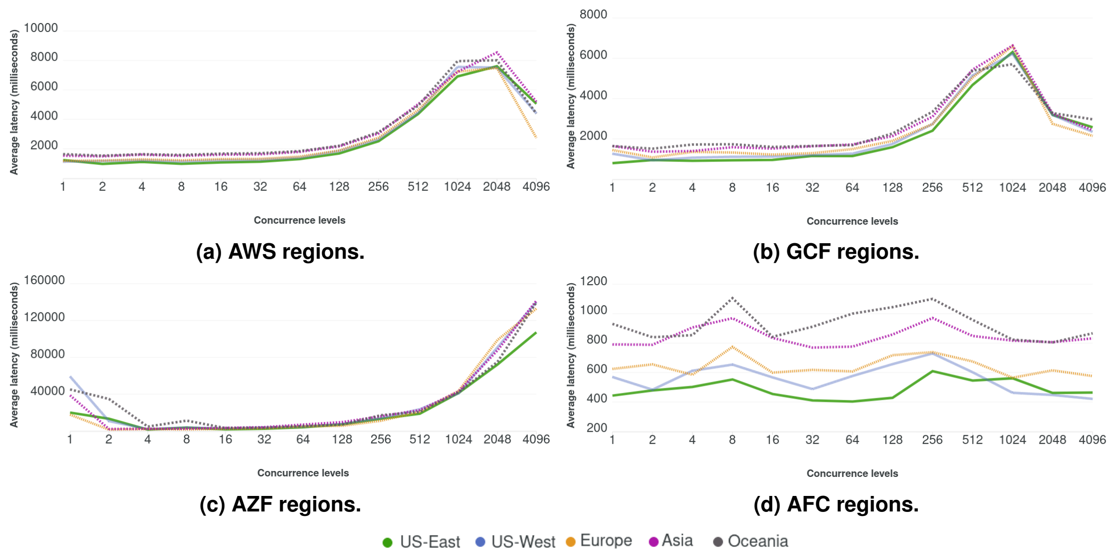
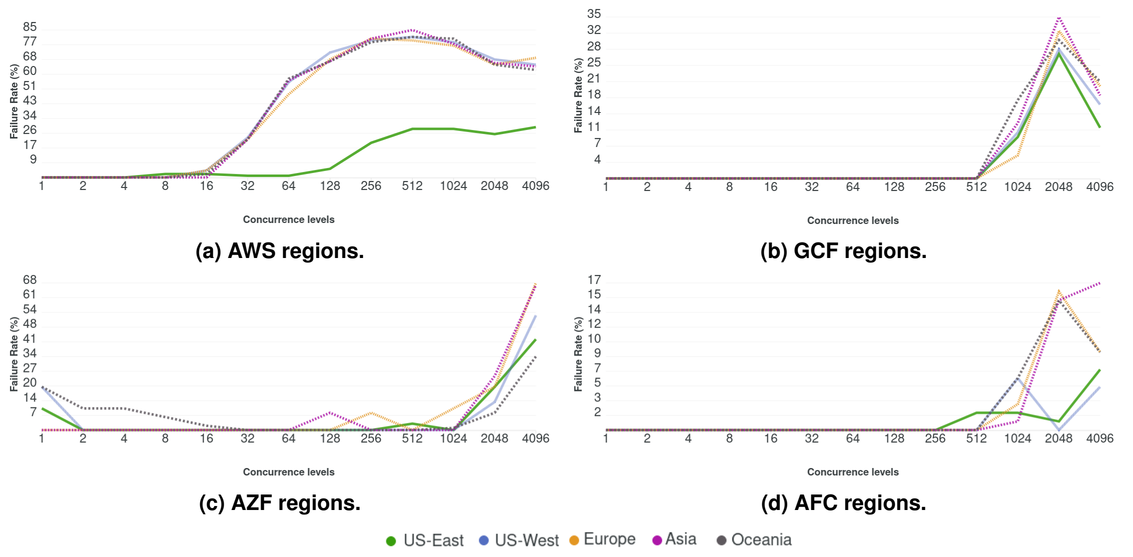


Figure 3. Average execution time by provider.

and 140k ms. This shows that Azure services, in addition to having higher average execution times than AWS and GCP, also deliver a wide range of execution time, since as the level of concurrence increased the average execution time also increased. There was no evidence of an intervention by the provider to reinforce the infrastructure to meet the increase in demand. In addition, given the repetitive and sequential nature of the tests performed, the overload of the highest levels of concurrence negatively impacted the initial levels, since unexpected increases in the average execution time can be observed at 1 and 2 concurrent requests. This same performance can be observed in all evaluated AZF regions.

In Fig. 3d the average execution times of the AFC regions are presented. It may be observed that the level presented is the lowest compared to the others, since the average times ranged from around 400 ms to around 1k ms. Unlike the increasing behavior of the execution time observed in Lambda, GCF and AZF results, in AFC the average execution time maintains a stable level, even at high levels of concurrence. This shows efficient management by the provider when dealing with oscillating and increasing demand. Although a comparison of the regions shows differences in thresholds, this difference can be attributed to the latency, which at lower thresholds is more evident.

While the test batteries were carried out, failures in the processing of requests were recorded. Fig. 4 presents the number of failures that occurred at each level of concurrence in the five regions of each provider. Lambda, as seen in Fig. 4a, registered failures starting from 16 simultaneous requests and this number increased to 512 when there was a small reduction, certainly due to the intervention of the provider in the infrastructure. Lambda web console has a "simultaneity" setting for the function that is fixed at 50, that is, only 50 instances of the function can be running simultaneously. This parameter can only be changed through a support request to AWS. Another significant fact presented in Fig 4a is the discrepancy in the incidence of failures between the US-East region and the others. As there was a lower failure rate in the US-East region and considering that this region is the pioneer of the provider, it is possible to assume that in this region this service has reached a higher degree of maturity in relation to the other AWS regions.



(a) AWS regions.

(b) GCF regions.

(c) AZF regions.

(d) AFC regions.

Figure 4. Failure rates (%) by provider.

GCF failure rates are shown in Fig. 4b where it may be seen that in this provider the errors started to occur from 1024 concurrent requests and grew to about 35% at 2048 simultaneous requests when there was a decrease in the error rate, due to the intervention of the provider in the infrastructure to readjust itself to the demand. It is noteworthy that while the error rate peaks in GCF at around 35% of requests, in Lambda this level is much higher, peaking at around 80%.

Fig. 4c shows the failure rates calculated at AZF where failure rates are perceived at all levels of concurrence, reaching peaks of 60% failures at 4096 simultaneous requests. Between 1024 and 4096 it is possible to observe an upward trend in the failure rate that accompanies increased concurrence. In addition, the error rates calculated in the first concurrence level are noteworthy, as it is likely that this rate was impacted by the failures that occurred at level 4096 of the previous repetition. The failure rates calculated in AFC are lower than the others, as seen in Fig. 4d. At the peak point, at 2048 concurrent requests, AFC recorded only around 16% failures. It is worth mentioning that the region where the service obtained the lowest failure rates was in the Eastern region of the United States.

Another way of viewing the occurrence of errors is shown in Fig. 5 where it is possible to see the distribution of errors between the regions. It may be noted that the distribution of failures in the Lambda and GCF regions is balanced, except in the US-East region where the rates are lower in both providers. This indicates that the degree of maturity in this region is higher and because of this, the strategies and resources available in these places contribute to the reduction of failures. In AZF more than half of the failures occurred in the regions of Europe or Asia while Oceania recorded the lowest failure rates. AFC failure rate was concentrated in the Asia region with almost 34% of the failures being located there. Given the good performance in terms of AFC execution time, it is likely that this concentration of failures precisely in the region where the provider has the greatest market focus is due to the concurrence with the other AFC customers.

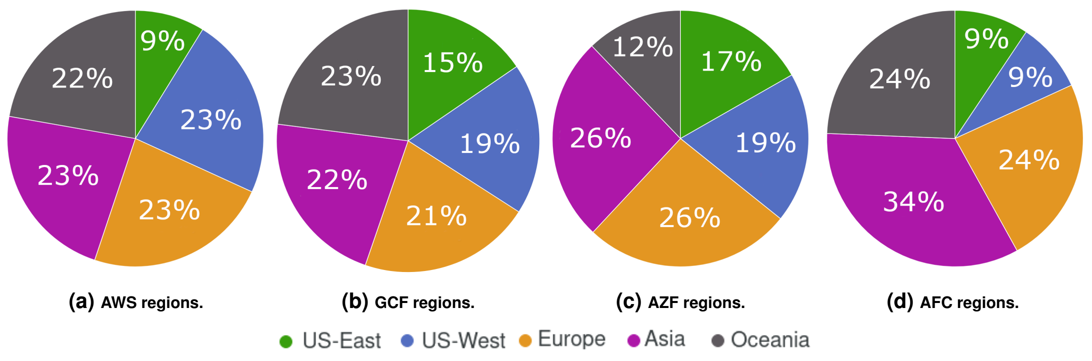


Figure 5. Regional failure distribution.

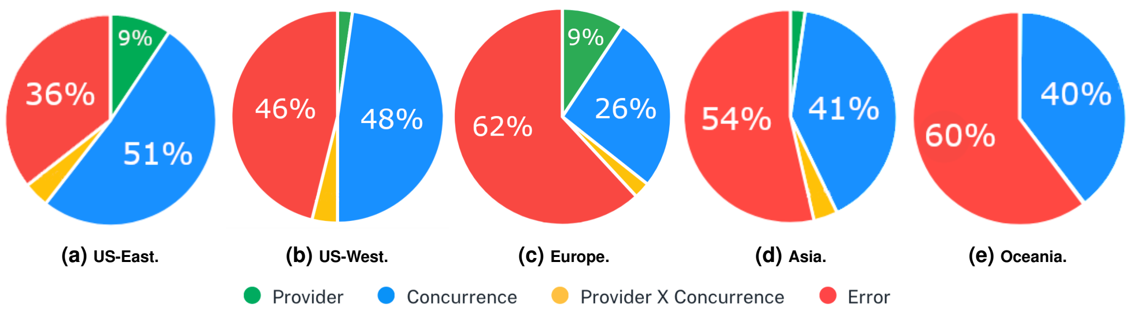


Figure 6. Factorial design results for Lambda vs. GCF.

As mentioned in Section 2, the Orama framework provides factorial design and the t-test as statistical analysis tools. The comparative analysis of the results reveals a great proximity between the Lambda and GCF results. In order to understand the effects of these results, a factorial design was created between the Lambda and GCF results using the lowest and highest level of concurrence (1 and 4096).

Fig. 6 shows the results of the factorial design effects, and highlights the prevalence of statistical error, which indicates the existence of another factor (in addition to the provider and the level of concurrence) influencing the result. Given the proximity of the approaches adopted by the providers, it is likely that this factor is the latency between the region in which the Orama framework was installed and the infrastructure of the providers. Another factor that predominates in the results is concurrence, indicating that the results were more affected by the difference in concurrence than by the difference between providers, that is, it may therefore be inferred that the strategies of AWS and Google in their FaaS were equivalent in the tests carried out in this paper.

The t-test results between Lambda and GCF are shown in Table 2. It is possible to observe that all differences found between providers have some level of confidence for statistical significance. In the European region, for example, the confidence level calculated by the Orama framework was only 70%, which is a relatively low confidence level. On the other hand, in the US-East region, the difference was considered more significant, with a 97.5% confidence level.

The results of the respective analyses provided in this section were obtained through the Orama framework, whose configurability allowed the adaptation of its orig-

Table 2. T-test results between Lambda and GCF.

Region	Difference	Standard deviation	Confidence level
US-East	1628.82	503.18	97.5%
US-West	889.78	432.67	95%
Europe	150.91	233.75	70%
Asia	966.60	425.49	95%
Oceania	792.39	465.21	90%

inal use case so that a proper evaluation of the FaaS environments of the main public cloud providers in this market could be obtained. Considering the consistent growth of the FaaS cloud computing model, it is essential to evaluate and understand the strategies adopted by providers in their service offerings of this nature. Since FaaS is foreseen as the main engine of the next generation of cloud computing, the more improved and better the delivery of providers on this paradigm, the more qualified will be the impact for the next generation of the most revolutionary archetype of computational infrastructure, that is, the cloud.

6. Conclusion

In this work, the performances of FaaS environments deployed in five different regions in each of the main public cloud providers were analyzed. AWS, Azure, Google and Alibaba cloud had their respective FaaS subjected to successive batteries of tests with the help of the Orama framework, which also assisted in the provisioning of the use case relating FaaS to DBaaS, as well as in the comparative and statistical analysis tools.

The results showed that, in general, the different regions of the evaluated providers deliver equivalent performances, except for the US-East region of Lambda, whose results outperformed the other regions of the provider, possibly due to its higher level of maturity.

The analysis of the average execution times showed that AFC led the results, presenting the lowest average times at all evaluated levels of concurrence, followed by Lambda and GCF, practically equal in their performances. Finally, AZF registered the highest average times in all tested regions.

Failure rate analysis confirmed AFC's lead in this assessment, registering the lowest rates followed by GCF. AWS and AZF delivered remarkably high failure rates, especially at higher levels of concurrence.

In future work, other test cases will be evaluated, such as FaaS integrated with object storage, for example. In addition, as other FaaS providers such as IBM and Oracle are integrated, these providers must be subject to the same evaluation conditions as those in this work in order to broaden the understanding of the aspects explored in this work.

References

AWS (2021). AWS lambda. `https://aws.amazon.com/en/lambda`. [online; 11-Aug-2021].

Back, T. and Andrikopoulos, V. (2018). Using a microbenchmark to compare function as a service solutions. In *ECSOCC*, pages 146–160. Springer.

Barcelona-Pons, D. and García-López, P. (2021). Benchmarking parallelism in faas platforms. *Future Generation Computer Systems*, 124:268–284.

Carvalho., L. and Araujo., A. (2022). Orama: A benchmark framework for function-as-a-service. In *Proceedings of the 12th CLOSER*, pages 313–322. INSTICC, SciTePress.

Cloud, A. (2021). Alibaba cloud function. `https://www.alibabacloud.com/product/function-compute`. [online; 11-Aug-2021].

García López, P., Sánchez-Artigas, M., París, G., Barcelona Pons, D., Ruiz Ollobarren, A., and Arroyo Pinto, D. (2018). Comparison of faas orchestration systems. In *2018 IEEE/ACM UCC Companion*, pages 148–153.

Google (2021). Cloud functions. `https://cloud.google.com/functions/`. [Online; 10-Aug-2021].

Grambow, M., Pfandzelter, T., Burchard, L., Schubert, C., Zhao, M., and Bermbach, D. (2021). Befaas: An application-centric benchmarking framework for faas platforms.

Jain, R. (1991). The art of computer systems: Techniques for experimental design, measurement, simulation, and modeling.

Kuhlenkamp, J., Werner, S., Borges, M. C., El Tal, K., and Tai, S. (2019). An evaluation of faas platforms as a foundation for serverless big data processing. In *Proceedings of the 12th IEEE/ACM*, UCC'19, page 1–9, NY, USA. ACM.

Malawski, M., Gajek, A., Zima, A., Balis, B., and Figiela, K. (2020). Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and Google Cloud Functions. *Future Generation Computer Systems*, 110:502–514.

MELL, P. and Grance, T. (2011). The NIST definition of cloud computing. *National Institute of Standards and Tecnology*.

Microsoft (2021). Azure functions. `https://azure.microsoft.com/pt-br/services/functions/`. [online; 11-Aug-2021].

Motta, M. A. C., Carvalho, L. R., Rosa, M. J. F., and Araujo, A. P. F. (2022). Comparison of faas platform performance in private clouds. In *Proceedings of the 12th CLOSER,*, pages 109–120. INSTICC, SciTePress.

Nupponen, J. and Taibi, D. (2020). Serverless: What it is, what to do and what not to do. In *2020 IEEE ICSA-C*, pages 49–50.

Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N. J., Popa, R. A., Gonzalez, J. E., Stoica, I., and Patterson, D. A. (2021). What serverless computing is and should become: The next phase of cloud computing. *ACM*, 64(5):76–84.

Somu, N., Daw, N., Bellur, U., and Kulkarni, P. (2020). Panopticon: A comprehensive benchmarking tool for serverless applications. In *2020 COMSNETS*, pages 144–151.

Wen, J., Liu, Y., Chen, Z., Ma, Y., Wang, H., and Liu, X. (2021). Understanding characteristics of commodity serverless computing platforms.

ZHENG, X. (2018). Database as a service - current issues and its future. *CoRR*, abs/1804.00465.

Capítulo 4

Como FaaS integrados a DBaaS se comportam em diferentes regiões: uma avaliação por meio do *framework* Orama (ERAD-CO)

Como FaaS integrados a DBaaS se comportam em diferentes regiões: uma avaliação por meio do framework Orama

Bruno Abreu Kamienski[1], **Leonardo Rebouças de Carvalho**[1], **Aleteia Araujo**[1]

[1] Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Brasília – DF – Brazil

{brunosabreu, leouesb}@gmail.com, aleteia@unb.br

Abstract. *Public cloud providers have made efforts to expand the coverage of their services based on the serverless paradigm in order to meet the need of next generation of cloud computing. However, it is very important studying the behavior of FaaS environments in different regions of providers. This work presents a study aided by the Orama framework in order to evaluate the performance of the main FaaS integrated with DBaaS services in different regions spread across the globe. The results indicate that the Alibaba provider was able to guarantee good equivalence between its regions, in addition to a lower average execution time. AWS and GCP had similar results, and Azure, on the other hand, had the worst performance and significant failure rates.*

Resumo. *Os provedores de nuvem pública têm feito esforços para expandir a cobertura de seus serviços baseados no paradigma sem servidor, a fim de atender à necessidade da próxima geração de computação em nuvem. No entanto, é muito importante estudar o comportamento dos ambientes FaaS em diferentes regiões de provedores. Este trabalho apresenta um estudo auxiliado pelo framework Orama a fim de avaliar o desempenho dos principais FaaS integrados aos serviços DBaaS em diferentes regiões espalhadas pelo globo. Os resultados indicam que o provedor Alibaba conseguiu garantir uma boa equivalência entre suas regiões, além de um menor tempo médio de execução. AWS e GCP tiveram resultados semelhantes, enquanto a Azure, por outro lado, teve o pior desempenho além de taxas de falha significativas.*

1. Introdução

A computação sem servidor [Nupponen and Taibi 2020] como o paradigma padrão de programação em nuvem tem se tornado uma ideia cada vez mais presente nas publicações recentes e isso mostra a importância que ela ganhou para a computação em nuvem. Function-as-a-Service (FaaS) [Schleier-Smith et al. 2021] permite que os usuários publiquem funções escritas em alguma linguagem de programação suportada pelo provedor e configurem um gatilho que, quando acionado, cabe ao provedor garantir o devido processamento, mesmo diante de altos níveis de concorrência.

Visto a necessidade dos provedores estarem o mais próximo possível do usuário final, frequentemente é utilizada pelas empresas de nuvem a estratégia de implantar infraestruturas geograficamente distribuídas ao redor do mundo. Além disso, é comum que diferentes serviços de nuvem sejam combinados para compor a solução. Portanto, provedores de nuvem oferecem diversas soluções para armazenamento de dados, entre as quais se destacam Database-as-a-Service (DBaaS) [ZHENG 2018] em que os provedores entregam ambientes de banco de dados totalmente gerenciados por eles.

2. Metodologia

Levando em consideração a perspectiva de crescimento da adoção de FaaS, bem como a possibilidade de diferentes implementações entre regiões impactarem o desempenho de aplicações operando em ambientes dessa natureza, foram avaliados os principais FaaS em diferentes regiões. Cinco importantes regiões do planeta onde Amazon WebServices (AWS), Google Cloud Platform (GCF), Azure (AZF) e Alibaba (AFC) possuem infraestruturas implantadas foram escolhidas para receber um dos casos de uso disponíveis do framework Orama [Carvalho and Araujo 2022]. Utilizando FaaS integrado com o respectivo DBaaS, foram executadas diversas baterias de testes simulando acessos simultâneos a serviços desde 1 requisição simultânea até 4096 acessos paralelos. Os processos de provisionamento de ambientes FaaS, execução de testes, análise de resultados e desprovisionamento de ambientes foram realizados utilizando o framework Orama.

O framework Orama [Carvalho and Araujo 2022] é uma ferramenta cujo objetivo é auxiliar na execução de benchmarks em ambientes FaaS. A estrutura permite alguns casos de uso integrados que podem ser provisionados e desprovisionados automaticamente. Além disso, o framework coordena a execução dos benchmarks a partir dessas configurações. O framework pode ser configurado para funcionar de forma autônoma, porém sua capacidade de ativar o FaaS estará limitada à quantidade de recursos disponíveis na máquina onde está instalado. Também é possível configurá-lo para atuar de forma distribuída seguindo a arquitetura "master/workers" em que os workers serão responsáveis por ativar o FaaS e assim a carga de concorrência pode ser dividida entre os nós workers configurados no ambiente do framework, aumentando assim a capacidade de concorrência da plataforma.

Soluções envolvendo o uso de FaaS em conjunto com banco de dados são comuns, justificando a escolha do caso de uso do framework Orama que implanta FaaS em diferentes provedores integrados com soluções DBaaS em provedores como DynamoDB da AWS, Firestore do GCF, CosmosDB da AZF e TableStore da AFC. Com o objetivo de submeter os ambientes FaaS a diferentes níveis de concorrência, foram definidos 13 cenários de teste com concorrência de 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 e 4096 solicitações simultâneas para FaaS. Cada bateria de testes foi configurada no framework Orama para ser repetida 10 vezes a fim de construir uma média de tempos de execução.

Uma vez implementado o framework, foram solicitados os deploys dos respectivos ambientes do caso de uso de FaaS com DBaaS para cada região de cada provedor, totalizando 20 provisionamentos. O Orama foi configurado para realizar uma requisição de pré-acionamento aos serviços e separando-a das demais requisições, permitindo uma análise da partida a frio. Cada provisionamento foi submetido a uma bateria de repetições, para que os resultados pudessem ser analisados.

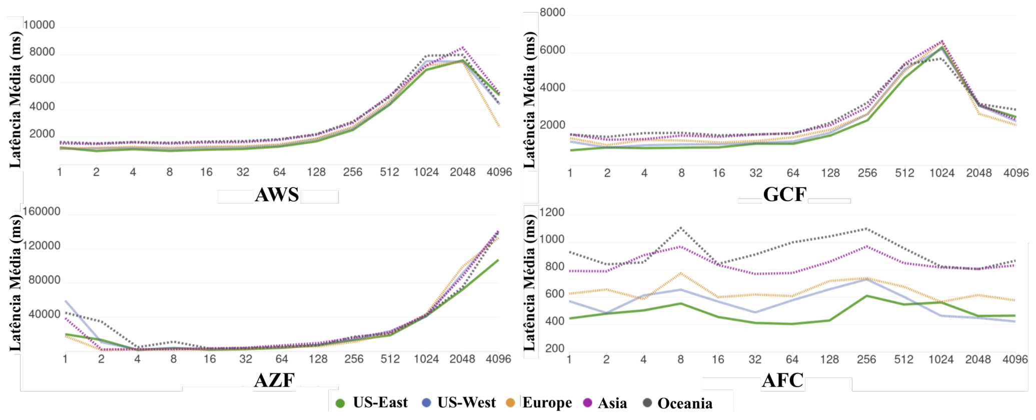


Figura 1. Tempos de execução dos provedores em função da concorrência.

3. Resultados e conclusão

Os resultados mostraram que, de maneira geral, as diferentes regiões dos provedores avaliados entregam desempenhos equivalentes, com exceção da região US-East de Lambda, cujos resultados superaram as demais regiões do provedor, possivelmente devido ao seu maior nível de maturidade.

A análise dos tempos médios de execução mostrou que o AFC liderou os resultados, apresentando os menores tempos médios em todos os níveis de concorrência avaliados, seguido pelo Lambda (AWS) e GCF, praticamente iguais em seus desempenhos. Por fim, AZF registrou os maiores tempos médios em todas as regiões testadas. A análise da taxa de falha confirmou a liderança da AFC nesta avaliação, registrando taxas abaixo de 17%, seguida pela GCF com valores de até 35%. AWS e AZF forneceram taxas de falha notavelmente altas, especialmente em níveis mais altos de concorrência, quando registraram taxas acima de 50%

A análise da partida a frio mostrou mais uma vez a eficiência das estratégias adotadas pela AFC. AWS e GCF registraram tempos de partida a frio equivalentes, enquanto o AZF teve um tempo muito alto para implantar seu ambiente de processamento FaaS em todas as suas regiões. Diante desses resultados, ficou evidente que a AFC registrou os melhores resultados em todos os itens analisados nesta avaliação, seguida da AWS e GCP em nível de qualidade equivalente, e por fim os menores resultados foram registrados pela AZF.

Referências

Carvalho, L. and Araujo, A. (2022). Orama: A benchmark framework for function-as-a-service. In *Proceedings of the 12th CLOSER*, pages 313–322. INSTICC, SciTePress.

Nupponen, J. and Taibi, D. (2020). Serverless: What it is, what to do and what not to do. In *2020 IEEE ICSA-C*, pages 49–50.

Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N. J., Popa, R. A., Gonzalez, J. E., Stoica, I., and Patterson, D. A. (2021). What serverless computing is and should become: The next phase of cloud computing. *ACM*, 64(5):76–84.

ZHENG, X. (2018). Database as a service - current issues and its future. *CoRR*, abs/1804.00465.

Capítulo 5

Conclusão e Trabalhos futuros

Nos experimentos realizados neste trabalho, ficou evidente a maior consistência e performance do provedor Alibaba, com tempos de resposta significativamente menores do que os outros provedores. Os provedores Amazon e Google tiveram desempenho muito similares entre si, e um tempo de inicialização muito próximo ao Alibaba. Entretanto, o provedor Azure registrou tempos de resposta e de inicialização muito acima dos outros provedores, podendo ter um impacto negativo na percepção dos usuários de seu serviço FaaS. Adicionalmente, foi observada uma forte influência da concorrência nas requisições em relação ao comportamento do FaaS nos provedores testados. Uma concorrência acima de mil requisições simultâneas aparentemente resultou em um reforço da infraestrutura FaaS e em uma ligeira queda nos tempos de resposta, exceto no Azure, no qual os tempos cresceram consistentemente de forma exponencial.

No cenário de persistência, de maneira geral, as diferentes regiões dos provedores avaliados entregam desempenhos equivalentes, com exceção da região US-East da AWS, que apresentou um resultado acima dos outros. Além disso, a análise da taxa de falha confirmou a liderança da AFC nesta avaliação, registrando as menores taxas seguidas pela GCF, AWS e AZF forneceram altas taxas de falha, especialmente, em níveis mais altos de concorrência. Por último, a análise da partida a frio mostrou mais uma vez a eficiência das estratégias adotadas pela AFC. Diante desses dados, ficou evidente que o AFC apresentou os melhores resultados em todos os itens analisados, seguida do AWS e GCF, que apresentaram nível de qualidade equivalente. Por fim, os piores resultados foram registrados pela AZF.

Como trabalhos futuros são propostos casos de uso estendidos para outros provedores de FaaS, tais como IBM, Oracle e Huawei, sendo estes submetidos às condições aqui apresentadas, a fim de ampliar o compreensão dos aspectos explorados. Adicionalmente, sugere-se a análise do custo-benefício dos planos pagos, seja no nível básico ou corporativo, uma vez que este trabalho se baseou no nível *free tier* oferecido pelos provedores.

Referências

[1] Nupponen, Jussi e Davide Taibi: *Serverless: What it is, what to do and what not to do.* Em *2020 IEEE ICSA-C*, páginas 49–50, 2020. 1

[2] Schleier-Smith, Johann, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica e David A. Patterson: *What serverless computing is and should become: The next phase of cloud computing.* ACM, 64(5):76–84, abril 2021, ISSN 0001-0782. `https://doi.org/10.1145/3406011`. 1

[3] ZHENG, Xi: *Database as a service - current issues and its future.* CoRR, abs/1804.00465, 2018. `http://arxiv.org/abs/1804.00465`, acesso em 05/07/2020. 1

[4] Carvalho., Leonardo e Aleteia Araujo.: *Orama: A benchmark framework for function-as-a-service.* Em *Proceedings of the 12th CLOSER*, páginas 313–322. INSTICC, SciTePress, 2022, ISBN 978-989-758-570-8. 1, 2