Serverless Platforms Performance Evaluation at the Network Edge

Vojdan Kjorveziroski[1*][0000-0003-0419-4300], Sonja Filiposka[1][0000-0003-0034-2855], and Vladimir Trajkovik[1][0000-0001-8103-8059]

[1] Faculty of Computer Science and Engineering,
Ss. Cyril and Methodius University, Rugjer Boshkovikj 16, 1000 Skopje, North Macedonia
{vojdan.kjorveziroski,sonja.filiposka,trvlado}@finki.ukim.mk

Abstract. Emerging computer paradigms aim to fulfill the ever-present ideal of running as many applications on existing infrastructure, as efficiently as possible. One such novel concept is serverless computing which abstracts away infrastructure management, scaling and deployment from developers, allowing them to host function instances with granular responsibilities. However, faced with the meteoric growth in the number of IoT devices, the cloud is no longer suitable to meet this demand and a shift to edge infrastructures is needed, providing reduced latencies. While there are existing serverless platforms, both commercial and open-source that can be deployed at the edge, a comprehensive performance analysis is needed to determine their advantages and drawbacks, define open-issues, and identify areas for improvement. This paper analyses three different serverless edge platforms with the help of an existing serverless test suite, outlining their architecture, as well as execution performance in both sequential and parallel invocation scenarios. Special focus is paid to solutions that can be deployed in a standalone fashion, without complex clustering requirements. Results show that while the serial execution performance is comparable among the analyzed platforms, there are noticeable differences in cases of concurrent executions.

Keywords: Serverless computing, Function-as-a-service, Edge computing, Performance comparison.

1 Introduction

Recent breakthroughs in both computer hardware and networking have led to a dramatic increase in the number of new devices [1], allowing novel use-cases, not possible before. Many of these devices interact with the nearby environment and other equipment within it, either because of an operator's command or upon an occurrence of a given event. To ensure a good user experience, such event-driven communication needs low latency [2], a requirement which is challenging to fulfill in the traditional cloud-based architectures. As a result, recently, there has been a gradual shift to the edge of the network [3], closer to the end devices and their users, thus ensuring seamless real-time communication. However, in such scenarios, the question of infrastructure management, execution performance, and application deployment arises, as a result of the more decentralized architecture.

A possible solution that would ease the use of such edge-based infrastructure is serverless computing [4], with its function-as-a-service semantics [5], allowing developers to write granular functional elements, which are easier to create, maintain, and scale. Cloud-based serverless solutions have existed for a number of years [6] and have proven very popular among businesses and developers alike, reducing the time to market, and providing a more cost-effective alternative to the more traditional Platform-as-a-Service (PaaS) or even Infrastructure-as-a-Service (IaaS) options of application deployment.

Taking into account that serverless at the edge is still a novel topic, there are already some open-source and commercial solutions, with a common aim of simplifying the infrastructure management and function deployment processes. However, their implementation, developer interfaces, and supported technologies vary significantly, and this lack of interoperability and standardized access methods is one of the main issues of serverless computing today. One consequence of this diversity is the difference in performance offered by the various serverless platform implementations, an aspect of paramount importance in environments requiring low response times, such as at the edge of the network.

The aim of this paper is to determine the performance characteristics of popular serverless platforms at the edge, both commercial and open source, by utilizing an existing set of benchmarks dedicated to serverless computing and adapting them to the platforms at hand.

The rest of the paper is structured as follows: in section 2 we outline related work to this topic and recent notable efforts of characterizing serverless execution performance in different environments, among different platforms. In section 3, we describe the employed methodology, the platform selection process, as well as the set of benchmarks being used. We then proceed to present the acquired results in section 4, analyzing the performance differences between the platforms. We conclude the paper with section 5, outlining plans for future work.

2 Related Work

With the rise in popularity of serverless computing, there has been an increasing interest from both the academic community, as well as the industry for the development of new solutions. These novel platforms tackle different aspects of the associated open issues with serverless computing [7], ranging from execution efficiency, to ease of use. Many academic implementations also show benchmark results which quantitatively compare their improved performance to other popular serverless platforms, either at the edge [8], the cloud [9] or in a hybrid hierarchical model [10], depending on the targeted execution location of the platform itself. However, the lack of standardized tests which would encompass all different aspects of the implementation at hand, and would facilitate easier results comparison, has led to the development of dedicated benchmark suites.

The authors of [11] present FunctionBench, a set of benchmark tools for cloud-based serverless platforms whose aim is to characterize network, disk, and compute performance of the targeted platforms. Similar to this, Das et al. [12] focus on the edge counterparts of these public cloud platforms, which can be deployed on private infrastructure, thus remotely orchestrating the deployment of new functions. Their results show that AWS Greengrass is more efficient in such constrained environments compared to Azure IoT hub, another commercial product for serverless computing at the edge.

Gorlatova et al. in [13] aim to bridge this gap, comparing both edge-based and cloud-based solutions, confirming that the cold start problem [14], experienced during the initial startup of a function after it has been scaled down to zero [15] plays a major role in functions' execution time. Finally, the authors of [16], unlike previous efforts, focus solely on open-source solutions, and their performance characteristics.

While these works offer an important insight into the different available serverless implementations, they focus solely on either commercial products or open-source ones, without thorough comparison between them. To the best of our knowledge no performance analysis focusing on both spectrums is currently available.

3 Methodology

Measuring the performance of different execution environments requires standardized tests based on common utilities that can be executed across different platforms. For this reason, we have decided to reuse the test suite presented in [11], whose code has been open-sourced and made publicly available [17]. Nevertheless, the lack of consistent programming interfaces between the different platforms required manual changes to the benchmarks, which have been initially developed for cloud-based serverless platforms.

In the subsections that follow we explain in detail the platform selection criteria, describe the different tests that have been utilized, and present the infrastructure where they have been executed.

3.1 Platform Selection

The primary criteria for choosing which platforms to include was the requirement for standalone deployment, without the need for multiple worker machines, or clustering setups. This led to the omission of many popular open-source solutions, such as Openwhisk[1] and OpenFaaS[2] since they rely on complex container orchestration platforms such as Kubernetes or Docker Swarm. Nonetheless, in our opinion, keeping the infrastructure as simple as possible, and deployable in severely constrained environments is an important aspect of edge architecture design, justifying the strict selection criteria.

[1] https://openwhisk.apache.org/ [Online] (Accessed: 31.05.2021)

[2] https://www.openfaas.com/ [Online] (Accessed: 31.05.2021)

We have included FaasD [18], [19], which is a lightweight version of OpenFaaS, whose primary purpose is deployment on edge devices with limited resources, thus completely fulfilling the above criteria. As a relatively new open-source solution, it is being actively developed, and has not been included in any of the previously mentioned platform evaluations, a fact that has further contributed to its selection.

FaasD is not the only scaled-down version of a popular serverless platform. There have been efforts to simplify the requirements of OpenWhisk in the past as well, resulting in OpenWhisk-Light [20], but unfortunately it is no longer actively maintained, making it an unsuitable choice for inclusion.

As representatives of the commercial serverless edge solutions, we have selected both AWS Greengrass and Azure IoT Hub, based on their popularity, and the fact that they have been included in other performance studies as well, allowing us to compare the obtained results more easily. Furthermore, the selected benchmark suite has been purposefully developed for the cloud counterparts of these products, providing an opportunity to reflect on the needed changes to adapt the test functions to the edge.

3.2 Test Types

Table 1. Test parameters and description

Test Category	Test Name	Parameters	Purpose
CPU & Memory	Float Operation	Random number	Common operations
	Matmul	Random number (matrix size)	Matrix multiplication
	Chameleon	500x500 (number of columns and rows)	HTML table rendering
	Image Processing	Object storage location	Image transformations
	Linpack	Random number (matrix size)	Linear equation solver
	PyAES	1000, 100 (length & iterations)	AES operations
	Model Training	Object storage location	ML model training
	Video Processing	Object storage location	Video encoding
Network	iperf3	IP address & duration	Network throughput
	JSONDumpsLoads	URL of JSON file[3]	JSON manipulations
	Object Download	Object storage location	Object storage performance
Disk	Dd	100M, 1 (block size & count)	Dd disk speed
	Random I/O	100, 1024 (file size & byte size)	Python random I/O
	Sequential I/O	100, 1024 (file size & byte size)	Python sequential I/O
	Gzip	50MB (compression size)	Gzip compression

[3] https://data.parliament.scot/api/departments [Online] (Accessed 31.05.2021)

The testing suite that has been adapted to run on the previously selected serverless edge platforms consists of 15 different functions, divided into three distinct categories: CPU, network, and disk benchmarks. Each of these tests requires different input parameters which represent the input data that is being worked upon. Considering the fact that these benchmarks have been created to test cloud-based platforms, we have replaced all cloud services such as object storage with either native features of the platform itself, or with other locally hosted alternatives. This allows more relevant latency information, where cloud communication can be completely avoided.

The lack of standardized serverless function formats meant that all 15 benchmarks had to be adapted manually to the three different platforms, resulting in 45 total functions. While there are efforts for provider cross-compatibility [10], [21], they are third party solutions that are not officially supported, often leading to a reduced feature set.

Table 1 shows more details about each of the tests, their purpose, as well as any inputs. In terms of the input parameters, to ensure equal testing conditions, all function instances that require an input number have been invoked with the same random number set. The other parameters have been chosen to strike a balance between load stressing and execution time, simulating common operations that might be performed at the network edge.

3.3 Execution Environment and Method

All three platforms have been deployed on x86 based virtual machines, each allocated with 2 vCPU cores and 4GB of RAM, simulating resource constrained execution devices such as widely available computing boards. Each test has been executed in multiple fashions: i) serially, simulating one request at a time; ii) parallelly, simulating up to 100 requests at a time; iii) serially, but with the function scaled down to 0 replicas to test the cold-start latency. All tests, across all execution scenarios have been executed: 1, 5, 10, 25, 50, 75, and 100 times, thus measuring the total response time, instead of simply the execution latency, which does not take into account network latency. This approach allows us to determine the efficiency of the underlying communication protocol as well, since it differs among the included platforms.

In the following section we present the obtained results, as well as any platform specific information that might have influenced the outcome.

4 Results

The three different platforms vary significantly in their developer interfaces, function provisioning, scaling mechanisms, communication protocols, and overall feature set. Each of them uses different tooling mechanisms to ease the development of new functions and provides a ready-made set of templates. However, one thing that they all have in common is reliance on containerization as a runtime environment for the functions.

4.1 FaasD

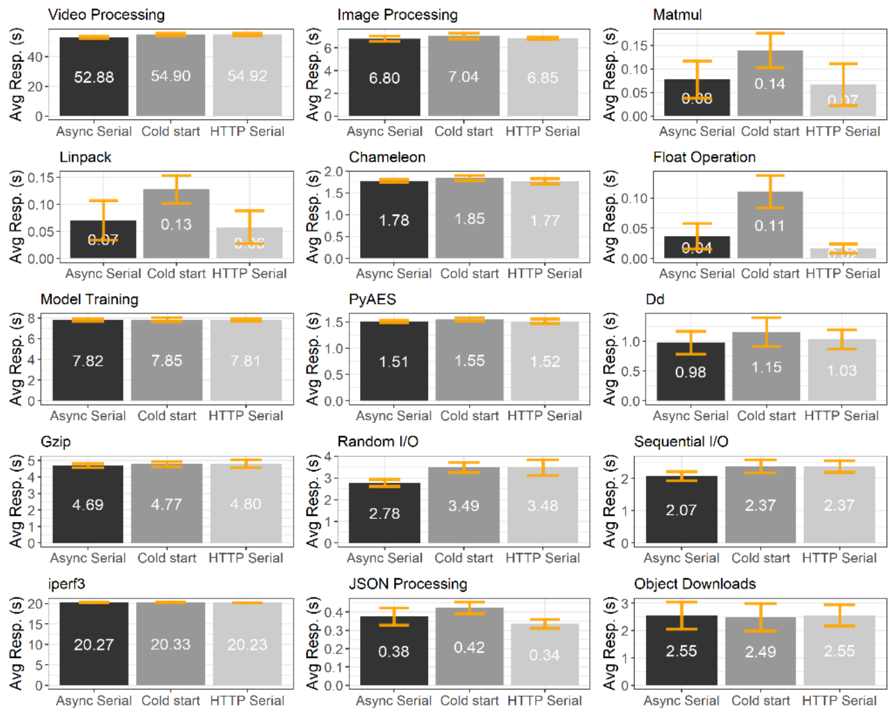

Fig. 1. FaasD execution performance comparison between different modes

FaasD is a new serverless platform created primarily for deployments at the network edge, on resource constrained devices. It is a more lightweight version than OpenFaaS, thus supporting the same function format. However, to achieve the desired simplicity, some notable features have been omitted, such as automatic scaling of function instances to more than a single replica, or automatic scaling to zero. Unfortunately, because of no container orchestration middleware, multiple function instances are not possible, while manual pausing of idling containers is possible using the application programming interface (API).

All functions are instantiated from manually built Docker images, and generic starter templates for different programming languages are provided. It supports two modes of invocation, either synchronous where the user waits until the response is returned or asynchronous where the serverless platform can issue a callback whenever a function has completed.

Figure 1 shows the average response time incurred during 100 serial invocations with one request at a time of the different tests, using the three invocation methods. The cold start latency is measured by synchronously invoking the function through the HTTP interface. It is evident that in most of the tests there is a cold start delay, a consequence of the paused container which must be resumed.

Parallel execution using asynchronous invocation is not possible in FaasD, since only one request can be processed at a time in this fashion, by a single container. However, this would not be an issue in environments with a container orchestrator, since spawning multiple container instances would solve the problem.

4.2 AWS Greengrass

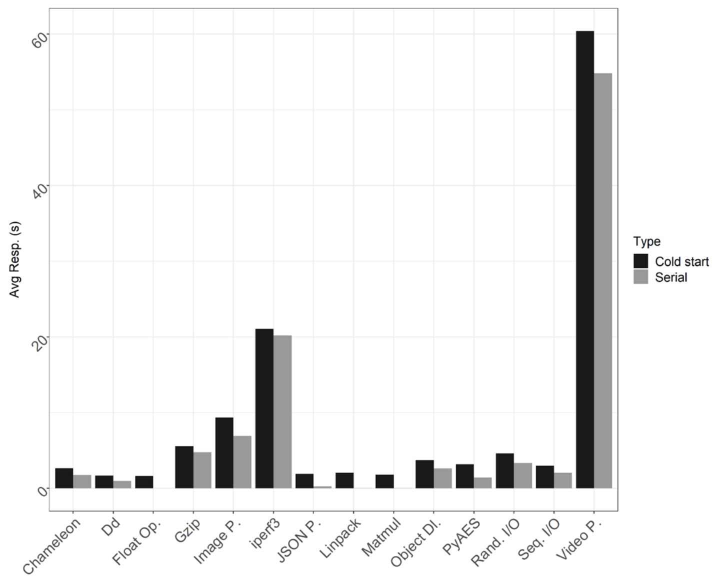

Fig. 2. AWS Greengrass execution latency between cold start and serial execution

AWS Greengrass is an Amazon Web Services product which allows lambda function deployment on customer owned edge devices. One of the primary features is the cross compatibility with the cloud-based Lambda service, where the same function code can be reused. It supports two modes of execution, either native or containerized, with options of strictly limiting the execution time, and memory consumption of the function itself, similar to the cloud counterpart of the service. Support for scaling to zero of unused functions as well as dynamically increasing the number of replicas in response to user demand is also supported, and transparent to the operator of the device. As a result of this, no request level isolation is guaranteed and multiple subsequent requests could be executed in the same container, sharing leftover temporary files.

The primary communication mechanisms with other devices and between Lambda functions is by message passing, where either the on-premise Greengrass device can be used as a message broker, or the cloud.

Unlike other serverless platforms, Greengrass does not rely on Docker images as a distribution format, instead all Lambdas are packaged as archive files, with a strict size limit imposed. This prevented the execution of the Model Training benchmark, whose resulting package was simply too large for this platform, explaining its omission from subsequent figures related to Greengrass.

Figure 2 shows a comparison between the regular latency and the cold-start latency during 100 invocations of all functions in the test suite. Similarly to FaasD, a cold-start delay is present, something that has been confirmed by other benchmarks focusing on commercial edge platforms as well [13].

4.3 Azure IoT Hub

Azure IoT Hub is a Microsoft service which supports function deployments to customer-owned edge devices as well. Like the other platforms, it relies on containerization for function execution, using the native Docker image format. There are no requirements to what interfaces the function must implement to be run in the serverless environment, and it is left to the developer to devise the communication mechanisms. Of course, message passing is one of the supported communication options, and in a similar fashion to Greengrass, the messages can either be routed locally or via the cloud. Unfortunately, neither scaling down to zero is supported nor horizontal scaling to multiple function instances. As a result of this, any concurrency is left to be implemented by the developers and built into the functions themselves, a popular option being the introduction of threading.

Cross-compatibility with the cloud counterpart of the serverless service is only possible when the functions are developed in the C# programming language, in all other instances, they must be packaged as Docker container by the users themselves. However, there is an option to run local instances of the Azure blob storage service and the relational database service, with the same API as those hosted in the cloud, thus removing the need for hosting on-premise alternatives.

There are no cold-start delays during the function invocation, since scale-down-to-zero is not supported, and the container instance hosting the function is always left in a running state.

4.4 Performance Comparison

Even though all selected platforms fulfill the primary goal of hosting serverless functions at the network edge, they vastly differ in their implementations. In this subsection we compare both their performance, and parallelization limits.

Synchronous Serial Execution

Figure 3 shows the serial synchronous execution performance of all platforms, across all different tests. In terms of FaasD, the HTTP serial synchronous execution is shown. As mentioned previously, results for the Model Training benchmark are absent for the Greengrass platform, since the resulting function package was too large to be deployed. This is a known problem in serverless environments, and one possible option is library size optimization through the removal of unused code [22] [23]. The results show that Greengrass exhibits the best performance for I/O bound workloads, such as the Dd, sequential I/O, random I/O and Gzip compression tests. A noticeable performance difference is seen in the model training and the float operation benchmarks, with the best performing platform being 25.76 and 80 per cent faster, respectively, than IoT Hub. All other tests show comparable levels of performance, with Greengrass being the fastest option in 12 test instances, and FaasD in 3, out of 15.

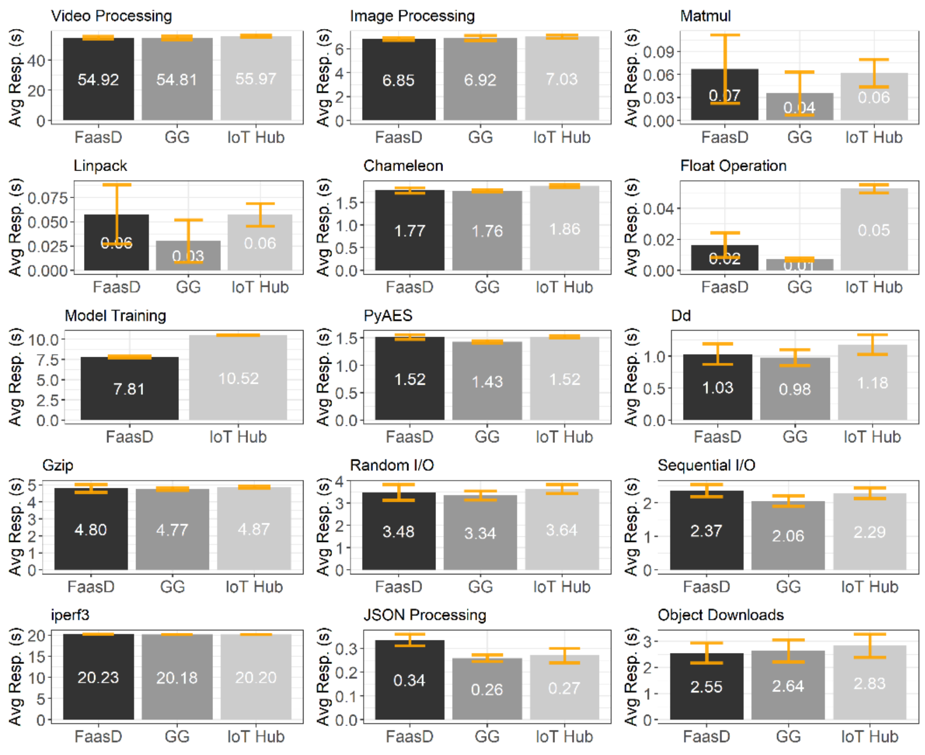

Fig. 3. Serial execution performance comparison, one request at a time

Parallel Execution

Comparing the parallel execution response time among the different platforms is challenging, because of the different levels of support for function parallelism. Both FaasD and Azure IoT Hub do not natively support automatic function scaling, a feature that is present in Greengrass. Furthermore, since all FaasD functions were implemented using the official Python template which includes a web server, the number of parallel requests that can be processed depends on the number of worker threads that have been spawned. If this value is left to its default settings, it is calculated based on the number of CPU cores that are available on the machine. Finally, in the case with IoT Hub, the developers have full control over any parallelization aspects.

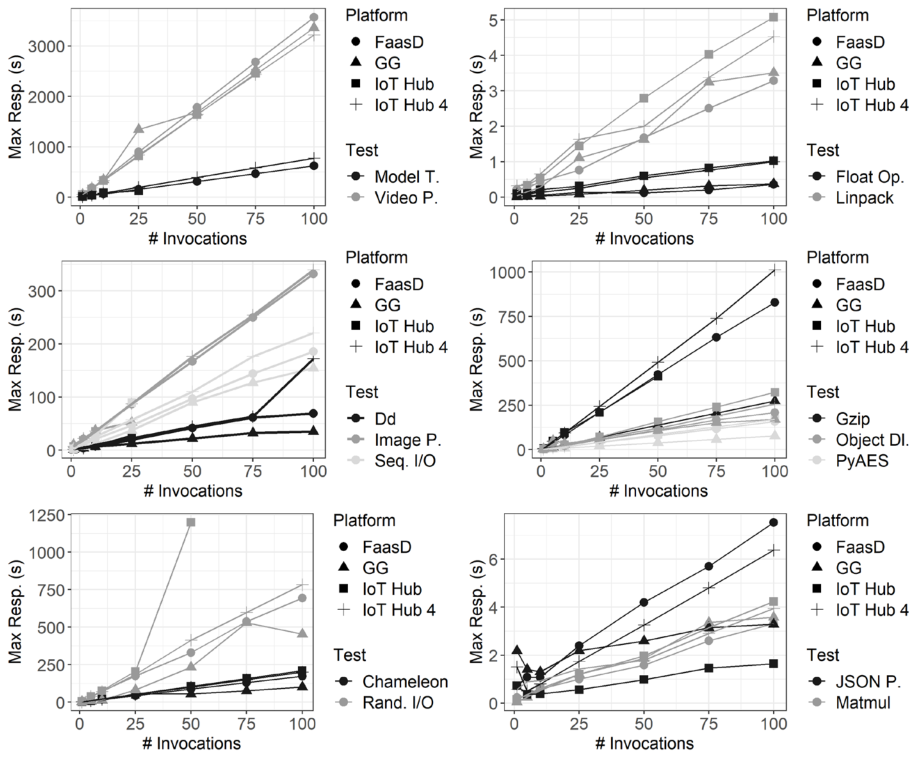

Fig. 4. Parallel execution performance comparison with varying degree of concurrency

Figure 4 shows the parallel execution performance of 14 out of the 15 total tests across all environments, where the maximum response time in seconds is given for 1, 5, 10, 25, 50, 75, and 100 parallel invocations. The iPerf3 test is omitted since it is not relevant in this case – iPerf3 supports only a single concurrent connection on a given port and conducting multiple tests at the same time would have required spawning of numerous parallel iPerf3 instances on the target machine. It is evident that a noticeable difference exists between the execution latencies of the platforms, and as expected, this is as a result of the varying levels of concurrency supported by each of them, as discussed in the previous subsections. The I/O advantage of Greengrass is also noticeable during parallel execution as well, while the limited parallelism of FaasD and IoT Hub 4 allow them to finish all tasks in a reliable manner, albeit, in some cases, slower. At the end, the concurrency level of each task should be tweaked as per its resource requirements, to mitigate situations which would lead to premature resource exhaustion, as is the case with the image processing benchmark for both Greengrass and IoT Hub with 100 concurrent workers, where 50 and 25 parallel executions, respectively, were not feasible. Total response time should be considered as well during such tweaking, which would prevent drastic latency increase during heavy I/O tasks, as is the case with IoT Hub and the Random I/O test.

With the aim of further investigating the maximum number of concurrent tasks executed by each platform, we have plotted the response times using histograms, as shown in figure 5. To normalize the different results among the different platforms, each response time during a parallel execution has been divided by the average response time during the serial, sequential, execution of the task, and the bin size has been set to the ratio between the average execution time during a parallel execution and the average execution time during a serial, sequential, execution. This ratio can also be seen as a slowdown coefficient between the parallel and sequential executions due to the higher concurrency. It is worth noting that the average response time takes into account any incurred communication latency, while the execution time relates only to the time taken to complete the task once it has been invoked, ignoring any communication delays.

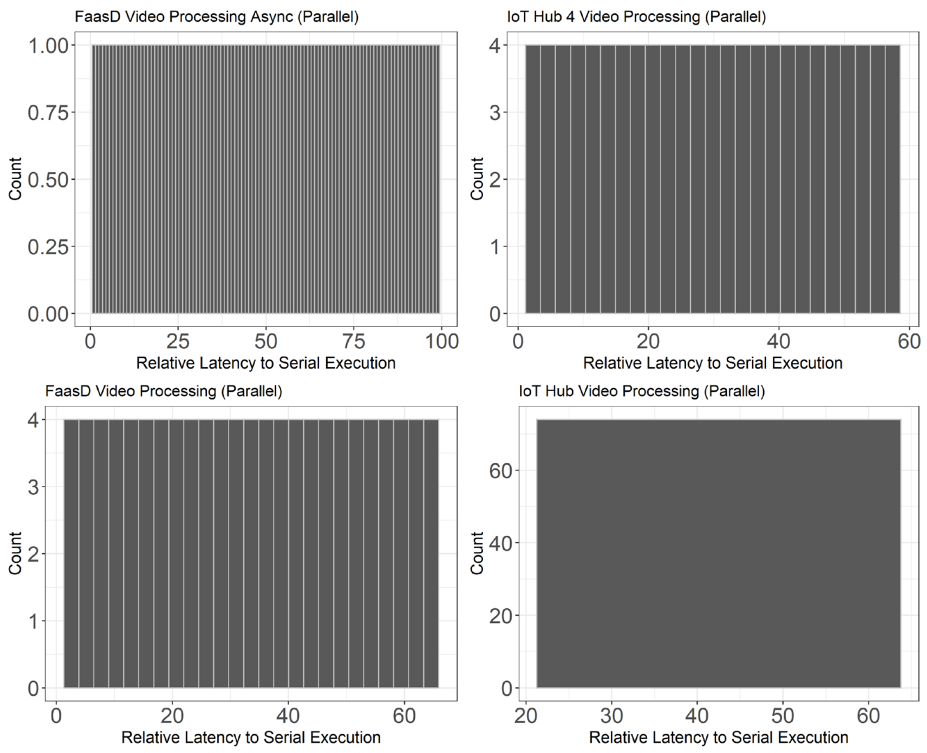

Fig. 5. Number of concurrent worker processes during parallel execution

By utilizing the previously described approach, we have confirmed that the FaasD function execution environments within the containers have defaulted to 4 worker instances when invoked parallelly using the synchronous HTTP interface, and that FaasD async execution supports only a single concurrent request per container instance. As a result of this, and exploiting the greater control that IoT Hub offers in terms of parallelism, we have decided to execute all tests on this platform using two levels of concurrency – 100 and 4. In this way, results that relate to the higher concurrency rating can be compared to Greengrass, and those with concurrency of 4 to FaasD. Unfortunately, some tests led to premature resource exhaustion on the test machines and could not be completed with all concurrency targets, explaining their omission in figure 4, further validating the point in terms of careful resource allocation, discussed earlier. These results have been used to show that, indeed, the manual concurrent implementation of the functions in Azure IoT Hub, using a maximum of 100 parallel workers, has met the desired concurrency level, as per the histogram shown in the bottom right corner in figure 5, analyzing the concurrency rating of the video processing test with 75 parallel requests issued.

5 Conclusion and Future Work

Using a set of existing benchmarks for cloud-based serverless solutions, and through their adaptation for edge-based platforms, we have compared the execution performance and response times of three different serverless products that can be deployed at the network edge. By including both open-source and commercial software, we have bridged the gap of other similar contributions, where the focus has been given exclusively to either option. In total, 15 different tests have been adapted, divided into three distinct categories, and each of these tests has been invoked in multiple scenarios, simulating both parallel and sequential task execution.

The results show that there is a comparable difference between the platforms when it comes to sequential, one at a time execution of functions, but a widening gap in terms of parallel execution. The main reasons for this are the varying levels of concurrency that each platform offers, with some choosing to abstract all scaling away from the developers, such as AWS Greengrass, while others catering only to the function deployment, and leaving concurrency to the implementation of the function itself, for example IoT Hub and to some extent FaasD. This varying rate of concurrency also impacts the execution performance and reliability, since all scaling must be done in respect to the available system resources, eliminating situations where tasks fail due to lack of disk space or memory.

When it comes to overall performance, AWS Greengrass shows a noticeable advantage in I/O related benchmarks, both in terms of serial and parallel execution, while FaasD offers more diverse execution options than the alternatives, supporting different communication protocols, and invocation strategies. The advantages of FaasD also extend to its observability, and integration options with other popular monitoring tools, instead of relying on similar options from commercial vendors. Finally, IoT Hub offers a more hands-on approach where the developer is responsible for the concurrency of the functions being executed.

Through the analysis of the test results, we have further confirmed some of the existing open issues in serverless architectures, such as the cold-start delay, non-standardized APIs [7] , and function size ballooning due to many dependencies [14], [15]. It is worth noting that serverless computing is still an area under active research and it is expected to further grow in the coming years, as solutions to the open problems are found [24].

In the future, we plan to extend our research of serverless platforms' performance with the inclusion of additional solutions that use alternative execution environments and do not focus on containers, thus helping to eliminate some of the issues related to containerization, such as the cold-start latency, overlay file system performance, and platform cross-compatibility.

References

[1] "Number of connected devices worldwide 2030," *Statista*. https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/ (accessed Jun. 01, 2021).

[2] T. Pfandzelter and D. Bermbach, "IoT Data Processing in the Fog: Functions, Streams, or Batch Processing?," in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic, Jun. 2019, pp. 201–206. doi: 10.1109/ICFC.2019.00033.

[3] L. Bittencourt *et al.*, "The Internet of Things, Fog and Cloud continuum: Integration and challenges," *Internet Things*, vol. 3–4, pp. 134–155, Oct. 2018, doi: 10.1016/j.iot.2018.09.005.

[4] N. Kratzke, "A Brief History of Cloud Application Architectures," *Appl. Sci.*, vol. 8, no. 8, p. 1368, Aug. 2018, doi: 10.3390/app8081368.

[5] N. El Ioini, D. Hästbacka, C. Pahl, and D. Taibi, "Platforms for Serverless at the Edge: A Review," in *Advances in Service-Oriented and Cloud Computing*, vol. 1360, C. Zirpins, I. Paraskakis, V. Andrikopoulos, N. Kratzke, C. Pahl, N. El Ioini, A. S. Andreou, G. Feuerlicht, W. Lamersdorf, G. Ortiz, W.-J. Van den Heuvel, J. Soldani, M. Villari, G. Casale, and P. Plebani, Eds. Cham: Springer International Publishing, 2021, pp. 29–40. Accessed: Apr. 09, 2021. [Online]. Available: http://link.springer.com/10.1007/978-3-030-71906-7_3

[6] "Introducing AWS Lambda," *Amazon Web Services, Inc.* https://aws.amazon.com/about-aws/whats-new/2014/11/13/introducing-aws-lambda/ (accessed Jun. 01, 2021).

[7] J. M. Hellerstein *et al.*, "Serverless Computing: One Step Forward, Two Steps Back," *ArXiv181203651 Cs*, Dec. 2018, Accessed: Feb. 05, 2021. [Online]. Available: http://arxiv.org/abs/1812.03651

[8] R. Wolski, C. Krintz, F. Bakir, G. George, and W.-T. Lin, "CSPOT: portable, multi-scale functions-as-a-service for IoT," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, Arlington Virginia, Nov. 2019, pp. 236–249. doi: 10.1145/3318216.3363314.

[9] W. Ling, L. Ma, C. Tian, and Z. Hu, "Pigeon: A Dynamic and Efficient Serverless and FaaS Framework for Private Cloud," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, Dec. 2019, pp. 1416–1421. doi: 10.1109/CSCI49370.2019.00265.

[10] Z. Huang, Z. Mi, and Z. Hua, "HCloud: A trusted JointCloud serverless platform for IoT systems with blockchain," *China Commun.*, vol. 17, no. 9, pp. 1–10, Sep. 2020, doi: 10.23919/JCC.2020.09.001.

[11] J. Kim and K. Lee, "FunctionBench: A Suite of Workloads for Serverless Cloud Function Service," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy, Jul. 2019, pp. 502–504. doi: 10.1109/CLOUD.2019.00091.

[12] A. Das, S. Patterson, and M. Wittie, "EdgeBench: Benchmarking Edge Computing Platforms," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Zurich, Dec. 2018, pp. 175–180. doi: 10.1109/UCC-Companion.2018.00053.

[13] M. Gorlatova, H. Inaltekin, and M. Chiang, "Characterizing task completion latencies in multi-point multi-quality fog computing systems," *Comput. Netw.*, vol. 181, p. 107526, Nov. 2020, doi: 10.1016/j.comnet.2020.107526.

[14] S. S. Gill *et al.*, "Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges," *Internet Things*, vol. 8, p. 100118, Dec. 2019, doi: 10.1016/j.iot.2019.100118.

[15] P. K. Gadepalli, G. Peach, L. Cherkasova, R. Aitken, and G. Parmer, "Challenges and Opportunities for Efficient Serverless Computing at the Edge," in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2019, pp. 261–2615. doi: 10.1109/SRDS47363.2019.00036.

[16] A. Palade, A. Kazmi, and S. Clarke, "An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge," in *2019 IEEE World Congress on Services (SERVICES)*, Jul. 2019, vol. 2642–939X, pp. 206–211. doi: 10.1109/SERVICES.2019.00057.

[17] *kmu-bigdata/serverless-faas-workbench*. BigData Lab. in KMU, 2021. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/kmu-bigdata/serverless-faas-workbench

[18] *openfaas/faasd*. OpenFaaS, 2021. Accessed: May 09, 2021. [Online]. Available: https://github.com/openfaas/faasd

[19] "faasd - OpenFaaS." https://docs.openfaas.com/deployment/faasd/ (accessed May 31, 2021).

[20] P. Kravchenko, *kpavel/openwhisk-light*. 2020. Accessed: May 09, 2021. [Online]. Available: https://github.com/kpavel/openwhisk-light

[21] "The Serverless Application Framework | Serverless.com," *serverless*. https://serverless.com/ (accessed Apr. 27, 2021).

[22] A. Christidis, R. Davies, and S. Moschoyiannis, "Serving Machine Learning Workloads in Resource Constrained Environments: a Serverless Deployment Example," in *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, Kaohsiung, Taiwan, Nov. 2019, pp. 55–63. doi: 10.1109/SOCA.2019.00016.

[23] I. Pelle, J. Czentye, J. Doka, A. Kern, B. P. Gero, and B. Sonkoly, "Operating Latency Sensitive Applications on Public Serverless Edge Cloud Platforms," *IEEE Internet Things J.*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.3042428.

[24] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Gener. Comput. Syst.*, vol. 79, pp. 849–861, Feb. 2018, doi: 10.1016/j.future.2017.09.020.