The State of Serverless Applications: Collection, Characterization, and Community Consensus

Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, Alexandru Iosup

Abstract—Over the last five years, all major cloud platform providers have increased their serverless offerings. Many early adopters report significant benefits for serverless-based over traditional applications, and many companies are considering moving to serverless themselves. However, currently there exist only few, scattered, and sometimes even conflicting reports on when serverless applications are well suited and what the best practices for their implementation are. We address this problem in the present study about the state of serverless applications. We collect descriptions of 89 serverless applications from open-source projects, academic literature, industrial literature, and domain-specific feedback. We analyze 16 characteristics that describe why and when successful adopters are using serverless applications, and how they are building them. We further compare the results of our characterization study to 10 existing, mostly industrial, studies and datasets; this allows us to identify points of consensus across multiple studies, investigate points of disagreement, and overall confirm the validity of our results. The results of this study can help managers to decide if they should adopt serverless technology, engineers to learn about current practices of building serverless applications, and researchers and platform providers to better understand the current landscape of serverless applications.

1 Introduction

Serverless computing is an emerging technology with increasing impact on our modern society, and increasing adoption by academia and industry [1], [2], [3]. The key promise of serverless computing is to make computing services more accessible, fine-grained, and affordable [4], [5], by having the infrastructure manage all operational concerns [6]. Major cloud providers, such as Amazon, Microsoft, Google, and IBM, already offer capable serverless platforms with well-defined responsibilities and pricing. However, serverless computing, and its common Function-as-a-Service (FaaS) realization, still raises many important challenges that may reduce adoption. These challenges have been recognized and discussed in fields such as software engineering, distributed systems, and performance engineering [7], [8], [9]. An important challenge that remains open is to present a clear view on the state of serverless applications for managers, engineers, and scientists. This work proposes a mixed-method approach to understand the state of serverless applications.

There exist only few, scattered, and sometimes conflicting reports addressing important questions such as *Why developers build serverless applications?*, *When are serverless applications useful?*, or *How are serverless applications implemented in practice?* For example, although some report significant cost-savings by switching to serverless applications [10], [11], others identify in some scenarios a higher cost compared to traditional hosting [12]. Similarly, although reports of successful serverless applications for data-intensive applications exist [13], [14], other reports claim that serverless is not well suited for data-intensive applications [9]. As a third and last example, although a recent study differentiates between containers and serverless and finds the former to be preferable for latency-critical tasks [15], others see them as connected [16], [17] or report successfully applying serverless to latency-critical, user-facing traffic [18]. Having concrete information on these topics would be valuable for developers, to guide decisions on whether serverless is a suitable paradigm for their specific application.

However much needed, systematic studies about serverless applications still do not exist. For serverless computing, existing research has focused on serverless platforms and their performance properties [19]. Pioneering studies about the features, architecture, and performance properties of these platforms [20], [21], [22], [23], [24], [25] do not study systematic collections of applications. Shahrad et al. [26] characterize the aggregated performance properties of the entire production FaaS workload from Microsoft Azure Functions, but do not provide details on individual applications. A recent mixed-method empirical study investigates how developers use serverless computing, focusing on mental models and the issues (pain points) developers experience [27]. Another multivocal literature review discusses simple patterns common in the architecture of serverless applications [28], but do not analyze the applications themselves. The only existing collection of serverless applications is linked to an article by Castro et al. [6], which introduces ten applications collected from non-peer-reviewed (*industrial*) literature.

- *Simon Eismann, Johannes Grohmann, and Nikolas Herbst are with the Department of Software Engineering, University of Würzburg, Germany. E-mail: firstname.lastname@uni-wuerzburg.de*
- *Joel Scheuner is with the Division of Software Engineering, Chalmers | University of Gothenburg, Sweden. Email: scheuner@chalmers.se*
- *Alexandru Iosup and Erwin van Eyk are with the Massivizing Computer Systems at the Vrije Universiteit Amsterdam. Email: A.Iosup@atlarge-research.com and E.vanEyk@atlarge-research.com*
- *Maximilian Schwinger is with the Department of Software Engineering, University of Würzburg, Germany, and the German Aerospace Center. Email: maximilian.schwinger@dlr.de*
- *Cristina L. Abad is with the Department of Electrical Engineering and Computer Science at Escuela Superior Politecnica del Litoral, Ecuador. Email: cabadr@espol.edu.ec*

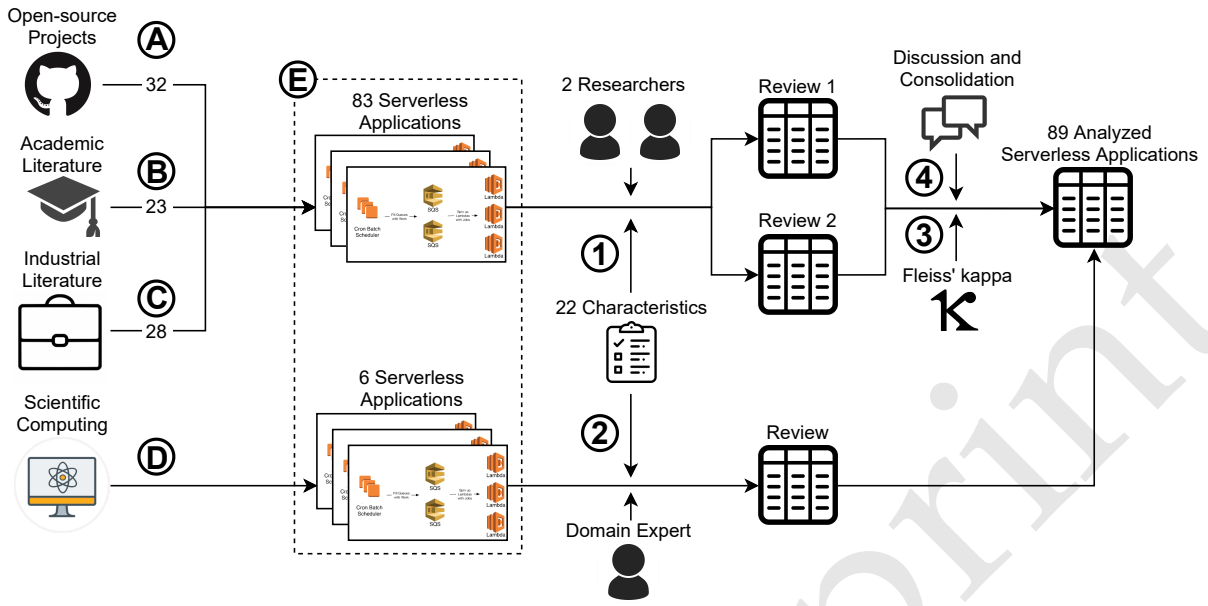


Fig. 1: Methodology for serverless application collection (left part, Section 2) and characterization (right part, Section 3).

This article makes three main contributions towards furthering the understanding of serverless applications:

1) **Systematic collection of serverless applications, the largest to date** (in Section 2): Building on resources created by the community [29], [30], we systematically collect a total of 89 serverless applications from four different sources. 32 applications are from open-source projects, 23 from academic literature, 28 from industrial literature, and 6 from the area of scientific computing; this is the largest collection of serverless applications to date, by a factor of 8.9x over the next largest [6].
2) **First systematic and comprehensive characterization of serverless applications** (in Section 3): We characterize each application from our collection, through a systematic and comprehensive pair-reviewing process, with regard to 16 characteristics, such as execution pattern, workflow coordination, use of external services, and motivation for adopting serverless. We have presented a small subset of these results for general magazine audience [31], but here we present the technical details and identify for the first time where serverless helps (e.g., from APIs to batch processing), common traffic patterns for serverless applications, workflow complexity and coordination, etc. The underlying dataset is described in detail in our technical report [32].
3) **First analysis of community consensus** (in Section 4): To understand whether the community can reach consensus across its attempts to characterize serverless applications, we systematically contrast our and previous community results. We conduct a literature search, finding 10, mostly industrial, web surveys and datasets on the characteristics of serverless applications. Next, we compare the results of these studies to this study, to identify characteristics for which there is a consensus among multiple studies and investigate points of disagreement (which give opportunities for further research).

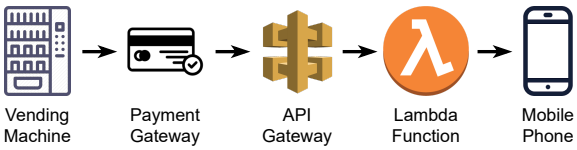
2 Serverless Application Collection

In this section, we create a process to collect serverless applications, and show the main result of using it—the largest public collection of serverless applications, to date.

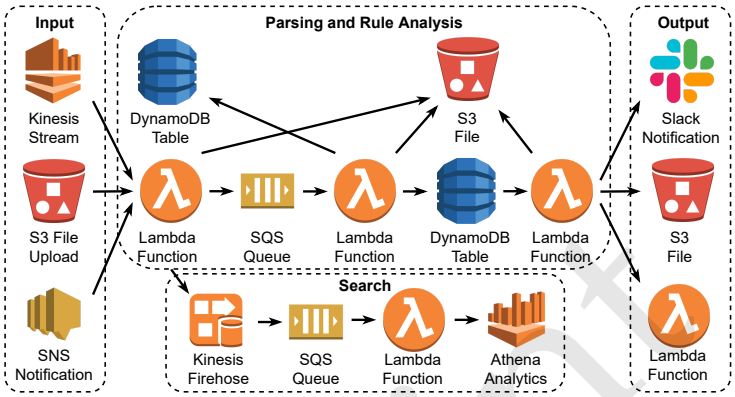
2.1 Methodology

Serverless applications have been described in many kinds of materials written for experts including peer-reviewed academic publications, open-source projects, blog posts, podcasts, talks, and provider-reported success stories. The field is only a few years old, so any of these types of materials could include meaningful and unique material. We aim to create a process for collecting a large amount of descriptions of serverless applications, spanning this range of materials judiciously and without a strong selection bias toward one or another. Our aim is not that the process should be exhaustive; doing so while the field is still growing and new applications are still emerging would not be useful long-term. Figure 1 shows the result of our use of this process—a large, varied sample, obtained from the following sources:

- **Open-source projects** (Figure 1, component A): We start with an existing dataset on open-source serverless projects [29]. We remove small and inactive projects based on the number of files, commits, contributors, and watchers. Next, we manually filter the resulting dataset to retain only projects that implement serverless applications. Finally, we select only projects that have an active and appreciative community (projects with over 50 stars). This results in a set of 32 serverless applications from open-source projects.
- **Academic literature** (Figure 1, B): We base our search on an existing, community-curated dataset on literature for serverless computing of over 180 peer-reviewed articles [30]. As the authors are familiar with 5 additional publications describing serverless applications,



(a) Vending machine backend at Coca-Cola.



(b) StreamAlert by AirBnB.

Fig. 2: Two examples of serverless applications from our collected dataset of 89 serverless applications.

we contribute them to the community-curated dataset and include them in this study. We first filter all the articles based on title and abstract, and remove any articles that implement only a single function for evaluation purposes or do not include sufficient detail to enable a review. This results in 23 serverless applications from academic literature.

- **Industrial literature** (Figure 1, C): There are many blog posts by companies or individuals, talks at industry conferences, and provider-reported success stories that describe serverless applications. We filter the case studies reported by the major serverless providers (AWS, Azure, Google, and IBM) and select from them solutions that depend on serverless technology. We also include the 10 applications reported in a recent article [6], which until this work is the largest public collection of serverless applications from industrial literature. We further extend this collection with industrial literature describing serverless applications in main industry events (e.g., KubeCon), etc. This process results in 28 serverless applications from industrial literature.
- **Scientific computing** (Figure 1, D): The scientific computing community is showing increasing interest in serverless solutions (e.g., at NASA [33] and CERN [34]). However, most of these applications are still at an early stage, with scarce public information. To address this deficit of public data, we collect and include in this work information from the German Aerospace Center (DLR) and from the German Electron Synchrotron (DESY). This results in 6 serverless applications from the area of scientific computing.

For each of these sources, we use the same predefined inclusion (I) and exclusion (E) criteria to determine if an application should be included in our dataset:

I1 Concrete application. Real world use is a plus.

I2 Application description has sufficient detail to conduct meaningful review. (Exclude high-level descriptions that lack technical detail.)

E1 Serverless platforms and frameworks, as these are not serverless applications.

E2 Boilerplate code and simple technology demonstrations, as they do not constitute real-world applications.

I3/E3 Include only one out of multiple academic papers describing the same use case. For example, many academic papers discuss serverless neural network serving [35], [36], [37], but we only include a single representative paper.

To ensure that our application collection process is transparent and reproducible, we have included further details on this process in our replication package.[1]

2.2 Resulting collection

Finding 1: About half of the 89 serverless applications in our dataset are used in production, and about half of them are open source. However, only few applications are both used in production and open source.

We collect a diverse dataset of 89 serverless applications from open-source projects, academic literature, industrial literature, and scientific computing based on the methodology in Section 2.1. This dataset (see Figure 1, component E) is publicly available as part of our replication package.[1] Out of the total of 89 applications, 55% are used in production, and 53% are open source. Researchers can use this dataset to study different applications, which facilitates extracting meaningful patterns and could trigger new designs. The dataset can also help with identifying representative applications, which can later be used for the evaluation of novel approaches and in empirical studies. Engineers can find in the dataset useful examples and identify areas in which serverless computing is successfully applied, to help decide whether to adopt serverless computing and to select blueprints for similar use-cases. Platform providers can extract knowledge on how their products are used and thus optimize them, and gaps in adoption that can point out deficits in current platform capabilities.

Figure 2 shows two example serverless applications from our dataset. Figure 2(a) depicts the serverless backend of Coca-Cola vending machines—an operation that handles 30 million requests per year. Figure 2(b) illustrates the

1. https://doi.org/10.5281/zenodo.5185054

open-source application StreamAlert, by AirBnB, which allows the validation of security rules on streams of log data. Both applications use the *cloud provider* AWS, but are implemented in different *programming languages*. The architecture of these applications is different: whereas the vending machine uses a single *external service*, a managed cloud API gateway, StreamAlert uses several, including managed databases, managed streaming, managed queues, and managed storage. They use different *trigger types*, HTTP requests for the vending-machine backend, cloud events for StreamAlert. The *number of serverless functions* also differs: whereas the vending machine backend uses a single serverless function, StreamAlert consists of many serverless functions. The workload of both applications further differs in *execution pattern*, *burstiness*, and *data volume*. The vending machine backend focuses on cost savings as the *motivation* behind adopting serverless, whereas StreamAlert seems to choose serverless to avoid operational overheads.

Motivated by this comparison, we focus in the next section on analyzing these and more characteristics for all serverless applications in our dataset.

3 Serverless Application Characteristics

This section describes our methodology to identify and analyze characteristics of serverless applications. We analyze the dataset collected in Section 2, using six general questions about serverless applications.

3.1 Methodology

Figure 1, points 1–4, gives an overview of our methodology to identify the characteristics of serverless applications. Although we apply the methodology described in the following on the dataset collected in Section 2, the characterization methodology we introduce here could be applied to other, similar datasets. This level of generality allows further comparison between studies, a feature we leverage to conduct our own cross-community study, in Section 4.

We first identify and formalize the set of investigated characteristics through a multi-round process. In an initial round, we start from a set of questions (headers in Section 3.2), and each author suggests characteristics independently, based on expertise. In the next round, we merge similar characteristics and retain only the characteristics that at least two authors consider relevant, in this work, 22 characteristics. Based on group discussion, we further define for each characteristic either the range of values or an exhaustive set of potential values, as applicable. For some characteristics, we cannot define a set of potential values before reviewing the applications. For these characteristics, we use text fragments during the review. Using thematic coding [38], [39], we extract codes and treat those as values for these characteristics.

We then conduct an initial round of reviews (Figure 1, label ①). Each application is assigned two reviewers out of a pool of seven available reviewers (all are authors). We manually adjust a few reviewer assignments to reduce the number of coinciding reviewer pairs. Subsequently, each reviewer individually assigns values to all characteristics of their assigned applications. For the scientific applications, a different approach was necessary, because many were not publicly available at the time of our review. Therefore, these applications are reviewed by a single domain expert, which was either involved in the development of the application or in direct contact with the development (Figure 1, label ②). Our replication package contains descriptions of the scientific use cases and outline which domain experts were consulted for each application.

Each review of an application characterizes it according to 22 characteristics: *cloud platform*, *programming languages*, *external services*, *trigger types*, *number of functions*, *execution pattern*, *burstiness*, *data volume*, *application type*, *function runtime*, *latency relevance*, *motivation*, *cost/performance tradeoff*, *resource bounds*, *locality requirements*, *update frequency*, *domain*, *is it a workflow?*, *workflow coordination*, *workflow structure*, *workflow size*, and *workflow internal parallelism*. For each, the result is typically a value, one of the possible values for that dimension. However, if the information to determine a characteristic for a serverless application is not available, we label the characteristic as "Unknown" for this application.

After completing the initial round of reviews, we calculate the Fleiss' kappa to quantify the level of agreement between the reviewers [40] (Figure 1, label ③). We exclude all characteristics that use thematic coding and all characteristic assignments where at least one reviewer assigned more than one value, as the Fleiss' kappa can not be calculated in these cases. As each characteristic has a different number of possible values, we calculate Fleiss' kappa value for each characteristic individually and then quantify the overall agreement with a weighted average over the individual Fleiss' kappa value of each characteristic. This results in a Fleiss' kappa value of 0.48, which can be interpreted as "moderate agreement" [41].

In the following discussion and consolidation phase (Figure 1, label ④), the reviewers compare their notes and try to reach consensus for the characteristics with conflicting assignments. For most conflicts, consolidation turns out to be a quick process, as the most frequent type of conflict was that one reviewer found additional documentation that the other reviewer did not find. In only a few cases, the two reviewers still have different interpretations of a characteristic; these conflicts are discussed among all authors to ensure that characteristic interpretations are consistent. Following this process, we were able to resolve all conflicts.

Our process is data-driven, so it also has to account for missing or malformed data. For 6 characteristics (*resource bounds*, *locality requirements*, *update frequency*, *domain*, *workflow internal parallelism*, and *cost/performance tradeoff*), many applications are assigned the "Unknown" value, i.e., the reviewers were not able to determine the value of this characteristic, as the required information was missing in the documentation. Therefore, we exclude these characteristics from this study.

For the remaining 16 characteristics, the percentage of "Unknowns" ranges from 0–19%, with two outliers at 25% and 30%. These "Unknowns" are excluded in the percentage values presented in this article. A breakdown per characteristic of the "Unknown" percentages is available in our repli-

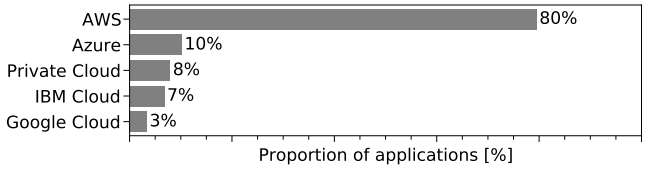


Fig. 3: Cloud provider used for serverless applications.

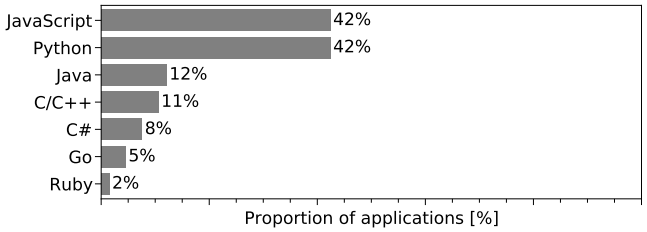


Fig. 4: Programming language used for serverless application.

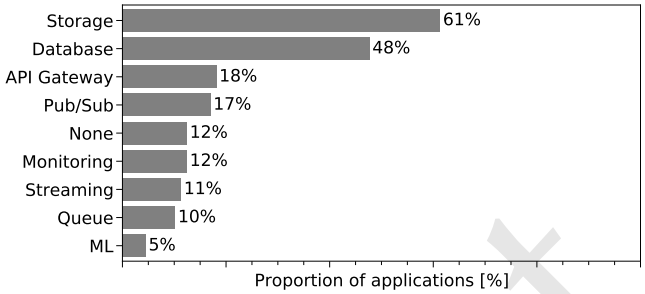


Fig. 5: Managed services used by serverless applications.

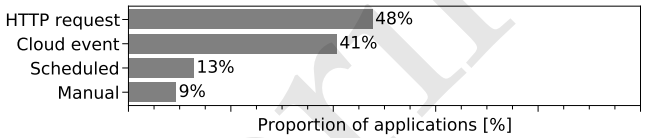


Fig. 6: Trigger types used in serverless applications.

cation package.[1] Additionally, for a single characteristic (the *application type*), the list of potential values turns out to be inadequate, so we repeat the mapping for this characteristic with a new set of potential values.

3.2 Resulting Characteristics

In the following, we describe the serverless application characterization results in the context of six common questions about serverless applications, in turn.

3.2.1 How are serverless applications implemented?

Finding 2: Currently, AWS is the dominating platform for serverless applications (80%), and most applications are implemented in either JavaScript or Python (42% each).

When AWS revealed Lambda in 2014, it was the only FaaS platform offered by a major cloud provider, and it only supported JavaScript functions. Since then, tens of serverless platforms have emerged [20, §3], offering support for diverse programming languages. The capabilities and performance features of these platforms and languages have been studied extensively [42]. We focus here on the state of practice in *implementing* serverless applications.

We analyze the collection of serverless applications introduced in Section 2. Figure 3 shows that 80% of the applications in our dataset are using the cloud provider AWS, whereas the other major cloud providers are used a lot less often (10% Azure, 7% IBM Cloud, and 3% Google Cloud). Although AWS also has the largest market share when it comes to IaaS at 47.8% [43], this difference alone is not enough to explain why so many of the applications in our dataset are using AWS. A potential explanation is that AWS Lambda was released two years before any other large cloud provider released their Function-as-a-Service solution, which means this platform is likely more mature and there was more time for customers to adopt its serverless features. Since then, many open-source Function-as-a-Service solutions launched [20], yet we do not see significant adoption for them in our dataset (a combined 8%, mostly by scientific applications). We observe that most applications in our dataset use managed cloud services that would not be available in a private cloud environment; this could explain the low adoption of the open-source Function-as-a-Service solutions and also spur innovators in serverless technology to consider more carefully the ecosystem where their platforms can work.

Interpreted languages could be better suited for serverless applications than compiled languages, because compiled languages suffer from longer cold-starts [44]. Figure 4 corroborates this rule-of-thumb: JavaScript (42%) and Python (42%) are the most popular programming languages. Serverless applications are also written in Java (12%), C/C++ (11%), or C# (8%); few use Go (5%) or Ruby (2%). However, this may change, as the usage of ahead-of-time compilation, e.g., for Java, has been shown to alleviate the difference in cold-start durations [45].

3.2.2 How does a typical serverless architecture look?

Finding 3: Serverless applications typically use cloud storage (61%), cloud databases (47%), and cloud messaging (38%). They use few cloud functions: 82% of serverless applications use 5 or fewer functions.

Developers looking to implement serverless applications need to make many architectural decisions, such as which external services to use, how many functions to use, and how they are triggered. Understanding patterns in serverless architecture can guide the general discourse on serverless applications and provide a valuable guideline for developers starting to build serverless applications.

Figure 5, label *None*, shows that only 12% of the serverless applications in our dataset do not use any managed service. This suggests serverless applications are typically created by combining serverless functions for compute and managed cloud services for other operations. The most frequently used managed services in our dataset are storage (61%) and databases (48%). Serverless functions are stateless, therefore all application state needs to be persisted

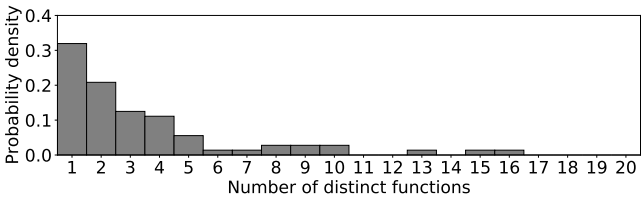


Fig. 7: Number of serverless functions per serverless application.

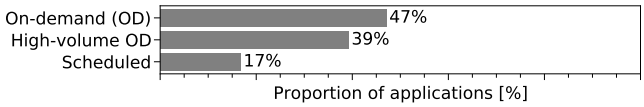


Fig. 8: Execution pattern of serverless applications.

in external storage and databases. The second class of most frequently used external services are managed messaging services, including publish/subscribe solutions (17%), streaming solutions (11%), and queuing solutions (10%). Serverless functions often use such messaging services to store their output if it needs to be processed further.

Cloud providers offer different ways to trigger the execution of serverless functions. As Figure 6 depicts, we find that 48% of the selected applications use HTTP triggers, and a further 41% use cloud events triggers (e.g., as a new message in a queue, or a new entry in a database). Generally, HTTP triggers are commonly used to expose functionality to users, whereas cloud events help coordinate multiple cloud functions. This is a significant change from microservices, which typically rely on API calls to coordinate multiple services. One of the reasons for this change towards an event-driven architecture could be that synchronous calls between serverless functions cause double billing [46]. A smaller number of applications use schedule-based triggers (13%) or manually triggered functionality (9%). These triggers are usually used for orchestration or management tasks.

Figure 7 shows the number of functions per application is relatively low: Only 7% of the applications in our dataset use more than 10 functions, and 82% use 5 or fewer functions. This suggests, firstly, that the use of external services reduces the amount of (internal) code required to build an application. Secondly, the functionality encapsulated by a serverless function is between a microservice and an API endpoint, as the applications we review do not wrap every programming function as a serverless function [47]. The term "serverless functions" might be misleading, as they are not related to the programming concept of functions.

3.2.3 What are common traffic patterns for serverless applications?

Finding 4: Most serverless applications have potentially bursty workloads (84%). Serverless applications are often used for high-traffic workloads (39%).

Traffic patterns—namely, execution patterns, burstiness characteristics, and data volumes—can reveal how serverless platforms are used. Applications can be executed *on-demand* when a user interacts with the application or a cloud

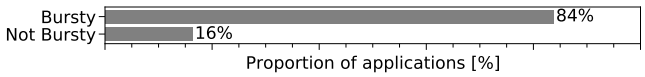


Fig. 9: Burstiness of the workload of serverless applications.

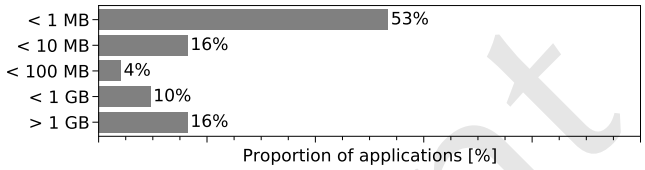


Fig. 10: Data volume handled by serverless applications.

event occurs; we further classify the on-demand execution as regular *on-demand* or *high-volume on-demand*. Applications can also be *scheduled* to run at specific times, e.g., to perform cleanup tasks during off-hours.

Regarding the *execution patterns*, Figure 8 shows that most applications are triggered on-demand (86%), out of which more than half are high-volume invocations, associated with business-critical functions. Only 17% of the applications are triggered by a periodic schedule. Through an in-depth analysis, we find that about half of the scheduled applications execute *operations & monitoring* functions (see also Section 3.2.4), highlighting how the serverless model has been adopted—in many cases—to automate operations, software management, and DevOps pipelines. We also note that the high prevalence of on-demand triggered applications, and specifically, high-volume on-demand patterns, is well supported by the industry trends of reducing overheads (i.e., function start-up time), and of providing quick and seamless function auto-scaling mechanisms.

Regarding *burstiness*, we classify applications as having potentially bursty workloads or non-bursty workloads. A *bursty application* follows a workload pattern that includes sudden and unexpected load spikes, or a significant amount of sustained noise and variation in intensity. We classify an application as *non-bursty* if the workload is guaranteed to rarely or never experience bursts (e.g., if all executions are scheduled and known in advance); otherwise, the workload is bursty. When humans trigger function executions, the workload pattern can be bursty, as user behavior can rarely be scheduled or reliably controlled. As Figure 9 shows, we classify more than 84% of the workload patterns we analyzed as bursty; only 16% have a clear non-bursty pattern. As one of the strengths of serverless computing is its seamless scalability, together with the general ease of operations, it comes as no surprise that most of the applications can indeed experience bursty workload patterns.

Finally, we analyze the *data volume* or load that the serverless apps issue on the network and storage devices. We classify applications into: volumes of *less than 1 MB* per execution, *less than 10 MB*, *less than 100 MB*, *less than 1 GB*, and *more than 1 GB*. Exact numbers rarely appear in our sources so this classification is based on reviewers' estimates. Figure 10 shows (i) more than half of the applications (53%) in the smallest category of data volumes and 16% in the next (<10 MB) and (ii) the second peak (16%)

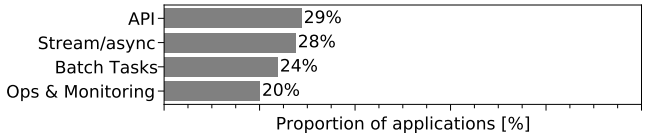


Fig. 11: Application type of serverless applications.

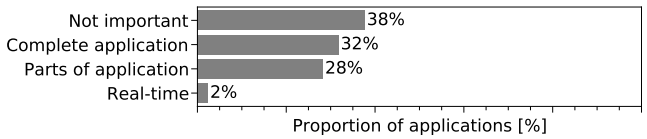


Fig. 12: Latency requirements of serverless applications.

in the largest category (> 1 GB). The resulting distribution appears bimodal, but this might be an artifact of the binning intervals.

3.2.4 What are serverless applications used for?

Finding 5: Serverless applications are not limited to any specific types of applications, as they are commonly used to implement APIs (29%), stream/async processing (28%), batch tasks (24%), and operations tasks (20%).

A common assumption is that serverless applications are suitable for operations tasks and batch jobs, as their traffic patterns profit from the pay-per-use model. For example, Netflix uses AWS Lambda for operations tasks, such as video encoding, file backup, security audits of EC2 instances, and monitoring. However, the core functionality—the website and app backend, and video delivery—is still running on traditional IaaS cloud services [48]. Contrary to this popular belief, and as Figure 11 depicts, we find that the most common serverless applications in our dataset are implementing APIs (29%) or are processing frequent events (streams) asynchronously (28%). Example use cases for these types of applications are serverless backends for web, mobile, or IoT applications. Still, a significant portion of serverless applications focus on processing batch tasks (24%) and on automating operations tasks (20%).

Another common assumption is that serverless applications are not suitable for complex analysis tasks. In contrast, in our dataset, 25% of applications contain functions with an estimated runtime of over one minute. Among these applications are scientific workloads, such as SNP Genotyping [14] or seismic imaging [13], showing an increased adoption of serverless for complex analysis tasks.

The high percentage of APIs is somewhat surprising, as a common argument against serverless applications is that cold starts make them unsuitable for applications with low-latency requirements or focus on tail-latency. However, we find that serverless applications are used for latency-critical tasks. As shown in Figure 12, 38% of the selected serverless applications have no latency requirements. However, 32% of the serverless applications have latency requirements for all functionality, 28% have partial latency requirements, and 2% even have real-time requirements.

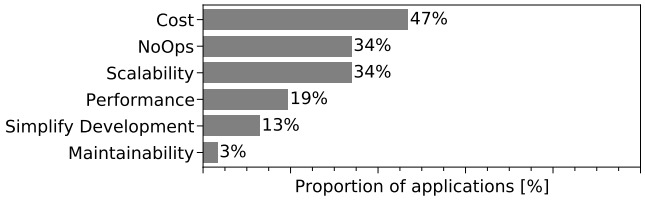


Fig. 13: Motivation for building serverless applications.

3.2.5 Why are practitioners choosing serverless?

Finding 6: Reduced hosting costs of serverless applications (47%), reduced operation effort (34%), and high scalability (34%) are the main drivers for serverless adoption.

Several potential benefits of serverless applications have been proposed: reduced operational effort, faster development due to the heavy use of Backend-as-a-Service, and near-infinite scalability of serverless applications. Many also discuss significant cost savings from switching to serverless. However, these benefits are not generally agreed upon, for example, cost savings have come under scrutiny [49]. To understand why practitioners choose to adopt serverless, we investigate the descriptions and documentation of applications in our dataset.

We could not determine the reasons behind the adoption of serverless for about 30% of the applications in our dataset, as the documentation did not mention explicitly why serverless was chosen. We analyze the remainder and depict the results in Figure 13. The main driver is cost—mentioned by 47% of the remainder applications. While serverless is not *per se* cheaper than IaaS hosting, the pay-per-use model and ability scale to zero reduce costs in scenarios where the IaaS resources are underutilized. Serverless applications can also offer seamless, virtually infinite scaling. Scalability is mentioned as a reason for serverless adoption by 34% of the applications in our dataset. The third main reason for choosing serverless over traditional hosting options is reduced operational overhead, because server management is no longer done by applications. A few applications also reported improved performance (19%) and faster development speed (13%) as reasons for serverless adoption.

3.2.6 How complex are serverless applications?

Finding 7: Almost a third (31%) of the serverless applications are workflows. Most workflows are of simple structure, small, and short-lived.

Although initially serverless focused on simple applications, comprised of mainly small functions, there has been increasing interest in using serverless for more complex applications. Such applications can be expressed as *serverless workflows*, which orchestrate the dependencies between multiple functions.

From our dataset, we find a significant percentage of applications already structured as serverless workflows (31%). Examples of such serverless workflows can range from simple workflows to large scientific workflows [13].

Similar to workflows in other fields, we can classify these serverless workflows into specific patterns based on

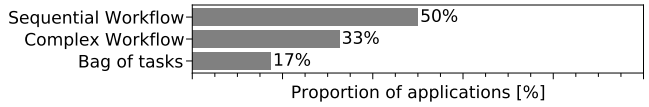


Fig. 14: Structure of serverless workflows.

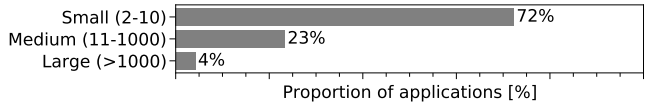


Fig. 15: Size of serverless workflows.

how the function calls (or tasks) are structured within these orchestrations. As Figure 14 depicts, we find that half of the workflows are sequential in nature, where tasks are executed one after the other. We also find *bags of tasks* (17%), where a set of tasks can execute without a particular order or inter-dependency. Finally, a third (33%) of the applications are defined as *complex workflows*, that is, workflows that include structures such as conditional branches and loops. The diversity and level of complexity indicate designers of workflow management systems are soon to be engaged in a competition over new features and optimizations.

Another key differentiator between the selected applications is workflow size (shown in Figure 15). Most applications (72%) have rather small workflows, consisting of 2 to 10 tasks. These tend to be applications for business processes and data pipelines, such as the multi-step provisioning of developer machines at Autodesk [50]. Almost a quarter of serverless workflows (23%) contain 11 up to 1,000 tasks. Finally, a few serverless workflows (4%) consist of more than 1,000 tasks—typically, large scientific workflows requiring custom workflow engines to run.

A final factor in serverless workflows is the approach used to orchestrate their execution. Overall, as Figure 16 highlights, we found that most serverless workflow applications (60%) use *event-based mechanisms*—such as file uploads triggering the execution of functions—to implicitly orchestrate entire workflows by ensuring that the result of one function triggers the next. About a third of the workflows (38%) are managed by a *dedicated workflow management* system, such as AWS Step Functions, Google Workflows, or Azure Durable Functions. Besides these cloud-based orchestration methods, we also identify a less-common approach, *local coordination* (2%), in which the orchestration complexity is deferred to the client-side. Although this is less robust than other methods, it is used for one-off workflows, e.g., the distributed build-workflows of gg [51].

4 Finding Community Consensus

Because the field of serverless applications is relatively new and fast-evolving, reaching community consensus about application patterns and best practices is both desirable and challenging. This section aims to analyze if existing studies that analyze one or several characteristics that we also study, and determine if overlapping studies corroborate our findings or contradict them. The results are a necessary first step towards reaching community consensus.

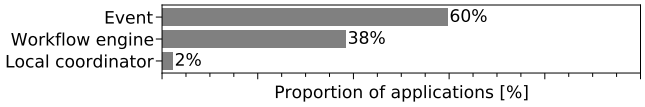


Fig. 16: Coordination of serverless workflows.

4.1 Methodology

We aim to find and compare with existing studies in the community on the characteristics of serverless applications. Our methodology consists of three parts: a literature search to identify related studies, mapping their findings to our framework, and quantifying the degree of agreement.

4.1.1 Identification of Related Study

To identify existing surveys and datasets that also investigate at least one of the characteristics investigated in this work, we conducted a literature search. As we are mostly looking for industrial studies and datasets, we use Google as the search engine with the following search term[2]:

$$(\text{“}serverless\text{”}\ OR\ \text{“}faas\text{”})\ AND$$
$$(\text{“}dataset\text{”}\ OR\ \text{“}survey\text{”}\ OR\ \text{“}report\text{”})$$
$$after{:}\ 2018\text{-}01\text{-}01$$

This search term looks for any combination of either serverless or FaaS alongside any of the terms: dataset, survey, or report. We further limit the search to articles since 2018, as serverless is a fast-moving field, and therefore any older studies are likely outdated. This search term results in a total of *173 unfiltered results*.

To validate if using only a single search engine is sufficient, and the search term is broad enough, we check if the 7 studies that the authors were already familiar with appear in the results. Because the search results include all these studies, we conclude our literature search is broad enough.

We identified relevant results as follows. In the first iteration, to keep primary sources, we filter out results that do not report original data. We remove all reports on secondary data, where the original study was already contained in the search results. This process results in a total of *16 primary studies*. Finally, we determine for each primary study if they investigate one of our characteristics. This resulted in a total of *10 related studies*, which Table 1 summarizes. The related studies include 7 surveys and 3 datasets, survey from 19 to 2 400 participants, and report between 2018 and 2020.

In the following, we give a short description of each related study, as the methodology and context of each study are important for the correct interpretation of their results.

Serverless Community Study (SCS): This is an online survey among 583 participants from the serverless community, conducted in April 2020. It mostly focuses on end-user concerns, such as how far the end-user is in adopting serverless and what challenges they experience.

2. https://www.google.com/search?q=%28%22serverless%22+OR+%22faas%22%29+AND+%28%22dataset%22+OR+%22survey%22+OR+%22report%22%29+after%3A2018-01-01

TABLE 1: Overview of the related studies.

Study	Year	Type	Participants	Source
SitW	2020	Dataset	-	https://bit.ly/3bl2vHM
TSoS	2020	Dataset	-	https://bit.ly/2Zp9zOh
FtLoS	2020	Dataset	-	https://bit.ly/3diWZrY
SCS	2020	Survey	583	https://bit.ly/37p56j4
FSS	2020	Survey	~150	https://bit.ly/2ZsIVUM
OSS	2019	Survey	>1500	https://bit.ly/3dnViJH
MMS	2018	Survey	182	https://bit.ly/3dpcJd6
DSS	2018	Survey	19	https://bit.ly/3qybX15
CNCF	2018	Survey	2400	https://bit.ly/2M2sjQz
GtST	2018	Survey	608	https://bit.ly/3biElxO

Serverless in the Wild (SitW): In 2020, researchers at Microsoft published one of the first comprehensive characterization studies of the workloads of a major (and closed-source) serverless platform. For this, they release all function invocations on the Azure Functions platform for two weeks.

Mixed-Method Study (MMS): The academic mixed-method study combines semi-structured practitioner interviews with 12 experts, a systematic review of 50 grey literature articles, and a quantitative survey covering 182 responses to investigate FaaS software development in industrial practice. Our study only compares against their web survey results from early 2018.

The State of Serverless (TSoS): This study compiles usage-data from the customer base of Datadog, a vendor of serverless monitoring solutions. This data was published in early 2020 and focuses solely on AWS Lambda.

O'Reilly Serverless Survey (OSS): In June 2019, O'Reilly surveyed over 1,500 participants from diverse locations, companies, and industries on the adoption of serverless computing.

Guide to Serverless Technologies (GtST): As part of the ebook, "Guide to Serverless Technologies", The New Stack surveyed 608 participants interested in serverless technology. The survey participants were primarily recruited through the company's newsletter and their social media reach-out.

For the Love of Serverless (FtLoS): New Relic, a vendor for a serverless monitoring solution, analyzed serverless trends in 2020, based on data covering a sample set of the trillions of serverless events that their product processes.

Fastly Serverless Survey (FSS): Soon after the launch of the beta version of Compute@Edge, Fastly conducted in the beta community a survey about trends and challenges.

Dashbird Serverless Survey (DSS): In 2018, Dashbird surveyed its customers on why they switched to serverless, what problems they were trying to solve, and the biggest benefits and drawbacks. The 19 companies in the survey use Dashbird's observability solution on AWS workloads.

CNCF Survey (CNCF): The Cloud Native Computing Foundation regularly surveys its community about the adoption of cloud-native technologies. The 2018 survey includes some questions on serverless adoption and platforms.

4.1.2 Mapping the Results to our Framework

Because the related studies (identified in Section 4.1.1) offer different answer-*options* than our study, we map their options to ours. In many cases, this is straightforward, e.g., when mapping "HTTP" to "HTTP Request".

When the granularities of offered options differ between studies, we aggregate lower-granularity options to match the higher-granularity. In case the lower-granularity options include multiple answers, we select only the highest value instead of aggregating values, to avoid counting a single study participant multiple times. We provide a detailed account of the mapping for each characteristic and related study as part of our replication package.[1]

4.1.3 Quantifying the Degree of Agreement

For many studies, some information required for traditional meta-analysis techniques [52], such as cohort size, is unavailable. This prevents the direct application of these meta-analysis techniques.

We propose an agreement metric, a total that equally weighs the agreement of the reported ranking and the agreement of the reported percentage values:

$$A_t = 0.5 \times A_p + 0.5 \times A_r$$

where A_t represents the total agreement, A_p the agreement of the reported percentage values, and A_r the agreement of the reported ranking, with A_p and A_r defined in the following.

We calculate the percentage agreement as the weighted mean absolute percentage error (MAPE), with the reported percentage value of each answer as the weight:

$$A_p = \sum_{i=1}^{N} Min(1, u_i) \times \frac{|u_i - t_i|}{u_i}$$

where N denotes the number of answer-options; and u_i/t_i are the percentage value reported for option i in our study and the related study, respectively. The formula caps the MAPE for each option at 100%, as otherwise options with very low percentage values would dominate the MAPE [53]. In some cases, one of the studies allows a participant to select multiple options, while the other study only allows for a single option. To compare these results, we calculate a scaling factor based on the percentage difference of the largest reported values by both studies and scale the results from the study with multiple answers per participant accordingly.

We calculate the agreement regarding the reported ranking as following:

$$A_r = \frac{S(u,t) + 1}{2}$$

We use Spearman's rank correlation coefficient $S(u,t)$, a common metric to quantify the similarity of two rankings [54]. As the Spearman's r value ranges from [-1, 1], we scale it to [0,1] so it has the same scale as A_p. Therefore, the resulting A_t also lies in the range [0, 1].

Finally, we categorize scores in the range [0.8, 1] as very *high agreement*, [0.6, 0.8) as *high agreement*, [0.4, 0.6) as *medium agreement*, [0.2, 0.4) as *low agreement*, and [0, 0.2) as *very low agreement*. We acknowledge that these categories are somewhat arbitrary. However, based on a manual inspection of the results, they seem to capture the individual studies'

TABLE 2: Degree of agreement with existing studies. A - denotes that the study did not investigate this characteristic and a (-) denotes that the results are incomparable due to differences in the question or answer options.

Characteristic	SCS	SitW	MMS	TSoS	OSS	GtST	FtLoS	FSS	DSS	CNCF
Platform	Very High	-	(-)	-	Very High	High	-	-	High	-
Language	Medium	-	High	Very High	-	Medium	Very High	-	-	-
External Services	(-)	(-)	(-)	(-)	-	-	-	-	-	-
Trigger Types	-	Very High	-	-	-	-	-	-	-	-
Number of Functions	(-)	Very High	High	-	-	Low	-	-	-	-
Execution Pattern	-	(-)	-	-	-	-	-	-	-	-
Burstiness	-	High	-	-	-	-	-	-	-	-
Data Volume	-	-	-	-	-	-	-	-	-	-
Application Type	Very High	-	(-)	-	-	High	-	(-)	-	(-)
Function Runtime	-	High	-	High	-	-	(-)	-	-	-
Is Latency relevant?	-	-	-	-	-	-	-	-	-	-
Motivation	Medium	-	Medium	-	High	High	-	Low	-	Low
Is it a workflow	-	-	-	-	-	-	-	-	-	-
Workflow coordination	-	-	-	-	-	-	-	-	-	-
Workflow structure	-	-	-	-	-	-	-	-	-	-
Workflow size	-	-	-	-	-	-	-	-	-	-

level of agreement quite well. Our replication package[1] includes the mapped data alongside the resulting scores, to enable readers to conduct manual inspections of the degree of agreement.

4.2 Results of Consensus Analysis

We analyze here the degree of agreement between the results from our study and from other studies. This meta-analysis can identify meaningful corroboration between the different studies: For the characteristics that appear both in our study and in others, a high degree of agreement with the existing studies would increase the credibility of these results. A high degree of agreement can also suggest that the results for characteristics that have not yet been investigated by any other study are also credible.

Table 2 summarizes the results; degrees of agreement are defined in Section 4.1.3. We find that 8 characteristics are also investigated by other studies, as indicated by rows where very low to very high items appear. Among these characteristics, *platform*, *language*, and *motivation* are analyzed by 4–6 other studies. For 6 characteristics, we are the first to investigate them in peer-reviewed material. For the remaining 2 characteristics, *external services* and *execution pattern*, our study uses options incomparable with other studies that considered the aspect.

In general, we find a high degree of agreement with the existing studies. For each characteristic where we observe only low or medium level of agreement with another study, we also observe high or very high agreement with another study, pointing towards differences between these studies. Only for the *motivation* characteristic, there are multiple studies relatively to which we observe low and medium level of agreement, suggesting there might be some information that our study missed.

In the following, we provide a qualitative comparison of the results from our study and the comparison studies for the eight characteristics analyzed by one or more of the comparison studies. Here, we focus on points of agreement and disagreement between the studies to obtain corroborating evidence for our findings and identify characteristics that require further investigation.

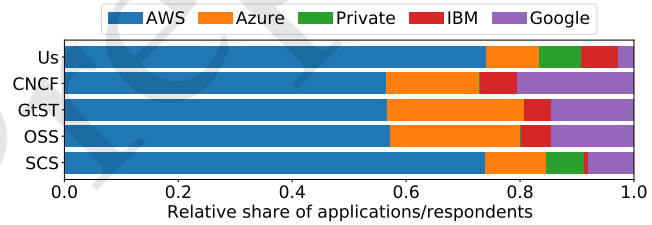


Fig. 17: Comparison of results for used cloud provider.

4.2.1 *Platform and Programming Language*

Consensus 1: AWS is the most popular serverless provider with an over 50% market share, followed by Microsoft Azure and Google Cloud. (See **Finding 2**.)

Consensus 2: JavaScript and Python are the most popular programming languages. Different studies find next a mix of Java, C#, and C/C++. (See **Finding 2**.)

Five independent studies indicate that AWS is the most popular serverless provider, followed by Microsoft Azure and Google Cloud. All five studies report a relative share of applications per respondent above 50% for AWS, as shown in Figure 17. Azure comes second in all studies except CNCF, where Google Cloud is slightly more popular. Google Cloud is ranked third in all studies except ours, which reports IBM Cloud to be more popular. IBM Cloud is the last major public serverless provider mentioned by all studies. CNCF, GtST, and SCS mention many other serverless platforms such as Cloudflare Workers, Twilio Functions, or Huawei FunctionStage. However, we excluded these hosted platforms due to low popularity (<5%) and only being mentioned by few studies. SCS and our study

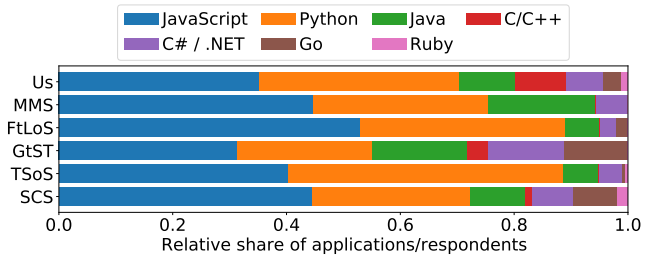


Fig. 18: Comparison of results for used programming language.

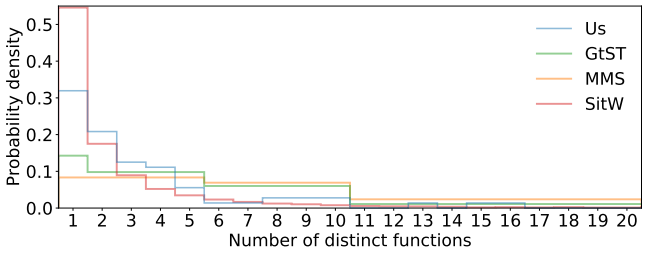


Fig. 19: Comparison of function numbers per application. The long tail of the distributions is not show (GtST: 10.3% $>$ 25 functions, MMS: 16% $>$ 20 functions, SitW: 0.25% $>$ 20 functions, Us: 1.1% $>$ 20).

grouped installable platforms into the private cloud category, including Apache OpenWhisk, Knative, Kubeless, and OpenFaas. For the other studies, the private cloud category could not be calculated due to incompatible reporting.

Figure 18 shows six studies agreeing that JavaScript and Python are the dominant programming languages in serverless applications, followed by Java and C#. The tie between JavaScript and Python in our study highlights that both languages are similarly popular across all six studies, with a minor trend towards JavaScript being more popular. Compared to broadly distributed surveys, Java appears to be more popular among enterprise newsletter respondents from GtST and the enterprise-focused survey and interview study MMS. The .NET platform with C# is also present in all six studies but generally less popular than Java. Four studies report Go as a strong contender for catching up with C#. Ruby remains a niche language listed by only three studies. In contrast to other studies, our study found C/C++ to be similarly popular to Java and C#. We assume that other studies mostly ignored this category because it refers to C/C++ binaries running under another officially supported runtime (e.g., using shims to invoke C++ binaries from JavaScript code). For GtST and TSoS, the share of programming languages is derived from telemetry data of deployed functions (rather than applications) based on their runtime configuration for the cloud provider AWS.

4.2.2 Number of Functions

Consensus 3: Nearly two-thirds of serverless applications have 10 or fewer functions. (See **Finding 3**.)

TABLE 3: Comparison of results for trigger types.

Study	HTTP	Event	Scheduled	Manual	Orchestration
SitW	0.641	0.363	0.292	-	0.094
Us	0.474	0.402	0.124	0.093	-

The number of functions per serverless application was also investigated by GtST, MMS, and SitW. Figure 19 shows a histogram of the number of functions determined by each study. The coarse binning used by the surveys prevents a detailed analysis, but a clear trend is visible. Both GtST and MMS find more functions than we do; the difference is larger for MMS. We hypothesize that this is due to differences in survey methodologies. SitW finds more single-function applications than all other studies. As SitW is specific to Azure, while the other studies predominantly cover AWS, we hypothesize that serverless applications on AWS are larger than on other platforms due to the higher maturity of the AWS serverless ecosystem. Despite disagreements on the exact distribution, all studies agree that at least 64% of serverless applications have ten or fewer functions.

4.2.3 Trigger Types

Consensus 4: HTTP requests and cloud events are the most common triggers for serverless functions. (See Section 3.2.2.)

The only other study on how serverless functions are triggered is SitW. Table 3 shows the results for the triggers HTTP request, cloud event, and scheduled triggers. For SitW, we aggregated the results for queue, storage, and event trigger as cloud events, as our definition of cloud event included those. Both studies agree that HTTP requests and cloud events are the most common triggers for serverless functions. However, SitW finds that scheduled triggers are similarly common, whereas we find them less common than HTTP and event triggers. SitW covers only functions deployed on Azure and reports a larger share of single-function apps than any other study. We hypothesize there is a difference in serverless usage between the providers, with serverless being used more for timer-based, single-function utility applications at Azure than at, e.g., AWS.

4.2.4 Burstiness

Consensus 5: More than 50% of serverless applications have potentially bursty workloads. (See **Finding 4**.)

The only other study that included information related to burstiness is the SitW study. We note that there is no standardized way of characterizing burstiness [55], [56]. The SitW study reports the coefficient of variation (c_v) of the inter-arrival times of application invocations, which we used to derive burstiness levels B in terms of the c_v following the metric proposed by Goh and Barabasi [55]. The SitW results are in high agreement with the results in our study: both studies agree that more than half of the serverless applications exhibit bursty workloads. Specifically, we found 81% of the applications exhibit bursty workloads, while 57% of the applications from the SitW study exhibited bursty behavior.

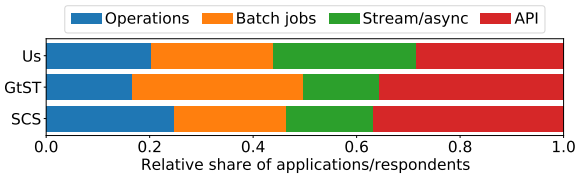


Fig. 20: Comparison of results for application type.

TABLE 4: Comparison of results for function runtime.

Study	Function Runtime <1 min	>= 1 min
SitW	0.960	0.040
TSoS	0.965	0.035
Us	0.750	0.250

4.2.5 Application Type

Consensus 6: There is no dominant application type, but several types are common. (See **Finding 5**.)

Comparing the different application type studies is not straightforward, as each study introduces its own classification type. We, therefore, had to map the categorizations of the other studies to match our taxonomy. The details can be found in our replication package. Figure 20 shows that GtST and SCS agree with the observation that serverless tasks are used for all areas of computing, including operations, batch jobs, streaming or asynchronous data, or standard API operations. Although the individual percentages differ, e.g., GtST and SCS both assign higher importance to API applications and less to stream operations, the overall picture is quite similar. In fact, all studies agree that at least 20% of serverless applications implement operations tasks, batch jobs, async processing, and APIs each.

4.2.6 Function runtime

Consensus 7: At least 75% of the serverless functions run for under 1 minute. (See Section 3.2.4.)

The runtime of serverless functions was not covered in any of the surveys, only the datasets from Azure (SitW), Datadog (TSoS), and New Relic (FtLoS) cover this information. Due to our study methodology, we only estimated if the runtime of any function of an application is less than a minute or if it is likely to run longer. Thus, we can not compare our results to the data from FtLoS, as they focus on the runtime distribution below five seconds. As Table 4 shows, the studies agree that most functions run for less than a minute, but SitW and TSoS find that over 95% of serverless functions run for less than a minute, compared to the 75% we found. The main reason for this difference should be that SitW and TSoS analyze per function runtime, whereas we analyze how many applications contain one or more functions that run longer than a minute, so the results are not directly comparable. We note that even though TSoS focuses on AWS and SitW on Azure, both agree that over 95% of serverless functions run for less than a minute.

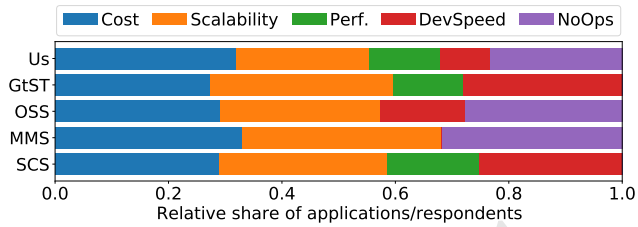


Fig. 21: Comparison of motivation for building serverless applications.

4.2.7 Motivation

Consensus 8: Cost, scalability, and NoOps are major drivers of serverless adoption. Some studies also find increased development speed as a major driver of serverless adoption. (See **Finding 6**.)

Figure 21 compares the motivations for building serverless applications. The studies had a wide range of possible and overlapping options; we grouped the answers of the comparison studies to fit the options that we identified in our study. This means we compared the studies based on motivations for cost reduction, scalability, performance, developer productivity (DevSpeed), and reduced operational complexity (NoOps). We find that cost reduction and scalability are common and key motivations mentioned in all studies. A reason for this could be that cost reduction and scalability are typical concerns for serverless developers and operators. In contrast, the other motivations—performance, developer productivity, and reduced operational complexity—vary in relative share or are even completely absent in some studies. We believe this is due to the surveys providing options for participants to select from; those options tend to be biased by what the survey is focused on. One survey (GtST) focused heavily on motivations related to the developer productivity, whereas another (MMS) targeted the operation of serverless applications.

5 Threats to Validity

We discuss potential threats to validity and mitigation strategies for internal validity, construct validity, and external validity.

5.1 Internal Validity

Manual data extraction can lead to inaccurate or incomplete data. To mitigate this threat, we established and discussed a review protocol before reviewing, continuously discussed upcoming questions during the review process, and performed redundant reviews through multiple reviewers. Our review protocol established an exhaustive list of potential values for each characteristic and configured automated validation, which immediately highlighted deviations from these values. For characteristics with thematic coding, we continuously refined their values in regular meetings during the review process. To address potential individual bias, we performed two independent reviews for each application, quantified the inter-rater agreement after an initial review

round through Fleiss' kappa, and resolved each disagreement in an extended discussion and consolidation phase.

The goal of this study is to capture and analyze the current state of serverless applications. However, due to our methodology, the collected sources can be several years old and therefore possibly represent already overhauled systems and architectures. As we published all the underlying data, follow-up studies can also focus on the development of different characteristics over time.

5.2 Construct Validity

To align this study's goal (i.e., comprehensive understanding of existing serverless applications) with the data extraction, we compiled a list of 22 characteristics covering 6 different aspect groups. We conducted and discussed this selection process in an international working group with authors from 5 different institutions. This kind of effort ensures that the construct has broad validity, but not necessarily that is valid for the entire community: other researchers might consider different characteristics as relevant.

For the purpose of this study, we excluded Container-as-a-Service, such as applications using AWS Fargate or Google Cloud Run, which also fall under a broader definition of serverless [57]. While analyzing these types of applications could also yield interesting findings, we consider it outside the scope of this work.

Serverless is an emerging technology, therefore it is possible and likely that the characteristics of serverless applications change within the next five years. However, the goal of this study is to provide a snapshot of the characteristics of serverless applications at the time of writing. We include a detailed replication package that enables the faithful replication of this study at a later time, which will allow to draw conclusions about how the state of serverless applications changed.

We analyzed consensus between our study and 10 related studies, and found many points of consensus and good (often high or very high) levels of agreement overall. However, to determine the level of agreement with existing studies, we could not use existing, established meta-analysis techniques, as some related studies did not disclose essential information (e.g., cohort size). Therefore, it is possible that the level of agreement we compute does not correctly reflect the actual degree of agreement between the studies. To account for this, we included a detailed breakdown of the study results and their comparison as part of our replication package[1], enabling the community to conduct other comparisons, independently.

5.3 External Validity

Our study was designed to cover applications from open source projects, academic literature, and industrial literature, but we cannot claim generalizability to all serverless applications. For open-source projects, we filtered non-trivial projects from the most popular open-source repository but might have missed projects published in other repositories.

Our academic literature collection is based on a curated dataset on serverless literature and complemented with articles known to the authors. Our comparison study uses a similar methodology. Although we validated the resulting collections against our knowledge and a small set of test-articles, the methodology does not guarantee validity: we might have missed more recent articles, or articles not found by our process and unknown to all authors.

Applications from industrial literature mostly focus on provider-reported case studies, an existing collection of industrial applications, and sources known to the authors. Our scientific computing applications are limited to institutions in a single country, Germany. We only partially cover applications in industry and science, as many of them remain unpublished, and others provide insufficient details to conduct a meaningful review. Other studies, e.g., on FaaS platforms [20], suffer from the same limitations.

6 Conclusion

The impact of serverless applications on the society is already large. Addressing a problem that could hamper further adoption, the goal of this study is to understand the state of serverless applications.

Building on open-access lists created by the community, we collect systematically 89 serverless applications from open-source projects, academic literature, industrial literature, and from scientific computing domain. Ours is the largest collection to date, by almost an order of magnitude over the next-largest. We analyze this collection alongside 16 characteristics in 6 groups: (I) implementation, (II) architecture, (III) traffic patterns, (IV) operational complexity, (V) usage scenarios, and (VI) motivation for adoption. Last, but not least, we corroborate our findings with 10 existing, mostly industrial, studies and data sets, and we investigate points of both agreement and disagreement.

Our analysis spotlights 7 main findings, such as: (I) The most commonly reported reasons for the adoption of serverless include cost-savings for irregular or bursty workloads, avoidance of operational concerns, built-in scalability, and increased speed of development, (II) Typical scenarios include short-running tasks with low data volume and bursty workloads, but we also frequently found latency-critical, high-volume core functionality as serverless applications, and (III) Serverless applications are mostly implemented on AWS, in either Python or JavaScript, and use BaaS.

We further investigate whether our study contributes to the combined knowledge of the community. We identify through a systematic process a set of 10 other seminal studies, whose focus is overlapping with ours. We analyze each of these studies and compare their findings with ours, identifying in the process the overall level of agreement, main points of consensus, and also some disagreements.

We see this study as a step towards community-wide sharing of and discussion around serverless applications. As future directions, this collection of serverless application could help with the development of serverless applications by showing best practices (and whether there is consensus around them), with the identification of serverless design-patterns, and with tuning and performance benchmarking based on realistic characteristics.

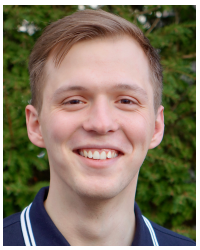
7 Acknowledgements

This work was partially supported by the NWO projects Vidi MagnaData and TOP2 OffSense and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] IDC, "FutureScape: Worldwide IT Industry 2019 Predictions," https://www.idc.com/getdoc.jsp?containerId=US44403818, 2018.
[2] Research and Markets, "$7.72 Billion Function-as-a-Service Market - Global Forecast to 2021," https://bwnews.pr/2VBDBgC, 2017.
[3] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless Computation with OpenLambda," in *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*, 2016, p. 33–39.
[4] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta, and A. Iosup, "Serverless is More: From PaaS to Present Cloud Computing," *IEEE Internet Comput.*, vol. 22, no. 5, pp. 8–17, 2018.
[5] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. J. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," *CoRR*, vol. abs/1902.03383, 2019.
[6] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The Rise of Serverless Computing," *CACM*, vol. 62, no. 12, p. 44–54, 2019.
[7] E. V. Eyk, A. Iosup, S. Seif, and M. Thömmes, "The SPEC cloud group's research vision on FaaS and serverless architectures," in *2nd International Workshop on Serverless Computing*, 2017, pp. 1–4.
[8] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures," in *Companion of the 2018 International Conference on Performance Engineering*, 2018, pp. 21–24.
[9] J. M. Hellerstein, J. M. Faleiro, J. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless Computing: One Step Forward, Two Steps Back," in *9th Biennial Conference on Innovative Data Systems Research*, 2019.
[10] G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 884–889.
[11] E. Levinson, "Serverless Community Survey 2020," 2020. [Online]. Available: https://bit.ly/SerComSurvey
[12] A. Eivy, "Be Wary of the Economics of" Serverless" Cloud Computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, 2017.
[13] P. A. Witte, M. Louboutin, C. Jones, and F. J. Herrmann, "Serverless seismic imaging in the cloud," *CoRR*, vol. abs/1911.12447, 2019.
[14] R. Crespo-Cepeda, G. Agapito, J. L. Vazquez-Poletti, and M. Cannataro, "Challenges and Opportunities of Amazon Serverless Lambda Services in Bioinformatics," in *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, 2019, p. 663–668.
[15] M. Chan, "Containers vs. Serverless: Which Should You Use, and When?" https://bit.ly/3rwMqpx, Aug 2018.
[16] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud Container Technologies: A State-of-the-Art Review," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 677–692, 2019.
[17] P. Maenhaut, B. Volckaert, V. Ongenae, and F. D. Turck, "Resource Management in a Containerized Cloud: Status and Challenges," *J. Netw. Syst. Manag.*, vol. 28, no. 2, pp. 197–246, 2020. [Online]. Available: https://doi.org/10.1007/s10922-019-09504-0
[18] A. Orfin, "How Droplr Scales to Millions With The Serverless Framework," https://bit.ly/3ejIWTu, 2018.
[19] V. Yussupov, U. Breitenbücher, F. Leymann, and M. Wurster, "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*, 2019, pp. 229–240.
[20] E. van Eyk, A. Iosup, J. Grohmann, S. Eismann, A. Bauer, L. Versluis, L. Toader, N. Schmitt, N. Herbst, and C. L. Abad, "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms," *IEEE Internet Comput.*, vol. 23, no. 6, pp. 7–18, 2019.
[21] T. Back and V. Andrikopoulos, "Using a Microbenchmark to Compare Function as a Service Solutions," in *Service-Oriented and Cloud Computing*, 2018, pp. 146–160.
[22] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 23, 2018.
[23] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 442–450.
[24] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 159–169.
[25] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 133–146.
[26] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," 2020.
[27] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of Function-as-a-Service software development in industrial practice," *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019.
[28] D. Taibi, N. El Ioini, C. Pahl, and J. R. S. Niederkofler, "Serverless Cloud Computing (Function-as-a-Service) Patterns: A Multivocal Literature Review," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20)*, 2020.
[29] I. Pavlov, S. Ali, and T. Mahmud, "Serverless Development Trends in Open Source: a Mixed-Research Study," Thesis, 11 2019.
[30] J. Spillner and M. Al-Ameen, "Serverless Literature Dataset," https://doi.org/10.5281/zenodo.1175423, 2019.
[31] S. Eismann, S. Joel, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup, "Serverless Applications:Why, When, and How?" *IEEE Software*, vol. 38, no. 1, p. 32–39, 2021.
[32] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "SPEC RG Technical Report: A Review of Serverless Use Cases and their Characteristics - Dataset," May 2020.
[33] J. Walter, "Systematic Data Transformation to Enable Web Coverage Services (WCS) and ArcGIS Image Services within ESDIS Cumulus Cloud," 2019. [Online]. Available: https://earthdata.nasa.gov/esds/competitive-programs/access/arcgis-cloud
[34] J. Blomer, G. Ganis, S. Mosciatti, and R. Popescu, "Towards a serverless CernVM-FS," in *EPJ Web of Conferences*, vol. 214. EDP Sciences, 2019, p. 09007.
[35] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 257–262.
[36] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "Barista: Efficient and scalable serverless serving system for deep learning prediction services," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 23–33.
[37] Z. Tu, M. Li, and J. Lin, "Pay-per-request deployment of neural network models using serverless architectures," in *NAACL: Demonstrations*, 2018, pp. 6–10.
[38] A. Coffey and P. Atkinson, *Making sense of qualitative data: complementary research strategies*. Sage Publications, Inc, 1996.
[39] G. Guest, K. M. MacQueen, and E. E. Namey, *Applied thematic analysis*. Sage Publications, 2011.
[40] K. L. Gwet, *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. LLC, 2014.
[41] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.
[42] J. Scheuner and P. Leitner, "Function-as-a-Service Performance Evaluation: A Multivocal Literature Review," *Journal of Systems and Software (JSS)*, 2020.
[43] Gartner, "Worldwide IaaS Public Cloud Services Market Grew 31.3%," 2018. [Online]. Available: https://bwnews.pr/2ZcI7o4
[44] N. Malishev, "AWS Lambda Cold Start Language Comparisons, 2019 edition," https://bit.ly/ColdStartComp, 2019.
[45] S. Moellering and S. Grunwald, "Field Notes: Optimize your Java application for AWS Lambda with Quarkus," https://amzn.to/3mqZYBg, 2020.
[46] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "The Serverless Trilemma: Function Composition for Serverless Computing," in *2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2017, p. 89–103.

[47] J. Spillner, C. Mateos, and D. A. Monge, "FaaSter, Better, Cheaper: The Prospect of Serverless Scientific Computing and HPC," in *High Performance Computing*, 2018, pp. 154–168.
[48] M. Laul, "Serverless Case Study - Netflix," 2018. [Online]. Available: https://dashbird.io/blog/serverless-case-study-netflix/
[49] A. Eivy and J. Weinman, "Be Wary of the Economics of "Serverless" Cloud Computing," *IEEE Cloud Computing*, vol. 4, pp. 6–12, 2017.
[50] A. Williams, "Autodesk Goes Serverless in the AWS Cloud, Reduces Account-Creation Time by 99%," https://amzn.to/2Q3X0pV, 2017.
[51] S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, and K. Winstein, "From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers," in *2019 USENIX Annual Technical Conference*, 2019, pp. 475–488.
[52] S. E. Brockwell and I. R. Gordon, "A comparison of statistical methods for meta-analysis," *Statistics in medicine*, vol. 20, no. 6, pp. 825–840, 2001.
[53] S. Makridakis, "Accuracy measures: theoretical and practical concerns," *International journal of forecasting*, vol. 9, pp. 527–529, 1993.
[54] J. L. Myers, A. Well, and R. F. Lorch, *Research design and statistical analysis*. Routledge, 2010.
[55] K.-I. Goh and A.-L. Barabási, "Burstiness and memory in complex systems," *EPL (Europhysics Letters)*, vol. 81, no. 4, p. 48002, 2008.
[56] A. Ali-Eldin, O. Seleznjev, S. Sjöstedt-de Luna, J. Tordsson, and E. Elmroth, "Measuring cloud workload burstiness," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 566–572.
[57] S. Kounev and et al., "Toward a Definition for Serverless Computing," in *Serverless Computing (Dagstuhl Seminar 21201)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, vol. 11, ch. Chapter 5.1, pp. 56–59.



Simon Eismann is currently a PhD student at the chair of software engineering at the University of Würzburg. He received the M.S. degree from the University of Würzburg in 2017. His research interests include cloud computing, serverless, and performance analysis/modeling. Contact him at simon.eismann@uni-wuerzburg.de.

Joel Scheuner is a PhD student at the division of software engineering at Chalmers University of Technology and the University of Gothenburg. He received his M.S. in Software Systems from the University of Zurich in 2017. His research interests include cloud computing, performance engineering, and software engineering. Contact him at scheuner@chalmers.se.

Erwin van Eyk is a PhD student at Vrije Universiteit Amsterdam, the Netherlands, and the chair of the SPEC-RG Cloud Serverless activity. In 2019, he received his M.Sc. degree from TU Delft, the Netherlands, for work on cloud computing and serverless workflows. Contact him at e.vaneyk@atlarge-research.com.

Maximilian Schwinger is a PhD student at the chair for software engineering at the University of Würzburg and received his Diploma in Computer Science from the TU Munich in 2006. Since then, he is working for the German Aerospace Center (DLR) as a Software and Systems Engineer. His research interest includes high-performance computing and cloud-based computin in the domain of satellite-based earth observation. Contact him at maximilian.schwinger@dlr.de.

Johannes Grohmann is currently a PhD student at the chair of software engineering at the University of Würzburg. He received the M.S. degree from the University of Würzburg in 2016. His research interests include serverless and cloud computing and performance model learning and analysis. Contact him at johannes.grohmann@uni-wuerzburg.de.

Nikolas Herbst is a research group leader at the chair of software engineering at the University of Würzburg. He received a PhD from the University of Würzburg in 2018 and serves as elected vice-chair of the SPEC Research Cloud Group. His research topics include predictive data analysis, elasticity, auto-scaling, resource management, performance evaluation of virtualized environments. Contact him at nikolas.herbst@uni-wuerzburg.de.

Cristina L. Abad is an associate professor at Escuela Superior Politecnica del Litoral, ESPOL, in Ecuador, where she leads the Distributed Systems Research Lab (DiSEL). She obtained MS and PhD in CS degrees from the University of Illinois at Urbana-Champaign. Her main research interests lie at the intersection of distributed systems and performance engineering. Contact her at cabad@fiec.espol.edu.ec.

Alexandru Iosup is the University Research Chair at Vrije Universiteit Amsterdam and member of the Young Royal Academy of Arts and Sciences of the Netherlands. He is the chair of the Massivizing Computer Systems research group at the VU and the SPEC-RG Cloud group. His work in distributed systems and ecosystems has received prestigious recognition, including the 2016 Netherlands ICT Researcher of the Year. He can be contacted at A.Iosup@vu.nl.