A Case Study on the Stability of Performance Tests for Serverless Applications

Simon Eismann[a], Diego Elias Costa[b], Lizhi Liao[b], Cor-Paul Bezemer[c], Weiyi Shang[b], André van Hoorn[d], Samuel Kounev[a]

[a] *University of Würzburg, Würzburg, Germany*
[b] *Concordia University, Montreal, Canada*
[c] *University of Alberta, Edmonton, Canada*
[d] *University of Hamburg, Hamburg, Germany*

Abstract

Context. While in serverless computing, application resource management and operational concerns are generally delegated to the cloud provider, ensuring that serverless applications meet their performance requirements is still a responsibility of the developers. Performance testing is a commonly used performance assessment practice; however, it traditionally requires visibility of the resource environment.

Objective. In this study, we investigate whether performance tests of serverless applications are stable, that is, if their results are reproducible, and what implications the serverless paradigm has for performance tests.

Method. We conduct a case study where we collect two datasets of performance test results: (a) repetitions of performance tests for varying memory size and load intensities and (b) three repetitions of the same performance test every day for ten months.

Results. We find that performance tests of serverless applications are comparatively stable if conducted on the same day. However, we also observe short-term performance variations and frequent long-term performance changes.

Conclusion. Performance tests for serverless applications can be stable;

Email addresses: simon.eismann@uni-wuerzburg.de (Simon Eismann), diego.costa@concordia.ca (Diego Elias Costa), l_lizhi@encs.concordia.ca (Lizhi Liao), bezemer@ualberta.ca (Cor-Paul Bezemer), shang@encs.concordia.ca (Weiyi Shang), andre.van.hoorn@uni-hamburg.de (André van Hoorn), samuel.kounev@uni-wuerzburg.de (Samuel Kounev)

however, the serverless model impacts the planning, execution, and analysis of performance tests.

---

1. Introduction

Serverless computing combines Function-as-a-Service (e.g., AWS Lambda, Google Cloud Functions, or Azure Functions) and Backend-as-a-Service (e.g., managed storage, databases, pub/sub, queueing, streaming, or workflows) offerings that taken together provide a high-level application programming model, offloading application resource management and operation aspects to the cloud provider [29, 16]. The cloud provider opaquely handles resource management tasks, such as deployment, resource allocation, or auto-scaling, and bills the user on a pay-per-use basis [6, 71]. While the cloud provider takes care of resource management, managing the performance of serverless applications remains a developer concern [36, 70]. Executing performance tests as part of a CI/CD pipeline to monitor the impact of code changes on system performance is a common and powerful approach to manage system performance [10, 34]. One of the key requirements for reliable performance tests is ensuring that an identical resource environment is used for all tests [14].

However, with serverless applications, developers have no control over the resource environment. Worse yet, cloud providers expose no information to developers about the resource environment [73]. Therefore, information such as the number of provisioned workers, worker utilization, worker version, virtualization stack, or underlying hardware is unavailable. Furthermore, cold starts (requests where a new worker has to be provisioned) are a widely discussed performance challenge [36, 70]. This begs the following question:

"*Are performance tests of serverless applications stable?*"

The performance variability of virtual machines in cloud environments has been studied extensively [24, 58, 32]. However, many serverless platforms are not deployed on traditional virtual machines [2]. Additionally, the opaque nature of serverless platforms means that it is extremely challenging, if not impossible, to control or know how many resources are allocated during a performance test [71]. Existing work on performance evaluation of serverless platforms focuses on determining the performance characteris-

tics of such platforms but does not investigate the stability of performance measurements [73, 49, 41, 59].

In this paper, we present an exploratory case study on the stability of performance tests of serverless applications. Using the serverless airline application [60], a representative, production-grade serverless application [37], we conduct two sets of performance measurements: (1) multiple repetitions of performance tests under varying configurations to investigate the performance impact of the latter, and (2) three daily measurements for ten months to create a longitudinal dataset and investigate the stability of performance tests over time. This study makes three main contributions towards furthering the understanding of the performance variability of serverless applications:

- **Contribution 1:** We present the first study on the performance variability of serverless applications over longer periods of time. In particular, we collect daily measurements on the performance variability of our application for ten months.
- **Contribution 2:** Our case study analyses the performance behavior of a complex, realistic application, unlike the micro-benchmarks and single-function applications often used by existing work.
- **Contribution 3:** We include a detailed replication package that enables the replication of our study on future configurations of serverless platforms.

We find that in our case study there are serverless-specific changes and pitfalls to all performance test phases: design, execution, and analysis. In the design phase, the load intensity of the test directly correlates to cost (Section 6.1.1), and reducing load intensity can deteriorate performance (Section 6.1.2). In the execution phase, daily performance fluctuations (Section 6.2.1) and long-term performance changes (Section 6.2.2) impact the decision when performance tests should be scheduled. In the analysis phase, developers need to consider that there is still a warm-up period after removing all cold starts (Section 6.3.1) and that cold starts can occur late in a performance test even under constant load (Section 6.3.2).

The rest of the paper is organized as follows: Section 2 gives an introduction to performance testing and serverless applications. Next, Section 3 introduces related work on performance evaluation of serverless platforms as

well as on performance variability of virtual machines. Section 4 describes the design of our case study and Section 5 discusses the results. Then, Section 6 discusses the implications of our findings on the performance testing of serverless applications. Section 7 presents the threats to the validity of our study, Section 8 introduces our comprehensive replication package, and finally, Section 9 concludes the paper.

2. Background

In the following, we give a short introduction to serverless applications and performance testing.

2.1. Serverless Applications

Serverless applications consist of business logic in the form of serverless functions—also known as Function-as-a-Service (FaaS)—and cloud provider-managed services such as databases, blob storage, queues, pub/sub messaging, machine learning, API gateways, or event streams.

Developers write business logic as isolated functions, configure the managed services via Infrastructure-as-Code, and define triggers for the execution of the business logic (serverless functions). Triggers can either be HTTP requests or cloud events such as a new message in a queue, a new database entry, a file upload, or an event on an event bus. The developer provides the code for the serverless functions, the configuration of the managed services, and the triggers; the cloud provider guarantees that the code is executed and the managed services are available whenever a trigger occurs, independent of the number of parallel executions. In contrast to classical IaaS platforms, developers are not billed for the time resources are *allocated* but rather for the time resources are actively *used*. Under this pay-per-use model, developers are billed based on the time the serverless functions run and per operation performed by the managed services.

Serverless functions promise seamless scaling of arbitrary code. In order to do so, each function is ephemeral, stateless, and is executed in a predefined runtime environment. Additionally, every worker (function instance) only handles a single request at a time. When a function trigger occurs, the request is routed to an available function instance. If no function instance is available, the request is not queued, instead, a new function instance is deployed and the request is routed to it (known as *cold start*, other executions are labeled as *warm start*). Cloud providers are continuously working on

reducing the cold start time through various techniques such as the utilization of a fleet of template function instances that already run the supported runtime environments, into which only the application-specific code needs to be loaded, and the use of more lightweight virtualization techniques [69].

A number of potential benefits of serverless applications compared to traditional cloud applications have been reported [18, 19]. The pay-per-use model is reported to reduce costs for bursty workloads, which often lead to over-provisioning and low resource utilization in traditional cloud-based systems. Furthermore, serverless applications are virtually infinitely scalable by design and reduce the operational overhead, as the cloud provider takes care of all resource management tasks. Finally, the heavy usage of managed services is reported to increase development speed.

2.2. Performance Testing

Performance testing is the process of measuring and ascertaining a system's performance-related aspects (e.g., response time, resource utilization, and throughput) under a particular workload [25]. Performance testing helps to determine compliance with performance goals and requirements [55, 74, 26], identify bottlenecks in a system [68, 52], and detect performance regressions [53, 46, 65]. A typical performance testing process starts with designing the performance tests according to the performance requirements. These performance tests are then executed in a dedicated performance testing environment, while the system under test (SUT) is continuously monitored to collect system runtime information including performance counters (e.g., response time and CPU utilization), the system's execution logs, and event traces. Finally, performance analysts analyze the results of the performance testing.

During the execution of a software system, it often takes some time to reach its stable performance level under given load. During performance testing, the period before the software system reaches steady-state is commonly known as the warm-up period, and the period after that is considered as the steady-state period. There are many reasons for the warm-up period, such as filling up buffers or caches, program JIT compilation, and absorbing temporary fluctuations in system state [48]. Since performance during the warm-up period may fluctuate, in practice, performance engineers often remove the duration of the unstable phase (i.e., warm-up period) of the performance test and only consider the steady-state period in the performance test results. The most intuitive way to determine the warm-up period is to

simply remove a fixed duration of time (e.g., 30 minutes [39]) from the beginning of the performance testing results. We refer to a review by Mahajan and Ingalls [43] for an overview of existing techniques to determine the warm-up period.

3. Related Work

Existing work related to this study can be grouped into performance evaluations of serverless platforms and studies on the performance variability of virtual machines.

3.1. Performance Evaluation of Serverless Platforms

A number of empirical measurement studies on the performance of serverless applications have been conducted. Lloyd et al. [41] examined the infrastructure elasticity, load balancing, provisioning variation, infrastructure retention, and memory reservation size of AWS Lambda and Azure Functions. They found that cold and warm execution times are correlated with the number of containers per host, which makes the number of containers per host a major source of performance variability. Wang et al. conducted a large measurement study that focuses on reverse engineering platform details [73]. They found variation in the underlying CPU model used and low performance isolation between multiple functions on the same host. When they repeated a subset of their measurements about half a year later, they found significant changes in the platform behavior. Lee et al. [33] analyzed the performance of CPU, memory, and disk-intensive functions with different invocation patterns. They found that file I/O decreases with increasing numbers of concurrent requests and that the response time distribution remained stable for a varying workload on AWS. Yu et al. [77] compared the performance of AWS Lambda to two open-source platforms, OpenWhisk and Fn. They found that Linux CPU shares offer insufficient performance isolation and that performance degrades when co-locating different applications. However, while the performance of FaaS platforms has been extensively studied, there has been little focus on the stability of these measurements over time. Djemame et al. [13] investigate the performance properties of Apache OpenWhisk, one of the most popular open-source function-as-a-service platforms. They compared the performance of OpenWhisk with docker-based and native execution, however, they do not consider performance variability in their study.

There have also been a number of measurement tools and benchmarks developed for serverless applications and platforms. Cordingly et al. [8] introduced the Serverless Application Analytics Framework (SAAF), a tool that allows profiling FaaS workload performance and resource utilization on public clouds; however it does not provide any example applications. Figiela et al. [21] introduced a benchmarking suite for serverless platforms and evaluated the performance of AWS Lambda, Azure Functions, Google Cloud Functions, and IBM Functions. The workloads included in the benchmarking suite consist of synthetic benchmark functions, such as a mersenne twister or linpack implementation. Kim et al. [27] proposed FunctionBench, a suite of function workloads for various cloud providers. The functions included in FunctionBench closely resemble realistic workloads, such as video processing or model serving, but they only cover single functions and not entire applications. Maissen et al. [45] propose Faasdom, an extensible benchmark suite for the comparison of serverless platforms using micro-benchmarks with native support for AWS, Azure, Google, and IBM. In addition to performance measurements, the benchmark tool also contains a cost calculator that determines the costs incurred for the execution of a specific benchmark. In their measurements, they find that performance varies between providers, with AWS usually leading the pack. There have also been a number of approaches proposed that aim to model the impact of various factors on the cost and performance of serverless applications [40, 17, 15, 20, 44].

Existing work mostly focuses on FaaS benchmarks and performance studies using micro-benchmarks and single function applications. In contrast, our case study uses a realistic application consisting of many functions and external services.

For further details on the current state of the performance evaluation of serverless offerings, we refer to an extensive multi-vocal literature review by Scheuner et al. [59]. This review also finds that the reproducibility of the surveyed studies is a major challenge. Therefore we include a comprehensive replication package (see Section 8) that enables other researchers to replicate and extend our study.

3.2. Performance Variability of Virtual Machines

Due to the extensive adoption of virtual machines (VMs) in practice, there exists much prior research on the performance variability of VMs. One early work in this area is from Menon et al. [50], which quantifies the network I/O related performance variation in Xen virtual machines. Their re-

sults also identify some key sources of such performance variability in VMs. Afterwards, Kraft et al. [30], Boutcher and Chandra [7] apply various techniques to assess the performance variation of VMs compared to a native system with respect to disk I/O. Taking the contention between different VMs into account, Koh et al. [28] analyze ten different system-level performance characteristics to study the performance interference effects in virtual environments. Their results show that the contention on shared physical resources brought by virtualization technology is one of the major causes of the performance variability in VMs.

Huber et al. [23] compared the performance variability (for CPU and memory) of two virtualization environments and use regression-based models to predict the performance overhead for executing services on these platforms. Schad et al. [58], Iosup et al. [24], and Leitner and Cito [35] assessed the performance variability across multiple regions and instance types of popular public clouds such as Amazon Web Services (AWS) and Google Compute Engine (GCE). Based on these findings, Asyabi [5] proposed a novel hypervisor CPU scheduler aiming to reduce the performance variability in virtualized cloud environments.

To investigate the impact of the performance variability of VMs on performance assurance activities (e.g., performance testing and microbenchmarking), Laaber et al. [31, 32] evaluated the variability of microbenchmarking results in different virtualization environments and analyzed the results from a statistical perspective. They found that not all cloud providers and instance types are equally suited for performance microbenchmarking. Costa et al. [9] summarized some bad practices of writing microbenchmarks using the JMH framework to mitigate the variation and instability of cloud environments when conducting performance microbenchmarking. Arif et al. [4] and Netto et al. [51] compared performance metrics generated via performance tests between virtual and physical environments. Their findings highlight the inconsistency between performance testing results in virtual and physical environments.

The focus of existing work is on traditional software systems in virtualized environments with static resource allocation and deployment. In this study, we investigate the performance variability of serverless applications where the resource allocation and deployment is not known to the developer and might change between experiments.

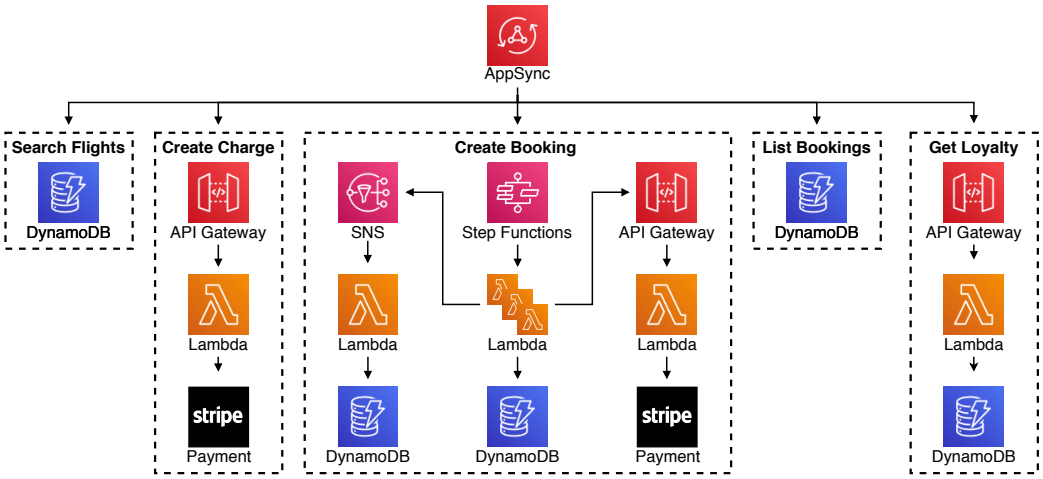

Figure 1: Architecture and API endpoints of the serverless airline booking application.

4. Case Study Design

In this section, we present the design of our case study. We first introduce our subject system, the Serverless Airline Booking (SAB) application, and describe why we selected this system for our case study. Then, we describe the experiment setup, the collected metrics, and the individual experiments.

4.1. Serverless Airline Booking (SAB)

The serverless airline booking application (SAB)[1] is a fully serverless web application that implements the flight booking aspect of an airline on AWS. It was presented at AWS re:Invent as an example for the implementation of a production-grade full-stack app using AWS Amplify [37]. The SAB was also the subject of the AWS Build On Serverless series [60]. Customers can search for flights, book flights, pay using a credit card, and earn loyalty points with each booking.

The frontend of the SAB is implemented using CloudFront, Amplify/S3, Vue.js, the Quasar framework, and Stripe Elements. This frontend sends GraphQL queries (resolved using AWS AppSync) to five backend APIs, as shown in Figure 1:

[1] `https://github.com/aws-samples/aws-serverless-airline-booking`

- The *Search Flights* API retrieves all flights for a given date, arrival airport, and departure airport from a DynamoDB table using the DynamoDB GraphQL resolver.

- The *Create Charge* API is implemented as an API gateway that triggers the execution of the *CreateStripeCharge* lambda function, which manages the call to the Stripe API.

- The *Create Booking* API reserves a seat on a flight, creates an unconfirmed booking, and attempts to collect the charge on the customer's credit card. If successful, it confirms the booking, and awards loyalty points to the customer. In case the payment collection fails, the reserved seat is freed again, and the booking is canceled. This workflow is implemented as an AWS Step Functions workflow that coordinates multiple lambda functions. The functions *ReserveBooking* and *CancelBooking* directly modify DynamoDB tables, the *NotifyBooking* function publishes a message to SNS, which is later consumed by the *IngestLoyalty* function that updates the loyalty points in a DynamoDB table. The *CollectPayment* and *RefundPayment* functions call the Stripe backend via an application from the Serverless Application Repository.

- The *List Bookings* API retrieves the existing bookings for a customer. Similar to the *Search Flights* API, this is implemented using a DynamoDB table and the DynamoDB GraphQL resolver.

- The *Get Loyalty* API retrieves the loyalty level and loyalty points for a customer. An API Gateway triggers the lambda function *FetchLoyalty*, which retrieves the loyalty status for a customer from a DynamoDB table.

We selected SAB for our case study after investigating potential applications from a review of serverless use cases [16], a serverless literature dataset [3], and a recent survey on FaaS performance evaluation [59]. We chose SAB over other potential applications due to its comparatively large size and its usage of many different managed services. It is also running on AWS, the by far most popular cloud provider for serverless applications [16, 36, 38], and it uses both Python and JavaScript to implement the serverless functions, the two most popular programming languages for serverless applications [16, 36, 38].

4.2. Experiment Setup

We deploy the frontend via Amplify [61] and the backend services via either the Serverless Application Model [63] or CloudFormation [62] templates depending on the service. The serverless nature of the application makes it impossible to specify the versions of any of the used services, as DynamoDB, Lambda, API Gateway, Simple Notification Service, Step Functions, and AppSync all do not provide any publicly available version numbers.

For the load profile, customers start by querying the *Search Flights* API for flights between two airports. If no flight exists for the specified airports and date, the customer queries the *Search Flights* API again, looking for a different flight. We populated the database so that most customers find a flight within their first query. Next, they call the *Create Charge* API and the *Create Booking* to book a flight and pay for it. After booking a flight, each customer checks their existing bookings and loyalty status via the *List Bookings* API and the *Get Loyalty* API. This load profile is implemented using the TeaStore load driver [72].

In terms of monitoring data, we collect the response time of each API call via the load driver. Additionally, we collect the duration, that is, the execution time of every lambda function. We exclude the duration of the lambdas *ChargeCard* and *FetchLoyalty*, as the response times of the APIs *Create Charge* and *Get Loyalty* mostly consist of the execution times of these lambdas. We cannot collect any resource-level metrics such as utilization or number of provisioned workers, as AWS and most other major serverless platforms do not report any resource level metrics.

For our experiments, we perform measurements with 5 req/s, 25 req/s, 50 req/s, 100 req/s, 250 req/s, and 500 req/s to cover a broad range of load levels. Additionally, we vary the memory size of the lambda functions between 256 MB, 512 MB, and 1024 MB, which covers the most commonly used memory sizes [12]. For each measurement, the SAB is deployed, put under load for 15 minutes, and then torn down again. We perform ten repetitions of each measurement to account for cloud performance variability. Additionally, we run the experiments as randomized multiple interleaved trials, which have been shown to further reduce the impact of cloud performance variability [1]. To minimize the risk of manual errors, we fully automate the experiments (for further details see Section 8). These measurements started on July 5th, 2020, and continuously ran until July 17th, 2020.

Additionally, we set up a longitudinal study that ran three measurement repetitions with 100 req/s and 512 MB every day at 19:00 from Aug 20th, 2020

to Jun 20th, 2021. The measurements were automated by a Step Functions workflow that is triggered daily by a CloudWatch alarm and starts the experiment controller VM, triggers the experiment, uploads the results to an S3 bucket, and shuts down the experiment controller VM again.

To ensure reproducibility of our results, the fully automated measurement harness and all collected data from these experiments are available in our replication package.[2]

5. Case Study Results

We now present the results of our empirical study in the context of our three research questions. For each research question, we present the motivation of answering the question, our approach to answering it, and the corresponding results.

5.1. RQ1: How do cold starts influence the warm-up period and stability of serverless performance tests?

Motivation. A common goal of performance tests is to measure the steady-state performance of a system under a given workload. Hence, it is essential that practitioners understand how long it takes for serverless applications to reach stable performance (i.e., how long is the warm-up period) in order to plan the duration of their performance tests accordingly. Aside from the general aspects that influence the initial performance instability, such as the environment and application optimizations (e.g., CPU adaptive clocking and cache setup), serverless applications also encounter cold starts. A cold start occurs when a request cannot be fulfilled by the available function instances, and a new instance has to be provisioned to process the upcoming request. Cold starts can incur significantly higher response times [73, 21]. Hence, in this RQ, we investigate: (1) how long is the warm-up period in our experiments and (2) the role of cold starts in the stability of the warm-up and steady-state experiment phases.

Approach. To determine the duration of the warm-up period, we initially tried to use the MSER-5 method [75], which is the most popular method to identify the warm-up period in simulations [43, 22]. However, this approach

[2] `https://github.com/ServerlessLoadTesting/ReplicationPackage`

Algorithm 1: Warm-up Period Identification Heuristic.

```
Result: warmupInSeconds
threshold = 0.01;
stable = False;
warmupInSeconds = 0;
global_mean = mean(ts);
while stable == False do
    ts = remove5secs(ts) // Remove 5 seconds of data;
    warmupInSeconds += 5;
    new_mean = mean(ts);
    delta = abs((new_mean - global_mean) / global_mean);
    if delta < threshold then
        stable = True;
    else
        global_mean = new_mean;
    end
end
```

was not applicable due to the large outliers present in our data, a well-documented flaw of MSER-5 [57]. Therefore, we employ a heuristic to identify the warm-up period. Our heuristic, shown in Algorithm 1, gradually removes data from the beginning of the experiment in windows of five seconds and evaluates the impact of doing so on the overall mean results. If the impact is above a threshold (we used 1% in our experiments), we continue the data removal procedure. Otherwise, we consider the seconds removed as the warm-up period and the remainder as the steady-state phase of the performance test experiment. Similar to MSER-5 [75], we label any measurement where the detected warm-up period is larger than 40% of the measurement as unstable. This regulation is necessary, as warm-up period detection approaches become unreliable once the steady-state period is not considerably longer than the warm-up period. In these scenarios, either longer measurements are required until a steady-state can be detected or the system under test never reaches a steady-state (e.g., due to a growing number of entries in a database).

To evaluate the impact of cold starts on the experiment stability, we analyze the distribution of cold start requests across the two phases of performance tests: warm-up period and steady-state period. Then, we evaluate the

influence of cold start requests on the overall mean response time, considering only cold start requests that occurred after the warm-up period. To test for statistically significant differences, we use the unpaired and non-parametric Mann-Whitney U test [47]. In cases where we observe a statistical difference, we evaluate the effect size of the difference using the Cliff's Delta effect size [42], and we use the following common thresholds [56] for interpreting the effect size:

$$\text{Effect size } d = \begin{cases} negligible(N), & \text{if } |d| \leq 0.147 \\ small(S), & \text{if } 0.147 < |d| \leq 0.33 \\ medium(M), & \text{if } 0.33 < |d| \leq 0.474 \\ large(L), & \text{if } 0.474 < |d| \leq 1 \end{cases}$$

Note that not all request classes provide information about cold starts. This information is only available for the six lambda functions, as the managed services either do not have cold starts or do not expose them. Therefore, we report the cold start analysis for the following six request classes: *CollectPayment*, *ConfirmBooking*, *CreateStripeCharge*, *IngestLoyalty*, *NotifyBooking*, and *ReserveBooking*. Finally, our experiment contains more than 45 hours of measurements, including performance tests with ten repetitions, different workload levels, and function sizes.

Findings. The warm-up period lasts less than 2 minutes in the vast majority of our experiments. Table 1 shows the maximum warm-up period in seconds, observed across all experiments per workload level. In most experiments, we observe that the maximum warm-up period out of the ten repetitions lasts less than 30 seconds (37 out of 48 experiment combinations). With exception of *IngestLoyalty*, all workload classes exhibit a shorter warm-up period as the load increases. The average warm-up period in experiments with 500 requests per second was 27 seconds, half of the warm-up period observed in runs with 5 requests per second (52 seconds). The function *Get Loyalty* never reaches a steady-state under high load, as it implements the performance anti-pattern "Ramp" due to a growing number of entries in the database [66]. We also note that, contrary to the workload, the function size (memory size) has no influence on the warm-up period: in most cases, the difference of the warm-up period across function sizes (256 MB, 512 MB, 1024 MB) is not significant ($p > 0.05$), with a negligible effect size for the few significantly different cases ($d < 0.147$). In the following, we opt to

Table 1: **Maximum** warm-up period in seconds across ten repetitions of all function sizes. We highlight warm-up periods over one minute with a dark background.

Request Class	Workload (reqs/s)					
	5	**25**	**50**	**100**	**250**	**500**
CollectPayment	15	10	10	10	10	10
ConfirmBooking	25	15	15	65	10	15
CreateStripeCharge	15	15	15	15	20	15
Get Loyalty	70	60	45	30	10	–
IngestLoyalty	15	25	55	75	125	155
List Bookings	115	70	55	45	25	15
NotifyBooking	20	40	40	15	60	10
Process Booking	65	45	20	10	10	15
ReserveBooking	20	20	10	10	10	15
Search Flights	135	80	50	30	30	20

conservatively consider the first 2 minutes of performance tests as part of the warm-up period for any subsequent analysis.

The vast majority (>99%) of cold starts occur during the first two minutes of the performance test (warm-up period). Cold start requests that occur after the warm-up period (<1%) do not impact the measurements. Table 2 depicts the average percentage of cold start requests across different request classes, the share of cold start requests that occur in the warm-up period, and whether cold starts after the warm-up period significantly impact the mean response time. We consider cold starts to impact the results, if there is a significant difference between the mean response time with and without cold starts in the steady-state experiment phase. As we observe similar results in all six request classes, below we discuss only the *CollectPayment* results. On average, cold start requests in *CollectPayment* make up for 0.93% of the total number of requests. However, since they mostly concentrate in the first two minutes of the experiment (99.5%), they are discarded from the final results as part of the warm-up period. The remaining cold start requests (0.5%) that occur throughout the run of our performance test did not significantly impact the response time (Mann-Whitney U test with $p > 0.05$).

In the majority of experiments, removing the cold starts does not shorten the warm-up period. Given that cold starts occur mostly

Table 2: Average occurrence of cold start requests in the performance tests per request class. We consider the first 2 minutes as the warm-up period. We report cold starts as impacting the results if there is a significant difference of the mean response time when accounting for cold start requests after the warm-up period.

Request Class	**% Cold Start**	**% Occurrence**		**Impact?**
		<=2 min	**>2 min**	
CollectPayment	0.93	99.5	0.05	No
ConfirmBooking	0.72	99.5	0.05	No
CreateStripeCharge	0.44	99.9	0.01	No
IngestLoyalty	1.01	99.2	0.02	No
NotifyBooking	1.04	99.9	0.01	No
ReserveBooking	0.40	99.8	0.02	No

during the warm-up period, we wanted to assess if the warm-up period is composed solely of cold start requests. Is it enough to simply drop cold start requests from the experiment and consider all other requests as part of the steady-state performance measurements? Table 3 shows the difference of the warm-up period considering all requests (the one shown in Table 1), versus the warm-up period calculated by filtering the cold start requests from the experiment. In the majority of the experiments (22 out of 36 combinations), we observe no difference between dropping or keeping the cold start requests in the duration of the warm-up period. Some request classes, however, exhibited shorter periods of warm-up once we filter out cold start requests, as the high response time of cold start requests contributes to the warm-up period. For instance, the experiment with *CreateStripesCharge* showed a consistent reduction of the warm-up period of at least 5 seconds (our heuristic's window size) for all the workload sizes. It is important to note, however, that the warm-up period — while shorter in some classes — is not only influenced by cold starts.

5.2. RQ2: How stable are the performance test results of a serverless application deployed on common serverless platforms?

Motivation. In RQ1, we found that within a run, the results of a performance test quickly become stable. The period of instability (warm-up) usually lasts less than two minutes, and the number of cold start requests that occur after this period does not impact the performance test results.

Table 3: Difference of the maximum warmup-period in seconds between experiments including all requests vs. experiments filtering out the cold start requests.

Request Class	Workload (reqs/s)					
	5	**25**	**50**	**100**	**250**	**500**
CollectPayment	-5	–	-5	–	–	–
ConfirmBooking	–	–	–	–	–	–
CreateStripeCharge	-10	-5	-5	-5	-5	-10
IngestLoyalty	–	–	-10	–	–	-10
NotifyBooking	–	-20	-20	–	–	–
ReserveBooking	-5	-5	–	–	–	–

However, results across multiple runs are likely to vary considerably. Practitioners have no way to ensure that two different performance tests are executed in similar resource environments, given that deployment details in serverless applications are hidden from developers. Hence, for this RQ, we study how the inherent variance in deployed serverless applications impacts the stability *between* performance tests.

Approach. In this analysis, we evaluate the variation of the mean response time across experiment runs and study the influence of experiment factors such as the load level and function size. We focus on evaluating the steady-state performance of performance tests. Hence, we discarded the data from the first two minutes of the performance test runs (warm-up period) and calculated the mean response time for the steady-state phase, that is, the remaining 13 minutes of experiment data.

To evaluate the stability of the mean response time across runs, we first exclude outliers within an experiment that fall above the .99 percentile. Then, we calculate the coefficient of variation of the response time across the ten repetitions, per workload level and function size. The coefficient of variation is the ratio of the standard variation to the mean and is commonly used as a metric of relative variability in performance experiments [8, 35]. Similarly to RQ1, we test statistically significant differences using the Mann-Whitney U test [47] and assess the effect size of the difference using the Cliff's Delta effect size [42].

Findings. We observe that the vast majority of experiments (160

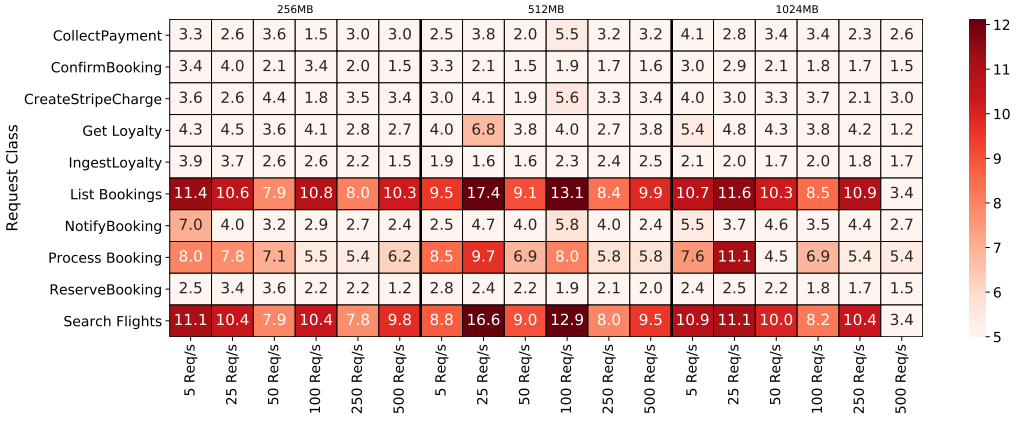

Figure 2: Coefficient of variation of the .99 mean across 10 repetitions per request class, load level, and function size. We highlight in the heatmap coefficients above 5% of the mean.

out of 180) exhibits a coefficient of variation below 10% of the mean response time. Figure 2 shows a heat map of the coefficient of variation observed in 10 repetitions of all experiments. With the exception of three request classes, *List Booking*, *Process Booking*, and *Search Flights*, most of the other experiments show a coefficient of variation of less than 5% of the mean (125 out of the 132 experiments). The observed coefficient of variation is also in line with reported variation in other serverless benchmarks [8], which was reported to be 5 to 10% when executing synthetic workloads in the AWS infrastructure. This suggests that the studied serverless application performance tests are more stable than most traditional performance tests of cloud applications (IaaS). Cito and Leitner [35] reported that performance variations of performance tests in cloud environments are consistently above 5%, reaching variations above 80% of the mean in several I/O-based workloads. The two classes with higher variability of the results, *List Booking*, and *Search Flights*, both use an Amplify resolver to retrieve data from DynamoDB without a lambda. Our findings indicate that this AWS-managed resolver might suffer from a larger performance variability.

We observe improvement in the response time and result stability in scenarios with higher workloads. Figure 3 shows the response time of the *ConfirmBooking* request class, in which we observe that as the workload increases, the average response time decreases for all function sizes.

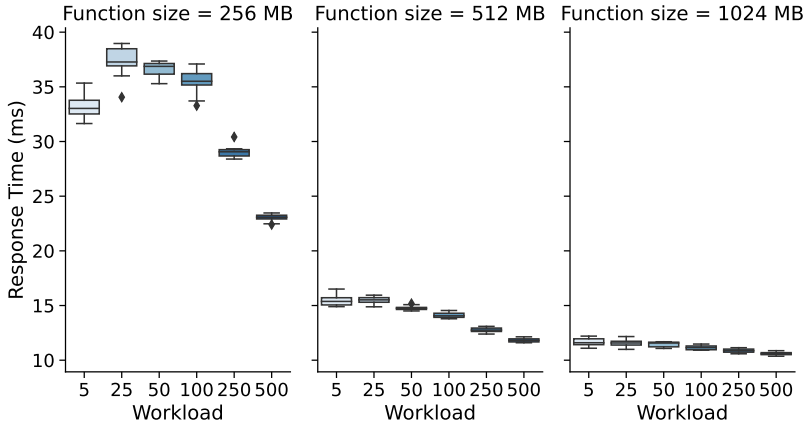

Figure 3: Response time of ten repetitions of *ConfirmBooking* performance tests, per workload level and function size.

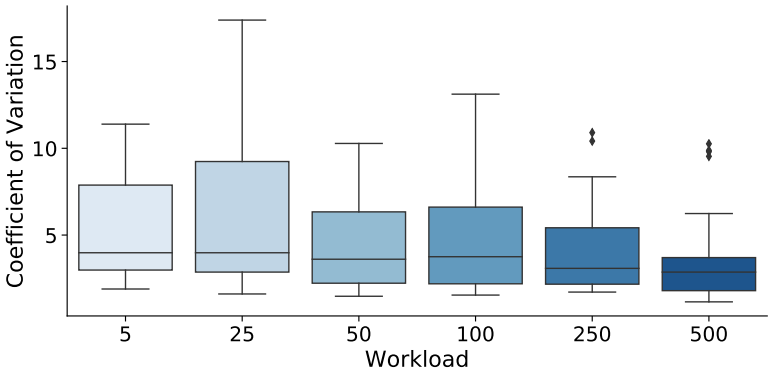

Figure 4: Distribution of coefficients of variation across all request classes and function size, per workload (reqs/s).

This is true across almost all experiments, where the response time observed in the scenario with 500 requests per second is significantly faster than scenarios with only 5 requests per second (Mann-Whitney U $p < 0.05$), often to large effect sizes (Cliff's delta $d > 0.474$). Moreover, the stability of the obtained average response time (across 10 repetitions) also improves slightly, from 4.6% on average across all experiments with 5 reqs/s, to 3.3% on experiments with 500 reqs/s (see Figure 4). Our findings suggest that the workload in the studied serverless application showed an inverse relationship to measured performance, that is, the higher the workload we tested the faster was the average response time, the opposite of what is expected in most typical systems (bare-metal, cloud environments). It is important to note that, given the cost model of serverless infrastructure, performance tests with 500 requests per second cost 100 x more than tests with 5 requests per second. Therefore, the small gain in stability is unlikely to justify the much higher costs of running performance tests in practice.

While the response time improves on larger function sizes, the stability of the tests is not affected significantly by the allocated memory. We note in Figure 3 that the *ConfirmBooking* average response time is considerably faster when the function size is 512 MB or larger. However, we do not observe any significant difference in the stability of the experiments (coefficient of variation) across different function sizes (Mann-Whitney U test $p > 0.05$). This means that the amount of memory allocated for the function has an impact on its response time (expected), but exerts no significant influence on the stability of experiments.

5.3. RQ3: Does the performance of serverless applications change over time?

Motivation. RQ1 and RQ2 focus on the stability of performance tests conducted within the same time frame. However, the opaque nature of the underlying resource environments introduces an additional challenge: the underlying resource environment may change without notice. This might result in both short-term performance fluctuations (e.g., due to changing load on the platform) or long-term performance changes (e.g., due to software/hardware changes). Therefore, in this RQ, we conduct a longitudinal study on the performance of our SUT, to investigate if we can detect short-term performance fluctuations and long-term performance changes.

Approach. We analyze the results of our longitudinal study (described in Section 4.2), which consists of three measurement repetitions with 100

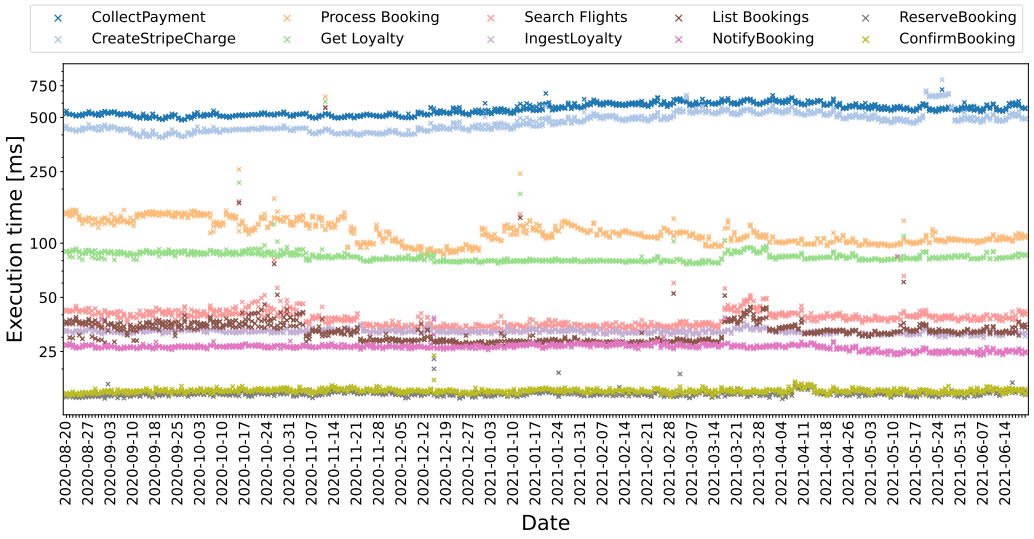

Figure 5: Mean response time for three daily performance measurements over a period of ten months.

requests per second and 512 MB memory size every day for ten months. First, to determine if there are any significant changes in the distribution of the measurement results over time, we employ the change point detection approach from Daly et al. [10]. To reduce the sensibility to short-term fluctuations, we use the median response time of the three daily measurements and configure the approach with $p = 0$ and 100,000 permutations. Second, upon visual inspection, it seemed that the variation between the three daily measurement repetitions was less than the overall variation between measurements. To investigate this, we conducted a Monte Carlo simulation that randomly picks 100,000 pairs of measurements that were conducted on the same day and 100,000 measurement pairs from different days. We calculated and compared the average variation between the sample pairs from the same day and from different days. Finally, to investigate if the observed performance variation could be misinterpreted as a real performance change (regression), we conducted a second Monte Carlo simulation. We randomly select two sets of ten consecutive measurements that do not overlap and test for a significant difference between the pairs using the Mann–Whitney U test [47]. For each detected significant difference, we calculate Cliff's Delta [42] to quantify the effect size. Similar to our first Monte Carlo simulation, we repeat this selection and comparison 100,000 times. Further implementation details are

Table 4: Comparison of average performance variation between two measurements from either the same day or different days based on a Monte Carlo simulation.

Request class	Same-day Variation	Overall Variation
ConfirmBooking	2.1% ± 2.6%	2.8% ± 3.0%
CreateStripeCharge	2.2% ± 2.1%	13.3% ± 11.0%
Get Loyalty	3.0% ± 20.0%	7.0% ± 22.0%
IngestLoyalty	1.6% ± 1.6%	2.9% ± 2.4%
List Bookings	7.0% ± 59.3%	16.2% ± 65.3%
NotifyBooking	2.3% ± 8.5%	4.3% ± 8.4%
Process Booking	3.5% ± 16.9%	17.2% ± 20.4%
ReserveBooking	2.4% ± 3.2%	3.2% ± 3.8%
CollectPayment	1.9% ± 2.0%	8.3% ± 6.1%
Search Flights	7.1% ± 50.1%	13.2% ± 49.6%

available in our replication package.[3]

Findings. There were short-term performance fluctuations during our longitudinal study, despite the fact that no changes were made to the application. Figure 5 presents the average response time of each API endpoint during the study periods. We can clearly observe fluctuations in performance. For example, the response time of the API *Process Booking* has demonstrated large fluctuation after October 2020. Table 4 compares the variation of performance between measurements from the same day and across different days (overall) using a Monte Carlo simulation. We find that in all of the API endpoints, the average variation between two random measurements is higher than the variation between measurements from the same day. For example, *Process Booking* has an average variation of 17.2% when considering all measurements, which is more than four times the average variation between measurements from the same day (3.5%).

We detect long-term performance changes during the observation period. Figure 6 presents the detected long-term performance changes in the different APIs, according to the change point detection. Although some API endpoints have more change points than others, all of the API

[3] `https://github.com/ServerlessLoadTesting/ReplicationPackage`

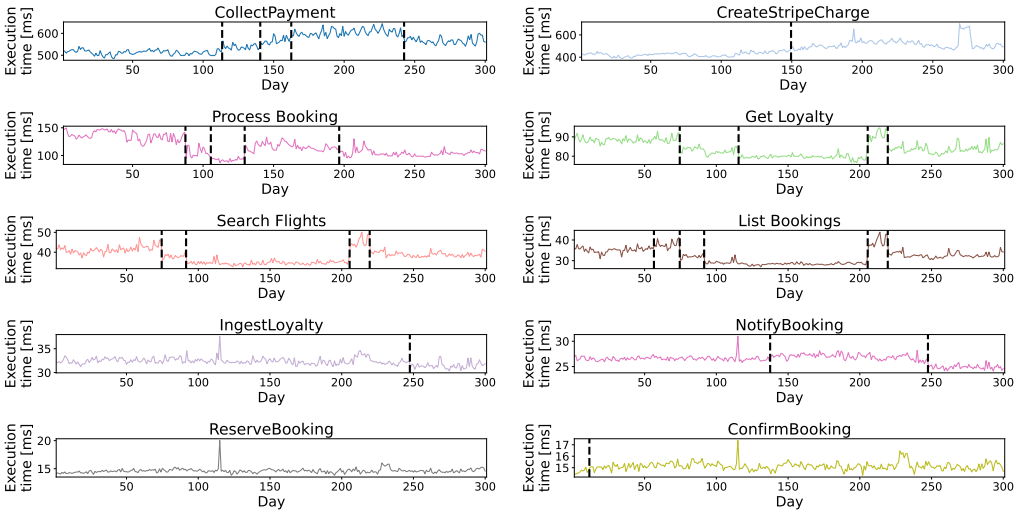

Figure 6: Detected change points for each workload class, note the different y-axis scales.

endpoints, except for *ReserveBooking*, have gone through at least one change point (the change point in *ConfirmBooking* might also be a false positive, as it is quite close to the experiment start). There exist as many as five change points during the observation period for an API endpoint. The impact of the performance change may be drastic. For example, the API endpoints *Search Flights* and *List Bookings* have similar performance changes where the response time is drastically reduced twice. On the other hand, the response time of some API endpoints, for example, *CollectPayment*, increases in each change point, leading to a potential unexpected negative impact on the end-user experience. Finally, most of the change points for the different API endpoints do not appear at the same time, which may further increase the challenge of maintaining the performance of the serverless applications.

The short-term performance fluctuations and long-term performance changes may have been considered as false performance regressions. Table 5 shows the results of conducting the Mann-Whitney U test and measuring Cliff's delta between two groups of consecutive, non-overlapping samples based on our Monte Carlo simulation. We find that for four API endpoints, almost half of the comparisons have a statistically significant performance difference, even though the serverless application itself was *identical* throughout the observation period. On the other hand, most of the differences have lower than medium effect sizes. In other words, the mag-

Table 5: Percentage of at least negligible, small, or medium differences according to Mann-Whitney U test and Cliff's delta based on Monte Carlo simulation.

Request class	Negligible+	Small+	Medium+
ConfirmBooking	54.0%	17.4%	6.8%
CreateStripeCharge	11.0%	3.8%	1.7%
Get Loyalty	21.6%	7.1%	2.9%
IngestLoyalty	43.3%	14.0%	5.6%
List Bookings	19.2%	6.8%	3.1%
NotifyBooking	37.1%	11.9%	4.7%
Process Booking	14.0%	5.0%	2.3%
ReserveBooking	57.0%	18.5%	7.0%
CollectPayment	13.2%	4.4%	1.9%
Search Flights	24.3%	7.9%	3.3%

nitude of the differences may be small and negligible, such that the impact on end users may not be drastic. However, there still exist cases whether large effect sizes are observed. Practitioners may need to be aware of such cases due to their large potential impact on end-user experience.

6. Discussion

According to Jiang et al. [25], performance tests consist of three stages: (1) designing the test, (2) running the test, and (3) analyzing the test results. Based on the findings from our case study, we identified multiple properties of performance tests of serverless applications that practitioners should consider in each of these stages, as shown in Figure 7.

6.1. Design Phase

During the design of a performance test, the key factors are the workload (which types of requests in which order), the load intensity (the number of requests), and the duration of the performance test.

6.1.1. Unintuitive performance scaling (D1).

One of the key selling points of serverless platforms is their ability to seamlessly, and virtually infinitely scale with increasing traffic [18]. Therefore, the classical approach of running performance tests at increasing load

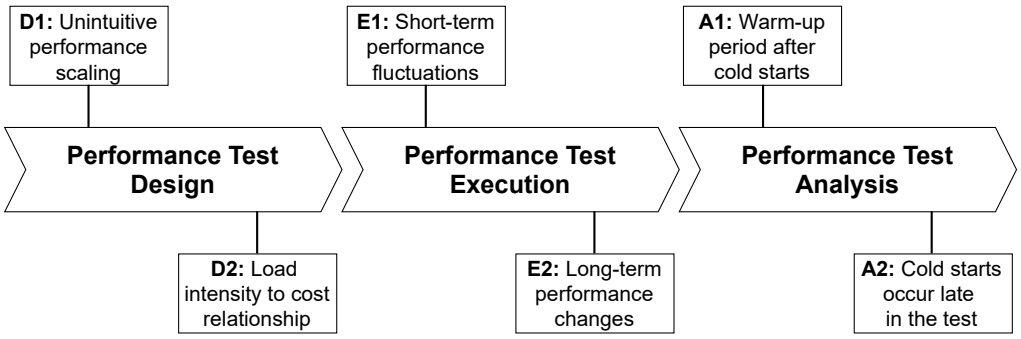

Figure 7: Properties of serverless that influence the different performance test stages.

intensities, to see how much the performance deteriorates, becomes obsolete. We find in our experiments that the performance still differs at different load levels, however, and perhaps counterintuitively, the execution time decreases with increasing load. This property impacts how to plan performance tests. For example, a developer might run a performance test at 200 requests per second and find that the performance satisfies the SLA; however when the application is deployed and receives only 100 requests per second, it might violate the SLA. *Therefore, developers need to consider that the worst performance is no longer observed at the highest load.* Depending on the use case, performance testing strategies could aim to: (a) quantify the expected performance by aiming to match the production load level, (b) understand how different load levels impact the performance by measuring a range of load intensities, or (c) aim to find the worst case performance with a search-based approach.

6.1.2. Load intensity to cost relationship (D2).

Traditionally, the cost of a performance test is independent of the load intensity and depends only on the number of deployed VMs and the duration of the experiment. For a serverless application, this relationship is inverted due to the pay-per-use pricing model of serverless. Due to this per-request pricing, the costs of a performance test has a linear relationship to the total number of requests in a performance test, for example, a performance test with 50 requests per second costs ten times as much as a performance test with 5 requests per second. This changes how developers should think about the costs of a performance test. Additionally, increasing the load intensity from five requests per second to 500 requests per second resulted in only a

minor increase in result stability in our case study. Therefore, *running more repetitions of a performance test at low load intensity instead of a single, large test could result in more stable results at the same cost.* However, further experiments in this direction are required to determine how much this increases the result stability.

6.2. Execution Phase

For the execution of a performance test, performance engineers need to decide when and how the test is executed. The technical implementation of a performance test is mostly unaffected by the switch to serverless applications, as most tooling for the performance testing of HTTP APIs (e.g., for microservice applications) can be reused. However, we find that there are two properties of serverless applications that influence the scheduling of performance tests.

6.2.1. Short-term performance fluctuations (E1).

We find that the performance of serverless applications can suffer from short-term (daily) performance variation. While performance variation has also been observed for virtual machines [24, 35], we find that the variation between measurements conducted on different days is larger than for measurements conducted on the same day for serverless applications. Depending on the goal of a performance test, this has different implications. If the goal is to compare the performance of two alternatives (e.g., to answer the question if the performance of an application changed between two commits), then the measurements for both alternatives should be conducted on the same day. On the other hand, *if the goal of a performance test is to quantify the performance of an application, the measurement repetitions should be spread across multiple days* as this will result in a more representative performance.

6.2.2. Long-term performance changes (E2).

We detect a number of long-term performance changes that caused the performance of the application to permanently change in our case study, despite no changes being made to the application itself. We hypothesize that these performance changes are caused by updates to the software stack of the serverless platform; however, most serverless services do not offer any publicly available versioning that could be used to corroborate this. Unlike the short-term fluctuations, this issue can not be combated by running a

larger number of measurement repetitions or by adopting robust measurement strategies such as multiple randomized interleaved trials [1]. When comparing two alternatives, they should be measured at the same time to minimize the chance of a long-term performance change occurring between the measurements, which is currently not necessarily the case, for example, for performance regression testing. *Quantifying the performance of a serverless application is no longer a discrete task, but rather a continuous process, as the performance of a serverless application can change over time.*

6.3. Analysis Phase

In this phase, the monitoring data collected during the execution phase is analyzed to answer questions related to the performance of the SUT. A key aspect of this phase is the removal of the warm-up period to properly quantify the steady-state performance.

6.3.1. Warm-up period after cold starts (A1).

The performance of a serverless application is generally separated into cold starts, which include initialization overheads, and warm starts, which are considered to have reached the steady-state phase and yield a more stable performance. We find that a performance test can still have a warm-up period even after excluding cold starts. A potential reason might be that, for example, caches of the underlying hardware still need to be filled before steady-state performance is reached. This indicates that *in the analysis of performance test results, the warm-up period still needs to be analyzed and excluded.* For our data, MSER-5, the current best practice to determine the warm-up period [43, 75], was not applicable due to large outliers present in the data, a well-documented flaw of MSER-5 [57]. Therefore, future research should investigate suitable approaches for detecting the warm-up period of serverless applications.

6.3.2. Cold starts occur late in the test (A2).

Another aspect about cold starts is that for a constant load, one could expect to find cold starts only during the warm-up period. In our experiments, we found that while the vast majority of cold starts occur during the warm-up period, some cold starts are scattered throughout the experiment. This might be, for example, due to worker instances getting recycled [41]. While these late cold starts did not significantly impact the mean execution time, they might impact more tail-sensitive measures such as the 99^{th} percentile.

Therefore, *performance testers need to keep the possibility of late cold starts in mind while analyzing performance testing results.*

7. Threats to Validity

This section first introduces the limitations of our study and then discusses the threats to validity that arise from these limitations. We consider the following to be the main limitations of this study:

L1 Single system under test. This study uses only a single system under test, the serverless airline application.

L2 Single cloud platform. This study is limited to AWS and does not consider any other cloud providers.

L3 Constant load. This study does not investigate the impact of varying load patterns as the experiments all use constant load.

L4 Black-box view. As this study is conducted on a public cloud, it is limited to the metrics exposed by the cloud provider.

In the following, we discuss the threats to the construct, internal, and external validity that arise from these limitations [76]. Construct validity examines the relation of the measurements to the proposed research questions. Internal validity examines the trustworthiness of the cause-and-effect relationship, that is, the existence of alternative explanations for findings, and external validity considers how well the results can be generalized.

7.1. Construct Validity

In our experiments, we measured only the response time and function execution time; other metrics might show different effects. Out of the commonly used performance metrics, we did not consider CPU utilization and throughput. However, measuring the throughput is unusual for serverless applications due to their built-in scalability, and CPU utilization is currently not exposed by AWS (**Limitation L4**). Further, we limited our experiments to performance tests with a constant load (**Limitation L3**); performance tests with varying load might behave differently. Constant load is commonly used for performance tests, whereas varying load is more commonly used for load and stress testing. However, further research is required to understand the effects of performance tests under varying load.

7.2. Internal Validity

As the MSER-5 method for determining the duration of the warm-up period was not applicable to our data, we used a custom heuristic. It might be possible that this heuristic does not appropriately capture the length of the warm-up period. Based on a visual inspection of a large subset of the experiments, we found that the heuristic seems to capture the warm-up period well. Our replication package can be used to repeat this visual inspection.

Another threat to the validity of our results is that performance experiments in the cloud can suffer from a high degree of uncertainty. To mitigate this threat, we followed recommended practices for conducting and reporting cloud experiments [54] and used randomized multiple interleaved trials [1] to reduce measurement variability. Further, we provide a fully automated measurement harness that enables the replication of our measurements. For the longitudinal study, we perform three measurement repetitions each day at the same time to mitigate measurement variability, but we do not attempt to further control for performance variability as the study was intended to investigate the variability.

7.3. External Validity

Our case study used only a single SUT (**Limitation L1**), which might limit the generalizability of our results. However, the serverless airline booking application is larger (uses more functions) than the average serverless application [18, 64], so independent parts of the application could also be considered multiple applications. Further, most of the properties we measure are more dependent on the underlying cloud platform than the application itself. However, it is possible that a different application, such as a scientific computing application with long-running functions might behave differently. While our experiments were conducted on one application only, our methodology is applicable to any application. Another threat is that we conduct measurements on a single cloud platform (**Limitation L2**). Although AWS is by far the most popular cloud provider for serverless applications, with 55%-70% of serverless applications running on AWS [18, 67, 11], further research is required to determine if our findings are transferable between cloud providers.

8. Replication Package

Performance measurements of public cloud environments are per definition only a snapshot of the performance at the time of measurement [35, 1, 24]. The performance properties can change whenever the cloud provider upgrades its hardware, switches to newer versions of the underlying operating system or virtualization technology, introduces new optimizations or features for the offered services, or changes any number of configuration parameters [14, 24]. To increase our results' longevity, we provide a replication package that allows other researchers to replicate our findings and enables tracking if and how the reported performance properties evolve over time. This is in line with the recently proposed methodological principles for the reproducible performance evaluation of public clouds by Papadopoulos et al. [54].

Our replication package[4] consists of two parts: (a) the experiment harness used to run the performance measurements and (b) the data and analysis scripts used in the presented analysis. We provide the experiment harness as a Docker container that replicates all measurements conducted in this study with a single CLI command from any Docker-capable machine. To simplify the reuse of this harness in other studies, experiments can be specified as JSON files, including measurement duration, load intensity, load pattern, measurement repetitions, and system configuration. The second part of our replication package is a CodeOcean capsule containing the collected measurement data and the scripts for the analysis presented in this paper. The CodeOcean capsule enables a one-click replication of our analysis either on the measurement data we collected or on new measurement data collected using our measurement harness.

9. Conclusion

Serverless applications delegate resource management tasks, such as deployment, resource allocation, or auto-scaling, to the cloud provider [29, 16], who bills users on a pay-per-use basis [6, 71]. A common and powerful approach to manage system performance is the regular execution of performance tests; however, performance tests require that an identical resource environment is used for all tests, which cannot be guaranteed for a serverless

[4] `https://github.com/ServerlessLoadTesting/ReplicationPackage`

application [14]. Therefore, we conducted an exploratory case study on the stability of performance tests of serverless applications, including a longitudinal study of daily measurements for ten months.

We find that in our case study there are serverless-specific changes and pitfalls to all performance test phases: design, execution, and analysis. In the design phase, the load intensity of the test directly correlates to cost, and reducing load intensity can decrease performance. In the execution phase, daily performance fluctuations and long-term performance changes impact the decision when performance tests should be scheduled. In the analysis phase, developers need to consider that there is still a warm-up period after removing all cold starts and that cold starts can occur late in a performance test under constant load.

Acknowledgments

The work was conducted by the SPEC RG DevOps Performance Working Group.[5] This work was supported by the AWS Cloud Credits for Research program. The authors would like to thank Heitor Lessa for his support with the serverless airline booking application, as well as David Daly and Alexander Costas for their input on the change point detection.

References

[1] A. Abedi and T. Brecht. Conducting repeatable experiments in highly variable cloud computing environments. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ICPE '17, page 287–292, New York, NY, USA, 2017. Association for Computing Machinery.

[2] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI'20)*, pages 419–434, 2020.

[3] M. Al-Ameen and J. Spillner. Systematic and open exploration of faas and serverless computing research. In *Proceedings of the European Sym-*

[5] `https://research.spec.org/devopswg`

posium on Serverless Computing and Applications (ESSCA '18), volume 2330, pages 30–35, 2018.

[4] M. M. Arif, W. Shang, and E. Shihab. Empirical study on the discrepancy between performance testing results from virtual and physical environments. *Empir. Softw. Eng.*, 23(3):1490–1518, 2018.

[5] E. Asyabi, M. Sharifi, and A. Bestavros. ppxen: A hypervisor CPU scheduler for mitigating performance variability in virtualized clouds. *Future Gener. Comput. Syst.*, 83:75–84, 2018.

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

[7] D. Boutcher and A. Chandra. Does virtualization make disk scheduling passé? *ACM SIGOPS Oper. Syst. Rev.*, 44(1):20–24, 2010.

[8] R. Cordingly, W. Shu, and W. J. Lloyd. Predicting performance and cost of serverless computing functions with SAAF. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 640–649. IEEE, 2020.

[9] D. E. Costa, C.-P. Bezemer, P. Leitner, and A. Andrzejak. What's wrong with my benchmark results? Studying bad practices in JMH benchmarks. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[10] D. Daly, W. Brown, H. Ingo, J. O'Leary, and D. Bradford. The use of change point detection to identify software performance regressions in a continuous integration system. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 67–75, New York, NY, USA, 2020. Association for Computing Machinery.

[11] J. Daly. Serverless community study. `https://github.com/jeremydaly/serverless-community-survey-2020`, 2020.

[12] Datadog. The state of serverless. `https://www.datadoghq.com/state-of-serverless/`, 2020.

[13] K. Djemame, M. Parker, and D. Datsev. Open-source serverless architectures: an evaluation of apache openwhisk. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pages 329–335. IEEE, 2020.

[14] S. Eismann, C.-P. Bezemer, W. Shang, D. Okanovic, and A. van Hoorn. Microservices: A Performance Tester's Dream or Nightmare? In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE)*, ICPE'20, pages 138—-149, April 2020.

[15] S. Eismann, J. Grohmann, E. van Eyk, N. Herbst, and S. Kounev. Predicting the costs of serverless workflows. In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20)*, page 265–276, New York, NY, USA, April 2020. Association for Computing Machinery (ACM).

[16] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup. A review of serverless use cases and their characteristics. Technical report, SPEC RG, June 2020.

[17] S. Eismann, L. Bui, J. Grohmann, C. Abad, N. Herbst, and S. Kounev. Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International MIDDLEWARE Conference*, 2021.

[18] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup. Serverless applications: Why, when, and how? *IEEE Software*, 38(1):32–39, 2021.

[19] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup. The state of serverless applications: Collection, characterization, and community consensus. *Transactions on Software Engineering*, 2021.

[20] T. Elgamal. Costless: Optimizing cost of serverless computing through function fusion and placement. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 300–312. IEEE, 2018.

[21] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski. Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience*, 30(23):e4792, 2018.

[22] K. Hoad, S. Robinson, and R. Davies. Automating warm-up length estimation. *Journal of the Operational Research Society*, 61(9):1389–1403, 2010.

[23] N. Huber, M. von Quast, M. Hauck, and S. Kounev. Evaluating and modeling virtualization performance overhead for cloud environments. In *CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science, Noordwijkerhout, Netherlands, 7-9 May, 2011*, pages 563–573. SciTePress, 2011.

[24] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 104–113, 2011.

[25] Z. M. Jiang and A. E. Hassan. A survey on load testing of large-scale software systems. *IEEE Trans. Software Eng.*, 41(11):1091–1118, 2015.

[26] M. Kalita and T. Bezboruah. Investigation on performance testing and evaluation of PReWebD: a .NET technique for implementing web application. *IET Softw.*, 5(4):357–365, 2011.

[27] J. Kim and K. Lee. Functionbench: A suite of workloads for serverless cloud function service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 502–504, 2019. doi: 10.1109/CLOUD.2019.00091.

[28] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '07)*, pages 200–209. IEEE Computer Society, 2007.

[29] S. Kounev, C. Abad, I. T. Foster, N. Herbst, A. Iosup, S. Al-Kiswany, A. A.-E. Hassan, B. Balis, A. Bauer, A. B. Bondi, K. Chard, R. L. Chard, R. Chatley, A. A. Chien, A. J. J. Davis, J. Donkervliet, S. Eismann, E. Elmroth, N. Ferrier, H.-A. Jacobsen, P. Jamshidi,

G. Kousiouris, P. Leitner, P. G. Lopez, M. Maggio, M. Malawski, B. Metzler, V. Muthusamy, A. V. Papadopoulos, P. Patros, G. Pierre, O. F. Rana, R. P. Ricci, J. Scheuner, M. Sedaghat, M. Shahrad, P. Shenoy, J. Spillner, D. Taibi, D. Thain, A. Trivedi, A. Uta, V. van Beek, E. van Eyk, A. van Hoorn, S. Vasani, F. Wamser, G. Wirtz, and V. Yussupov. Toward a Definition for Serverless Computing. In C. Abad, I. T. Foster, N. Herbst, and A. Iosup, editors, *Serverless Computing (Dagstuhl Seminar 21201)*, volume 11 (5), chapter Chapter 5.1, page TBA. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021. doi: 10.4230/DagRep.11.5.1.

[30] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. IO performance prediction in consolidated virtualized environments. In *ICPE'11 - Second Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE '11)*, pages 295–306. ACM, 2011.

[31] C. Laaber and P. Leitner. An evaluation of open-source software microbenchmark suites for continuous performance assessment. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pages 119–130. ACM, 2018.

[32] C. Laaber, J. Scheuner, and P. Leitner. Software microbenchmarking in the cloud. How bad is it really? *Empirical Software Engineering (EMSE)*, 24(4):2469–2508, April 2019.

[33] H. Lee, K. Satyam, and G. Fox. Evaluation of production serverless computing environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 442–450, 2018.

[34] P. Leitner and C.-P. Bezemer. An exploratory study of the state of practice of performance testing in Java-based open source projects. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ICPE '17, page 373–384, New York, NY, USA, 2017. Association for Computing Machinery.

[35] P. Leitner and J. Cito. Patterns in the chaos—a study of performance variation and predictability in public IaaS clouds. *ACM Trans. Internet Technol.*, 16(3), Apr. 2016.

[36] 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*, 149:340–359, 2019.

[37] H. Lessa. Production-grade full-stack apps with AWS Amplify. `https://www.youtube.com/watch?v=DcrtvgaVdCU`, 2019.

[38] E. Levinson. Serverless community survey 2020. `https://www.nuweba.com/blog/serverless-community-survey-2020-results`, 2020.

[39] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, and S. Sajedi. Using black-box performance models to detect performance regressions under varying workloads: an empirical study. *Empir. Softw. Eng.*, 25(5):4130–4160, 2020.

[40] C. Lin and H. Khazaei. Modeling and optimization of performance and cost of serverless applications. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):615–632, 2020.

[41] 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)*, pages 159–169. IEEE, 2018.

[42] J. D. Long, D. Feng, and N. Cliff. *Ordinal Analysis of Behavioral Data*. John Wiley & Sons, Inc., 2003.

[43] P. S. Mahajan and R. G. Ingalls. Evaluation of methods used to detect warm-up period in steady state simulation. In *Proceedings of the 36th conference on Winter simulation (WSC '04)*, pages 663–671. IEEE Computer Society, 2004.

[44] N. Mahmoudi and H. Khazaei. Performance modeling of serverless computing platforms. *IEEE Transactions on Cloud Computing*, 2020.

[45] P. Maissen, P. Felber, P. Kropf, and V. Schiavoni. Faasdom: A benchmark suite for serverless computing. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20)*, pages 73–84, 2020.

[46] H. Malik, H. Hemmati, and A. E. Hassan. Automatic detection of performance deviations in the load testing of large scale systems. In *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pages 1012–1021. IEEE Computer Society, 2013.

[47] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18 (1):50–60, 03 1947.

[48] R. K. Mansharamani, A. Khanapurkar, B. Mathew, and R. Subramanyan. Performance testing: Far from steady state. In *Workshop Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC Workshops 2010*, pages 341–346. IEEE Computer Society, 2010.

[49] G. McGrath and P. R. Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW '17)*, pages 405–410, 2017.

[50] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st International Conference on Virtual Execution Environments (VEE '05)*, pages 13–23. ACM, 2005.

[51] M. A. S. Netto, S. Menon, H. V. Vieira, L. T. Costa, F. M. de Oliveira, R. S. Saad, and A. F. Zorzo. Evaluating load generation in virtualized environments for software performance testing. In *Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011), Workshop Proceedings*, pages 993–1000. IEEE, 2011.

[52] T. H. Nguyen, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora. An industrial case study of automatically identifying performance regression-causes. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pages 232–241, 2014.

[53] T. H. D. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. N. Nasser, and P. Flora. Automated detection of performance regressions using

statistical process control techniques. In *Third Joint WOSP/SIPEW International Conference on Performance Engineering (ICPE'12)*, pages 299–310. ACM, 2012.

[54] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-eldin, C. Abad, J. N. Amaral, P. Tůma, and A. Iosup. Methodological principles for reproducible performance evaluation in cloud computing. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[55] B. A. Pozin and I. V. Galakhov. Models in performance testing. *Program. Comput. Softw.*, 37(1):15–25, 2011.

[56] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices. In *Annual meeting of the Southern Association for Institutional Research*, pages 1–51, 2006.

[57] B. Sandıkçı and İ. Sabuncuoğlu. Analysis of the behavior of the transient period in non-terminating simulations. *European Journal of Operational Research*, 173(1):252–267, 2006.

[58] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endow.*, 3(1–2):460–471, Sept. 2010.

[59] J. Scheuner and P. Leitner. Function-as-a-service performance evaluation: A multivocal literature review. *Journal of Systems and Software (JSS)*, 2020.

[60] A. W. Services. Build on serverless - architect an airline booking application. `https://pages.awscloud.com/GLOBAL-devstrategy-OE-BuildOnServerless-2019-reg-event.html`, 2019.

[61] A. W. Services. Aws amplify - fastest, easiest way to build mobile and web apps that scale. `https://aws.amazon.com/amplify/`, 2021.

[62] A. W. Services. Aws cloud formation - speed up cloud provisioning with infrastructure as code. `https://aws.amazon.com/cloudformation/`, 2021.

[63] A. W. Services. Aws serverless application model -build serverless applications in simple and clean syntax. `https://aws.amazon.com/serverless/sam/`, 2021.

[64] M. Shahrad, R. Fonseca, Í. 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. In *2020 USENIX Annual Technical Conference (USENIXATC 20)*, pages 205–218, 2020.

[65] W. Shang, A. E. Hassan, M. N. Nasser, and P. Flora. Automated detection of performance regressions using regression models on clustered performance counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE '15)*, pages 15–26. ACM, 2015.

[66] C. U. Smith and L. G. Williams. New software performance antipatterns: More ways to shoot yourself in the foot. In *Int. CMG Conference*, pages 667–674. Citeseer, 2002.

[67] T. N. Stack. Guide to serverless technologies. `https://thenewstack.io/ebooks/serverless/guide-to-serverless-technologies/`, 2018.

[68] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. N. Nasser, and P. Flora. Leveraging performance counters and execution logs to diagnose memory-related performance issues. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM '13)*, pages 110–119. IEEE Computer Society, 2013.

[69] P. Vahidinia, B. Farahani, and F. S. Aliee. Cold start in serverless computing: Current trends and mitigation strategies. In *Proceedings of the 2020 International Conference on Omni-layer Intelligent Systems (COINS '20)*, pages 1–7. IEEE, 2020.

[70] E. van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann. A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*, page 21–24, New York, NY, USA, 2018. ACM.

[71] E. van Eyk, J. Grohmann, S. Eismann, A. Bauer, L. Versluis, L. Toader, N. Schmitt, N. Herbst, C. L. Abad, and A. Iosup. The SPEC-RG reference architecture for FaaS: From microservices and containers to serverless platforms. *IEEE Internet Computing*, 23(6):7–18, nov 2019.

[72] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '18, page 223–236, September 2018.

[73] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIXATC '18)*, pages 133–146, 2018.

[74] E. J. Weyuker and F. I. Vokolos. Experience with performance testing of software systems: Issues, an approach, and case study. *IEEE Trans. Software Eng.*, 26(12):1147–1156, 2000.

[75] K. White, M. Cobb, and S. Spratt. A comparison of five steady-state truncation heuristics for simulation. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, volume 1, pages 755–760 vol.1, 2000.

[76] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell. *Experimentation in Software Engineering*. Springer, 2012.

[77] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*, pages 30–44, 2020.