The Journal of Systems & Software 170 (2020) 110708

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Function-as-a-Service performance evaluation: A multivocal literature review

Joel Scheuner [*], Philipp Leitner

Software Engineering Division, Chalmers | University of Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 7 April 2020
Received in revised form 1 June 2020
Accepted 24 June 2020
Available online 28 June 2020

Keywords:
Cloud computing
Serverless
Function-as-a-Service
Performance
Benchmarking
Multivocal literature review

ABSTRACT

Function-as-a-Service (FaaS) is one form of the serverless cloud computing paradigm and is defined through FaaS platforms (e.g., AWS Lambda) executing event-triggered code snippets (i.e., functions). Many studies that empirically evaluate the performance of such FaaS platforms have started to appear but we are currently lacking a comprehensive understanding of the overall domain. To address this gap, we conducted a multivocal literature review (MLR) covering 112 studies from academic (51) and grey (61) literature. We find that existing work mainly studies the AWS Lambda platform and focuses on micro-benchmarks using simple functions to measure CPU speed and FaaS platform overhead (i.e., container cold starts). Further, we discover a mismatch between academic and industrial sources on tested platform configurations, find that function triggers remain insufficiently studied, and identify HTTP API gateways and cloud storages as the most used external service integrations. Following existing guidelines on experimentation in cloud systems, we discover many flaws threatening the reproducibility of experiments presented in the surveyed studies. We conclude with a discussion of gaps in literature and highlight methodological suggestions that may serve to improve future FaaS performance evaluation studies.

1. Introduction

Cloud computing continues to evolve, moving from low-level services such as Amazon Web Services (AWS) EC2, towards integrated ecosystems of specialized high-level services. Early Infrastructure-as-a-Service (IaaS) cloud services are generalist solutions, which only provide a low-level abstraction of computing resources, typically in the form of self-administered virtual machines. In contrast, the emerging serverless[1] paradigm aims to liberate users entirely from operational concerns, such as managing or scaling server infrastructure, by offering a fully-managed high-level service with fine-grained billing (Eyk et al., 2017). As a type of specialist service, serverless offerings range from simple object storage (e.g., Amazon S3) to deep learning-powered conversational agents (e.g., Amazon Lex, the technology behind Alexa).

To connect the different services (e.g., feed images from S3 into a transcoding service), a serverless-but-generalist service is required as a 'glue' to bridge the gaps (in triggers, data formats, etc.) between services. This is the primary niche that Function-as-a-Service (FaaS) platforms, such as AWS Lambda,[2] have emerged to fill.

In FaaS, developers provide small snippets of source code (often JavaScript or Python) in the form of programming language functions adhering to a well-defined interface. These functions can be connected to trigger events, such as incoming HTTP requests, or data being added to a storage service. The cloud provider executes the function (with the triggering event as input) on-demand and automatically scales underlying virtualized resources to serve elastic workloads of varying concurrency.

FaaS is used for a wide variety of tasks (Leitner et al., 2019), including as a 'glue' holding together a larger serverless application, as a backend technology to implement REST services, and for a variety of data analytics (e.g., PyWren Jonas et al., 2017) and machine learning tasks (e.g., serving deep learning models Ishakian et al., 2018). This makes their performance crucial to the efficient functioning of a wide range of cloud applications.

Previous research has indicated performance-related challenges common to many FaaS platforms. Among others, cold start times (the time required to launch a new container to execute a function) can lead to execution delays of multiple seconds (Manner et al., 2018), hardware heterogeneity makes

[*] Corresponding author.
E-mail addresses: scheuner@chalmers.se (J. Scheuner), philipp.leitner@chalmers.se (P. Leitner).

[1] https://martinfowler.com/articles/serverless.html.

[2] https://aws.amazon.com/lambda/.

https://doi.org/10.1016/j.jss.2020.110708

predicting the execution time of a function difficult (Figiela et al., 2018), and complex triggering mechanisms can lead to significant delays in function executions on some platforms (Pelle et al., 2019). So far, reports about performance-related challenges in FaaS are disparate and originate from different studies, executed with different setups and different experimental assumptions. The FaaS community is lacking a consolidated view on the state of research on FaaS performance.

This paper addresses this gap. We conduct a multivocal literature review (MLR) (Garousi et al., 2019) to consolidate academic and industrial (i.e., grey literature) sources that were published between 2016 and 2019 and report performance measurements of FaaS offerings of different platforms. The area of our study is the performance evaluation (also referred to as performance benchmarking) of FaaS offerings, both of commercial public services and open source systems intended to be installed in private data centres. Our research goal is two-fold. Firstly, we characterize the landscape of existing isolated FaaS performance studies. Secondly, we identify gaps in current research (and, consequently, in our understanding of FaaS performance). We also provide methodological recommendations aimed at future FaaS performance evaluation studies.

The remainder of this paper is structured as follows. Section 2 introduces FaaS performance benchmarking. Section 3 defines and motivates our research questions. Section 4 describes our MLR study design before we present and discuss the results in Section 5. The main findings then lead to the implications of our study in Section 6, where we also identify gaps in current literature. Section 7 relates our work and results to other research in the field. Finally, Section 8 summarizes and concludes this paper.

2. Background

This section introduces FaaS performance benchmarking based on the two benchmark types covered in this paper. Micro-level benchmarks target a narrow performance aspect (e.g., floating-point CPU performance) with artificial workloads, whereas application-level benchmarks aim to cover the overall performance (i.e., typically end-to-end response time) of real-world application scenarios. We clarify this distinction of benchmark types based on example workloads from our analysed studies.

2.1. Micro-benchmarks

Listing 1 shows a simple CPU-intensive AWS Lambda function written in the Python programming language. This example function serves as a CPU micro-benchmark in one of our surveyed studies [A21]. It implements a provider-specific handler function to obtain the parameter n from its triggering invocation event (see line 13). The floating-point operations helper function (see line 4) exemplifies how common FaaS micro-benchmarks measure latency for a series of CPU-intensive operations.

```
1  import math
2  from time import time
3
4  def float_operations(n):
5      start = time()
6      for i in range(0, n):
7          sin_i = math.sin(i)
8          cos_i = math.cos(i)
9          sqrt_i = math.sqrt(i)
10     latency = time() - start
11     return latency
12
13 def lambda_handler(event, context):
14     n = int(event['n'])
15     result = float_operations(n)
16     print(result)
17     return result
```

Listing 1: Lambda function with CPU micro-benchmark [A21]

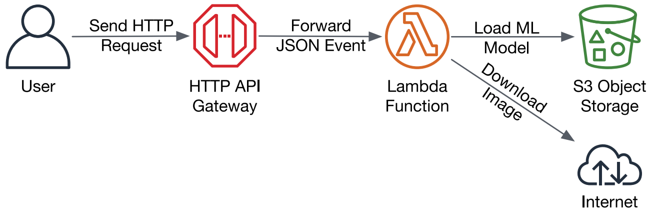

Fig. 1. FaaS application for Machine Learning (ML) inference.

2.2. Application-benchmarks

Fig. 1 depicts the architecture of an AWS Lambda FaaS application that performs machine learning (ML) inferencing. The diagram is based on the mxnet-lambda reference implementation[3] used in adjusted form by one study [A16] to benchmark ML inferencing. The application predicts image labels for a user-provided image using a pre-trained deep learning model. A user interacts with the application by sending an HTTP request to the HTTP API gateway, which transforms the incoming HTTP request into a cloud event and triggers an associated lambda function. The API gateway serves as an example of a common function trigger. However, lambda functions can also be triggered programmatically (e.g., via CLI or SDK), by other cloud events, such as file uploads (e.g., creation or modification of objects in S3), or various other trigger types.

Lambda functions implement the actual application logic, in our example application by loading the pre-trained ML model from S3, downloading the image from the user-provided URL from the internet, and then performing the inference computation within the lambda function. Such lambda functions commonly make use of cloud services for data storage (e.g., object storage S3, document database DynamoDB), logging (e.g., CloudWatch monitoring), analytics (e.g., Amazon EMR including Hadoop, Spark, HBase, and other big data frameworks), machine learning (e.g., natural language translation with AWS Translate) and many more purposes. The image download exemplifies other potential interactions with third-party services, such as REST APIs. Finally, our interactive example application returns the address and geographical coordinates (i.e., the predicted image labels) to the user through the HTTP API gateway as an HTTP response. In other non-interactive scenarios, lambda functions typically deliver their results to other cloud services, which might themselves trigger further actions or even other lambda functions as part of a workflow.

3. Research questions

In the context of studies on FaaS performance evaluation, our research questions address publication trends (RQ 1), benchmarked platforms (RQ 2), evaluated performance characteristics (RQ 3), used platform configurations (RQ 4), and reproducibility (RQ 5):

[3] https://github.com/awslabs/mxnet-lambda.

RQ 1 **Publication Trends:** What are the publication trends related to FaaS performance evaluations?

This question helps us understand how active research on FaaS performance evaluation has been and gives us insights on publication types and academic venues. This type of question is common to systematic mapping studies and has been studied in previous work for FaaS in general (Yussupov et al., 2019) and for other domains (Garousi et al., 2017).

RQ 2 **Benchmarked Platforms:** Which FaaS platforms are commonly benchmarked?

This question intends to identify FaaS platforms that are particularly well-understood or under-researched.

RQ 3 **Evaluated Performance Characteristics:** Which performance characteristics have been benchmarked for which FaaS platforms?

This question aims to characterize the landscape of existing work on FaaS performance to systematically map prior work and guide future research.

We divide the performance characteristics into the following sub-questions:

RQ 3.1 **Evaluated Benchmark Types:** Are experiments typically using micro- or application-level benchmarks?

RQ 3.2 **Evaluated Micro-Benchmarks:** Which micro-benchmarks (e.g., CPU or IO benchmarks) are commonly evaluated?

RQ 3.3 **Evaluated General Characteristics:** Which general performance characteristics (e.g., platform overhead/cold starts) are commonly evaluated?

RQ 4 **Used Platform Configurations:** Which platform configurations are commonly used?

This question targets the depth of the current understanding on FaaS performance. We want to examine whether many studies conduct similar experiments or explore diverse configurations.

We divide the platform configurations into the following three sub-questions:

RQ 4.1 **Used Language Runtimes:** Which language runtimes are commonly used?

RQ 4.2 **Used Function Triggers:** Which function trigger types are commonly used?

RQ 4.3 **Used External Services:** Which external services are commonly used?

RQ 5 **Reproducibility:** How reproducible are the reported experiments?

This question addresses an inherently important quality of experimental designs by assessing how well the FaaS community follows existing guidelines on reproducible experimentation in cloud systems (Papadopoulos et al., 2019).

4. Study design

This section describes the methodology of our Multivocal Literature Review (MLR) based on the guidelines from Garousi et al. (2019). We first summarize the overall process, then detail the strategies for search, selection, and data extraction and synthesis, followed by a discussion of threats to validity.

4.1. MLR process overview

The MLR process is divided into a part for academic and grey literature. We classify peer-reviewed papers (e.g., papers published in journals, conferences, workshops) as academic literature (i.e., white literature) and other studies (e.g., preprints of unpublished papers, student theses, blog posts) as grey literature.

The search process and source selection for academic literature follow a conventional systematic literature review (SLR) process (Kitchenham and Charters, 2007). Fig. 2 summarizes this multi-stage process originating from three different search sources and annotates the number of studies after each stage.

The process for grey literature studies is summarized in Fig. 3 with sources originating prevalently from web search. Notice that the number of relevant studies are already deduplicated, meaning that we found 25 relevant studies through Google search and the additional +8 studies from Twitter search only include new, non-duplicate studies. A key motivation for the inclusion of grey literature is the strong industrial interest in FaaS performance and the goal to identify potential mismatches between academic and industrial perspectives.

4.2. Search strategies

We first describe manual and database search for academic publications, then highlight the adjustments for web search, and finally discuss how alert-based search and snowballing complement the classic search strategies. For manual, database, and web search, we were able to perform exhaustive search by applying the selection strategy to all initial search results.

4.2.1. Manual search for academic literature

We use manual search to establish an initial seed of relevant sources to refine the database search query and to complement database search results with sources from third-party literature collections. We screen the following sources for potentially relevant studies:

(a) Studies from the preliminary results of an SLR targeting benchmarking of FaaS platforms (Kuhlenkamp and Werner, 2018): Their references from Table 1 are all relevant for our MLR but limited to 9 FaaS benchmarking studies, from which we removed one due to duplication (a journal extension covering more experiments than the initial extended abstract).

(b) Studies from the *Serverless Literature Dataset* (Spillner and Al-Ameen, 2019) (Version 0.4) listing 188 serverless-related papers published between 2016 and 2019: This extensive list of references covers many different serverless topics and therefore only a small subset of studies are potentially relevant for our MLR.

(c) Studies from a systematic mapping (SM) study on engineering FaaS platforms and tools (Yussupov et al., 2019): Their 62 selected publications focus on novel approaches and thus explicitly exclude benchmarking studies “without proposing any modifications” (Yussupov et al., 2019). We still identify a total of 10 relevant studies for our MLR in their categories related to benchmarking and performance and by screening their complete references.

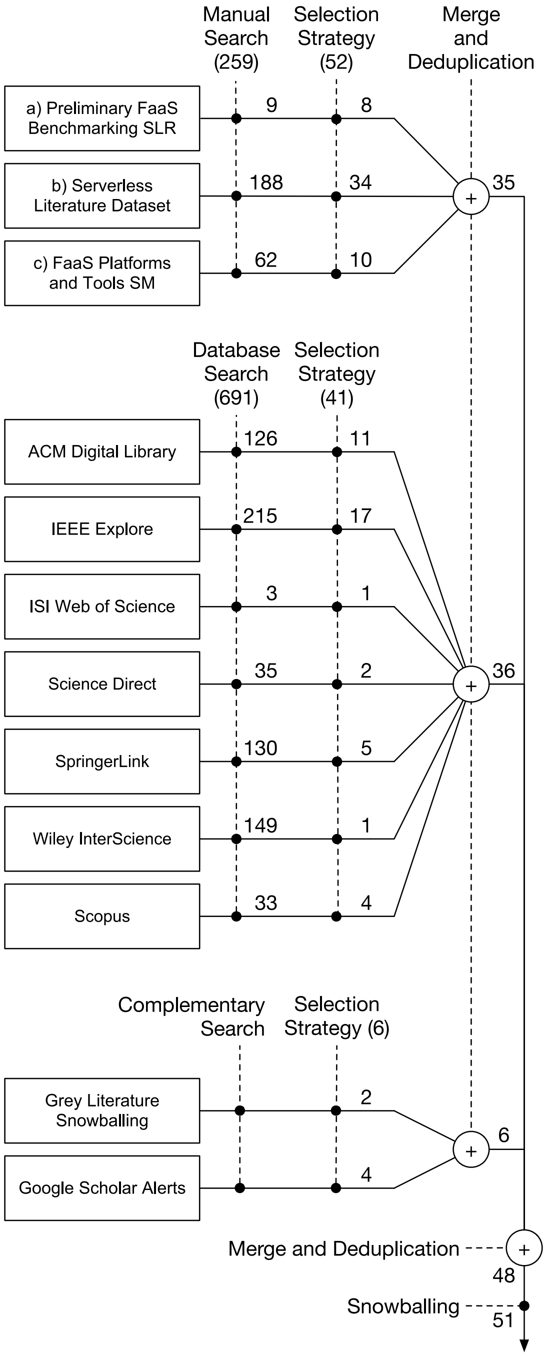

Fig. 2. Process for systematic literature review.

4.2.2. Database search for academic literature

Following standard SLR guidelines (Kitchenham and Charters, 2007), we define a search string to query common digital libraries for potentially relevant papers. We make use of logical OR operators to consider alternative terms given the lack of terminology standardization in the field of serverless computing. Within the area of serverless computing (i.e., `(serverless OR faas)`), our search string targets performance-related (i.e., `performance OR benchmark`) empirical (i.e., `experiment`) research. We refine the search string based on the insights from manual search, as suggested by Zhang et al. (2011), by adding the additional keyword `lambda` (appeared in all full texts) but omitting double quotes for exact matching. Our final search string is defined as follows:

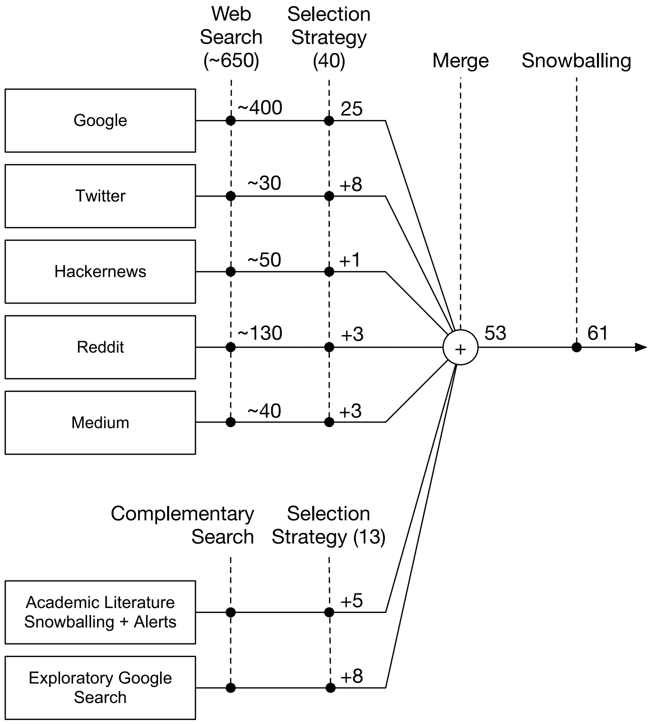

Fig. 3. Process for grey literature review.

`(serverless OR faas) AND (performance OR benchmark) AND experiment AND lambda`

We apply the search string to 7 common digital libraries, namely ACM Digital Library, IEEE Explore, ISI Web of Science, Science Direct, Springer Link, Wiley InterScience, and Scopus. The libraries are configured in their advanced query modes (if available) to search within full texts and metadata fields for maximal search coverage. The exact search query for each library can be found in the online appendix[4] including direct links and instructions for reproducing the search. The search was performed in October 2019 and all raw search results are exported into the bibtex format.

4.2.3. Web search for grey literature

For querying grey literature, we modified our original search string to account for less formal language in online articles. We replicate our academic search for one Google query but omit the terms *experiment* and *lambda* for all remaining queries using the following simplified search string: `(serverless OR faas) AND (performance OR benchmark)`

We apply the search string to 5 search engines, namely Google Search, Twitter Search, Hacker News Algolia Search, Reddit Search, and Medium Search. These engines (with the exception of Google Search) lack support for logical *OR* expressions. Therefore, we compose and combine four logically equivalent subqueries equivalent to the defined search string. Most searches were performed in Dec 2019 and, for replicability, we save the output of every search query as PDF and HTML files. Notice that the numbers of initial search results for web search are rough estimates due to the nature of web search engines. We refer to our replication package for technical details (Scheuner and Leitner, 2020).

[4] https://github.com/joe4dev/faas-performance-mlr.

4.2.4. Complementary search

Our previous search strategies often missed recent literature because manual search heavily relies on previous work and database search might suffer from outdated query indices or omit academic literature in press (i.e., accepted but not yet published). To discover recently published literature, we configured Google Scholar web-alerts[5] for the broad search term `serverless` and the more specific search term `serverless benchmark` over a period of 5 months (2019–10 till 2020–02) and screened hundreds of articles for potential relevance. Alert-based search discovered 6 relevant preprints (e.g., from arXiv.org) for which we explicitly checked whether they were accepted manuscripts (4 academic literature) or unpublished preprints (2 grey literature). Hence, complementary search for academic literature also contributed relevant studies to grey literature through alerts and snowballing, as well as vice versa. For grey literature, we spotted further relevant studies through exploratory Google search following looser adaptations of the search terms in particular contexts (e.g., related to a benchmarking tool).

4.2.5. Snowballing

After applying the selection criteria, we perform snowballing for academic and grey literature. For academic literature, we apply backward snowballing by screening their reference lists and forward snowballing by querying citations to relevant papers through Google Scholar. For grey literature, we prevalently apply backward snowballing by following outgoing links and occasionally (particularly for popular and highly relevant sources) apply forward snowballing by querying incoming for links through a backlink checker.[6]

4.3. Selection strategy

Following established SLR study guidelines (Kitchenham and Charters, 2007), we define the following inclusion (I) and exclusion (E) criteria for our study:

I1 Studies performed at least one performance-related experiment (i.e., excluding purely theoretical works, simulations, and works where a performance experiment was only mentioned as a sidenote) with a real FaaS environment as System-Under-Test (SUT). The FaaS environment can be fully managed or self-hosted.
I2 Studies presented empirical results of at least one performance metric.
I3 Studies published after Jan 1st 2015, as the first FaaS offering (AWS Lambda) was officially released for production use on April 9, 2015.[7]
E1 Studies written in any other language than English
E2 Secondary or tertiary studies (e.g., SLRs, surveys)
E3 Re-posted or republished content (e.g., sponsored re-post, conference paper with a journal extension)

As suggested by Wohlin et al. (2012), we only consider the most complete study as relevant primary study in cases of partial republication, for instance in the case of a journal extension of a conference paper. The two authors classified each potentially relevant study either as *relevant*, *uncertain* (with an indication whether rather relevant or not), or *not relevant*. All studies classified as *uncertain* were examined again and the rationale for the final decision was documented following the selection strategy presented above. If the title, keywords, and abstract were insufficient for obviously excluding a study, we read the full text of the study to take a final decision as practised for all included studies.

4.4. Data extraction and synthesis

Guided by the research questions, we extract the corresponding information based on a structured review sheet.

Publication trends (RQ 1). To capture how many studies of which type are published, we extract the following metadata: (i) the publication date (ii) the venue type for academic literature (i.e., journal, conference, workshop, doctoral symposium) and grey literature (i.e., preprint, thesis, blog post) (iii) the name of the venue (e.g., IEEE CLOUD, USENIX ATC), and a ranking of the venue (i.e., A*, A, B, C, W for workshop, unranked). The venue ranking follows the CORE ranking for conferences (CORE2018[8]) and journals (ERA2010[9]).

Benchmarked platforms (RQ 2). To assess which offerings are particularly well-understood or insufficiently researched, we extract the names of all FaaS platforms that are empirically investigated in a study.

Evaluated performance characteristics (RQ 3). To understand which performance characteristics have been benchmarked, we distinguish between micro- and application-benchmarks, collect a list of micro-benchmarks (e.g., CPU speed, network performance), and capture more general performance characteristics (e.g., use of concurrent execution, inspection of infrastructure). We start with an initial list of characteristics and iteratively add popular characteristics from an open *Others* field.

Used platform configurations (RQ 4). To describe which platform configurations have been evaluated, we extract the list of used language runtimes (RQ 4.1), function triggers (RQ 4.2), and external services (RQ 4.3). We generalize vendor-specific services to cross-platform terminology (e.g., AWS S3 was generalized to cloud storage).

Reproducibility (RQ 5). To review the potential regarding reproducibility, we follow existing guidelines on experimentation in cloud systems (Papadopoulos et al., 2019). The authors propose eight fundamental methodological principles on how to measure and report performance in the cloud and conduct an SLR to analyse the current practice concerning these principles covering top venues in the general field of cloud experimentation. As part of our work, we replicate their survey study in the more specific field of FaaS experimentation. We largely follow the same study protocol by classifying for each principle whether it is fully met (*yes*), partially present (*partial*) but not comprehensively following all criteria, or not present (*no*). Additionally, we collect some more fine-grained data for certain principles. For example, we distinguish between dataset availability and benchmark code availability for P4 (open access artefact) because we consider public datasets to be essential for replicating (statistical) analyses and public benchmark code is practically essential for reproducing the empirical experiment. For P3 (experimental setup description), we additionally capture whether a study describes the time of experiment (i.e., dates when the experiment was conducted), cloud provider region (i.e., location of data centre), and function size (i.e., used memory configurations).

4.5. Threats to validity

We discuss potential threats to validity and mitigation strategies for selection bias, data extraction and internal validity, replicability of the study, and external validity.

[5] https://scholar.google.com/intl/en/scholar/help.html#alerts.
[6] https://ahrefs.com/backlink-checker.
[7] https://docs.aws.amazon.com/lambda/latest/dg/lambda-releases.html.
[8] http://portal.core.edu.au/conf-ranks/.
[9] http://portal.core.edu.au/jnl-ranks/.

Selection bias. The representativeness of our selected studies is arguably one of the main threats to this study. We used a multi-stage process (see Section 4.1) with sources originating from different search strategies. Initial manual search based on existing academic literature collections allowed us to fine-tune the query string for database searches against 7 well-established electronic research databases. We optimize our search string for more informal grey literature and query 5 search engines specializing in general-purpose search, social search, and developer-community search. Additionally, our complementary search strategies aim to discover studies that were recently published, found in the other context (i.e., academic vs. grey), or spotted through more exploratory search (e.g., a looser adaptation of search terms).

Data extraction and internal validity. Tedious manual data extraction could potentially lead to inaccuracies in the extracted data. To mitigate this threat, we define our MLR process based on well-established guidelines for SLR (Kitchenham and Charters, 2007) and MLR (Garousi et al., 2019) studies, methodologically related publications (Garousi et al., 2017), and topically relevant publications (Yussupov et al., 2019; Kuhlenkamp and Werner, 2018). Further, we set up a structured review sheet with practical classification guidelines and further documentation, which was incrementally refined (e.g., with advice on classifying borderline cases). We implemented traceability through over 700 additional comments, at least for all borderline cases. The data extraction process was conducted by both authors, with the first author as the main data extractor and the second author focusing on discussing and verifying borderline cases. We also repeatedly went over all sources to verify certain data (e.g., based on refined classification scheme) and collect more details (e.g., individual aspects of more vague P3 on experimental setup description). For the reproducibility part (see RQ 5), we refer to the statistical evaluation on inter-reviewer agreement in the original study (Papadopoulos et al., 2019), which achieved very high agreement.

Replicability of the study. We publish a replication package (Scheuner and Leitner, 2020) to foster verification and replication of our MLR study. Our package includes all search queries with direct links and step-by-step instructions on how to replicate the exact same queries, query results in machine-readable (BibTeX/HTML) and human-readable (PDF) formats, a structured review sheet containing all extracted data and over 700 comments with guidance, decision rationales, and extra information, and code to reproduce all figures in our study. The latest version of the replication package and further documentation is also available online.[10]

External validity. Our study is designed to systematically cover the field of FaaS performance benchmarking for peer-reviewed academic white literature and unpublished grey literature including preprints, theses, and articles on the internet. However, we cannot claim generalizability to all academic or white literature as we might have missed some studies with our search strategies. The inclusion of grey literature aims to address an industrial perspective but is limited to published and indexed content freely available and discoverable on the internet (e.g., excluding paywall articles or internal corporate feasibility studies).

5. Study results and discussion

This section presents and discusses the main outcomes of our MLR study guided by our research questions stated in Section 3. The results are based on the extracted and synthesized (according to Section 4.4) survey data from 112 selected (according to Section 4.3) primary studies including 51 academic publications and 61 grey literature sources. For each research question, we briefly describe context, motivation, and methodology, followed by relevant results and their subsequent discussion.

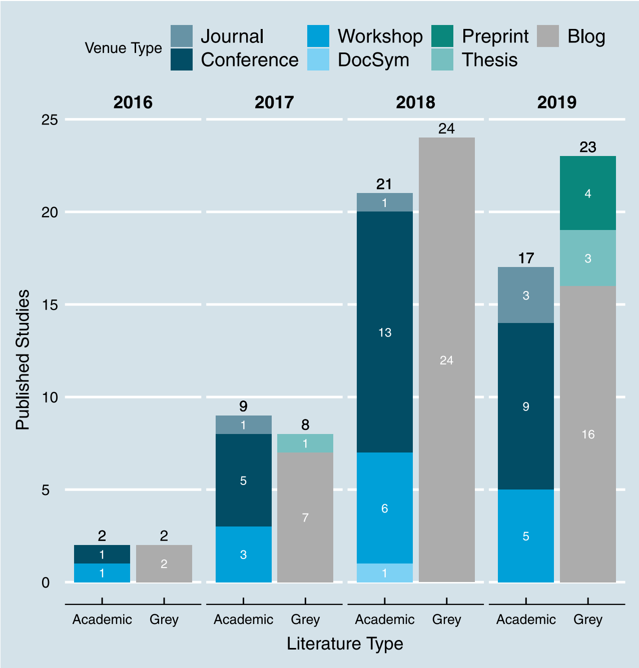

Fig. 4. Distribution of academic (N = 51) and grey (N = 61) literature studies over years and venue type.

5.1. Publication trends (RQ 1)

Description. We describe the publication trends on FaaS performance evaluations by summarizing the publication statistics over years and venue types, the venue rankings for academic literature, and the most popular publication venues. The venue ranking follows the CORE ranking for conferences (CORE2018) and journals (ERA2010).

Results. Fig. 4 shows the distribution of published studies for academic and grey literature over years and venue types. We observe a growing interest for both types of literature, with early studies appearing in mid 2016 [A15, A35], followed by a drastic increase in 2017, and a surge of new studies in 2018. The year 2019 indicates a minor decrease in overall publication activity but covers more diverse publication types. Notice that the initial searches were performed in October 2019 for academic literature and December 2019 for grey literature and therefore cover 2019 only partially, also considering the indexing delay. Further, Fig. 4 omits 3 blog posts with unspecified publication dates and also the just-started year 2020 consisting of 2 academic conference publications and 1 grey literature preprint.

Fig. 5 summarizes the venue ranking across all academic studies. A good share of studies is published in top-ranked venues, surprisingly few studies appear in C-ranked venues, and the majority of studies are published in workshops or other unranked venues.

Table 1 lists the 10 most popular publication venues where at least two studies have been published. The most popular venue with 7 publications is the International Workshop on Serverless Computing (WoSC), founded in 2017 and held two times per year in 2017 and 2018.

[10] https://github.com/joe4dev/faas-performance-mlr.

Table 1
List of 10 most popular publication venues.

N	Acronym	Venue
7	WoSC	International Workshops on Serverless Computing
5	ATC	USENIX Annual Technical Conference
3	CLOUD	IEEE International Conference on Cloud Computing
2	SAC	ACM/SIGAPP Symposium On Applied Computing
2	FGCS	Future Generation Computer Systems
2	ICSOC	International Conference on Service Oriented Computing
2	BigData	IEEE International Conference on Big Data
2	IC2E	IEEE International Conference on Cloud Engineering
2	HotCloud	USENIX Workshop on Hot Topics in Cloud Computing
2	NSDI	USENIX Symposium on Networked Systems Design and Implementation

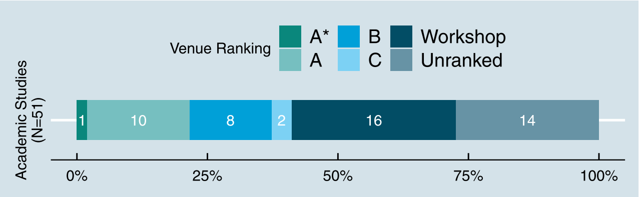

Fig. 5. Academic venue rankings (N = 51).

Discussion. Our results are generally in line with the related systematic mapping study from Yussupov et al. (2019). However, we see a stronger emphasis on workshop publications, which appears plausible for a more narrow topic of investigation. Additionally, our work indicates that grey literature follows a similar but possibly more pronounced hype trend with blog posts spiking in 2018 and declining stronger in 2019 than cumulative academic literature.

Related to academic venue rankings, we interpret the relative over-representation of top-ranked publications (in comparison to relatively few full papers in C-ranked venues) as a positive sign for this young field of research. The strong representation of workshop papers, particularly at WoSC, is plausible for a relatively narrow topic in a young line of research.

5.2. Benchmarked platforms (RQ2)

Description. The two main types of FaaS platforms are hosted platforms and platforms intended to be installed in a private cloud. Hosted platforms are fully managed by a cloud provider and often referred to as FaaS providers. All major public cloud providers offer FaaS platforms, including AWS Lambda, Microsoft Azure Functions, Google Cloud Functions, and IBM Cloud Functions. Installable platforms are provided as open source software and can be self-hosted in on-premise deployments. Prominent open source platforms include Apache OpenWhisk, Fission, Knative, or OpenFaaS. Self-hosting requires extra setup, configuration, and maintenance efforts, but allows for full control and inspection during experimentation. Dozens more hosted services[11] and many more FaaS development frameworks[12] and installable platforms have emerged in this fast-growing market.

Results. The first row of the bubbleplot in Fig. 6a summarizes the total number of performance evaluation experiments in absolute frequency counts for the 5 most popular hosted FaaS platforms in our study. Self-hosted platforms are only depicted in aggregation due to their low prevalence in literature. The *x*-axis is ordered by cumulative platform frequency, where AWS Lambda leads with a total of 99 studies divided into 45 academic and 54 grey literature studies. Thus, 88% of all our selected studies perform experiments on AWS Lambda, followed by Azure (26%), Google (23%), self-hosted platforms (14%), IBM (13%), and CloudFlare (4%). For hosted platforms, we omit Lambda@Edge[13] (3) and Binaris[14] (1) because Lambda@Edge is covered in the same experiments as CloudFlare and Binaris only occurred once. Within self-hosted platforms, academic literature mostly focuses on OpenWhisk (70%), whereas grey literature covers other platforms, such as Fission, Fn, or OpenFaaS.

Discussion. In comparison to other surveys, our overall results for percentage by provider closely (±5%) match the self-reported experience per cloud provider in a 2018 FaaS survey (N = 182) (Leitner et al., 2019). Our results are also reasonably close (±5% except for AWS +13%) to self-reported use in organizations in a 2019 O'Reilly survey on serverless architecture adoption (N > 1500).[15] Initial results of the latest 2020 survey (N > 120 for the first day)[16] indicate similar results for FaaS products currently in use, with a very close match for AWS (1% deviation). However, this survey shows even lower numbers (up to −12%) for other providers.

Hence, our results show that AWS is currently overstudied in absolute numbers (by a factor of 3x). However, the strong emphasis on AWS appears to be justified in relative numbers given the industrial importance of AWS in this domain.

We observe that the attention by literature type is appropriately balanced (less than ±10%) for most platforms, except for proportionally higher academic coverage by IBM (+12%) and self-hosted platforms (+17%). IBM appears to be over-represented in academic studies, potentially motivated by the ability to compare a provider-hosted platform with its underlying open source platform Apache OpenWhisk in a self-hosted setup (e.g., A1, A3). In contrast to hosted platforms, self-hosted platforms allow for full control of the experimental setup (e.g., A28) and detailed performance and resource profiling (e.g., Shahrad et al., 2019) but raise other challenges regarding fair comparability and configuration.

5.3. Evaluated performance characteristics (RQ3)

To answer RQ3, the facetted bubbleplot in Fig. 6 combines performance characteristics for (a) benchmark types (b) microbenchmarks, and (c) general characteristics across FaaS platforms. All these plots can be interpreted as a heatmap ranging from few studies in the bottom-left corner to many studies in the top-right corner for a given characteristic-platform combination. We provide relative percentages against the total number per literature type (i.e., $N_{academic} = 51$ vs. $N_{grey} = 61$) because the absolute numbers are not directly comparable.

[11] https://landscape.cncf.io/format=serverless.

[12] https://github.com/anaibol/awesome-serverless#frameworks.

[13] https://aws.amazon.com/lambda/edge/.

[14] Binaris (https://binaris.com/) was renamed to reshuffle in Oct 2019.

[15] Figure 12 in https://www.oreilly.com/radar/oreilly-serverless-survey-2019-concerns-what-works-and-what-to-expect/.

[16] Question 14 in https://www.nuweba.com/blog/serverless-community-survey-2020-results.

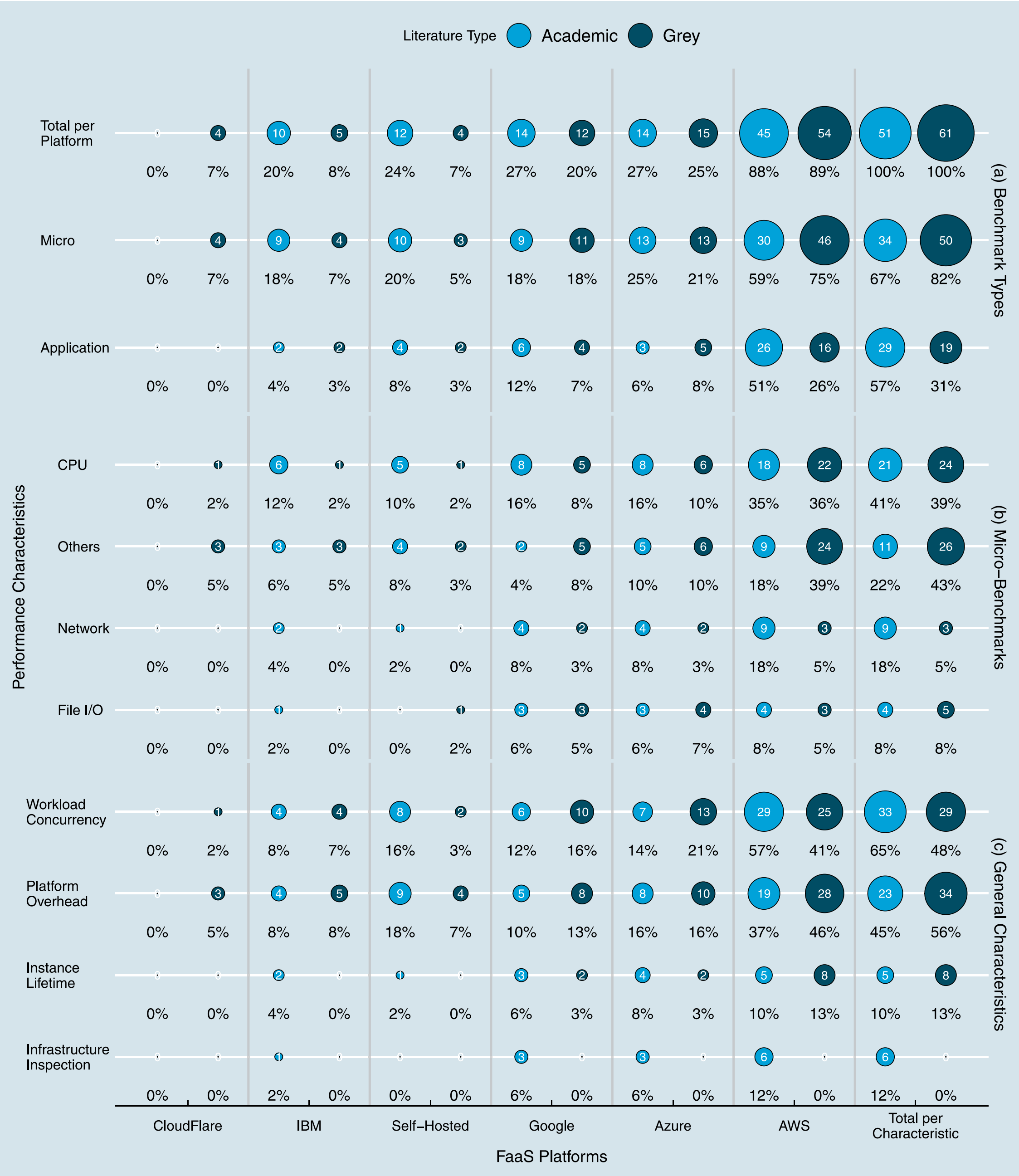

Fig. 6. Evaluated performance characteristics per FaaS platform.

5.3.1. Evaluated benchmark types (RQ 3.1)

Description. We distinguish between narrow micro- and holistic application-benchmarks as introduced in Section 2. Fig. 7 summarizes our FaaS benchmark taxonomy including the most common micro-benchmarks derived through open coding as described in Section 4.4 and reflected in Fig. 6.

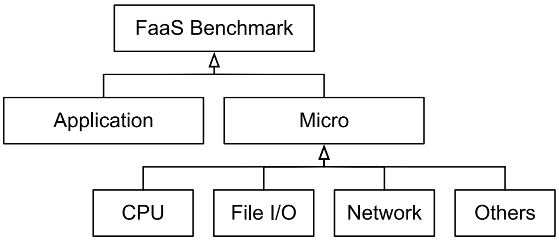

Fig. 7. FaaS benchmark taxonomy.

Results. Fig. 6a summarizes which high-level types of benchmarks are used across which FaaS platforms. The rightmost *Total per Characteristic* column indicates that micro-benchmarks are the most common benchmark type, used by 75% of all selected studies (84/112). Interestingly, we observe a particularly strong emphasis on micro-benchmarks in grey literature (50 studies, or 82%). However, also two-thirds of the selected academic literature conduct studies using micro-benchmarks. Application-level benchmarks are used by 48 (43%) of all selected studies and are more prevalent among academic literature with 29 (57%) studies compared to grey literature with only 19 (31%) studies. Further, 12 (24%) academic studies combine micro-benchmarks and application-benchmarks, which can be derived by the difference (i.e., 12) between the total number of academic literature studies

(51) and the sum of total micro-benchmarks and application-benchmarks (34 + 29 = 63). For grey literature, only 8 (13%) studies combine the two benchmark types and thus the vast majority of studies (87%) uses micro- or application-benchmarks in isolation. Finally, micro-benchmarks are more commonly used across different providers, whereas application-benchmarks are prevalently (>84%) used for benchmarking AWS.

Discussion. Micro-benchmarks are the most common benchmark type across both literature types but academic literature uses application-benchmarks more frequently (+26%) than grey literature. We expected application-benchmarks to be under-represented but were surprised to see relatively many academic studies using application-benchmarks. Closer investigation revealed that many of these academic studies demonstrate or evaluate their proposed prototypes on a single FaaS platform (e.g., "MapReduce on AWS Lambda" [A12]) focusing on thorough insights and leaving cross-platform comparisons for future work. While we agree that such studies on a single FaaS platform can be great demonstrators of ideas and capabilities, the general usefulness of application-benchmarks evaluated in isolation on a single platform is limited, as the inability to relate results from such work against any baseline or other reference platform makes it hard to derive meaningful conclusions. This threat is particularly relevant for hosted platforms, as the performance observed from such experiments depends strongly on the underlying hard- and software infrastructure. Therefore, we argue that reproducibility (see RQ 5) is particularly important for this type of study.

Some studies clearly intend to conduct end-to-end (i.e., application-level) measurements, however, their applications and workloads are insufficiently described such that it is unclear what exactly they do. This unclarity is reinforced by the tendency of application-benchmarks to remain closed source, with only 35% of the studies publishing at least partial benchmark code compared to 50% overall.

5.3.2. *Evaluated micro-benchmarks (RQ 3.2)*

Description. We cover micro-benchmarks targeting CPU, file I/O, and network performance. The *Others* category summarizes other types of micro-benchmarks such as cold start evaluations (i.e., platform overhead). Notice that we grouped platform overhead as general performance characteristics because some studies alternatively use application-benchmarks with detailed tracing.

Results. Fig. 6b summarizes which micro-benchmark performance characteristics are used across which FaaS platforms. The right-most *Total per Characteristic* column shows that CPU is by far the most evaluated micro-benchmark characteristic, used by 40% of all studies. Network and file I/O performance are less common for academic literature studies, and even more rare in grey literature. These two less common characteristics are all evaluated on the AWS platform (except for two file-I/O grey literature studies) but practically uncovered on self-hosted platforms (only two studies overall). The *Others* category mainly consists of platform overhead and workload concurrency evaluated through micro-benchmarks. While many studies evaluate different memory configurations or monitor memory usage, we have not seen studies evaluating memory performance (e.g., bandwidth, latency) itself.

Discussion. Academic literature tends to focus more on traditional performance characteristics, such as CPU, network, file I/O, in contrast to grey literature focusing more on other FaaS-specific characteristics, such as cold starts, concurrency, and trigger comparisons.

Our results suggest that CPU performance is an overstudied performance characteristic among FaaS micro-benchmarks. Many studies confirm that CPU processing speed scales proportionally to the amount of allocated memory (i.e., configured function size) for AWS [A8, A49, A3, G10, G43] and Google [A8, A49, A3, G10]. This empirically validated behaviour is also explicitly stated in the documentation of the providers. For instance, the AWS Lambda documentation states that *"Lambda allocates CPU power linearly in proportion to the amount of memory configured".*[17] The Google Cloud Functions documentation also used to mention proportional scaling explicitly. A few of these studies [A8, A3, G2] also cover Azure and IBM and conclude that these platforms assign the same computational power for all functions. Notice that Azure does not expose an explicit memory size configuration option as common for the other providers, but rather determines available memory sizes based on a customer's service subscription plan.[18]

5.3.3. *Evaluated general characteristics (RQ 3.3)*

Description. We cover four general performance characteristics, namely platform overhead, workload concurrency, instance lifetime, and infrastructure inspection. These general characteristics are orthogonal to previously discussed characteristics, and can be measured using either micro- or application-level benchmarks. Platform overhead (e.g., provisioning of new function instances) mainly focuses on startup latency and in particular on quantifying the latency of cold starts. Workload concurrency refers to workloads that issue parallel requests, or to benchmarks evaluating platform elasticity or scaling behaviour (e.g., [A26, A36]). Instance lifetime or infrastructure retention attempts to re-engineer the provider policy on how long function instances are kept alive until they get recycled and trigger a cold start upon a new function invocation. Infrastructure inspection aims to re-engineer underlying hardware characteristics (e.g., CPU model to detect hardware heterogeneity) or instance placement policy (e.g., instance identifier and IP address to detect co-residency on the same VM/container [A49]).

Results. Fig. 6c summarizes which general performance characteristics are benchmarked across which FaaS platforms. Workload concurrency is a commonly studied characteristic, but more so in academic literature (65%) than in grey literature (48%). On the other hand, grey literature seems to focus more on platform overhead (56%) than academic literature (45%). Infrastructure inspection is exclusively analysed in academic literature studies (12%). Note that this line of enquiry does not make sense for self-hosted platforms, and hence is not studied in this context. Finally, the data from the *Others* column shows that there is currently a lack of cross-platform comparisons of function triggers.

Discussion. General performance characteristics focus on particularly relevant aspects of FaaS and only a few studies aim towards reverse-engineering hosted platforms. Elasticity and automatic scalability have been identified as the most significant advantage of using FaaS in a previous survey (Leitner et al., 2019), which justifies the widespread evaluation of concurrency behaviour. Given the importance of this characteristic, we argue that concurrent workloads should be an inherent part of all FaaS performance evaluations going forward (going beyond the 50% of studies observed in our corpus). Container start-up latency has been identified as one of the major challenges for using FaaS services in prior work (Leitner et al., 2019), receives comparably even higher attention from grey literature (+11%), and thus motivates a large body of work related to quantifying platform overheads.

In prior IaaS cloud performance evaluation research, reverse-engineering cloud providers was a common theme and lead to

[17] https://docs.aws.amazon.com/lambda/latest/dg/configuration-console.html.

[18] https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale#service-limits.

exploitation approaches for hardware heterogeneity (Farley et al., 2012; Ou et al., 2012). However, as hardware heterogeneity became less relevant over time (Scheuner and Leitner, 2018), we refrain from interpreting the current lack of infrastructure inspection studies as a research gap that requires more attention. The lack of studies from grey literature might also hint that this characteristic is currently of less interest to practitioners.

5.4. Used platform configurations (RQ 4)

To answer RQ 4, we present a facetted barplot (Fig. 8), visualizing the share of studies using a given configuration. We report the share as percentage against all academic and all grey literature studies.

5.4.1. Used language runtimes (RQ 4.1)

Description. The language runtime is the execution environment of a FaaS function. Fully managed platforms offer a list of specific runtimes (e.g., Node.js, Python) determining the operating system, programming language, and software libraries. Some providers support the definition of custom runtimes by following a documented interface, often in the form of Docker images. If customization is not available in a platform, shims can be used to invoke an embedded target language runtime through a support runtime via system calls (e.g., invoking binaries through Node.js).

Results. Fig. 8a shows how frequently different language runtimes are evaluated. Overall, Python and Node.js are the most popular runtimes, followed by Java. Interestingly, Node.js and Java are twice as popular among grey literature compared to academic literature. Grey literature generally covers more (up to 7), and more diverse languages in comparison to academic literature. In particular, 46% of the grey literature studies use more than one language in comparison to only 16% for academic studies. The category of *Others* includes a list of 13 languages (e.g., F#, Scala, Haskell) evaluated through custom runtimes or shims.

Discussion. The large differences between academic and grey literature indicate a potential mismatch of academic and industrial interests. This assumption is supported by other studies reporting the use of FaaS languages that similarly conclude Node.js to be roughly 20% more popular than Python[19] (Leitner et al., 2019).

5.4.2. Used function triggers (RQ 4.2)

Description. Function triggers cover alternative ways of invoking FaaS functions. FaaS functions can be triggered explicitly (e.g., through code) or implicitly through events happening in other services (e.g., image uploaded to cloud storage triggers function). HTTP triggers invoke functions on incoming HTTP requests. SDK and CLI triggers use software libraries to explicitly invoke functions. Storage and database triggers invoke functions on object modifications (e.g., creation of a new object). Stream, publish/subscribe (pub/sub), and queues process messages and invoke functions according to certain order or batching strategies. Timer triggers invoke functions on specified schedules (e.g., cron expressions). Workflow engines implement some kind of state machine to orchestrate complex processes across multiple functions.

Results. Fig. 8b shows how frequently different types of function triggers are evaluated. HTTP triggers are by far the most commonly evaluated type of trigger and are used by 57% of all studies. Invocation through storage triggers is surprisingly uncommon for grey literature (10%). In general, only two studies cover more than two trigger types [A41, A27], with the vast majority focusing on a single type of trigger.

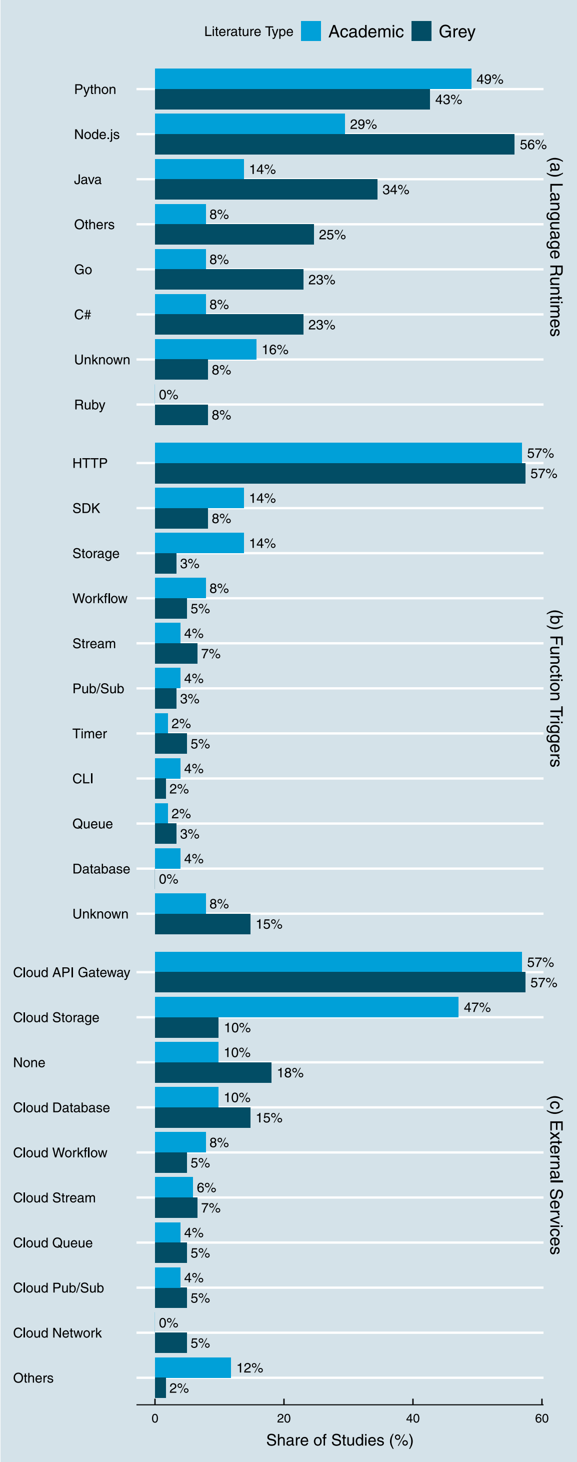

Fig. 8. Used platform configurations for 51 academic and 61 grey literature studies.

Discussion. It appears function triggering has received little attention given that most studies go for the de-facto default option of exposing a function via HTTP. There are a wide range of other ways to trigger function execution (e.g., through a message queue, data streaming service, scheduled timer, database event, an SDK, etc.), which are currently not widely used and evaluated. The overall strong focus on HTTP triggers makes it hard to derive any meaningful differences between academic and grey literature,

[19] Question 25 in https://www.nuweba.com/blog/serverless-community-survey-2020-results.

except for a stronger academic focus on storage (+11%) and database triggers (4% vs. 0%).

5.4.3. Used external services (RQ 4.3)

Description. We now discuss which external services are commonly used in FaaS performance evaluations. Cloud API gateways offer a fully managed HTTP service, which is commonly used to trigger functions upon incoming HTTP requests. Cloud storages offer object storage for blob data, such as images. Cloud databases offer structured data storage and querying. Cloud workflow engines manage the state of complex processes across multiple functions. Cloud stream, cloud queue, and cloud pub/sub are different types of message processing services. Cloud networks refer to configurable private network services, such as AWS Virtual Private Cloud (VPC).

Results. Fig. 8c shows how frequently different external services are used. Cloud API gateways are the most commonly used external service, which is unsurprising given that most studies use HTTP events to trigger functions. About half of the academic literature studies use cloud storage compared to only 10% of grey literature studies. Overall, database services are among the most popular integrations. The *Others* category includes caching services, self-hosted databases, and special services such as artificial intelligence APIs. In general, given how central service ecosystems are to the value proposition of cloud functions, it is surprising how rarely FaaS benchmarking studies incorporate external services beyond API gateways.

Discussion. The result from function triggers explains the strong emphasis on cloud API gateway services for both, academic and grey literature. Most surprisingly, cloud storage receives very little attention in grey literature (−37%) compared to academic literature. This is in contrast to other studies, indicating that database services are more commonly used in conjunction with FaaS[20] (Leitner et al., 2019). A possible explanation lies in the strong focus of grey literature on micro-benchmarks, which typically use no external services or only an API gateway for easy invocation. We conclude that the integration of external services in FaaS performance evaluations in a meaningful way remains a gap in current literature.

5.5. Reproducibility (RQ 5)

Description. To evaluate the maturity of literature concerning reproducibility, we rely on recent work by Papadopoulos et al. (2019). They propose eight methodological principles for reproducible performance evaluation in cloud computing, which we now summarize and apply to our corpus:

- P1 *Repeated Experiments:* Repeat the experiment with the same configuration and quantify the confidence in the final result.
- P2 *Workload and Configuration Coverage:* Conduct experiments with different (preferably randomized) workloads and configurations motivated by real-world scenarios.
- P3 *Experimental Setup Description:* For each experiment, describe the hardware and software setup, all relevant configuration and environmental parameters, and its objective.
- P4 *Open Access Artefact:* Publish technical artefacts related to the experiment including software (e.g., benchmark and analysis code) and datasets (e.g., raw and cleaned data).

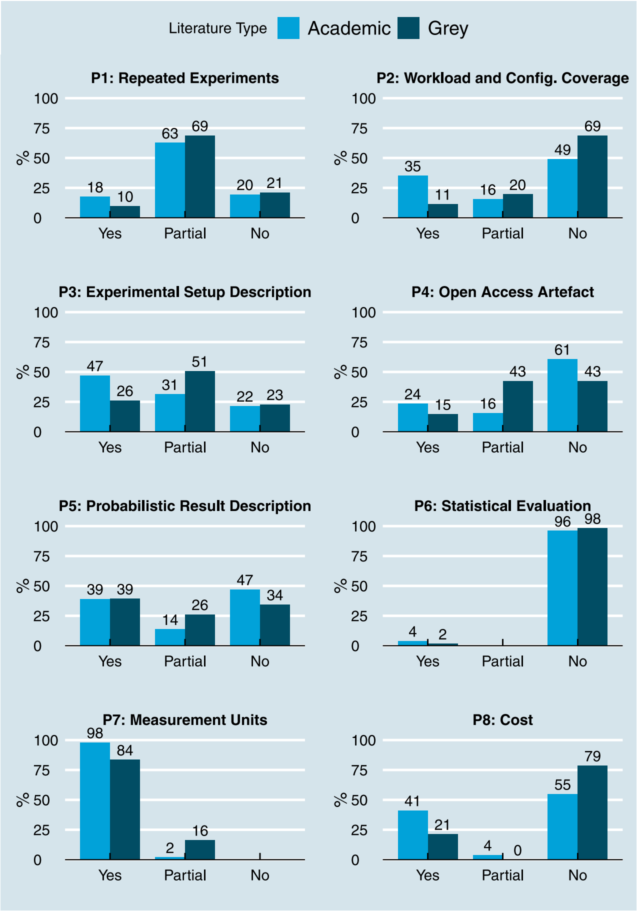

Fig. 9. Evaluation of reproducibility principles P1–P8 (Papadopoulos et al., 2019) for 51 academic and 61 grey literature studies.

- P5 *Probabilistic Result Description:* Describe and visualize the empirical distribution of the measured performance appropriately (e.g., using violin or CDF plots for complex distributions), including suitable aggregations (e.g., median, 95th percentile) and measures of dispersion (e.g., coefficient of variation also known as relative standard deviation).
- P6 *Statistical Evaluation:* Use appropriate statistical tests (e.g., Wilcoxon rank-sum) to evaluate the significance of the obtained results.
- P7 *Measurement Units:* For all the reported measurements also report the corresponding unit.
- P8 *Cost:* Report the calculated (i.e., according to cost model) and charged (i.e., based on accounted resource usage) costs for every experiment.

Results. Fig. 9 shows to what extent the reproducibility principles from Papadopoulos et al. (2019) are followed by our selected academic and grey literature. Overall, we find that 7 out of 8 principles are not followed by the majority of studies and, although academic literature performs better (>20%) than grey literature for 3 principles (i.e., P2, P3, P8), we do not see a clear trend that academic work follows the proposed principles more strictly. Interestingly, grey literature is even better than academic literature with regards to providing open access (P4) and probabilistic result descriptions (P5).

P1: Repeated experiments. The first subplot shows that the majority (≈65%) of our selected studies only partially fulfils P1 by performing some kind of repetition in their experiments, but

[20] Question 18 in https://www.nuweba.com/blog/serverless-community-survey-2020-results.

without justifying or reporting confidence values. The results for academic and grey literature are comparable. While most of the studies perform some kind of repetition with the same configuration, the actual repetition count or duration time varies wildly. Results range from hundreds of repetitions or minutes of experimentation to thousands of repetitions or hours of experimentation collected over the course of multiple days. Most of the studies partially fulfilling P1 do not discuss the confidence of their results. However, there are also a few individual cases that use appropriate probabilistic result descriptions (e.g., CDFs), but omit the absolute number of repetitions. Around 21% of the studies perform insufficient repetitions (e.g., only tens of individual data points) or do not provide any information about whether or how their experiments use repetitions. Only 10%–15% of the studies discuss the confidence of their results and collect a large-scale dataset with up to 360 000 entries per histogram [A8] or up to 1 000 000 executions across 5 regions collected over 4 months [G9]. The following three studies further address the challenge of outdated benchmark results by offering a web service that continuously publishes updated results: The λ Serverless Benchmark[21] by Strehl [G11], FaaSTest[22] by the Nuweba company [G57], and the cloud functions benchmarking dashboard[23] by Figiela et al. [A8].

P2: Workload and configuration coverage. About 50% of the academic and 70% of the grey literature studies do not use different workloads and configurations motivated by real-world scenarios. These studies mostly use a single trivial example application, such as some version of a Hello World FaaS function[24] that returns a constant string, a timestamp, or a random value. The second-most commonly used workload is some kind of CPU-intensive calculation, such as the prevalent prime number calculations with the iterative Sieve of Eratosthenes algorithm. The partial category (16%–20%) mainly consists of studies using only microbenchmarks or a single application-benchmark in only one configuration. Application-level workloads motivated by real-world use cases and covering multiple configurations are used by 35% of the academic but only 11% of the grey literature studies. One academic short paper by Kim and Lee [A21] specifically aims at introducing a suite of micro- and application-level serverless function benchmarks.

P3: Experimental setup description. More than half of all studies insufficiently describe their experimental setup. However, as can be expected academic literature describes their experiments more comprehensively than grey literature. About 31% of the academic and 51% of the grey literature studies omit important details in their experiment description and another 22% of all studies exhibit severe flaws in their study description (e.g., significant details are missing), thus not fulfilling P3. With only minor omissions (e.g., time of experiment not provided), 47% of academic but only 26% of grey literature studies satisfy P3. The clear lead of academic literature highlights our overall impression that academic literature tends to describe their experimental setup in a more structured manner, often in dedicated sections (e.g., Kuhlenkamp et al. [A26]). Further, academic studies tend to define their experiment goals more formally based on testable hypotheses (e.g., Figiela et al. [A8] or Manner et al. [A34]).

P4: Open access artefact. Technical artefacts are unavailable for 61% of the academic and 43% of the grey literature studies. Grey literature more commonly publishes their benchmark code (43% vs. 16%) but more academic studies provide complete open source access to benchmark code and collected datasets (24% vs. 15%). The partial fulfilment category has only two exceptions of grey literature studies solely publishing their dataset but not their benchmark code instead of vice versa.

We discovered one of the following three practical issues related to handling open access artefacts in 9% of all studies. Firstly, we found inaccessible links in 3 studies that claim their artefacts are open source. Secondly, we noticed obviously incomplete implementations (e.g., only for one provider, isolated legacy code snippet, code within inaccessible pictures) in another 3 studies. Thirdly, we discovered open source artefacts that were not explicitly linked in 4 studies but could be discovered via manual Google or Github search or were implicitly linked in user comments (e.g., upon request of commenters).

The violation of P4 is particularly severe in combination with insufficient experimental setup description (P3). A total of 19 (17%) studies neither provide any technical artefacts nor any proper experimental setup description, rendering these studies practically impossible to replicate in practice. Another 20 (18%) studies violate P4 and omit relevant details in their experimental setup description. Thus, these studies are hard to replicate under similar conditions (but a “similar” experiment could be conducted).

P5: Probabilistic result description. About 40% of all studies appropriately visualize or characterize their empirical performance data, but roughly the same percentage of all studies ignore complex distributions and primarily focus on reporting averages. These nearly 40% of the studies fulfilling P5 commonly use CDFs, histograms, or boxplots complemented with additional percentiles. The 15% of academic and 26% of grey literature studies partially fulfilling P5 often give some selective characterization of the empirical distribution by plotting (raw) data over time or by violating P1 (i.e., insufficient repetitions).

P6: Statistical evaluation. Almost none of the selected studies perform any statistical evaluations. Only two academic papers and one preprint use statistical tools such as the Spearman correlation coefficient [A34] or a nonparametric Mann–Whitney U test [G49].

P7: Measurement units. Overall, P7 is followed almost perfectly with no major violations. Grey literature occasionally (16%) omits measurement units (most commonly in figures) but the missing unit can be derived relatively easy from the context (or is mentioned in the text).

P8: Cost. Cost models are missing in 55% of the academic and 79% of grey literature. Two academic studies fulfil P8 partially by discussing costs in a general sense (e.g., as a motivational example), but without discussing actual costs of the experiments. While some studies that specifically focus on costs (e.g., Kuhlenkamp and Klems [A24]), most studies typically calculate costs based on accounted or self-measured resource usage (e.g., runtime), but omit the actually charged cost.

Discussion. We expected peer-reviewed academic literature to consistently achieve more methodological rigour than largely individually-authored grey literature. Surprisingly, we do not see a clear trend that academic literature disregards the principles less often than grey literature. It is concerning that even simple principles such as publishing technical artefacts are frequently neglected, and grey literature is even better in providing at least partial open access. Methodologically long-known principles from academic literature are still commonly overlooked in academia,

[21] https://serverless-benchmark.com/.

[22] https://www.faastest.com/.

[23] http://cloud-functions.icsr.agh.edu.pl/.

[24] https://aws.amazon.com/getting-started/tutorials/run-serverless-code/.

exemplified by statistical guidance from 1986 on avoiding misleading arithmetic mean values (Fleming and Wallace, 1986). The presumably "more informal" grey literature is often on par or even better in appropriately describing performance results.

On the other hand, we emphasize that the clear lead of academic literature for three principles (i.e., P2, P3, P8) goes beyond the expressiveness of a 3-point discrete scale (i.e., yes, partial, no). Experimental setup description (P3) has many facets and our results prevalently cover the presence or absence of relevant conditions, but fail to appropriately account for other important facets, such as clear structure and presentation. Grey literature includes examples of unstructured studies, where results are presented without any discussion of methodology and scarce details about the experiment setup are scattered throughout a blog post. In terms of P2, grey literature frequently picks one of the easiest available workloads, whereas academic studies more often motivate their workloads and attempt to link them to real-world applications.

We found that although many studies seemingly evaluate similar performance characteristics, comparing actual performance results is very difficult due to large parameter space, continuously changing environments, and insufficient experimental setup descriptions (P3). We collected some exemplary results for the hosted AWS platform and find dramatic differences in numbers reported for platform overhead/cold starts ranging from 2 ms (80th percentile, Python, 512 mb but presumably reporting something else, maybe warm-start execution runtime of an empty function) [G5] up to 5 s (median, Clojure via Java JVM, 256 mb) [G54]. More common results for end-to-end (i.e., including network latency of typically pre-warmed HTTPS connection) cold start overhead (i.e., excluding actual function runtime) for the Nodejs runtime on AWS (according to live data from 2020–02) are in the orders of ≈50 ms (median) to ≈100 ms (90th percentile) [A8,G11]. Studies from 2019 tend to report slightly higher numbers mostly around 200–300 ms (median) [G11,G33,G3].

In the following, we highlight some insights into practical reproducibility related to P3 and P4. We strongly agree with Papadopoulos et al. (2019) that preserving and publishing experiment artefacts (P4) may be the only way to achieve practical reproducibility given that an exhaustive description (P3) of a complex experiment is often unrealistic. We further argue that at least any time-consuming repetitive manual steps (but preferably any error-prone manual setup step that could lead to potential misconfiguration and affect the outcome of a study) should be fully automated (Scheuner et al., 2014). We are positive to discover many automated setup and evaluation approaches in open source artefacts (P4) accompanying our studies, but still encounter too many studies with inexistent or tedious manual setup instructions.

6. Implications and gaps in literature

We now discuss the main findings and implications of our study and identify gaps in current literature.

6.1. Publication trends (RQ 1)

FaaS performance evaluation is a growing field of research in academic as well as grey literature, with a surge of new studies appearing in 2018. Our results show that a healthy 20% of the selected academic studies are published in top-ranked conferences or journals.

6.2. Benchmarked platforms (RQ 2)

The most evaluated platforms are AWS Lambda (88%), Azure Functions (26%), Google Cloud Functions (23%), IBM Cloud Functions (13%), and self-hosted platforms (14%), predominantly Apache OpenWhisk. In absolute numbers, AWS is currently overstudied (by a factor of 3x). However, other sources have reported that AWS is also predominant in actual production usage by a similar margin (see Section 5.2-Discussion). Despite current industrial practice, future FaaS benchmarking studies should go beyond performance evaluations for the most popular platforms (e.g., avoid studying only AWS) to broaden our understanding of the field in general. Further, other quickly rising cloud providers (e.g., Alibaba Cloud as the leading Asian cloud provider[25]) currently see no attention in literature.

6.3. Evaluated performance characteristics (RQ 3)

The lack of cross-platform benchmarks is a common theme across the following performance characteristics.

6.3.1. Evaluated benchmark types (RQ 3.1)

The predominant use of micro-benchmarks in 75% of all studies indicates an over-emphasis on simple easy-to-build benchmarks, compared to application-benchmarks, which are used in 57% of the academic and 31% of the grey literature studies (i.e., overall 18% use both). This insight is supported by the large percentage of studies conducting platform overhead benchmarks with trivial functions (e.g., returning a constant) and CPU benchmarks using common workloads (e.g., prime number calculations). Future work needs to go beyond such oversimplified benchmarks, and focus on more realistic benchmarks and workloads. We also identify a need to develop cross-platform application-level benchmarks as the current focus on a single platform (88% of all application-benchmarks are evaluated on AWS) limits their usefulness for comparing platforms. However, such cross-platform benchmarks are challenging to develop due to heterogeneous platforms and their complex ecosystems (van Eyk et al., 2020).

6.3.2. Evaluated micro-benchmarks (RQ 3.2)

Most micro-benchmarks (40%) evaluate CPU performance, and show that CPU performance in FaaS systems is indeed proportional to the memory size of the selected function type for certain providers (i.e,. AWS, Google). This is disappointing, as this behaviour is well-documented by the cloud providers themselves and does not justify much further study. We understand the need for periodic re-evaluations due to the dynamic nature of continuously evolving FaaS platforms (Leitner and Cito, 2016) and want to emphasize the importance of studies targeting continuous benchmarking efforts (see examples in Section 5.5–P1). However, given the large scientific support that CPU performance of FaaS services behaves as documented, we suggest future studies to de-emphasize this aspect and focus on other characteristics such as network or function trigger performance (or real-world application-benchmarks).

[25] https://www.parkmycloud.com/blog/alibaba-cloud-market-share/.

6.3.3. Evaluated general characteristics (RQ 3.3)

The most evaluated general performance characteristics are FaaS platform overhead (i.e., cold starts) and workload concurrency (i.e., invoking the same function in parallel), both used by about half of the studies. This makes sense, as these aspects link to FaaS specifics and the most significant advantages of using FaaS, as reported in other surveys (Leitner et al., 2019). No study currently evaluates function triggers across platforms. We think the integration through triggers is an important aspect for FaaS performance, where insights can guide decisions about function invocation, function coordination, and usage of appropriate external services. A major open research challenge towards such cross-platform benchmarks is the heterogeneous landscape of FaaS systems (van Eyk et al., 2020).

6.4. Used platform configurations (RQ 4)

Our study indicates a broader coverage of language runtimes, but shows that other platform configurations focus on very few function triggers and external services.

6.4.1. Used language runtimes (RQ 4.1)

We identify a mismatch between academic and industrial sources, as Node.js, Java, Go, and C# are evaluated two times more frequently in grey literature than in academic work. Grey literature is generally more focused on covering more and more diverse runtimes than academic literature. We suggest future academic literature studies to diversify their choice of runtimes, potentially also including insufficiently researched runtimes, such as Go or C#.

6.4.2. Used function triggers (RQ 4.2)

At the moment, a majority of studies (57%) focuses on HTTP triggers. We conclude that many trigger types remain largely insufficiently researched and suggest future studies to explore alternative triggers, such as message queues, data streams, timers, or SDKs.

6.4.3. Used external services (RQ 4.3)

Integrating external services in a meaningful way into FaaS performance evaluation studies remains an open challenge. Despite their importance to overall serverless application performance, most current evaluations choose to abstract away from external services. The only services we have seen used with some frequency are cloud API gateways (57%), cloud storage (47% academic vs. 10% grey literature), and cloud databases (10%–15%).

6.5. Reproducibility (RQ 5)

We find that 7 of 8 reproducibility principles are not followed by the majority of the analysed studies. This is in line with the results of the original study (Papadopoulos et al., 2019) on cloud experimentation in general. We classify one third of all studies as practically impossible or hard to replicate under reasonably similar conditions due to the simultaneous lack of sufficient experimental setup description and available artefacts. Overall, academic studies tend to satisfy the principles more comprehensively than grey literature but we do not see a clear trend that academic literature is less susceptible to disregarding the principles. Academic work is considerably better (principle fully met >20%) than grey literature in choosing appropriate workloads (P2), describing the experimental setup (P3), and reporting costs (P8). However, grey literature is considerably better in providing at least partial open access to experimental artefacts (i.e., code and data). We support the trend towards artifact evaluations[26] and recommend focusing on artefact availability first (e.g., explicitly include availability check in reviewer guidelines) and subsequently target more qualitative attributes (e.g., ACM Functional, defined as documented, consistent, complete, exercisable). We conclude with actionable recommendations on what are the next steps regarding each principle for future FaaS studies to improve:

P1 Explicitly report the number of iterations.
P2 Motivate workloads through industrial use cases.
P3 Report the time of experiment and follow good examples [A26, A8, A34] (see Section 5.5–P3).
P4 Publish the dataset in addition to the benchmark code.
P5 Stop reporting mean values exclusively, but use appropriate statistical tools, such as CDFs, instead.
P6 Use appropriate statistical tests, such as Wilcoxon rank-sum or overlapping bootstrapped confidence intervals, for stronger conclusions (Laaber et al., 2019).
P7 Include measurement units in all figures.
P8 Report a cost model.

7. Related work

We compare and relate our results to existing literature reviews on FaaS and more generally on cloud performance evaluations, and compare our FaaS-specific results on reproducibility principles to the original study on cloud experimentation.

7.1. Literature reviews on FaaS

Kuhlenkamp and Werner (2018) introduce a methodology for a collaborative SLR on FaaS benchmarking and report on preliminary results of 9 studies. They capture more fine-grained experiments within each paper and extract data regarding workload generator, function implementation, platform configuration, and whether external services are used. A completeness score of these categories represents the reproducibility of FaaS experiments. Their results indicate insufficient experimental description. Somu et al. (2020) summarize the capabilities of 7 FaaS benchmarking studies along 34 characteristics for parameters, benchmarks, and metrics. Their results indicate a strong focus on the AWS Lambda platform and a lack of support for function chaining, especially in combination with different trigger types. These two most related papers hint towards some of our results but cannot confidently identify overall trends due to their limited scope.

Taibi et al. (2020) conduct an MLR on serverless cloud computing patterns to catalogue 32 patterns originating from 24 sources. Their MLR has a strong practitioner perspective but is limited to 7 peer-reviewed sources. Our work focuses on performance whereas their pattern catalogue only occasionally mentions performance as part of discussing a pattern.

Yussupov et al. (2019) conduct a systematic mapping study on FaaS platforms and tools to identify overall research trends and underlying main challenges and drivers in this field across 62 selected publications. Their work covers a broader range of FaaS research and explicitly excludes FaaS benchmarking studies "without proposing any modifications" (Yussupov et al., 2019) through their exclusion criteria. Nevertheless, they identify 5 benchmarking studies and 26 function execution studies on performance optimization. Al-Ameen and Spillner (2018) introduced a curated "Serverless Literature Dataset" that initially covered 60 scientific publications and preprints related to FaaS and Serverless computing in general, but in its latest Version 0.4 (2019-10-23) (Spillner and Al-Ameen, 2019) the dataset has been extended

[26] https://www.acm.org/publications/policies/artifact-review-badging.

to 188 articles. The authors classify their work as no survey itself, but rather envision its potential as input for future surveys such as ours. We demonstrate this potential in the manual search process for academic literature where the serverless literature dataset covers 34 out of 35 relevant studies. These two general studies identify publication trends, common technologies, and categories of research but do not extract and synthesize more specific data on FaaS benchmarking aspects we cover in our work. To the best of our knowledge, we present the first comprehensive and systematic literature review on FaaS performance evaluation covering academic as well as grey literature.

7.2. Literature reviews on cloud performance

We relate our results to existing literature reviews on general cloud performance topics. These studies apply similar methods to us but in the context of cloud performance evaluation in general. Li et al. (2013) conducted an SLR on evaluating commercial cloud services for 82 relevant studies. Their work is methodologically closely related to our MLR but targets a more general field of research than our FaaS benchmarking study. Their SLR has a strong focus on publication trends and performance metrics building upon the authors' previous work on cataloguing (Li et al., 2012b) and classifying (Li et al., 2012a) performance evaluation metrics. In contrast, our work specializes on performance characteristics in the field of FaaS, extends the scope beyond academic research by including grey literature, and reports on the reproducibility of the analysed studies. Leitner and Cito (2016) used an SLR methodology and open coding for identifying hypotheses seeding their principled experimental validation study on performance predictability of public IaaS clouds. They performed experimental validation on common patterns of results and conclusions but did not extract further data on benchmarking studies. A recent preprint (March 2020) (Bjørndal et al., 2020) conducts an SLR on benchmarks and metrics within software engineering in the context of migrating from monolithic to microservice architectures. The most frequent metrics for their 33 selected articles are latency, CPU, throughput, and network indicating that their study partially uses similar characteristics but in a less structured way (e.g., network and throughput are orthogonal aspects).

7.3. Reproducibility principles

We compare our FaaS-specific results to the results of the original study by Papadopoulos et al. (2019) on more general experimentation in cloud environments. Our MLR study specifically targets FaaS experiments for academic and grey literature resulting in a largely disjoint set of studies with only 2 of our studies matching their stricter venue and impact criteria (i.e., $\geq$15 citations). Overall, our results for academic literature studies are reasonably similar ($\pm$10%) except for P1 and P5. For P1, we speculate that we might have been more lenient in classifying studies, especially when no long-time experiments were present. For P5, we see an improvement and notice more widespread use of CDFs, histograms, and boxplots or dotplots with error margins and accompanying percentiles. Smaller trends suggest that more of our selected studies tend to open source technical artefacts (P4) and report costs (P8), but perform slightly worse in workload and configuration coverage (P2).

8. Conclusion

This paper presented results from the first systematic and comprehensive survey on FaaS performance evaluation studies. We conducted a multivocal literature review (MLR) across 112 studies from academic (51) and grey (61) literature. We identify gaps in literature and give actionable recommendations highlighting the next steps towards compliance with reproducibility principles on cloud experimentation. Our main findings are that AWS Lambda is the most evaluated FaaS platform (88%), that micro-benchmarks are the most common type of benchmark (75%), and that application benchmarks are currently prevalently evaluated on a single platform. We further find that the majority of studies do not follow reproducibility principles on cloud experimentation from prior work. Academic studies tend to satisfy the principles more comprehensively than grey literature, but we do not see a clear trend that academic literature is less susceptible to disregarding the principles. We recommend future studies to broaden their scope of platforms beyond AWS as a single platform and in particular contribute cross-platform application-level benchmarks. FaaS performance evaluation studies need to address flaws threatening their reproducibility and should particularly focus on choosing relevant workloads, publishing collected datasets, and statistically evaluating their results. Our survey consolidates existing work and can guide future research directions. It provides a useful instrument for the systematic discovery of related studies and thus helps future research to relate and discuss their results in a wider context.

CRediT authorship contribution statement

Joel Scheuner: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Philipp Leitner:** Conceptualization, Methodology, Investigation, Writing - review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank the SPEC RG Cloud Working group for insightful discussions on this topic and Jan-Philipp Steghöfer for his valuable inputs. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, Sweden, and by the Swedish Research Council VR under grant number 2018-04127.

Appendix

The appendix provides a complete list of all 112 studies analysed as part of the multivocal literature review including hyperlinks to 51 academic (see Table A.1) and 61 grey (Table A.2) literature sources. The complete curated dataset is available online for interactive querying[27] in its latest version and published as a versioned dataset on Zenodo (Scheuner and Leitner, 2020).

[27] https://github.com/joe4dev/faas-performance-mlr.

Table A.1
Complete list of analysed academic literature studies.

ID	Ref	Authors	Article title and link (preferably DOI, last accessed 2020-07-09)	Published
A01	Akkus et al. (2018)	Akkus et al.	SAND: Towards High-Performance Serverless Computing	2018-07
A02	Albuquerque et al. (2017)	Albuquerque et al.	Function-as-a-Service X Platform-as-a-Service: Towards a Comparative Study on FaaS and PaaS	2017-10
A03	Back and Andrikopoulos (2018)	Back and Andrikopoulos	Using a Microbenchmark to Compare Function as a Service Solutions	2018-08
A04	Balla et al. (2020)	Balla et al.	Tuning Runtimes in Open Source FaaS	2020-01
A05	Bardsley et al. (2018)	Bardsley et al.	Serverless Performance and Optimization Strategies	2018-09
A06	Bortolini and Obelheiro (2019)	Bortolini and Obelheiro	Investigating Performance and Cost in Function-as-a-Service Platforms	2019-10
A07	Carreira et al. (2018)	Carreira et al.	A Case for Serverless Machine Learning	2018-12
A08	Figiela et al. (2018)	Figiela et al.	Performance evaluation of heterogeneous cloud functions	2018-08
A09	Fouladi et al. (2017)	Fouladi et al.	Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads.	2017-03
A10	Fouladi et al. (2019)	Fouladi et al.	From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers	2019-07
A11	Gan et al. (2019)	Gan et al.	An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems	2019-04
A12	Giménez-Alventosa et al. (2019)	Giménez-Alventosa et al.	A framework and a performance assessment for serverless MapReduce on AWS Lambda	2019-08
A13	Gupta et al. (2018)	Gupta et al.	OverSketch: Approximate Matrix Multiplication for the Cloud	2018-12
A14	Hall and Ramachandran (2019)	Hall and Ramachandran	An Execution Model for Serverless Functions at the Edge	2019-04
A15	Hendrickson et al. (2016)	Hendrickson et al.	Serverless Computation with OpenLambda	2016-05
A16	Ishakian et al. (2018)	Ishakian et al.	Serving Deep Learning Models in a Serverless Platform	2018-04
A17	Ivan et al. (2019)	Ivan et al.	Serverless Computing: An Investigation of Deployment Environments for Web APIs	2019-06
A18	Jackson and Clynch (2018)	Jackson and Clynch	An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions	2018-12
A19	Jiang et al. (2017)	Jiang et al.	Serverless Execution of Scientific Workflows	2017-10
A20	Jonas et al. (2017)	Jonas et al.	Occupy the cloud: distributed computing for the 99%	2017-09
A21	Kim and Lee (2019)	Kim and Lee	FunctionBench: A Suite of Workloads for Serverless Cloud Function Service	2019-07
A22	Kim et al. (2019)	Kim et al.	Network Resource Isolation in Serverless Cloud Function Service	2019-06
A23	Klimovic et al. (2018)	Klimovic et al.	Understanding Ephemeral Storage for Serverless Analytics	2018-07
A24	Kuhlenkamp and Klems (2017)	Kuhlenkamp and Klems	Costradamus: A Cost-Tracing System for Cloud-Based Software Services	2017-10
A25	Kuhlenkamp et al. (2019)	Kuhlenkamp et al.	An Evaluation of FaaS Platforms as a Foundation for Serverless Big Data Processing	2019-12
A26	Kuhlenkamp et al. (2020)	Kuhlenkamp et al.	Benchmarking Elasticity of FaaS Platforms as a Foundation for Objective-driven Design of Serverless Applications	2020-03
A27	Lee et al. (2018)	Lee et al.	Evaluation of Production Serverless Computing Environments	2018-07
A28	Li et al. (2019)	Li et al.	Understanding Open Source Serverless Platforms: Design Considerations and Performance	2019-12
A29	Lloyd et al. (2018a)	Lloyd et al.	Serverless Computing: An Investigation of Factors Influencing Microservice Performance	2018-04
A30	Lloyd et al. (2018)	Lloyd et al.	Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads	2018-12
A31	López et al. (2018)	López et al.	Comparison of FaaS Orchestration Systems	2018-12
A32	Malawski et al. (2017)	Malawski et al.	Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions	2017-11
A33	Malla and Christensen (2019)	Malla and Christensen	HPC in the Cloud: Performance Comparison of Function as a Service (FaaS) vs Infrastructure as a Service (IaaS)	2019-11
A34	Manner et al. (2018)	Manner et al.	Cold Start Influencing Factors in Function as a Service	2018-12
A35	McGrath et al. (2016)	McGrath et al.	Cloud Event Programming Paradigms: Applications and Analysis	2016-06
A36	McGrath and Brenner (2017)	McGrath and Brenner	Serverless Computing: Design, Implementation, and Performance	2017-06
A37	Mohan et al. (2019)	Mohan et al.	Agile Cold Starts for Scalable Serverless	2019-07
A38	Mohanty et al. (2018)	Mohanty et al.	An Evaluation of Open Source Serverless Computing Frameworks.	2018-12
A39	Niu et al. (2019)	Niu et al.	Leveraging Serverless Computing to Improve Performance for Sequence Comparison	2019-09
A40	Oakes et al. (2018)	Oakes et al.	SOCK: Rapid Task Provisioning with Serverless-Optimized Containers	2018-07
A41	Pelle et al. (2019)	Pelle et al.	Towards Latency Sensitive Cloud Native Applications: A Performance Study on AWS	2019-07
A42	Pérez et al. (2019)	Pérez et al.	A Programming Model and Middleware for High Throughput Serverless Computing Applications	2019-04
A43	Pu et al. (2019)	Pu et al.	Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure	2019-02
A44	Puripunpinyo and Samadzadeh (2017)	Puripunpinyo and Samadzadeh	Effect of optimizing Java deployment artifacts on AWS Lambda	2017-05
A45	Saha and Jindal (2018)	Saha and Jindal	EMARS: Efficient Management and Allocation of Resources in Serverless	2018-07
A46	Shillaker (2018)	Shillaker	A provider-friendly serverless framework for latency-critical applications	2018-04
A47	Singhvi et al. (2017)	Singhvi et al.	Granular Computing and Network Intensive Applications: Friends or Foes?	2017-11
A48	Spillner et al. (2018)	Spillner et al.	FaaSter, Better, Cheaper: The Prospect of Serverless Scientific Computing and HPC	2018-12
A49	Wang et al. (2018)	Wang et al.	Peeking behind the curtains of serverless platforms	2018-07
A50	Werner et al. (2018)	Werner et al.	Serverless Big Data Processing using Matrix Multiplication as Example	2018-12
A51	Zhang et al. (2019)	Zhang et al.	Video Processing with Serverless Computing: A Measurement Study	2019-06

Table A.2
Complete list of analysed grey literature studies.

ID	Authors	Article title and link (last accessed 2020-03-10)	Published
G01	Adam Matan	How to Make It Fa(a)ster?	2018-02-07
G02	Adir Shemesh	Is Azure Functions 3.0 Ready For Production?	2019-12-17
G03	Alessandro Morandi	Investigating the effect of VPC on AWS Lambda cold-start	2019-03-28
G04	Alex DeBrie	AWS API Performance Comparison: Serverless vs. Containers vs. API Gateway integration	2019-02-20
G05	Algirdas Grumuldis	Evaluation of "Serverless" Application Programming Model : How and when to start Serverles	2019-04-25
G06	Andre Luckow and Shantenu Jha	Performance Characterization and Modeling of Serverless and HPC Streaming Applications	2019-09-13
G07	Andrew Hoang	Analysis of microservices and serverless architecture for mobile application enablement	2017-06-13
G08	Andrew Smiryakhin et al.	Java vs NodeJS on AWS Lambda: Benchmark Survey	Unknown
G09	Andy Warzon	How Reliable is Lambda Cron?	2017-01-06
G10	Bernd Strehl	The largest benchmark of Serverless providers.	2018-09-07
G11	Bernd Strehl	λ Serverless Benchmark (Serverless Benchmark2.0)	Live
G12	Bruno Schionato et al.	What We Learned by Serving Machine Learning Models Using AWS Lambda	2018-10-30
G13	Can Tepakidareekul	Serverless Platform Comparison: Google Cloud Function vs. AWS Lambda	2018-11-26
G14	Cloudflare, Inc.	How Can Serverless Computing Improve Performance? | Lambda Performance	Unknown
G15	Driss Amri	How to minimize AWS Lambda Java cold starts	2019-05-19
G16	Erica Windisch	Understanding AWS Lambda Coldstarts	2017-02-09
G17	Erwin Van Eyk	The Design, Productization, and Evaluation of a Serverless Workflow-Management System	2019-06-21
G18	Frederik Willaert	The Serverless Promise: AWS and Azure Function Performance and Scaling	2018-01-14
G19	Ingo Müller et al.	Lambada: Interactive Data Analytics on Cold Data using Serverless Cloud Infrastructure	2019-12-02
G20	James Randall	Azure Functions - Significant Improvements in HTTP Trigger Scaling	2018-03-09
G21	James Randall	Azure Functions - Significant Improvements in HTTP Trigger Scaling	2018-03-09
G22	Jannik Kollmann	Serverless Architecture with AWS Lambda	2018-02-06
G23	Jim Conning	AWS Lambda: Faster Is Cheaper	2017-01-26
G24	John Chapin	The Occasional Chaos of AWS Lambda Runtime Performance	2017-02-24
G25	Josef Spillner	FaaS: Function hosting services and their technical characteristics	2019-10-31
G26	Jun Sung Park	AWS Lambda Triggers: Kinesis vs SQS	2019-05-06
G27	Kevin S Lin	Benchmarking Lambda's Idle Timeout Before A Cold Start	2019-02-05
G28	Leonardo Zanivan	Serverless Native Java Functions using GraalVM and Fn Project	2018-06-24
G29	Luke Demi	Exploring serverless technology by benchmarking AWS Lambda	2019-03-28
G30	Mark Fowler	Battle of the Serverless - Part 2: AWS Lambda Cold Start Times	2019-10-28
G31	Matthieu Napoli	Serverless and PHP: Performances	2018-05-24
G32	Mikhail Shilkov	Serverless at Scale: Serving StackOverflow-like Traffic	2019-01-20
G33	Mikhail Shilkov	Comparison of Cold Starts in Serverless Functions across AWS, Azure, and GCP	2019-09-26
G34	Mikhail Shilkov	Azure Functions V2 Is Released, How Performant Is It?	2018-10-10
G35	Mikhail Shilkov	From 0 to 1000 Instances: How Serverless Providers Scale Queue Processing	2018-11-19
G36	Mustafa Akin	How does proportional CPU allocation work with AWS Lambda?	2018-01-25
G37	Mustafa Akin	sing GraalVM to run Native Java in AWS Lambda with Golang	2018-11-06
G38	Nathan Malishev	How to manage Lambda VPC cold starts and deal with that killer latency	2018-07-14
G39	Nathan Malishev	AWS Lambda Cold Start Language Comparisons, 2019 edition	2019-09-04
G40	Paul Batum	Processing 100,000 events per second on Azure Functions	2017-09-19
G41	Pranjal Gupta and Shreesha Addala	Experimental Evaluation of Serverless Functions	2019-09-13
G42	Ran Ribenzaft	The right way to distribute messages effectively in serverless applications	2018-03-31
G43	Ran Ribenzaft	How to make Lambda faster: memory performance benchmark	2018-02-15
G44	Remy Chantenay	Blazing Fast Microservice with Go and Lambda	2018-02-28
G45	Robert Vojta	AWS journey - API Gateway & Lambda & VPC performance	2016-10-30
G46	Rodric Rabbah	How to see performance details for cloud functions (FaaS)	2018-02-28
G47	Ryan Chard et al.	Serverless Supercomputing: High Performance Function as a Service for Science	2019-08-14
G48	Sachin Shrestha	Comparing Programming Languages used in AWS Lambda for Serverless Architecture	2019-06-19
G49	Sebastián Quevedo et al.	Evaluating Apache OpenWhisk-FaaS	2020-02-05
G50	Tai Nguyen Bui	Benchmarking AWS Lambda runtimes in 2019 (Part I)	2019-07-04
G51	Tai Nguyen Bui	Benchmarking AWS Lambda runtimes in 2019 (Part II)	2019-07-15
G52	Tim Nolet	AWS Lambda Go vs. Node.js performance benchmark: updated	2018-01-15
G53	Vlad Holubiev	My Accidental 3--5x Speed Increase of AWS Lambda Functions	2016-12-12
G54	Wojciech Gawroński	Functional Programming in Serverless World	2018-10-18
G55	Yan Cui	aws lambda - compare coldstart time with different languages, memory and code sizes	2017-06-13
G56	Yan Cui	How long does AWS Lambda keep your idle functions around before a cold start?	2017-07-04
G57	Yan Cybulski	FaaS Benchmarking Made Easy: Introducing FaaStest.com and Faasbenchmark	2019-10-02
G58	Yun Zhi Lin	Comparing AWS Lambda performance of Node.js, Python, Java, C# and Go	2018-03-08
G59	Zac Charles	NET Core 3.0 AWS Lambda Benchmarks and Recommendations	2019-10-25
G60	Zack Bloom	Comparing Serverless Performance for CPU Bound Tasks	2018-07-09
G61	Zack Bloom	Serverless Performance: Cloudflare Workers, Lambda and Lambda@Edge	2018-07-02

References

Akkus, I.E., Chen, R., Rimac, I., Stein, M., Satzke, K., Beck, A., Aditya, P., Hilt, V., 2018. SAND: Towards high-performance serverless computing. In: Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference. In: USENIX ATC '18, USENIX Association, USA, pp. 923–935, URL https://www.usenix.org/conference/atc18/presentation/akkus.

Al-Ameen, M., Spillner, J., 2018. Systematic and open exploration of faas and serverless computing research. In: Proceedings of the European Symposium on Serverless Computing and Applications (ESSCA). In: CEUR Workshop Proceedings, vol. 2330, CEUR-WS.org, pp. 30–35, URL http://ceur-ws.org/Vol-2330/short2.pdf.

Albuquerque, Jr., L.F., Ferraz, F.S., Oliveira, R.F., Galdino, S.M., 2017. Function-as-a-service x platform-as-a-service: Towards a comparative study on FaaS and PaaS. In: ICSEA 2017. p. 217, URL https://www.thinkmind.org/download.php?articleid=icsea_2017_9_30_10096.

Back, T., Andrikopoulos, V., 2018. Using a microbenchmark to compare function as a service solutions. In: Kritikos, K., Plebani, P., de Paoli, F. (Eds.), European Conference on Service-Oriented and Cloud Computing. Springer International Publishing, Cham, pp. 146–160. http://dx.doi.org/10.1007/978-3-319-99819-0_11.

Balla, D., Maliosz, M., Simon, C., Gehberger, D., 2020. Tuning runtimes in open source faas. In: Hsu, C.-H., Kallel, S., Lan, K.-C., Zheng, Z. (Eds.), Internet of Vehicles. Toward Smart Cities, Springer International Publishing, Cham, pp. 250–266. http://dx.doi.org/10.1007/978-3-030-38651-1_21.

Bardsley, D., Ryan, L., Howard, J., 2018. Serverless performance and optimization strategies. In: 2018 IEEE International Conference on Smart Cloud (SmartCloud). pp. 19–26. http://dx.doi.org/10.1109/SmartCloud.2018.00012.

Bjørndal, N., Bucchiarone, A., Mazzara, M., Dragoni, N., Dustdar, S., Kessler, F.B., Wien, T., 2020. Migration from monolith to microservices: Benchmarking a case study. http://dx.doi.org/10.13140/RG.2.2.27715.14883, unpublished.

Bortolini, D., Obelheiro, R.R., 2019. Investigating performance and cost in function-as-a-service platforms. In: Barolli, L., Hellinckx, P., Natwichai, J. (Eds.), Advances on P2P, Parallel, Grid, Cloud and Internet Computing. Springer International Publishing, Cham, http://dx.doi.org/10.1007/978-3-030-33509-0_16.

Carreira, J., Fonseca, P., Tumanov, A., Zhang, A., Katz, R., 2018. A case for serverless machine learning. In: Workshop on Systems for ML and Open Source Software at NeurIPS. URL http://learningsys.org/nips18/assets/papers/101CameraReadySubmissioncirrus_nips_final2.pdf.

Eyk, E.V., Iosup, A., Seif, S., Thömmes, M., 2017. The SPEC cloud group's research vision on faas and serverless architectures. In: Proceedings of the 2nd International Workshop on Serverless Computing (WOSC). ACM, pp. 1–4. http://dx.doi.org/10.1145/3154847.3154848.

Farley, B., Juels, A., Varadarajan, V., Ristenpart, T., Bowers, K.D., Swift, M.M., 2012. More for your money: Exploiting performance heterogeneity in public clouds. In: Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12). pp. 20:1–20:14. http://dx.doi.org/10.1145/2391229.2391249.

Figiela, K., Gajek, A., Zima, A., Obrok, B., Malawski, M., 2018. Performance evaluation of heterogeneous cloud functions. In: Concurrency and Computation: Practice and Experience, Vol. 30. http://dx.doi.org/10.1002/cpe.4792.

Fleming, P.J., Wallace, J.J., 1986. How not to lie with statistics: The correct way to summarize benchmark results. Commun. ACM 218–221. http://dx.doi.org/10.1145/5666.5673.

Fouladi, S., Romero, F., Iter, D., Li, Q., Chatterjee, S., Kozyrakis, C., Zaharia, M., Winstein, K., 2019. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In: 2019 USENIX Annual Technical Conference (USENIX ATC 19). USENIX Association, Renton, WA, pp. 475–488, URL https://www.usenix.org/conference/atc19/presentation/fouladi.

Fouladi, S., Wahby, R.S., Shacklett, B., Balasubramaniam, K., Zeng, W., Bhalerao, R., Sivaraman, A., Porter, G., Winstein, K., 2017. Encoding, fast and slow: low-latency video processing using thousands of tiny threads. In: NSDI. pp. 363–376, URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi.

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., Hu, K., Pancholi, M., He, Y., Clancy, B., Colen, C., Wen, F., Leung, C., Wang, S., Zaruvinsky, L., Espinosa, M., Lin, R., Liu, Z., Padilla, J., Delimitrou, C., 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: Bahar, I., Herlihy, M., Witchel, E., Lebeck, A.R. (Eds.), Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19). In: ASPLOS '19, ACM, ACM, New York, NY, USA, pp. 3–18. http://dx.doi.org/10.1145/3297858.3304013.

Garousi, V., Felderer, M., Hacaloglu, T., 2017. Software test maturity assessment and test process improvement: A multivocal literature review. Inf. Softw. Technol. 16–42. http://dx.doi.org/10.1016/j.infsof.2017.01.001.

Garousi, V., Felderer, M., Mäntylä, M.V., 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. 101–121. http://dx.doi.org/10.1016/j.infsof.2018.09.006.

Giménez-Alventosa, V., Moltó, G., Caballer, M., 2019. A framework and a performance assessment for serverless mapreduce on aws lambda. Future Gener. Comput. Syst. 97, 259–274. http://dx.doi.org/10.1016/j.future.2019.02.057.

Gupta, V., Wang, S., Courtade, T., Ramchandran, K., 2018. OverSketch: Approximate matrix multiplication for the cloud. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 298–304. http://dx.doi.org/10.1109/BigData.2018.8622139.

Hall, A., Ramachandran, U., 2019. An execution model for serverless functions at the edge. In: Proceedings of the International Conference on Internet of Things Design and Implementation. In: IoTDI '19, Association for Computing Machinery, New York, NY, USA, pp. 225–236. http://dx.doi.org/10.1145/3302505.3310084.

Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., 2016. Serverless computation with OpenLambda. In: 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), Vol. 60. USENIX Association, Denver, CO, p. 80, URL https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/hendrickson.

Ishakian, V., Muthusamy, V., Slominski, A., 2018. Serving deep learning models in a serverless platform. In: 2018 IEEE International Conference on Cloud Engineering (IC2E). pp. 257–262. http://dx.doi.org/10.1109/IC2E.2018.00052.

Ivan, C., Vasile, R., Dadarlat, V., 2019. Serverless computing: An investigation of deployment environments for web apis. Computers 8, http://dx.doi.org/10.3390/computers8020050.

Jackson, D., Clynch, G., 2018. An investigation of the impact of language runtime on the performance and cost of serverless functions. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). pp. 154–160. http://dx.doi.org/10.1109/UCC-Companion.2018.00050.

Jiang, Q., Lee, Y.C., Zomaya, A.Y., 2017. Serverless execution of scientific workflows. In: Service-Oriented Computing, Vol. 10601. Springer International Publishing, Cham, pp. 706–721. http://dx.doi.org/10.1007/978-3-319-69035-3_51.

Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., Recht, B., 2017. Occupy the cloud: distributed computing for the 99%. In: Proceedings of the 2017 Symposium on Cloud Computing. ACM, pp. 445–451. http://dx.doi.org/10.1145/3127479.3128601.

Kim, J., Lee, K., 2019. Functionbench: A suite of workloads for serverless cloud function service. In: IEEE International Conference on Cloud Computing (WIP paper). pp. 502–504. http://dx.doi.org/10.1109/CLOUD.2019.00091.

Kim, J., Park, J., Lee, K., 2019. Network resource isolation in serverless cloud function service. In: AMGCC 2019 in conjuction with IEEE International Conference on Autonomic Computing. pp. 182–187. http://dx.doi.org/10.1109/FAS-W.2019.00051.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, Keele University, URL https://community.dur.ac.uk/ebse/resources/guidelines/Systematic-reviews-5-8.pdf.

Klimovic, A., Wang, Y., Kozyrakis, C., Stuedi, P., Pfefferle, J., Trivedi, A., 2018. Understanding ephemeral storage for serverless analytics. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). USENIX Association, Boston, MA, pp. 789–794, URL https://www.usenix.org/conference/atc18/presentation/klimovic-serverless.

Kuhlenkamp, J., Klems, M., 2017. Costradamus: A cost-tracing system for cloud-based software services. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (Eds.), Service-Oriented Computing. Service-Oriented Computing, Springer International Publishing, Cham, pp. 657–672. http://dx.doi.org/10.1007/978-3-319-69035-3_48.

Kuhlenkamp, J., Werner, S., 2018. Benchmarking FaaS platforms: Call for community participation. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). IEEE, pp. 189–194. http://dx.doi.org/10.1109/UCC-Companion.2018.00055.

Kuhlenkamp, J., Werner, S., Borges, M.C., El Tal, K., Tai, S., 2019. An evaluation of FaaS platforms as a foundation for serverless big data processing. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing. In: UCC'19, Association for Computing Machinery, New York, NY, USA, pp. 1–9. http://dx.doi.org/10.1145/3344341.3368796.

Kuhlenkamp, J., Werner, S., Borges, M.C., Ernst, D., Wenzel, D., 2020. Benchmarking elasticity of faas platforms as a foundation for objective-driven design of serverless applications. In: Hung, C., Cerný, T., Shin, D., Bechini, A. (Eds.), SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing. ACM, pp. 1576–1585. http://dx.doi.org/10.1145/3341105.3373948.

Laaber, C., Scheuner, J., Leitner, P., 2019. Software microbenchmarking in the cloud. how bad is it really?. Empir. Softw. Eng. 24, 2469–2508. http://dx.doi.org/10.1007/s10664-019-09681-1.

Lee, H., Satyam, K., Fox, G.C., 2018. Evaluation of production serverless computing environments. In: Third International Workshop on Serverless Computing (WoSC). pp. 442–450. http://dx.doi.org/10.1109/CLOUD.2018.00062.

Leitner, P., Cito, J., 2016. Patterns in the chaos – a study of performance variation and predictability in public IaaS clouds. ACM Trans. Internet Technol. 15:1–15:23. http://dx.doi.org/10.1145/2885497.

Leitner, P., Wittern, E., Spillner, J., Hummer, W., 2019. A mixed-method empirical study of function-as-a-service software development in industrial practice. J. Syst. Softw. 340–359. http://dx.doi.org/10.1016/j.jss.2018.12.013.

Li, J., Kulkarni, S.G., Ramakrishnan, K.K., Li, D., 2019. Understanding open source serverless platforms: Design considerations and performance. In: Proceedings of the 5th International Workshop on Serverless Computing. In: WOSC '19, ACM, New York, NY, USA, pp. 37–42. http://dx.doi.org/10.1145/3366623.3368139.

Li, Z., O'Brien, L., Cai, R., Zhang, H., 2012a. Towards a taxonomy of performance evaluation of commercial cloud services. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. pp. 344–351. http://dx.doi.org/10.1109/CLOUD.2012.74.

Li, Z., O'Brien, L., Zhang, H., Cai, R., 2012b. On a catalogue of metrics for evaluating commercial cloud services. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing (GRID '12). pp. 164–173. http://dx.doi.org/10.1109/Grid.2012.15.

Li, Z., Zhang, H., O'Brien, L., Cai, R., Flint, S., 2013. On evaluating commercial cloud services: A systematic review. J. Syst. Softw. 2371–2393. http://dx.doi.org/10.1016/j.jss.2013.04.021.

Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., Pallickara, S., 2018a. Serverless computing: An investigation of factors influencing microservice performance. In: IEEE International Conference on Cloud Engineering (IC2E). http://dx.doi.org/10.1109/IC2E.2018.00039.

Lloyd, W., Vu, M., Zhang, B., David, O., Leavesley, G., 2018. Improving application migration to serverless computing platforms: Latency mitigation with keep-alive workloads. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). pp. 195–200. http://dx.doi.org/10.1109/UCC-Companion.2018.00056.

López, P.G., Sánchez-Artigas, M., París, G., Pons, D.B., Ollobarren, Á.R., Pinto, D.A., 2018. Comparison of faas orchestration systems. In: 4th International Workshop on Serverless Computing (WoSC). pp. 148–153. http://dx.doi.org/10.1109/UCC-Companion.2018.00049.

Malawski, M., Gajek, A., Zima, A., Balis, B., Figiela, K., 2017. Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and google cloud functions. Future Gener. Comput. Syst. http://dx.doi.org/10.1016/j.future.2017.10.029.

Malla, S., Christensen, K., 2019. HPC in the cloud: Performance comparison of function as a service (FaaS) vs infrastructure as a service (IaaS). In: Internet Technology Letters. http://dx.doi.org/10.1002/itl2.137.

Manner, J., Endreß, M., Heckel, T., Wirtz, G., 2018. Cold start influencing factors in function as a service. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). pp. 181–188. http://dx.doi.org/10.1109/UCC-Companion.2018.00054.

McGrath, G., Brenner, P.R., 2017. Serverless computing: Design, implementation, and performance. In: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). pp. 405–410. http://dx.doi.org/10.1109/ICDCSW.2017.36.

McGrath, G., Short, J., Ennis, S., Judson, B., Brenner, P., 2016. Cloud event programming paradigms: Applications and analysis. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD). pp. 400–406. http://dx.doi.org/10.1109/CLOUD.2016.0060.

Mohan, A., Sane, H., Doshi, K., Edupuganti, S., Nayak, N., Sukhomlinov, V., 2019. Agile cold starts for scalable serverless. In: 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19). USENIX Association, Renton, WA, URL https://www.usenix.org/conference/hotcloud19/presentation/mohan.

Mohanty, S.K., Premsankar, G., Di Francesco, M., et al., 2018. An evaluation of open source serverless computing frameworks. In: CloudCom. pp. 115–120. http://dx.doi.org/10.1109/CloudCom2018.2018.00033.

Niu, X., Kumanov, D., Hung, L.-H., Lloyd, W., Yeung, K.Y., 2019. Leveraging serverless computing to improve performance for sequence comparison. In: Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics. In: BCB '19, Association for Com- puting Machinery, New York, NY, USA, pp. 683–687. http://dx.doi.org/10.1145/3307339.3343465.

Oakes, E., Yang, L., Zhou, D., Houck, K., Harter, T., Arpaci-Dusseau, A., Arpaci-Dusseau, R., 2018. SOCK: rapid task provisioning with serverless-optimized containers. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). USENIX Association, Boston, MA, pp. 57–70, URL https://www.usenix.org/conference/atc18/presentation/oakes.

Ou, Z., Zhuang, H., Nurminen, J.K., Ylä-Jääski, A., Hui, P., 2012. Exploiting hardware heterogeneity within the same instance type of amazon EC2. In: Fonseca, R., Maltz, D.A. (Eds.), Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'12). USENIX Association, URL https://www.usenix.org/conference/hotcloud12/workshop-program/presentation/ou.

Papadopoulos, A.V., Versluis, L., Bauer, A., Herbst, N., von Kistowski, J., Ali-Eldin, A., Abad, C.L., Amaral, J.N., Tuma, P., Iosup, A., 2019. Methodological principles for reproducible performance evaluation in cloud computing. IEEE Trans. Softw. Eng. (TSE) 93–94. http://dx.doi.org/10.1109/TSE.2019.2927908.

Pelle, I., Czentye, J., Dóka, J., Sonkoly, B., 2019. Towards latency sensitive cloud native applications: A performance study on AWS. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). pp. 272–280. http://dx.doi.org/10.1109/CLOUD.2019.00054.

Pérez, A., Moltó, G., Caballer, M., Calatrava, A., 2019. A programming model and middleware for high throughput serverless computing applications. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. In: SAC '19, ACM, New York, NY, USA, pp. 106–113. http://dx.doi.org/10.1145/3297280.3297292.

Pu, Q., Venkataraman, S., Stoica, I., 2019. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association, Boston, MA, pp. 193–206, URL https://www.usenix.org/conference/nsdi19/presentation/pu.

Puripunpinyo, H., Samadzadeh, M.H., 2017. Effect of optimizing java deployment artifacts on aws lambda. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 438–443. http://dx.doi.org/10.1109/INFCOMW.2017.8116416.

Saha, A., Jindal, S., 2018. EMARS: efficient management and allocation of resources in serverless. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 827–830. http://dx.doi.org/10.1109/CLOUD.2018.00113.

Scheuner, J., Leitner, P., 2018. Estimating cloud application performance based on micro-benchmark profiling. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, pp. 90–97. http://dx.doi.org/10.1109/CLOUD.2018.00019.

Scheuner, J., Leitner, P., 2020. Replication package for function-as-a-service performance evaluation: a multivocal literature review, v1.0. http://dx.doi.org/10.5281/zenodo.3906613, dataset.

Scheuner, J., Leitner, P., Cito, J., Gall, H., 2014. Cloud WorkBench – infrastructure-as-Code based cloud Benchmarking. In: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science. IEEE, pp. 246–253. http://dx.doi.org/10.1109/CloudCom.2014.98.

Shahrad, M., Balkind, J., Wentzlaff, D., 2019. Architectural implications of function-as-a-service computing. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO. ACM, pp. 1063–1075. http://dx.doi.org/10.1145/3352460.3358296.

Shillaker, S., 2018. A provider-friendly serverless framework for latency-critical applications. In: 12th Eurosys Doctoral Workshop, Porto, Portugal. URL http://conferences.inf.ed.ac.uk/EuroDW2018/papers/eurodw18-Shillaker.pdf.

Singhvi, A., Banerjee, S., Harchol, Y., Akella, A., Peek, M., Rydin, P., 2017. Granular computing and network intensive applications: Friends or foes?. In: Proceedings of the 16th ACM Workshop on Hot Topics in Networks. In: HotNets-XVI, ACM, New York, NY, USA, pp. 157–163. http://dx.doi.org/10.1145/3152434.3152450.

Somu, N., Daw, N., Bellur, U., Kulkarni, P., 2020. Panopticon: A comprehensive benchmarking tool for serverless applications. In: 2020 International Conference on COMmunication Systems NETworkS (COMSNETS). pp. 144–151. http://dx.doi.org/10.1109/COMSNETS48256.2020.9027346.

Spillner, J., Al-Ameen, M., 2019. Serverless literature dataset. http://dx.doi.org/10.5281/zenodo.2649001.

Spillner, J., Mateos, C., Monge, D.A., 2018. FaaSter, better, cheaper: The prospect of serverless scientific computing and HPC. In: Mocskos, E., Nesmachnow, S. (Eds.), High Performance Computing, Vol. 796. Springer International Publishing, Cham, pp. 154–168. http://dx.doi.org/10.1007/978-3-319-73353-1_11.

Taibi, D., El Ioini, N., Pahl, C., Niederkofler, J.R.S., 2020. Patterns for serverless functions (function-as-a-service): A multivocal literature review. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20). http://dx.doi.org/10.5220/0009578501810192.

van Eyk, E., Scheuner, J., Eismann, S., Abad, C.L., Iosup, A., 2020. Beyond microbenchmarks: The spec-rg vision for a comprehensive serverless benchmark. In: Companion of the ACM/SPEC International Conference on Performance Engineering, ICPE '20, ACM/SPEC. pp. 26–31. http://dx.doi.org/10.1145/3375555.3384381.

Wang, L., Li, M., Zhang, Y., Ristenpart, T., Swift, M., 2018. Peeking behind the curtains of serverless platforms. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). USENIX Association, Boston, MA, pp. 133–146, URL https://www.usenix.org/conference/atc18/presentation/wang-liang.

Werner, S., Kuhlenkamp, J., Klems, M., Müller, J., Tai, S., 2018. Serverless big data processing using matrix multiplication as example. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 358–365. http://dx.doi.org/10.1109/BigData.2018.8622362.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., 2012. Experimentation in Software Engineering, first ed. Springer, http://dx.doi.org/10.1007/978-3-642-29044-2.

Yussupov, V., Breitenbücher, U., Leymann, F., Wurster, M., 2019. A systematic mapping study on engineering function-as-a-service platforms and tools. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, UCC'19. ACM, New York, NY, USA, pp. 229–240. http://dx.doi.org/10.1145/3344341.3368803.

Zhang, H., Babar, M.A., Tell, P., 2011. Identifying relevant studies in software engineering. Inf. Softw. Technol. 625–637. http://dx.doi.org/10.1016/j.infsof.2010.12.010.

Zhang, M., Zhu, Y., Zhang, C., Liu, J., 2019. Video processing with serverless computing: A measurement study. In: Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. In: NOSSDAV '19, ACM, New York, NY, USA, pp. 61–66. http://dx.doi.org/10.1145/3304112.3325608.