Colder Than the Warm Start and Warmer Than the Cold Start! Experience the Spawn Start in FaaS Providers

Sashko Ristov
sashko.ristov@uibk.ac.at
University of Innsbruck
Innsbruck, Austria

Christian Hollaus
christian.hollaus@student.uibk.ac.at
University of Innsbruck
Innsbruck, Austria

Mika Hautz
mika.hautz@student.uibk.ac.at
University of Innsbruck
Innsbruck, Austria

ABSTRACT

Many researchers reported considerable delay of up to a few seconds when invoking serverless functions for the first time. This phenomenon, which is known as a *cold start*, affects even more when users are running multiple serverless functions orchestrated in a workflow. However, in many cases users need to instantly spawn numerous serverless functions, usually as a part of parallel loops. In this paper, we introduce the *spawn start* and analyze the behavior of three Function-as-a-Service (FaaS) providers AWS Lambda, Google Cloud Functions, and IBM Cloud Functions when running parallel loops, both as warm and cold starts. We conducted a series of experiments and observed three insights that are beneficial for the research community. Firstly, cold start on IBM Cloud Functions, which is up to 2 s delay compared to the warm start, is negligible compared to the spawn start because the latter generates additional 20 s delay. Secondly, Google Cloud Functions' cold start is "warmer" than the warm start of the same serverless function. Finally, while Google Cloud Functions and IBM Cloud Functions run the same serverless function with low concurrency faster than AWS Lambda, the spawn start effect on Google Cloud Functions and IBM Cloud Functions makes AWS the preferred provider when spawning numerous serverless functions.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Theory of computation** → *Distributed algorithms*.

KEYWORDS

Function-as-a-Service, cold start, overhead, performance, serverless, spawn start

ACM Reference Format:
Sashko Ristov, Christian Hollaus, and Mika Hautz. 2022. Colder Than the Warm Start and Warmer Than the Cold Start! Experience the Spawn Start in FaaS Providers. In *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED '22), July 25, 2022, Salerno, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3524053.3542751

1 INTRODUCTION

Function-as-a-Service (FaaS) is the latest serverless cloud paradigm, which brings many benefits to cloud users from many perspectives. It became very popular to be used in scientific computing [14], for data analytics, both at edge [21] and cloud [16], event-based systems [2], batch processing in a form of serverless workflows [24], or real-time processing with streams [15]. Moreover, it allows on-demand and real "pay-as-you-go" model, rounded to the closest 100 ms and even to 1 ms on Amazon Lambda[1] recently.

While these benefits alleviate management and development challenges, and usually reduce the monetary costs [12], still FaaS raises several non-functional challenges. For instance, FaaS providers limit their customers both at the deployment time (e.g., code size or assigned memory) and runtime (e.g., function duration or input / output data size). Further on, many researchers reported considerable delays due to the *cold start* effect, and other non-documented obstacles by FaaS providers.

In this paper we focus on various delays that appear while spawning numerous serverless functions simultaneously and closely investigate them, both for warm and cold start. Several recent works determined delayed start of serverless functions when multiple functions are invoked at the same time [16, 20, 24, 25]. This observation motivated us to introduce the *spawn start*, which appears when the user invokes serverless functions up to the concurrency limit of 1,000 serverless functions. After the exhaustive evaluation on nine cloud regions of three FaaS providers AWS Lambda, Google Cloud Functions, and IBM Cloud Functions, we determined several insights regarding the spawn start that can be very useful for the research community. They include:

- Although cold start overhead on IBM Cloud Functions is considerable, e.g., up to 2 s, it is negligible compared to the spawn start delay;
- The spawn start, although warm, may generate up to 20 s overhead compared to the cold start of a single serverless function on IBM Cloud Functions;
- Spawn start is negligible on AWS Lambda;
- Spawn start is also negligible on Google Cloud Functions. However, the cold start on Google Cloud Functions is "much warmer" than its warm start;
- Spawn start on IBM and "warm start" on Google are worse than the cold start on AWS.

This paper is organized in several sections. Section 2 presents the state-of-the-art in terms of additional overheads and techniques how to minimize them. In Section 3, we motivate our work and the setup for evaluation. Section 4 evaluates the three FaaS providers AWS Lambda, IBM Cloud Function, and Google Cloud Functions and derives several conclusions. Finally, we conclude our work and present plans for the future work in Section 5.

ApPLIED '22, July 25, 2022, Salerno, Italy

ACM ISBN 978-1-4503-9280-8/22/07.
https://doi.org/10.1145/3524053.3542751

[1]https://aws.amazon.com/lambda/pricing/

2 RELATED WORK

This section discusses several challenges in serverless computing, especially in terms of cold start effects and delayed start of serverless functions.

2.1 Cold start effects

Serverless computing, in particular FaaS, attracted many practitioners, companies, and developers in recent years. Despite the huge popularity, FaaS still needs to overcome several challenges. One of the main challenges is the so called cold start, which means that long and unpredictable delays are introduced when a new serverless function replica is invoked. In general, serverless functions that are developed in interpreted programming languages, such as JavaScript and Python, report lower cold start latency than those written in compiled programming languages, such as Java [11]. The effects of cold start have been identified as a challenge for serverless computing [31] and multiple researchers observed and analyzed cold start latency [6, 17, 19], usually from hundreds of milliseconds to multiple seconds. Cold starts can cause a significant rise in monetary cost by poisoning the execution performance in the nested function chain [8, 33]. Cold starts appear more for burst loads [5] and the effect is higher for short-running serverless functions [9], thus avoiding the cold start may lead to more than 3× speedup.

The effect of cold start may be mitigated at the application level. For instance, Lloyd [18] schedules cloud-based event triggers to keep alive the serverless platform and thus minimize the number of cold start. Despite the cost for keeping the containers alive, this approach achieves near VM-based stable performance for multiple times lower costs. Similarly, the WLEC [29] container management system architecture minimizes both the cold start delay by leveraging a container-aware three-queue model. Silva et al. [28] use "prebaking", i.e., they perform checkpoints and then restore serverless functions of previously started functions runtimes. However, since this approach may result in high start-up latency due to lazy page faults or poor data locality in SSD accesses, Ustiugov et al. [30] introduced a record-and-prefetch mechanism REAP, which eagerly loads pages used by a serverless function from a pre-recorded trace.

Other approaches, such as Oakes [22] and Yechuri[2] tend to shorten cold starts by reducing the deployment package size. With a completely opposite approach, SAND [1] tries to collocate multiple serverless functions of a workflow application into a single deployment package which reduces the number of cold starts.

Finally, some cloud providers (e.g., Google[3]) offer a minimum number of function instances that are kept alive and are idle. Although customers are charged at a reduced rate while these instances are idle, still, this approach generates additional monetary costs.

2.2 Workload effects

The main benefit of orchestrating serverless functions in a serverless workflow is the option to spawn numerous functions simultaneously. However, many researchers reported that sometimes, even several seconds are needed to spawn hundreds of serverless functions. Among others, Ristov et al. reported a delay of more than 0.6 s when running 1,000 serverless functions simultaneously on a single region of AWS Lambda using their xAFCL FC management system [25]. The same authors detected considerable delay while running serverless workflows on AWS Step Functions [24]. Jonas et al. [16] reported several seconds delay to invoke 100 serverless functions on IBM. In order to alleviate the effect of the overloading, the authors recommended to invoke the serverless functions in a cascade, which significantly reduced the overall delay. On the other side, Manner and Wirtz [20] reported that serverless functions' behavior is affected by the workload. As a consequence, models that use average values may generate a large inaccuracy when estimating runtime of serverless functions.

3 MOTIVATION

While the related work analyzed and recommended techniques to alleviate the cold start and workload effects, to the best of our knowledge, we could not find work that analyzed both challenges simultaneously. Mainly, the cold start effects are investigated for small workload, while the big workload is analyzed mainly for warm starts. Therefore, in this paper, we conducted a series of experiments to investigate how the workload with spawn start affects both cold and warm starts, across multiple cloud regions of three different FaaS providers AWS Lambda, Google Cloud Functions, and IBM Cloud Functions.

We set our goal to conduct a comprehensive evaluation of cold and spawn start effects by running a real-life compute and data bound serverless application that is widely used by the research community in serverless computing. For this purpose, we carefully designed our experiment setup.

We selected the embarrassingly parallel Monte Carlo workflow that approximates π (Figure. 1). It scales with the number of serverless functions `monteCarlo` in the parallel loop. Afterwards, the function `averagePi` collects results and approximates π. The Monte Carlo workflow is a compute bound workflow and its serverless functions run the same work because they do not use external cloud services and we set the same amount of points as input to each serverless function `monteCarlo`. We ran the Monte Carlo simulation with 100 serverless functions as a part of a parallel loop. We selected the Monte Carlo simulation because it is a widely used problem by researchers, even in serverless computing [3, 4, 10, 13, 23, 25–27]. However, since either all 100 invocations would be a cold start in the first execution, or all would be a warm start in the following executions, we set the concurrency limit to 30. This means that

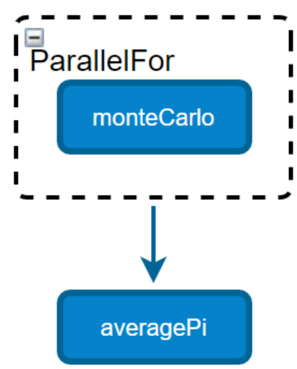

Figure 1: The Monte Carlo serverless workflow

[2]https://www.serverless.com/blog/madhuri-yechuri-unikernels-event-driven-serverless-emit-2017/

[3]https://cloud.google.com/functions/docs/configuring/min-instances

the first 30 serverless functions `monteCarlo` are invoked simultaneously, while the remaining 70 will be invoked one by one after some of the active serverless functions `monteCarlo` finish. With this approach, the first 30 invocations of the first execution of the `monteCarlo` serverless function are always cold start, while the remaining 70 invocations of the serverless function `monteCarlo` are warm start.

The Monte Carlo workflow was developed with the Abstract Function Choreography Language [24] and its serverless functions were deployed on three FaaS providers AWS Lambda, Google Cloud Functions, and IBM Cloud Functions. Each serverless function was deployed with 256 MB. We used three regions of each FaaS provider across three continents America, Europe, and Asia Pacific. With this methodology, we used nine cloud regions in total. For execution, we used the xAFCL serverless workflow management system [25].

4 EVALUATION

This section presents the evaluation of joint effect from cold and spawn start on three FaaS providers.

4.1 AWS Lambda

Figure 2 shows the execution time of the `monteCarlo` serverless function on three AWS regions Frankfurt (EU), North Virginia (US), and Tokyo (AC). We observe that all three evaluated AWS cloud regions reported similar behavior in terms of execution time. Our detailed analysis, which we determined using the Serverless Application Analytics Framework (SAAF) [7], reported that only one CPU Intel Xeon E5-2670 v2 @ 2.50GHz was obtained in all three evaluated AWS cloud regions. The first 30 invocations, which are all cold start, run longer than the remaining 70 serverless functions with warm start. As expected, we observed the overhead for the cold start. That is, the average execution time of cold starts is 7.45 s, while the average execution time of the serverless functions with a *warm* start is 7.24 s. The overhead of the cold start is 210 ms or 2.9 %. For the long-running serverless functions, the relative overhead may be considered as negligible, but not for the short-running serverless functions.

In terms of the spawn start, there is no visible impact in all three regions. This observation is inline with other works who reported that AWS scales well up to 200 concurrent invocations [32] and is valid both for cold and warm spawn start.

4.2 IBM Cloud Functions

More interesting and surprising results were achieved on IBM Cloud Functions. Figure 3 shows the execution time of the `monteCarlo` serverless function in three IBM cloud regions Frankfurt (EU), Washington (US), and Tokyo (AC). We observed completely different behavior of the `monteCarlo` serverless function on IBM Cloud Functions compared to its compatriot on AWS cloud regions. Still, similar as AWS Lambda, IBM Cloud Functions showed similar behavior for all three evaluated regions.

We observed two classes of execution time. Firstly, the cold start overhead is up to two seconds compared to the warm start, which however is imperceptible due to additional 20 seconds delay introduced by the spawn start. Execution times vary greatly between different regions. Executions on the *AC* cloud region on average take 16.59 s for cold starts, and 16.12 s for warm starts, resulting in a difference of 470 ms or 2.8 %. However, executions on the *US* region on average range from 21.05 s (cold start) to 18.08 s (warm start), leading to an overhead of 2.97 s or 16.4 %. Finally, executions on the *EU* region were in the range from 14.45 s for cold starts to 10.51 s for warm starts on average, causing a delay of 3.94 s or 37.49 %. Still, despite the higher average execution time for the cold spawn start, we observed that serverless functions at the beginning or at the end of the workflow are executed faster than intermediate ones in the burst period when the FaaS provider is loaded with 30 concurrent serverless functions.

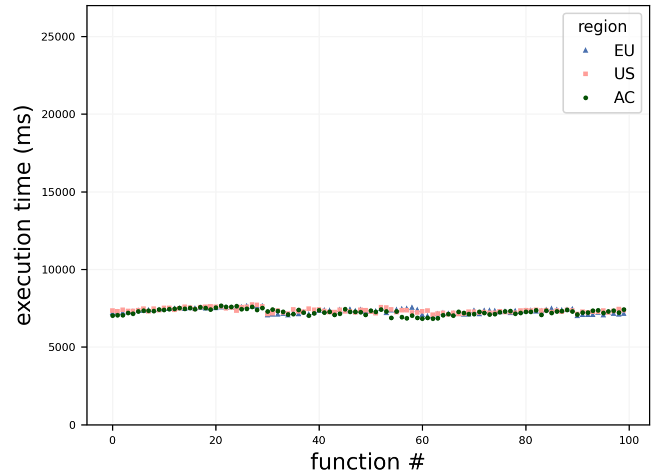

Figure 2: Execution time in ms **of 100 instances of the Monte Carlo serverless function that runs on three AWS Lambda cloud regions Frankfurt (EU), North Virginia (US), and Tokyo (AC).**

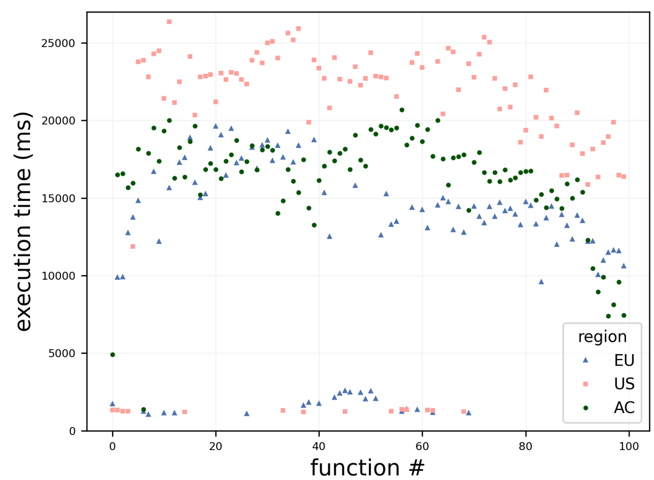

Figure 3: Execution time in ms **of** 100 **instances of the Monte Carlo serverless function that runs on three IBM Cloud Functions cloud regions Frankfurt (EU), Washington (US), and Tokyo (AC).**

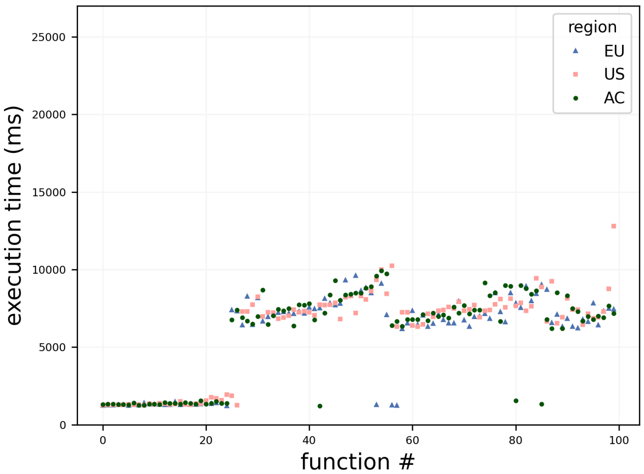

Figure 4: Execution time in ms **of 100 instances of the Monte Carlo serverless function on three Google Cloud Functions cloud regions Frankfurt (EU), North Virginia (US), and Tokyo (AC).**

As a summary, for IBM Cloud Functions, the spawn start affects mainly the serverless functions with warm start, opposite to AWS Lambda. Moreover, although some serverless functions finish on IBM Cloud Functions within 1 s even with the cold start, still, the overall parallel loop finishes when all serverless functions finish, and that is up to 26 s in the Washington (US) cloud region. The higher execution time on IBM Washington was caused by different CPUs (Intel Xeon Gold 6140 @ 2.30GHz, Intel Xeon E5-2690 v4 @ 2.60GHz, and Intel Xeon Gold 5120 @ 2.20GHz). On the other hand, only one CPU was achieved in the other two cloud regions, that is, Intel Xeon Gold 6140 @ 2.30GHz in IBM Tokyo, while Intel Xeon E5-2690 v4 @ 2.60GHz in IBM Frankfurt.

4.3 Google Cloud Functions

The third evaluated FaaS provider, Google Cloud Functions, reported even more strange behavior from the other two evaluated FaaS providers AWS Lambda and IBM Cloud Functions. Figure 4 shows the execution time of the `monteCarlo` serverless function that runs on three Google regions Frankfurt (EU), North Virginia (US), and Tokyo (AC). The paradoxical result is that Google's cold start is "warmer" than the warm start. Namely, we observed that the execution time of the first 30 invocations are much faster than the other 70 serverless functions. While the execution of a cold start serverless function on Google Cloud Functions only takes 2.86 s for all three evaluated cloud regions, the same computation takes 7.52 s for serverless functions with warm start. This results in a negative overhead of −4.66 s or −61.97 %. This means that the spawn start on Google Cloud Functions affects the serverless functions with warm starts only, rather than the cold starts.

Even more, some serverless functions with warm starts reported considerable low execution time in all three evaluated regions. We analyzed the paradoxical behavior in more detail and determined that all cold starts, including the "fast warm" starts, were scheduled on the new container, while the other "slow warm" starts on the existing container.

4.4 Discussion

After evaluating how the `monteCarlo` serverless function behaves in each cloud provider individually, we discuss how cloud providers affect the execution of serverless functions. For running under low concurrency, IBM Cloud Functions and Google Cloud Functions cause lower overhead than AWS Lambda, even including the cold start effect. Google Cloud Functions should be selected as a FaaS provider for all serverless functions that run mainly with a cold start. For example, serverless functions that are rarely invoked. Finally, AWS Lambda reported the fastest execution time for serverless functions with warm start, even for spawn start, compared to the other two evaluated FaaS providers.

5 CONCLUSION AND FUTURE WORK

This paper analyzed the spawn start effects, both under cold and warm start. We uncovered many strange and paradoxical results, which are not documented so far by the researchers and FaaS providers. The excessive set of experiments that compared the behavior of multiple cloud regions of three FaaS providers AWS Lambda, IBM Cloud Functions, and Google Cloud Functions demonstrated that the spawn start effect has to be considered when selecting the target FaaS provider, especially IBM Cloud Functions, where its impact is even higher than the regular cold start. AWS Lambda is resistant to the spawn start, but not on the cold start, which affects not only the overhead to start the container, but also the execution time of the serverless function that runs inside the container. This effect is more emphasized on IBM Cloud Functions. Finally, Google Cloud Functions' warm start is much colder that its cold start.

We will extend our investigation to generate a mathematical model for the spawn start for more complex workflows of serverless functions which access other cloud services, such as Amazon S3 storage or AWS Rekognition. Such model will be used to build a scheduling algorithm that will decide how to transform a parallel loop into multiple parallel sections, each with distinct serverless functions, such that the makespan and the monetary cost are minimized. With the novel mathematical model, we will develop a simulator for FaaS, which will accurately simulate the behavior of serverless workflows in federated clouds. Finally, we will develop a scheduler to determine the optimal execution of serverless workflows across the top five FaaS providers, which considers the effects of both cold and spawn starts.

ACKNOWLEDGEMENT

This research is funded by Land Tirol, funding contract F.35499, Bundesministerium für Digitalisierung und Wirtschaftsstandort der Republik Österreich, under the project Digital Innovation Hub West (DIH-WEST) and from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951745, in particular, the FF4EUROHPC CALL-2, for the project entitled "CardioHPC Improving DL-based Arrhythmia Classification Algorithm and Simulation of Real-Time Heart Monitoring of Thousands of Patients".

REFERENCES

[1] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *Usenix ATC*. Boston, USA, 923–935.

[2] Aitor Arjona, Pedro García López, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2021. Triggerflow: Trigger-based orchestration of serverless workflows. *Future Generation Computer Systems* 124 (2021), 215–229. https://doi.org/10.1016/j.future.2021.06.004

[3] Daniel Barcelona-Pons and Pedro García-López. 2021. Benchmarking parallelism in FaaS platforms. *Future Generation Computer Systems* (2021).

[4] Daniel Barcelona-Pons, Marc Sánchez-Artigas, Gerard París, Pierre Sutra, and Pedro García-López. 2019. On the FaaS Track: Building Stateful Distributed Applications with Serverless Architectures *(Middleware '19)*. ACM, Davis, CA, USA, 41–54.

[5] David Bermbach, Ahmet-Serdar Karakaya, and Simon Buchholz. 2020. Using Application Knowledge to Reduce Cold Starts in FaaS Services. In *ACM Symposium on Applied Computing*. Czech Republic, 134–143. https://doi.org/10.1145/3341105.3373909

[6] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. 2020. SEUSS: Skip Redundant Paths to Make Serverless Fast. In *European Conference on Computer Systems (EuroSys '20)*. ACM, Heraklion, Greece, Article 32, 15 pages. https://doi.org/10.1145/3342195.3392698

[7] Robert Cordingly, Hanfei Yu, Varik Hoang, Zohreh Sadeghi, David Foster, David Perez, Rashad Hatchett, and Wes Lloyd. 2020. The Serverless Application Analytics Framework: Enabling Design Trade-off Evaluation for Serverless Software. In *International Workshop on Serverless Computing (WoSC'20)*. ACM, Delft, Netherlands, 67–72. https://doi.org/10.1145/3429880.3430103

[8] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating Cascading Cold Starts in Serverless Function Chain Deployments. In *Middleware*. ACM, Delft, Netherlands, 356–370. https://doi.org/10.1145/3423211.3425690

[9] Vojislav Dukic, Rodrigo Bruno, Ankit Singla, and Gustavo Alonso. 2020. Photons: Lambdas on a Diet. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*. ACM, 45–59. https://doi.org/10.1145/3419111.3421297

[10] Simon Eismann, Johannes Grohmann, Erwin van Eyk, Nikolas Herbst, and Samuel Kounev. 2020. Predicting the Costs of Serverless Workflows. In *Int. Conf. on Performance Engineering (ICPE)*. ACM, Canada, 265–276.

[11] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2021. Serverless Applications: Why, When, and How? *IEEE Software* 38, 1 (2021), 32–39. https://doi.org/10.1109/MS.2020.3023302

[12] Adam Eivy and Joe Weinman. 2017. Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing* 4, 2 (2017), 6–12. https://doi.org/10.1109/MCC.2017.32

[13] Pedro García-López, Aleksander Slominski, Simon Shillaker, Michael Behrendt, and Barnard Metzler. 2020. Serverless End Game: Disaggregation enabling Transparency. *arXiv preprint arXiv:2006.01251* (2020).

[14] Michael T Heath. 2018. *Scientific Computing: An Introductory Survey, Revised Second Edition*. SIAM.

[15] Sanghyun Hong, Abhinav Srivastava, William Shambrook, and Tudor Dumitras. 2018. Go Serverless: Securing Cloud via Serverless Design Patterns. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. USENIX Association, Boston, MA.

[16] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *Symposium on Cloud Computing*. 445–451.

[17] Wes Lloyd, Shruti Ramesh, Swetha Chinthalapati, Lan Ly, and Shrideep Pallickara. 2018. Serverless Computing: An Investigation of Factors Influencing Microservice Performance. In *IEEE Int. Conf. on Cloud Engineering (IC2E)*. 159–169. https://doi.org/10.1109/IC2E.2018.00039

[18] Wes Lloyd, Minh Vu, Baojia Zhang, Olaf David, and George Leavesley. 2018. Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads. In *IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 195–200.

[19] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. 2020. FaaSdom: A Benchmark Suite for Serverless Computing. In *ACM International Conference on Distributed and Event-Based Systems (DEBS '20)*. ACM, Montreal, Canada, 73–84. https://doi.org/10.1145/3401025.3401738

[20] Johannes Manner and Guido Wirtz. 2019. Impact of Application Load in Function as a Service. (2019).

[21] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. 2017. A Serverless Real-Time Data Analytics Platform for Edge Computing. *IEEE Internet Computing* 21, 4 (2017), 64–71. https://doi.org/10.1109/MIC.2017.2911430

[22] Edward Oakes, Leon Yang, Kevin Houck, Tyler Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. Pipsqueak: Lean Lambdas with Large Libraries. In *IEEE Int. Conf. on Distributed Computing Systems Workshops (ICDCSW)*. 395–400. https://doi.org/10.1109/ICDCSW.2017.32

[23] Sasko Ristov, Dragi Kimovski, and Thomas Fahringer. 2022. FaaScinating Resilience for Serverless Function Choreographies in Federated Clouds. *IEEE Transactions on Network and Service Management* (2022), 1–1. https://doi.org/10.1109/TNSM.2022.3162036

[24] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2021. AFCL: An Abstract Function Choreography Language for serverless workflow specification. *Fut. Gen. Comp. Syst.* 114 (2021), 368 – 382.

[25] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2021. xAFCL: Run Scalable Function Choreographies Across Multiple FaaS Systems. *IEEE Transactions on Services Computing* (2021), 1–1. https://doi.org/10.1109/TSC.2021.3128137

[26] Josep Sampe, Pedro Garcia-Lopez, Marc Sanchez-Artigas, Gil Vernik, Pol Roca-Llaberia, and Aitor Arjona. 2021. Toward Multicloud Access Transparency in Serverless Computing. *IEEE Soft.* 38, 1 (2021), 68–74. https://doi.org/10.1109/MS.2020.3029994

[27] J. Sampe, M. Sanchez-Artigas, G. Vernik, I. Yehekzel, and P. Garcia-Lopez. 2021. Outsourcing Data Processing Jobs with Lithops. *IEEE Transactions on Cloud Computing* (Nov. 2021), 1–1. https://doi.org/10.1109/TCC.2021.3129000

[28] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking Functions to Warm the Serverless Cold Start. In *Middleware*. ACM, Delft, Netherlands, 1–13. https://doi.org/10.1145/3423211.3425682

[29] Khondokar Solaiman and Muhammad Abdullah Adnan. 2020. WLEC: A Not So Cold Architecture to Mitigate Cold Start Problem in Serverless Computing. In *IEEE Int. Conference on Cloud Engineering (IC2E)*. 144–153. https://doi.org/10.1109/IC2E48712.2020.00022

[30] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021. Benchmarking, Analysis, and Optimization of Serverless Function Snapshots. In *ACM ASPLOS*. 559–572. https://doi.org/10.1145/3445814.3446714

[31] Erwin van Eyk, Alexandru Iosup, Simon Seif, and Markus Thömmes. 2017. The SPEC Cloud Group's Research Vision on FaaS and Serverless Architectures. In *International Workshop on Serverless Computing (WoSC '17)*. ACM, Las Vegas, Nevada, 1–4. https://doi.org/10.1145/3154847.3154848

[32] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the Curtains of Serverless Platforms. In *USENIX Annual Technical Conference*. Boston, MA, USA, 133–145.

[33] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing Serverless Platforms with Serverlessbench. In *ACM SoCC*. ACM, 30–44. https://doi.org/10.1145/3419111.3421280