2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)

Benchmarking FaaS Platforms: Call for Community Participation

Jörn Kuhlenkamp
Information Systems Engineering,
TU Berlin, Germany
jk@ise.tu-berlin.de

Sebastian Werner
Information Systems Engineering,
TU Berlin, Germany
sw@ise.tu-berlin.de

***Abstract*—The number of available FaaS platforms increases with the rising popularity of a "serverless" architecture and development paradigm. As a consequence, a high demand for benchmarking FaaS platforms exists. In response to this demand, new benchmarking approaches that focus on different objectives continuously emerge. In this paper, we call for community participation to conduct a collaborative systematic literature review with the goal to establish a community-driven knowledge base.**

***Keywords*-serverless, FaaS, benchmarking, secondary study**

I. Introduction

Serverless computing [1] is an emerging architecture and development paradigm for building cloud-based software services that rely entirely on fully managed cloud infrastructure services. One of the main benefits of serverless computing is the potential to significantly reduce the cost of service development and operations [2], [3]. Thus, a large number of open source FaaS platforms [1], [4] and fully managed FaaS service offers have emerged over the last years. As a consequence, the need to benchmark [5] different qualities and features of FaaS platforms has been recognized by research and industry [6]–[8]. Therefore, a variety of recent publications present benchmarking approaches for FaaS platforms (see table 2). However, it remains challenging to efficiently identify an appropriate benchmarking approach for a question at hand. We argue that this impedes the verification and reproduction of existing results and hinders an extension of the current scope of existing benchmarks.

With our work, we provide the first contributions to answer the research question: How can experimenters that benchmark FaaS platforms efficiently identify the current state-of-the-art with high quality?

In our approach, we adopt generic procedures for conducting a systematic literature review (SLR) [9]. In addition, we propose to increase the quality and relevance of the obtained results by enabling the community to participate in designing the SLR method and verifying extracted data.

In our work, we present three contributions over the state-of-the-art:

- We present a method for conducting SLR that captures existing approaches for benchmarking FaaS platforms.
- We present a extended method for conducting a continuous and collaborative SLR including a call for community participation.
- We present preliminary results of the SLR.

The remainder of this paper is structured as follows. We present an SLR method (section II) and extensions to the method in support of a call for community participation (III). We present and discuss preliminary results (section IV) followed by a conclusion (section V).

II. Method

We present our method for a systematic literature review (SLR) [9] based on generic procedures proposed by Kitchenham [10]. Kitchenham describes *tasks* that are organized in *phases*. Figure 1 illustrates tasks that we consider for our SLR and *artifacts* that are produced by a specific task. Furthermore, figure 1 shows artifacts are required to execute a task. Moreover, we distinguish between *committee tasks* and *community tasks*. Committee tasks are exclusively executed by researches that initiate an SLR. Community tasks are executed under the participation of a community (see section III), respectively. Detailed procedures for conducting an SLR are documented in a *review protocol* [10]. The following subsections provide a detailed discussion of our proposed review protocol.

A. Research Questions

We address three research questions with our study:

- **RQ1 What FaaS experiments exist since 2015?**
- **RQ2 What are designs of FaaS experiments?**
- **RQ3 How reproducible are FaaS experiments?**

In 2015, the first productive and broadly available fully managed FaaS offer, i.e., AWS Lambda, was launched. Therefore, we start our survey at the beginning of 2015. We discuss RQ1-3 in detail in section IV.

1) Objectives : RQ1 aims to provide an overview of existing scientific studies that present empirical evidence related to FaaS platforms collected by experimentation. Precisely, we are interested in identifying researched FaaS platforms, treatments, e.g., workloads, configurations, and implementations, and reported effects of treatments on qualities, e.g., performance, scalability, and availability.

In addition, we are interested in the motivation behind an experiment. For example, Bermbach et al. [5] differentiate

978-1-7281-0359-4/18/$31.00 ©2018 IEEE
DOI 10.1109/UCC-Companion.2018.00055

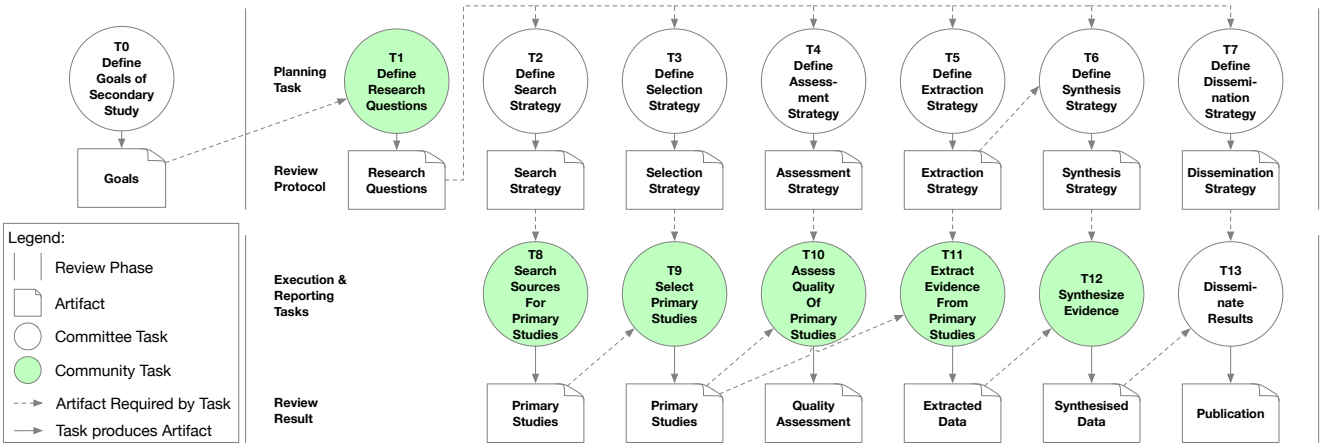

Figure 1. Illustration of Tasks, Artifacts Produced by Tasks, and Dependencies for the Applied Structured Literature Review Process based on [10]

between application-driven and micro-benchmarks. Similarly, an experiment can focus on evaluating (i) a real-world FaaS-based application from a specific application domain or (ii) a specific feature of a FaaS platform. Application examples include image recognition [11], chat bots [12] and IoT [3]. Examples of features are: support of different event triggers and fully managed elastic scaling strategies.

2) Designs : A detailed understanding of an experiment design is critical to verify, reproduce, and assess the relevance of the presented results. Due to space constraints, we focus on four design elements to address RQ2. We refer the reader to our knowleadge base (see section III) for additional information. The four design elements are: (i) workload, (ii) deployment package, (iii) platform configuration, and (iv) composed third-party services. A FaaS workload can comprise of different types and numbers of events that are generated at specific times. Deployment packages include bundled source code - provided by a FaaS user - that is executed in functions in response to events. The configuration of a FaaS platform can include memory or event triggers. Functions can be part of a service composition. Thus, functions can execute in response to events generated by a third-party service, e.g., an API gateway, or call a third-party service during execution, e.g., database service.

3) Reproducibility : Reproducibility is a quality of foremost importance for experiments that allows to verify and affirm gathered results. RQ3 aims to assess the reproducibility of experiment designs (see IV).

B. Search Strategy

The design of our search strategy is inspired by [10], [13], [14]. The seed of our study consists of 5 seminal publications, for which we assume wide dissemination. The seed comprises publications that focus on serverless benchmarking [6], [15] or open challenges in serverless computing [8], [16], [17]. All forward and backward references are followed recursively to identify additional publication candidates. A publication candidate is included in the SLR if it suffices the criteria presented in section II-C.

C. Selection Strategy

A publication must suffice all criteria C1-C6 to be included in the SLR. We excluded publications from journals listed on[1] as proposed by [13]. The criteria are:

- **C1**: The publication is peer-reviewed and scientific.
- **C2**: The publication date is after Jan 1st 2015.
- **C3**: The publication includes at least one experiment.
- **C4**: At least one experiment uses a fully managed FaaS offer or open-source FaaS system as SUT.
- **C5**: At least one design element (see section II-A2) of an experiment that suffices C4 is described.
- **C6**: The publication presents results for at least one experiment that suffices C5.

D. Data Extraction Strategy

A data extraction strategy defines what evidence is extracted from each primary study included in an SLR. We partially discuss the schema that we propose for data extraction. The complete schema is referenced in section III. For each publication, four fields are extracted: `title`, `author`, `affiliation`, and `year`. For each experiment, the fields are:

- **D1 SUT** The SUT in the experiment.
- **D2 Abstraction** Indicates if the experiment focuses on a FaaS-based application or a FaaS platform feature.
- **D3 Quality** The qualities under investigation.
- **D4 Feature** FaaS platform features under investigation.
- **D5 Parameter** The values of controlled parameters.
- **D6 Platform** The target programming platform for deployment packages.
- **D7 Functionality** The functionality implemented by function handlers in deployment packages.

[1] https://beallslist.weebly.com/

- **D8 Composition** Indicates if a service external to the FaaS platform is called during the execution of a function handler.
- **D9 Ressources** The configured amount of compute resources for the execution of a function handler.
- **D10 Generator** The system that is used to generate workloads during an experiment.
- **D11 Determinism** Indicates if the method for generating a workload is deterministic or probabilistic.
- **D12 Distance** The collocation of the workload generator and SUT.

The value – indicates a field that does not require a value. The value `N/A` indicates a field that requires a value that is not provided by a publication.

E. Synthesis Strategy

To answer RQ1, we aggregate values for the three fields D1, D3, and D5 in a bubble plot as suggested by Petersen et al. [18]. To answer RQ3, we derive a reproducibility score (R-Score) for each experiment based on missing information indicated by the number of `N/A` values per experiment.

III. Call for Participation

FaaS platforms and FaaS-based applications are under rapid development. Therefore, we argue that a rapid dissemination and holistic awareness of experiment designs and results is highly desirable for the community comprising academia and industry. We argue that our initial SLR method (see section II) implies a number of limitations to deliver on this need and discuss three examples of such limitations. First, results of an SLR only present a snapshot in time. Therefore, the likelihood to miss important experiments quickly increases with elapsed time in a rapidly evolving field such as FaaS platforms. Second, due to standard publication processes, results only becomes available to a community with a delay. Third, the time of a single researcher is limited and task T9-11 proved to be time consuming due to missing means for automation. To address some these limitations, we call for community participation and propose to establish a community-driven knowledge base.

In support of this call, we provide a living document[2]. The document contains the complete review protocol and current results. We periodically snapshot the document and store all snapshots in support of referencing. We discuss select features of the living document that allow community members from academia and industry to participate in community tasks (see figure 1).

1) Propose Research Question:

- **Community Task**: Define Research Questions (T1)
- **Motivation**: The research questions do not address the goal of the secondary study holisticly.
- **Consequences**: A new research question requires the execution of task T2-13.

2) Propose Publication:

- **Community Task**: Search Sources (T8)
- **Motivation**: The search strategy or searches do not identify all appropriate publications.
- **Consequences**: Adding a new publication candidate requires the execution of task T9-13.

3) Propose Value for Experiment:

- **Community Task**: Extract Evidence (T11)
- **Motivation**: Reviewers can fail to extract data correctly or experimenters want to provide missing information.
- **Consequences**: Adding a new value requires the execution of task T12-13 and repeating voting proceedures [19] in T11.

[2]http://www.tu-berlin.de/?id=199198

IV. Preliminary Results and Discussion

In this section, we present and discuss the preliminary results of our study. We expect that with community participation more additional results will follow.

RQ1 *What FaaS experiments exist since 2015?*

In this snapshot, we identified a total of 30 publications (see Table I) of which 9 met the selection criteria (II-C). The selected primary studies described a total of 26 experiments (see Table I). Figure 2 shows that most experiment research targets all four major FaaS providers. We can also observe that 78 percent of all benchmarks presented in the literature are focused on established qualities such as scalability and performance.

Our results indicate a lack of benchmarks of more complex qualities such as cost-efficiency and availability. However, we argue that such qualities provide the core value proposition FaaS. Furthermore, our results suggest a lack of benchmarks that measure qualities with regards to the serverless application model. One key difference of serverless architectures to alternatives like a micro-services architecture is that serverless functions exclusively rely on external services. These services are used to provided state, event distribution, and storage. Thus it is crucial to detect how different supporting services impact qualities, and how services differ between FaaS providers. However, only one experiment specifically measured the impact of different trigger methods on performance.

We also found that authors measured these qualities out of different motivations, some authors aimed to measure specific features of the FaaS infrastructure, while others measured how qualities differ for application use-cases. Investigated application use-cases span from machine learning, big data processing, and computer graphics. These benchmarks involved workloads such as face detection (E3c), weather forecasting (E3e), and image manipulation (E9a). Besides application related benchmarks, authors also reported interests in FaaS platform features like the propagation of code

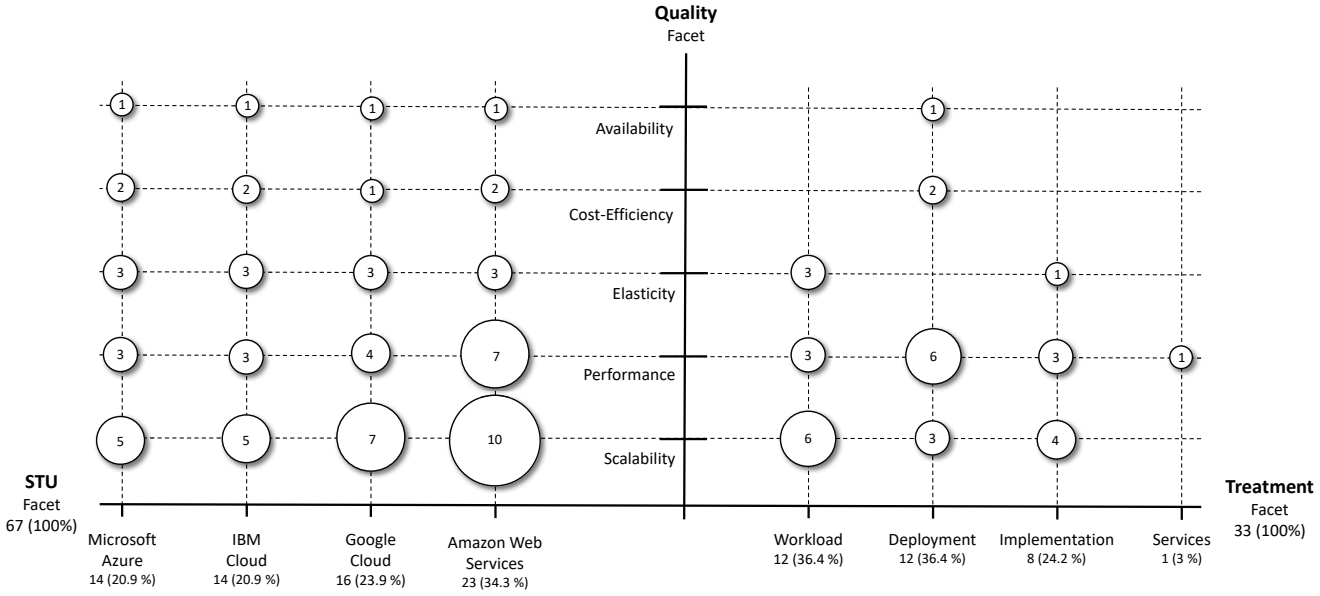

Figure 2. Bubble Plot that Illustrates FaaS Platforms,Treatments, and Observed Qualities for Experiments Presented in Primary Studies Included in Preliminary Results

updates (E2f), the event delivery behavior (E2h, E5a, E7b) and internals of elasticity control (E1a, E1b, E2e). Authors also discuss the cold and hot start problem of serverless applications and the pausing and deprovisioning behavior function containers.

RQ2 *What are designs of FaaS experiments?*

Our results indicate that no standard designs for FaaS benchmarks exist, yet. Although efforts towards a common standard exist [2], [7], our results do not show that these efforts have been adopted, yet.

Table I indicates a wide difference in tooling, workload generation, as well as environment setups. However, we could observe some common designs within the performed benchmarks. For example, our results indicate that STU function implementations can be categorized into four types. Implementations are either trivial functions, such as sleep or No-Op functions, trivial algorithms provided in pseudo-code or complex algorithms, provided either in native FaaS programming code or as binary packages which are executed using a FaaS wrapper function. For the binary packages, we identified that the Linpack binary is used by many authors to benchmark CPU performance. Furthermore, with regards to workloads, we identified two common models, *direct execution*, using command line tooling offered by the cloud provider, and *indirect execution*, using other cloud services to trigger functions (see Table I column: services used). We could not identify a common workload generator. Instead, each study used a different approach to run their experiments. However, a few studies utilized a common deployment mechanism provided by the serverless-framework. Therefore, our results indicate, a set of popular tools and approaches for the design of a FaaS benchmark exists.

RQ3 *How reproducible are FaaS experiments?*

Our results also indicate that many primary studies only reported some of the mandatory information to validate the reported results. Out of the 26 collected experiments only 3 supplied all information we recognized as necessary to reproduce a benchmark.

In order to characterize reproducibility, we create the R-Score, which ranks experiments from 0-4. For each of the categories reported in Table I (workload generator, function implementation, platform configuration, and service used), we assign 1 point if all information of the category is available and assigned a half point if at least some information is available.

The average R-Score of all the reviewed literature is 2.6 which indicates that much information is missing regarding reproducibility. The category that is the least discussed is the workload generator, where only 30% reported any information. Besides that, only a few authors reported function configuration parameters like memory. Even though most report what kind of function was used in their experiments and what services were used with that function, none of the experiments specified how the used services were configured. One possible explanation behind the lack of information could be that the main scope of some of

Table I
OVERVIEW OF EXTRACTED EXPERIMENTS.

Reference		Workload Generator		Function Implementation			Platform Configuration		Services used	R-Score
Pub.	Exp.	Tool	Distance	Functionality	Type	Sources	Programming Environment	Memory		
[15]	E1a	Perf Tool	Region	Empty	Trivial	yes	js	512	No	4.0
	E1b	Perf Tool	Region	Empty	Trivial	yes	js	512	No	4.0
[20]	E2a	N/A	N/A	N/A	N/A	no	*	512,1536	N/A	1.5
	E2b	N/A	N/A	Matrix Mult.	Native	no	N/A	512,1536	No	2.5
	E2c	N/A	N/A	N/A (IO)	Native	no	N/A	512,1536	No	2.5
	E2d	N/A	N/A	N/A (net)	Native	no	*	512,1536	Yes	3.0
	E2e	N/A	N/A	Fast	Native	no	js	512,1536	No	3.0
	E2f	N/A	N/A	N/A	Native	no	*	512,1536	No	2.5
	E2g	N/A	N/A	N/A	Native	no	Py, js	3000	N/A	2.5
	E2h	N/A	N/A	N/A	N/A	no	N/A	N/A	Yes	1.5
	E2i	N/A	N/A	Wait	Trivial	no	*	N/A	No	2.5
[11]	E3a	N/A	N/A	Fibonacci	Pseudo	yes	Py	128-1024	No	3.0
	E3b	N/A	N/A	PI calculation	Native	no	Py, Py3	N/A	No	2.5
	E3c	N/A	N/A	Face detection	Native	no	Py	N/A	Yes	2.5
	E3d	N/A	N/A	Pwd Cracking	Native	no	Py	512	N/A	2.0
	E3e	HyperFlow	N/A	Weather	Binary	no	Py	N/A	N/A	2.0
[16]	E4a	N/A	N/A	Idle 200ms	Trivial	no	N/A	N/A	Yes	2.0
	E4b	N/A	N/A	Idle 200ms	Trivial	no	N/A	N/A	Yes	2.0
[21]	E5a	Custom	N/A	Linpack	Binary	no	N/A	512	N/A	2.0
	E5b	Custom	N/A	Linpack	Binary	no	N/A	256-2048	N/A	2.0
[22]	E6a	mu	Multi-region	Linpack	Binary	no	mu	N/A	Yes	3.5
[7]	E7a	N/A	Remote	Random gen.	Binary	no	js	128-1024	Yes	3.0
	E7b	N/A	Remote	Linpack	Binary	no	N/A	N/A	Yes	2.5
	E7c	HyperFlow	Remote	Linpack	Binary	no	js	128-1024	No	4.0
[17]	E8a	PyWren	Region	Matrix Mult.	Native	no	Py	N/A	Yes	3.5
[23]	E9a	N/A	N/A	Image Crop	N/A	no	N/A	N/A	Yes	2.5

* = js, Java, C#, Py, Py3

the collected papers is not in benchmarking but rather in architecture improvements or architecture design, where an experiment is only used to showcase a specific improvement.

Additionally, some of the workloads described in the literature were based on HTTP requests, which might indicate that the authors used standard traffic generation tools to perform their experiments. With some effort, the results of these studies could be reproduced even without the source code and configuration of a given workload generator as all experiments offered a rough outline about there workload.

V. CONCLUSION

In this paper, we presented the preliminary results for a systematic literature review in support of benchmarking FaaS platforms.

Based on a preliminary review of the results we identified limitations regarding the quality and relevance of the extracted results. We conclude that these limitations are due to the fast innovation cycles in FaaS and slow publication cycle of scientific findings. Therefore, we call for community participation to overcome these limitations. We present an extended SLR method with a living document[3] in support of this. Our preliminary results indicate a lack of benchmarks that measure effects on complex quality relationships such as cost-efficiency. Furthermore, our results indicate a lack of benchmarks that observe functions not in isolation but in the composed environment of a cloud service.

For future work, we aim to periodically report the results of continued SLR. Moreover, we will provide improved tool support for conducting a community-driven literature review. Furthermore, we plan to provide reproducible benchmarks that quantify complex qualities. In addition, we plan to provide application-level benchmarks for complex FaaS based applications that interact with cloud services outside of the FaaS platform.

[3] http://www.tu-berlin.de/?id=199198

ACKNOWLEDGMENTS

The work in this paper was performed in the context of the DITAS and BloGPV.Blossom project. DITAS is partially supported by the European Commission through the Horizon 2020 Research and Innovation program under contract 731945. BloGPV.Blossom is partially funded by the Germany Federal Ministry for Economic Affairs and Energy (BMWi) under grant no. 01MD18001E. The authors assume responsibility for the content.

REFERENCES

[1] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless Computation with OpenLambda," *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '16)*, no. Figure 1, pp. 14–19, 2016.

[2] A. Eivy, "Be wary of the economics of" serverless" cloud computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, 2017.

[3] J. Kuhlenkamp and M. Klems, "Costradamus: A Cost-Tracing System for Cloud-based Software Services," in *Proceedings of the 15th International Conference on Service Oriented Computing (ICSOC17)*. Springer, 2017.

[4] Apache Software Foundation, "Apache OpenWhisk," [Online; accessed 15.10.2017].

[5] D. Bermbach, E. Wittern, and S. Tai, *Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective*. Springer, 2017.

[6] E. van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A spec rg cloud group's vision on the performance challenges of faas cloud architectures," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: ACM, 2018, pp. 21–24.

[7] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *European Conference on Parallel Processing*. Springer, 2017, pp. 415–426.

[8] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*. Springer, 2017, pp. 1–20.

[9] B. A. Kitchenham, "Procedures for performing systematic reviews," Tech. Rep., 2004.

[10] ——, "Guidelines for performing systematic literature reviews in software engineering," Tech. Rep., 2007.

[11] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and hpc," in *Latin American High Performance Computing Conference*. Springer, 2017, pp. 154–168.

[12] 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.

[13] P. Leitner and J. Cito, "Patterns in the chaos - a study of performance variation and predictability in public iaas clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, pp. 15:1–15:23, Apr. 2016.

[14] B. A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 273–281.

[15] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 405–410.

[16] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," *Elastic*, vol. 60, p. 80, 2016.

[17] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 445–451.

[18] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *EASE*, vol. 8, 2008, pp. 68–77.

[19] M. Kuhrmann, D. M. Fernández, and M. Daneva, "On the pragmatic design of literature studies in software engineering: an experience-based guideline," *Empirical software engineering*, vol. 22, no. 6, pp. 2852–2891, 2017.

[20] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," Technical report, April 2018. https://doi. org/10.13140/RG. 2.2. 28642. 84165, Tech. Rep.

[21] M. Pawlik, K. Figiela, and M. Malawski, "Performance evaluation of parallel cloud functions," 2018.

[22] S. Fouladi, R. S. Wahby, B. Shacklett, K. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads." in *NSDI*, 2017, pp. 363–376.

[23] G. McGrath, J. Short, S. Ennis, B. Judson, and P. Brenner, "Cloud event programming paradigms: Applications and analysis," in *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 400–406.