Function-as-a-Service X Platform-as-a-Service: Towards a Comparative Study on FaaS and PaaS

Lucas F. Albuquerque Jr.[1,2], Felipe Silva Ferraz[3], Rodrigo F. A. P. Oliveira[1], and Sergio M. L. Galdino[1]

[1]Polytechnic School of Pernambuco, University of Pernambuco, Recife, Brazil
Email: {lfaj,rfapo}@ecomp.poli.br, sergio.galdino@ieee.org
[2]IFPE - Federal Institute of Technology, Palmares, Brazil
Email: lucasjr@palmares.ifpe.edu.br
[3]Recife Center for Advanced Studies and Systems (CESAR), Recife, Brazil
Email: fsf@cesar.org.br

***Abstract*—The adoption of cloud computing for service delivery is a market trend and attracts customers seeking elastic, scalable, and cost-effective infrastructures. Instance-based models, such as Platform-as-a-Service (PaaS), are being used to support mobile applications. Despite the management facilities, the PaaS receives criticism for the inefficient use of resources. Studies point to a new model, known as Function-as-a-Service (FaaS), as an alternative that would offer a more efficient use of resources and lower costs. The present work has proposed to perform a comparative evaluation between FaaS and PaaS service delivery models regarding performance, scalability and costs issues in support of mobile applications based on microservices. The conclusions obtained showed that FaaS presented an equivalent performance, a more efficient scalability and the costs influenced by workload type.**

Keywords–Cloud Computing; FaaS; PaaS; Serverless; Mobile; Microservices

I. Introduction

The term virtualization is associated with the abstraction of computational resources for the purpose of optimizing their use, allowing users and applications to transparently share resources [1]. Thus, with virtualization of the infrastructure, servers become a mere abstraction of resources, being more easily managed [2]. Virtualization technologies form the basis of what we now know as cloud computing, which is the provision of information or computing resources as a service accessible through the network [3]. Cloud models, such as Infrastructure-as-a-Service(IaaS) and Platform-as-a-Service (PaaS) use the concept of instance to define the amount of computational resources allocated to carry out their tasks.

In parallel to the advances related to cloud computing, the dissemination of mobile devices led to the emergence of the Mobile Cloud Computing (MCC) [4] concept, which is the use of cloud computing by mobile devices for service delivery anytime, anywhere, managing a large volume of data from a variety of device platforms. To meet performance and scalability requirements in mobile applications, microservice architectures have emerged to enable the development of decoupled applications in separate, scalable and portable modules that communicate through common protocols [5].

Considering mobile application support, PaaS model has been used as the cloud computing alternative for many microservices applications [6]. One of the advantages of PaaS would be its independence from operational issues, allowing the customer to focus on code development. However, the PaaS model is criticized for being a instance-based model, requiring pre-allocating resources, increasing costs for certain types of workloads [7]. In this context, the Function-as-a-Service (FaaS) model, commercially known as Serverless Computing, has been cited as an alternative model for meeting the requirements of mobile applications in microservices [8], offering a scalable, on-demand infrastructure that operates in response to events, adopting a granular demand-based billing model.

FaaS has been cited in several studies as a computational model with potential to meet many of the challenges of mobile computing, as an alternative to the PaaS model. Works such as [9]–[13] point out that due to platform variability, data volume and temporal data characteristics of MCCs, event-based models like FaaS, would be an alternative model in support of mobile devices, Internet of Things (IoT), real-time processing, artificial intelligence, among others. However, as a newly proposed model, many questions remain open about the benefits of using and applying FaaS model.

Considering the several open questions related to the FaaS models, this paper presents a comparative analysis between the PaaS and FaaS models in the mobile application support, as well as a performance and scalability evaluation between these two models. Finally, the paper also proposes to present a cost comparison between the PaaS and FaaS models from a case study based on a geolocation microservices-based application.

This paper is structured as follows. In Section 2, we introduce the basic idea behind FaaS (Function-as-a-Service) and present a comparative analysis to PaaS. In Section 3, we discuss the experiment setup and test plan performed. In Section 4, we discuss the findings, analyzing the performance, scalability results (Section 4.1) and costs (Section 4.2). Section 5 discusses related research, and finally Section 6 concludes the paper with lessons learned and an outlook on future work.

II. Function-as-a-Service

Serverless computing was initially associated with two scenarios:

- Applications that depend on external services for their operation, having their business rules concentrated on the client side. This development model was initially called Backend-as-a-Service (BaaS) and included the

 ISBN: 978-1-61208-590-6

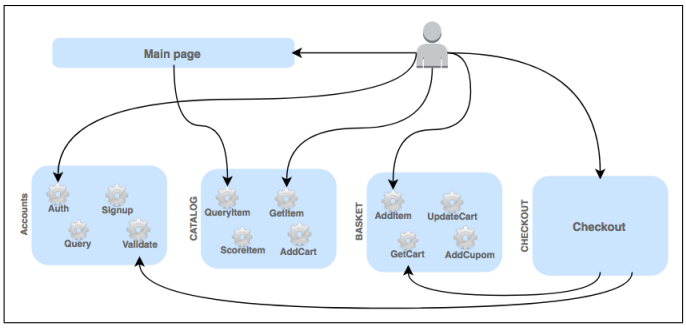

Figure 1. A Faasified application where each service is decomposed into functions that can be performed or escalated independently.

use of external services as databases, authentication services, messaging services, among others [14].

- Applications whose business rules are located in the cloud, running on demand only, in response to events and in an ephemeral way (no relation between events). This approach is more recent and usually referred as FaaS (Function-as-a-Service) or Event-Driven Computing [**?**].

The BaaS model was an important driver for both cloud computing and popularization of mobile devices, but it did carry with it some complications. Business rules on the client side were making it difficult to update and deploy new features as well as reverse engineering risks. In FaaS, business rules can be server-centric or divided between the server and the client, and the application is decomposed into small, specific, well-defined tasks, called functions. Each executed function is treated as an ephemeral event (independent and stateless) and its lifetime is the same as the task being executed. The client has an environment that responds to events, rather than dedicated full-time infrastructure [8].

FaaS enables the decomposition of service in micro-functions, which can be performed and scaled independently, introducing the concept of nanoservices [15]. Another concept introduced by FaaS is related to the transformation process of monolithic or microservice applications to functions (Figure 1), process known as FaaSification [16] [17].

Studies that have already been carried out, place FaaS as a variant of the PaaS model, or a type of specialized PaaS [6], but do not present in a consolidated form the characteristics of each of the models. From Table I it is possible to observe the main differences that can be pointed out in relation to the PaaS and FaaS models considering several aspects.

III. Experimentation

The purpose of this section is to present the environment used to perform the experiments (Subsection III-A) and the results obtained (Subsection III-B). At the end of the Section, performance, scalability and cost analyzes will also be presented for the scenarios evaluated (Subsections III-C and III-D, respectively).

A. Experiment Setup

For the experiments, we developed an application using the architecture based on microservices (Figure 2) composed

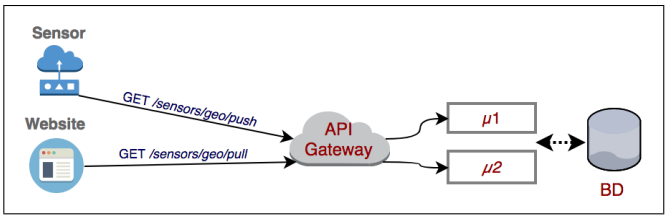

Figure 2. Diagram of the microservice application for storage and availability of Geolocation data

of two services (µ1 e µ2), responsible for receiving and making available geolocation information collected from mobile devices. The µ1 was responsible for receiving and storing data received through HTTP (Hypertext Transfer Protocol) REST (Representational State Transfer) requests, while µ2 receives requests and returns the results in JSON ((JavaScript Object Notation) format (Figure 2).

To perform the experiments, we use Amazon Web Services (AWS) Elastic Beanstalk as the PaaS environment and AWS Lambda as FaaS solution for running an application developed in Node.JS with MongoDB database as persistence layer. For the tests, the following operations were performed, in three rounds, with an interval of one hour between them:

- **Write** - Write operations targeting µ1 for both environments.
- **Read** - Read operations targeting µ2 for both environments.
- **Write/Read** - Write and read operations for µ1 and µ1 simultaneously, for both environments.

Beyond the one hour interval, upon completion of each round, both environments were destroyed and re-implemented, to ensure that results from previous rounds did not interfere in the results of subsequent rounds.

In order to carry out the performance tests, we use JMeter 3.1 [18] configured in an EC2 *c4.large* instance connected to the same Virtual Private Cloud (VPC) as the environments to be evaluated. The performance tests were executed with the objective of identifying the maximum number of requests supported by each scenario and the scalability efficiency. The tests simulated concurrent requests (threads) that were gradually increased to the limit of 100 users. Tests started with 10 threads and started another 10 every 10 seconds (with ramp-up of 2 Seconds). After reaching 100 requests, the test remained active for 120s, finally being finalized in a controlled way, at a rate of 5 req/sec (Figure 3).

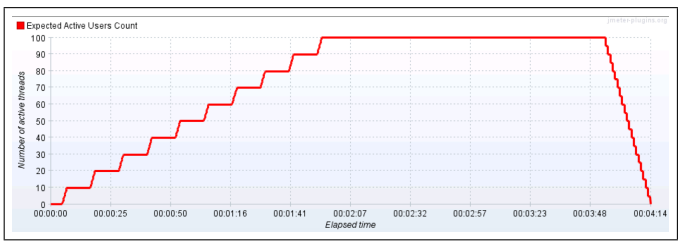

Figure 3. JMeter test plan used during experiments

In test plan, it was defined that the tests would be interrupted if any of the conditions listed below were met:

TABLE I. COMPARATIVE ANALYSIS BETWEEN PAAS AND FAAS MODELS CONSIDERING DIFFERENT ASPECTS.

Aspects	PaaS	FaaS
Coding and Delivery	Focused on services, with development teams being responsible for one or more parts of the application. Service-based delivery.	Focused on tasks (functions) with development teams being responsible for a set of functions. Function-based delivery.
Sizing	Based on the number of instances and resources required by the services. Risks of overestimating or underestimating workloads.	Based on the amount of resources for the execution of each event.
Environment	Fully operated by the provider with possibility of customization by the client.	Fully operated by the provider with no possibility of customization by the client.
Application Maintenance	Interventions in the code need to take into account the entire service.	Interventions in the code need to take into account a specific function. Less code.
Resources Allocation	Pre-allocation of resources with the possibility of allocating additional instances on demand.	No pre-allocated resources. Transparent and on-demand allocation.
Execution	Permanent waiting state, no restrictions on the duration of events.	No waiting state. The function is executed when required. Restriction of maximum duration per event.
Billing	By instantiated resources, whether used or not.	Per event executed. Without commitment.

1) If the latency for any of the requests destined for μ1 and μ2 reaches the Maximum Latency Accepted (MLA) for the test scenario being performed;
2) If any of the tested requests return errored responses to the requests made;
3) If the time planned for the tests is completed, considering the test plan defined on JMeter for μ1 and μ2;

B. Results

The results obtained during the tests will be presented in tables throughout the section. In each table, the rounds in bold with $^{(1)}$ indicate that the test round did not return positive results, representing 100% errors in the samples. The rounds in bold with $^{(2)}$, however, indicate successful samples, but errors were observed during execution that forced premature interruption of threads, as indicated in Section III-A. And, finally, rounds without indication mean that the round was completed successfully, with no observed errors.

For Lambda(A) scenario (Table II), the application executed in the FaaS environment presented errors in 100% of the samples tested, the round being closed due to the requests having exceeded the MLA limit. The Cloudwatch logs showed that the initial requests made to the Application Programming Interface (API) Gateway reached a latency of 14.50 seconds, which extrapolated the MLA for the scenario (5000 ms). This latency observed in the execution of the FaaS functions was an issue already cited in other works, being known as *Cold Start* [19] [20], this behavior is observed in FaaS implementations, and affects functions of eventual use or that present long temporary lapses between the requisitions, causing the deallocation of resources. In order to try to overcome *cold start*, for the Lambda(B) scenario the MLA was increased to 10000ms, but some rounds still showed errors, returning 100% of failures.

For the Lambda(C) scenario (Table III), the MLA was increased to 15000ms, which allowed the *cold start* to be overcome and the samples returned successfully. But despite overcoming *cold start*, the samples presented errors during execution. When analyzing the Lambda logs, it was observed that the errors occurred because the Lambda was taking a lot of time to process the requests, exceeding the default maximum duration for Lambda environment (3 secs), causing timeout errors. In the Lambda (D) scenario, with the memory increased to 256MB, the FaaS environment started to complete the rounds successfully. The results confirmed the existence of a direct proportionality between the allocated memory and CPU resources.

TABLE II. RESULTS FOR THE SCENARIOS (A) AND (B) FOR LAMBDA ENVIRONMENT, SETTING WITH MLA of 5000MS AND 10000MS, RESPECTIVELY.

Operation		Lambda (A) 128MB			Average Latency (ms)	Lambda (B) 128MB			Average Latency (ms)
Write	Round	#1(1)	#2(1)	#3(1)	10854	#1(1)	#2(2)	#3(1)	9821
	Suceeded Requests	0	0	0		0(1)	202	0(1)	
	Latency (ms)	11210	10020	11332		11425	4057	13981	
Read	Round	#1(1)	#2(1)	#3(1)	11915	#1(2)	#2(1)	#3(1)	9501
	Suceeded Requests	0	0	0		162	0	0	
	Latency (ms)	13642	12901	9201		4322	10939	13241	
		Timeout: 5000ms				Timeout: 10000ms			

TABLE III. RESULTS FOR THE SCENARIOS (C) AND (D) FOR LAMBDA ENVIRONMENT, SETTING MEMORY TO 128MB AND 256MB RESPECTIVELY, AND 15000MS OF MLA

Operation		Lambda (C) 128MB			Average Latency (ms)	Lambda (D) 256MB			Average Latency (ms)
Write	Round	#1(2)	#2(2)	#3(2)	4134	#1	#2	#3	129
	Suceeded Requests	152	189	177		>10000	>10000	>10000	
	Latency (ms)	4232	4057	4112		139	127	121	
Read	Round	#1(2)	#2(2)	#3(2)	4338	#1	#2	#3	585
	Suceeded Requests	150	150	141		>10000	>10000	>10000	
	Latency (ms)	4322	4292	4401		571	604	580	
		Timeout: 15000ms				Timeout: 15000ms			

In the case of the Beanstalk(A) scenario (Table IV), the same issues observed in Lambda(C) were also observed for this scenario. During the execution of the tests, the PaaS environment started to return errors, caused by the resource saturation of the instance used (*t1.micro*). For the Beanstalk (B) scenario, the multi-instance feature was enabled, allocating more instances on demand, allowing the successful completion of the test rounds.

Considering that the Lambda(D) and Beanstalk(B) scenarios completed the tests successfully, other scenarios were analyzed, adding extra features to the tested environments in order to evaluate the existence of a direct relation between resources and performance considering the proposed application. For the FaaS environment, the scenarios included the increase of memory per function performed and the use of SSD disks. For

 ISBN: 978-1-61208-590-6

TABLE IV. RESULTS FOR THE SCENARIOS (A) AND (B) FOR BEANSTALK ENVIRONMENT, USING SINGLE-INSTANCE AND MULTI-INSTANCE, AND 15000MS OF MLA

Operation		Beanstalk (A)				Beanstalk (B)			
		Single Instance (t1.micro)			Average Latency (ms)	Multi Instance (t1.micro)			Average Latency (ms)
Write	Round	#1 [2]	#2 [2]	#3 [2]	114	#1	#2	#3	162
	Suceeded Requests	>10000	>10000	>10000		>10000	>10000	>10000	
	Latency (ms)	111	96	136		158	183	144	
Read	Round	#1 [2]	#2 [2]	#3 [2]	179	#1	#2	#3	523
	Suceeded Requests	>10000	>10000	>10000		>10000	>10000	>10000	
	Latency (ms)	203	157	177		458	579	531	
		Timeout: 5000ms				Timeout: 15000ms			

the PaaS environment, tests were performed using instances with more resources (2x) and Solid State Disks (SSD) also. The results (Table V) show that, for the tested scenarios, based on the Lambda(D) and Beanstalk(B), latency reduction did not show a proportionality between the addition of resources and performance, depending also on the type of operation (writing or reading).

TABLE V. COMPARISON BETWEEN SEVERAL LAMBDA AND BEANSTALK SCENARIOS SHOWING THE REDUCTION IN LATENCY OBTAINED IN DIFFERENT SCENARIOS

L - 256 - HDD	WRITE	L - 512 - HDD %	L - 256 - SSD %	L - 512 - SSD %
		-69,51	-57,62	-78,30
	READ	L - 512 - HDD %	L - 256 - SSD %	L - 512 - SSD %
		2,74	-48,83	-54,76

L-256-HDD = Lambda - 256MB - HDD
L-512-HDD = Lambda - 512MB - HDD
L-256-SSD = Lambda - 256MB - SSD
L-512-SSD = Lambda - 256MB - SSD

B - T1 - HDD	WRITE	B - T2 - HDD %	B-T1-SSD %	B-T2-SSD %
		-43,51	-25,98	-45,98
	READ	B - T2 - HDD %	B-T1-SSD %	B-T2-SSD %
		64,73	-50,57	-50,89

B-T1 Micro-HDD = Beanstalk - 256MB - HDD
B-T2 Small-HDD = Beanstalk - 512MB - HDD
B-T1 Micro-SSD = Beanstalk - 256MB - SSD
B-T2 Small-SSD = Beanstalk - 256MB - SSD

C. Performance and Scalability Analysis

Considering the results for the scenarios tested, some considerations:

- The results showed that the *cold start* observed in FaaS environments affected the performance of applications executed in FaaS environments. This behavior is observed in applications of occasional use or that present long temporary lapses between the requisitions, causing the deallocation of resources and is common to all serverless implementations evaluated.
- The results showed that the allocation of more resources to the tested environments had a positive impact on overall performance. For FaaS environments, there was a direct relationship between the amount of memory and the processing resources allocated by function. However, at the application level, the allocation of more resources was not proportional to the performance gains;
- The results showed that the scalability mechanisms adopted by the PaaS and FaaS environments were efficient in all scenarios evaluated. The scalability of the PaaS environment was based on instance and occured in resource jumps, being less granular. In FaaS environments scalability was linear, occurring based on volume of events.

D. Cost Analysis

Some types of applications tend not to benefit from the characteristics of PaaS, especially those that present variations in workload. For these types of applications, which can range from zero requests to thousands in a few seconds, the permanent instantiation of resources is not advantageous, considering that the instances are kept alive and are charged regardless of usage. For applications that present variations in workloads, the FaaS model presents itself as more adequate, considering the dynamic allocation of resources. The economic benefits of serverless computing heavily depend on the execution behavior and volumes of the application workloads.

Considering the scenarios tested, it was possible to extract a cost basis in order to compare infrastructure costs. For comparison purposes, the monthly quantity of requests were used as a metric, since a direct cost comparison was not possible. In order to obtain the number of monthly requests supported by a Beanstalk instance, the number of 10,000 requests per minute was used as a reference, based on the maximum number of requests supported per minute by one instance during the tests performed. In order to obtain the maximum number of requisitions per month, the maximum number of requisitions supported (10,000) was multiplied by 43,200, which is the number of minutes per month, totaling 432,000,000 monthly requisitions.

For the costing of the FaaS environment, the durations of 100ms and 300ms were considered for writing and reading operations, respectively, so that cost simulations were based on 50/50 (50% write/50% read), 70/30 (70% write/30% read) and 90/10 (90% write/10% read). The proposed durations were based on the mean reading and writing values obtained from the median latencies during the tests performed. Table VI presents the results of the comparison for the three proposed scenarios, as well as the monthly cost of an instance *t1.micro*. As can be seen in the results, for the 50/50 scenario the monthly cost was US$44.65, while for the 70/30 scenario the monthly cost was US$ 37.45, both scenarios presented a higher monthly cost than the AWS Beanstalk, which was US$ 33.86. However, for the third scenario (90/10), the monthly cost was US$ 30.25, falling below the monthly value for Beanstalk.

TABLE VI. COMPARISON OF COSTS BETWEEN AWS LAMBDA AND AWS BEANSTALK CONSIDERING A TOTAL OF 43,200,000 MONTH REQUESTS IN THREE DIFFERENT SCENARIOS

	50/50		70/30		90/10	
	21.600.000	21.600.000	30.240.00	12.960.000	38.880.000	4.320.000
	100ms	300ms	100ms	300ms	100ms	300ms
Operation	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**
AWS Lambda 256MB	US$ 13.32	US$ 31.33	US$ 18.65	US$ 18.80	US$ 23.98	US$ 6.27
Montlly Cost	**US$ 44.65**		**US$ 37.45**		**US$ 30.25**	

AWS Beanstalk (*t1.micro*)	744 hours/month	**US$ 33.86**

From the costs shown in Table VI, the following conclusions were obtained regarding the costs related to the case study in question:

- FaaS and PaaS environments presented cost variations considering the different scenarios presented (50/50, 70/30 and 90/10), depending on the type of predominant operation. The cost of the FaaS environments was

higher in the first two scenarios due to the duration observed during the writing operations;

- For FaaS, the longer the duration of the functions, the higher the cost. Investing in the use of more performative database instances and caching features could reduce the duration of read operations, enabling a reduction in cost per read event;
- Different FaaS providers may offer lower costs per event (Table VII). Providers seeking to build a portfolio of clients or seek to consolidate their product on the market can offer attractive prices;

TABLE VII. COMPARISON OF COSTS BETWEEN FAAS PROVIDERS CONSIDERING A TOTAL OF 43,200,000 MONTH REQUESTS FOR THE 90/10 SCENARIO

	90/10		
	38.880.000	4.320.000	
	100ms	300ms	
Provider	**Read**	**Write**	**Montly Cost**
AWS Lambda	US$ 23.98	US$ 6.27	**US$ 30.25**
Azure Functions	US$ 23.33	US$ 6.05	**US$ 29.38**
Google Functions	US$ 33.53	US$ 7.72	**US$ 41.25**
IBM OpenWhisk	US$ 16.52	US$ 5.51	**US$ 22.03**

- The commercial policies adopted by the PaaS and FaaS providers follow a similar model to their respective scalability characteristics. For the PaaS environment, the costs increase based on the allocated instances (in jumps) (Figure 4). In the case of FaaS environments, the cost increases due to the number of requests received and executed.

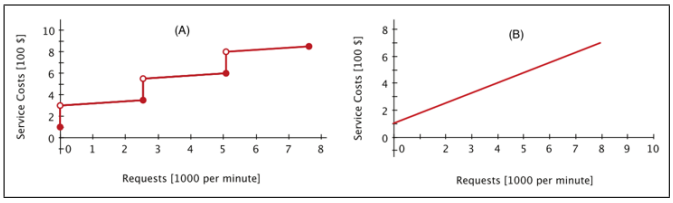

Figure 4. Graphs showing the evolution of costs in the Paas (A) and FaaS (B) environment in relation to the number of requisitions [21]

This section was dedicated to present and discuss the results obtained during the performed experiments regarding to performance and costs for the proposed scenarios. In the next section will be presented the works that served as reference or are related with the present work.

IV. Related Work

As a newly proposed model, scientific work related to FaaS is still scarce, and most use the term Serverless to refer to the model. Works such as [11] and [19] propose Serverless implementations as proof of concept and experimental, to be used for research purposes and for evaluation of applications in event-based environments, addressing conceptual issues regarding scalability and performance. Some papers are dedicated to proposing Serverless implementations and present results of experiments performed, such as [22], which addresses the inefficiency of instance-based models, and [23] that analyzes scalability issues, *cold start* and execution in FaaS environments.

Some papers focus on cost issues in Serverless environments, such as [24] that addresses performance, scalability, and cost issues in FaaS environments. In [25] and [26], the authors present results comparing a single application implemented as monolithic, an instance-based microservices and an event-based microservices (Lambda), and presenting significant cost reductions with the adoption of the FaaS model. As in [21], the authors present CostHat, a graphical model for evaluating costs for instance and event-based microservices, which simulates the impact of changes in applications.

In [16], the authors addressed FaaSification, which is a process of migrating applications to nanoservices, or event-based architectures that, although functional, still requires more in-depth research. Works related to the use of Serverless computing in several applications, such as for rejuvenation of environments [27], support for weareables [28], cognitive services [20], among others.

Considering previous works, the present paper has the purpose to contribute with a comparison between the FaaS and PaaS model with a focus on performance, scalability and cost on support of microservices applications. The work seeks to cover the characteristics of both the models, intrinsical aspects of FaaS and its pitfalls, such as cold start and execution limitations in different workload scenarios, points not covered by other published papers.

V. Conclusion

The growth of cloud computing has been boosting and enabling the emergence of parallel research areas, which use computational clouds to support various applications, such as mobile devices support. Variations in the volume of requests, the use of heterogeneous platforms and the availability requirements are peculiar characteristics of mobile computing environments that, in association with the use of microservice architectures, allow scale gains, independence and the availability required to support modern applications.

Although the PaaS is a consolidated model and recognized as efficient in supporting applications in microservices, the FaaS model has been identified as an alternative model for meeting mobile computing scenarios, providing more efficient use of resources and lower costs. This paper proposed to perform a comparative evaluation between FaaS and PaaS cloud service delivery models regarding performance issues, scalability and costs in the use of mobile applications based on microservices.

Based on the experiments carried out, the comparative analyzes and the case study, it was possible to reach the following conclusions:

- The FaaS application performance during the experiments was shown to be equivalent to the PaaS for most of the scenarios tested. And the addition of resources did not represent proportional gains in performance.
- *Cold Start* issues observed in FaaS environments must be taken into account prior to the adoption of the model and can significantly impact the performance of the application, despite the existence of techniques to reduce these latencies;
- In terms of scalability, FaaS has proven to be most interesting for services whose workloads are variable

 ISBN: 978-1-61208-590-6

or unpredictable, while PaaS best applies to constant or predictable workloads;

- In terms of costs, FaaS presented a better cost benefit in the treatment of requests that require short and predictable execution times, while PaaS was better suited for requests that require longer execution times or variable duration;
- In order to keep costs low using FaaS, in addition to the concern with the execution time of the requests, the results showed that the dependence on the use of external services, such as database, authentication services, among others, can interfere considerably with the expenses . In these cases, it is worth investing in better-performing external services, to reduce the time to perform functions while reducing costs.

In order to reduce the cold start impacts, some preliminar techniques can be applied. (1) in applications that have a higher latency tolerance, adjust the response threshold times, as performed during the performed experiments; (2) maintain an external routine (heartbeat) to keep resources permanently active through requisitions at regular times; (3) associate microservices of frequent and occasional use under the same API endpoint, so that access to the most used microservices guarantees the instantiation of resources to those of less access.

The results showed that the use of FaaS can help reduce costs depending on the workload. As FaaS market implementations take a charge per event taking into account the execution time of the function, the longer the time required to process a request, the higher the cost of the operation. In addition, applications whose execution times are short and predictable tend to benefit from the use of FaaS, which, together with the infinite scalability of the model, can aid in the support of seasonal workloads. The results obtained can help solution architects in the decision making regarding the use of FaaS to support their applications.

A. Future Research Directions

The presented work requires future work in three directions:

- To deepen the cost studies in FaaS environments from a service provider's point of view, in order to evaluate if, in comparison with other models, FaaS enables a more efficient use of resources, thus reducing costs reduction with infrastructure.
- Evaluation of techniques to reduce cold start in FaaS environments in order to reduce latency in occasional applications;
- Evaluate FaaSification techniques for the portability of microservice applications for FaaS environments.

Acknowledgment

This research has been supported by an AWS in Education Research Grant which helped us to run our experiments on AWS Lambda as representative public commercial FaaS.

References

[1] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in 2010 Second International Conference on Computer and Network Technology, April 2010, pp. 222–226.

[2] M. Portnoy, Virtualization Essentials. Sybex, 2012.

[3] Y. Tsuruoka, "Cloud computing - current status and future directions," vol. 24, no. 2. Information Processing Society of Japan, 2016, pp. 183–194.

[4] Y. Wang, I.-R. Chen, and D.-C. Wang, "A survey of mobile cloud computing applications: Perspectives and challenges," Wireless Personal Communications, vol. 80, no. 4, Feb 2015, pp. 1607–1623.

[5] M. Fazio et al., "Open Issues in Scheduling Microservices in the Cloud," in IEEE Cloud Computing, vol. 3, no. 5, sep 2016, pp. 81–88.

[6] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1, INSTICC. ScitePress, 2016, pp. 137–146.

[7] T. Reeder, "What is serverless computing and why is it important — iron.io," https://goo.gl/0oDIcT, Jul 2016, (Accessed on 06/16/2017).

[8] M. Roberts, "Serverless architectures," http://martinfowler.com/articles/serverless.html, 2016, (Accessed on 06/16/2017).

[9] F. Renna, J. Doyle, V. Giotsas, and Y. Andreopoulos, "Query Processing For The Internet-of-Things: Coupling Of Device Energy Consumption And Cloud Infrastructure Billing," ArXiv e-prints, Feb. 2016.

[10] M. Diaz, C. Martin, and B. Rubio, "State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing," in Academic Press Ltd., vol. 67, no. C. Academic Press Ltd., May 2016, pp. 99–117.

[11] I. Nakagawa, M. Hiji, and H. Esaki, "Dripcast - Server-less java programming framework for billions of IoT devices," Proceedings - IEEE 38th Annual International Computers, Software and Applications Conference Workshops, COMPSACW 2014, vol. 23, no. 4, 2014.

[12] S. Nastic, S. Sehic, D.-H. Le, H.-L. Truong, and S. Dustdar, "Provisioning software-defined iot cloud systems," in Proceedings of the 2014 International Conference on Future Internet of Things and Cloud, ser. FICLOUD '14. IEEE Computer Society, 2014, pp. 288–295.

[13] Y. Jararweh et al., "Sdiot: a software defined based internet of things framework," Journal of Ambient Intelligence and Humanized Computing, vol. 6, no. 4, 2015.

[14] K. Lane, Overview of the backend as a service (BaaS) space. API Evangelist, 2015.

[15] E. Wolff, Microservices: Flexible Software Architecture. Pearson Education, 2016.

[16] J. Spillner and S. Dorodko, "Java Code Analysis and Transformation into AWS Lambda Functions," ArXiv e-prints, Feb. 2017.

[17] J. Spillner, "Transformation of Python Applications into Function-as-a-Service Deployments," ArXiv e-prints, May 2017.

[18] D. Rahmel, "Advanced joomla!" in Apress. Apress, 2013, ch. 8, pp. 211–247.

[19] S. Hendrickson et al., "Serverless computation with openlambda," in Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing, ser. HotCloud'16. USENIX Association, 2016, pp. 33–39.

[20] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in Proceedings of the 1st International Workshop on Mashups of Things and APIs, ser. MOTA '16. New York, NY, USA: ACM, 2016, pp. 5:1–5:4.

[21] P. Leitner, J. Cito, and E. Stckli, "Modelling and managing deployment costs of microservice-based cloud applications," in 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), Dec 2016, pp. 165–174.

[22] J. Spillner, "Snafu: Function-as-a-Service (FaaS) Runtime Design and Implementation," ArXiv e-prints, Mar. 2017.

[23] E. Jonas, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the Cloud: Distributed Computing for the 99%," ArXiv e-prints, Feb. 2017.

[24] T. Hoff, "The serverless start-up - down withservers!" http://highscalability.com/blog/2015/12/7/the-serverless-start-up-down-with-servers.html, Dec 2015, (Accessed on 06/16/2017).

[25] M. Villamizar et al., "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 2016, pp. 179–182.

 ISBN: 978-1-61208-590-6

[26] M. Villamizar, O. Garcs, H. Castro, and M. Verano, “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in 2015 10th Computing Colombian Conference (10CCC), Sept 2015, pp. 583–590.

[27] B. Wagner and A. Sood, “Economics of resilient cloud services,” in 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Aug 2016, pp. 368–374.

[28] I. Baldini et al., “Cloud-native , event-based programming for mobile applications,” 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems, 2016, pp. 287–288.

 ISBN: 978-1-61208-590-6