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

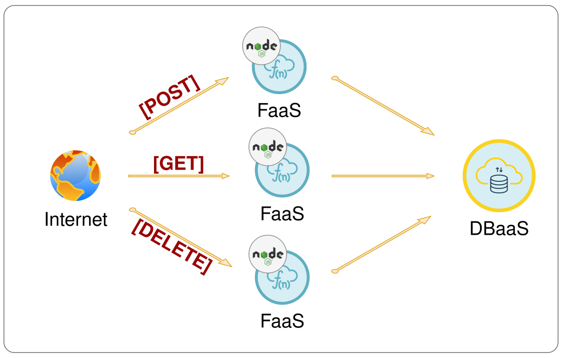

Figure 1. FaaS for DBaaS Orama framework built-in use case.

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.

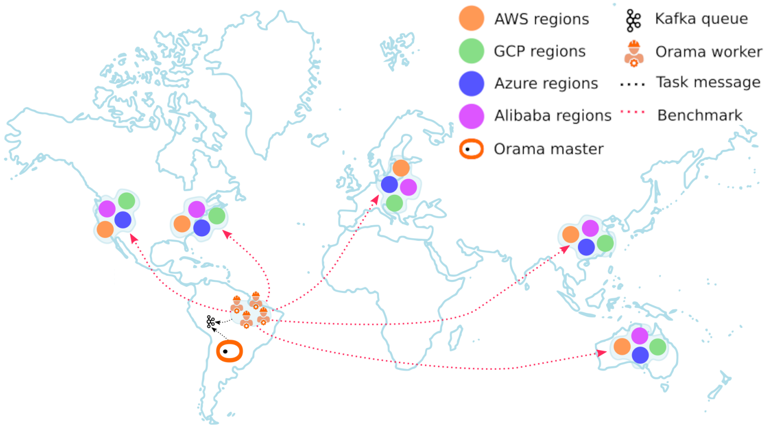

Figure 2. Overall architecture of the experiment.

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

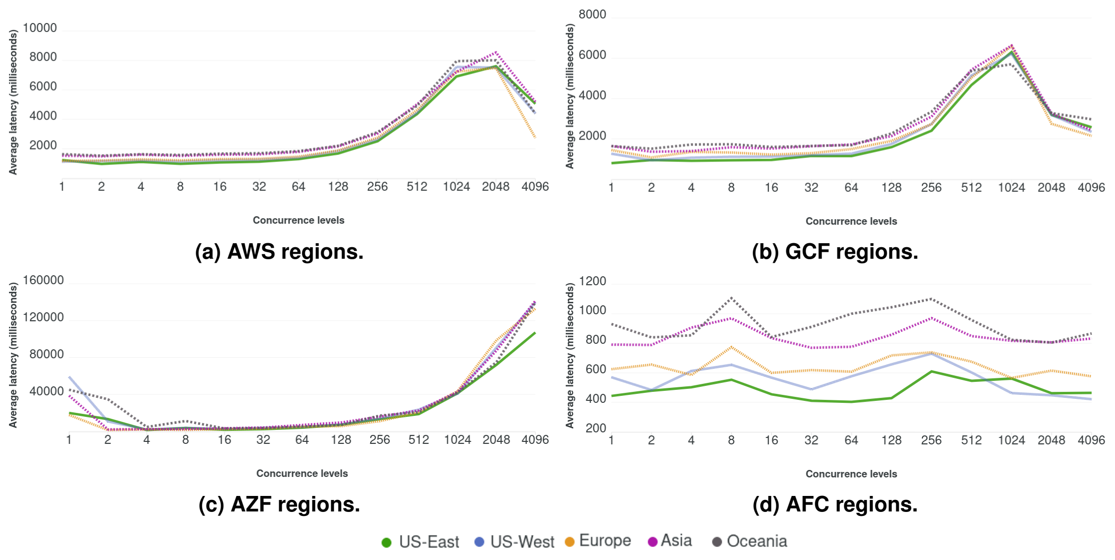

Figure 3. Average execution time by provider.

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.

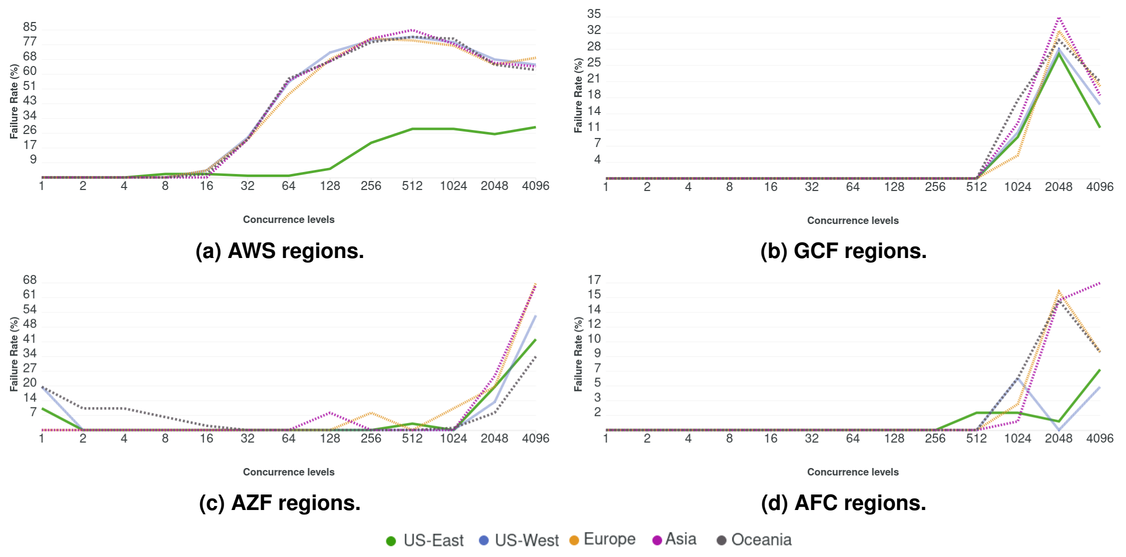

Figure 4. Failure rates (%) by provider.

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.

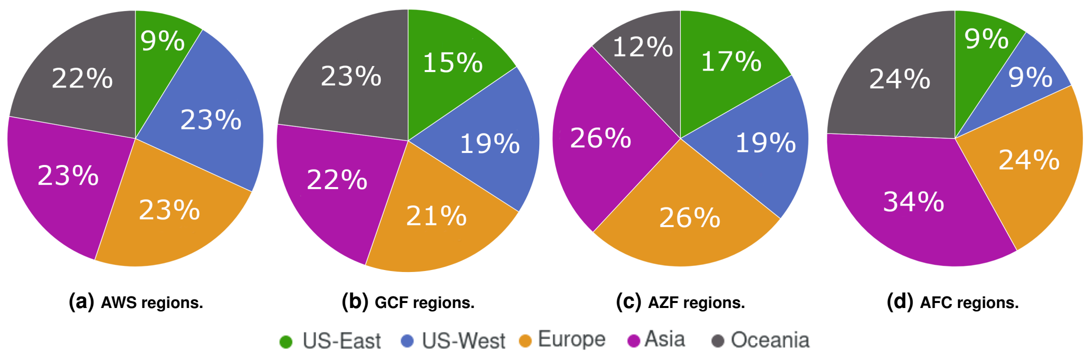

Figure 5. Regional failure distribution.

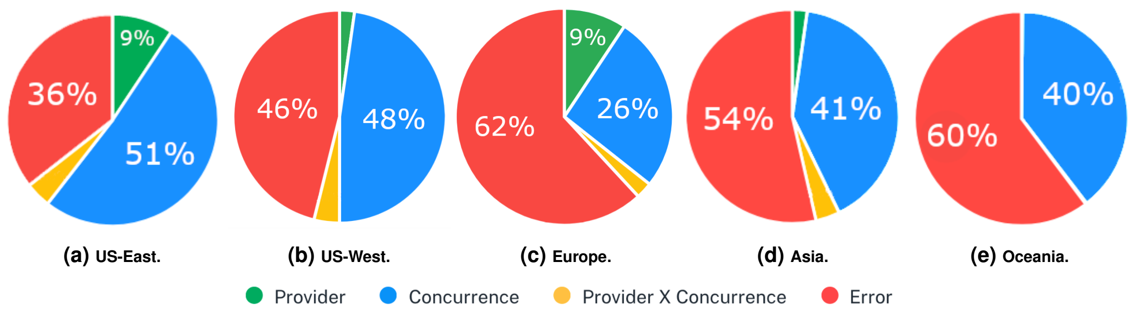

Figure 6. Factorial design results for Lambda vs. GCF.

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.