A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures

Erwin van Eyk[1,3]
Alexandru Iosup[1,2]
[1]TU Delft and [2] VU Amsterdam, NLD
[3]Platform9 Inc., USA
E.vanEyk@atlarge-research.com

Cristina L. Abad[4]
[4]Escuela Superior Politecnica del Litoral, ESPOL, Ecuador
cabad@fiec.espol.edu.ec

Johannes Grohmann[5]
Simon Eismann[5]
[5]University of Wuerzburg, Germany
first_name.last_name@uni-wuerzburg.de

ABSTRACT

As a key part of the serverless computing paradigm, Function-as-a-Service (FaaS) platforms enable users to run arbitrary functions without being concerned about operational issues. However, there are several performance-related issues surrounding the state-of-the-art FaaS platforms that can deter widespread adoption of FaaS, including sizeable overheads, unreliable performance, and new forms of the cost-performance trade-off. In this work we, the SPEC RG Cloud Group, identify six performance-related challenges that arise specifically in this FaaS model, and present our roadmap to tackle these problems in the near future. This paper aims at motivating the community to solve these challenges together.

ACM Reference Format:
Erwin van Eyk[1,3], Alexandru Iosup[1,2], Cristina L. Abad[4], Johannes Grohmann[5], and Simon Eismann[5]. 2018. A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3185768.3186308

1 INTRODUCTION

Software-development paradigms are increasingly transitioning from monolithic applications to compositions of smaller services that are more granular and distributed (e.g., through Software Oriented Architectures and, recently, microservice architectures). Correspondingly, cloud vendors have started offering *serverless computing* services, hosting on granularly billed resources the emerging application services. The notion of Function-as-a-Service (FaaS) can be seen as a combination of both developments, which offers new benefits but also raises new performance challenges. In this vision paper we focus on the latter.

The context of serverless computing, including FaaS and cloud(-native) functions, does not yet have a community-wide terminology; we use in this work the following terms as defined in [22]. First, *serverless computing* is a form of cloud computing which allows users to run event-driven and granularly billed applications without having to manage the operational logic. Second, a *cloud function* is as a small, stateless, on-demand service with a single functional responsibility. Thus, conceptually a cloud function is a type of microservice with a single endpoint or responsibility. Lastly, *Function-as-a-Service* (FaaS) is a form of serverless computing that manages the resources, lifecycle, and event-driven execution of user-provided cloud functions. The FaaS model lets developers compose applications using cloud functions, and as a result, enabling easy and effective operational cloud control for the provider.

A common motivating example for serverless computing is the data- and/or compute-intensive process of resizing images, possibly arriving as a (video) stream. Common scenarios using this process include hosting visually rich websites, video surveillance and traffic shaping, identification of objects in warehousing and manufacturing, and matching image sizes to the needs of users (desktop vs. mobile presentation). The process in which a resizing function is invoked can be: an image source (e.g. user) uploads an image, triggering an event; this triggers the execution of the resizing cloud function; the function runs and stores the resulting image in the cloud. The serverless paradigm is a good fit for this use case for several reasons. First, the cloud function is stateless and well-scoped. Second, it runs on-demand, creating a bursty workload, which can make optimal use of the inherent elasticity of the FaaS platform. Finally, there are often no strict latency requirements, which makes the performance overhead (see Section 2.1) of FaaS acceptable.

Although the FaaS model opens new opportunities, it also introduces additional challenges. From the perspective of performance engineering, the increasingly granular and modular structures enable specifically tailored and fine-grained solutions. However, as we emphasize in this vision paper, any design in this domain must leverage a thorough understanding of the complexities of the FaaS architectures. Toward this end, our contribution is two-fold:

(1) We identify and examine six performance-related challenges (Section 2). For each challenge, we describe the underlying problem, examine why the challenge is relevant, and present related work and possible approaches towards resolving it.
(2) We define a roadmap for addressing these challenges in the SPEC RG Cloud Group (Section 3). Our roadmap has two main milestones for 2018: a (validated) FaaS reference architecture and a benchmark for FaaS platforms.

2 PERFORMANCE CHALLENGES

We identify six performance challenges for the serverless domain.

2.1 Reducing FaaS Overhead

Especially for performance-critical use cases, such as web and IoT applications, FaaS adoption depends in part on proving that the

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186308

loss of performance that could be incurred by splitting applications into fine-grained functions is negligible. For resource-intensive, coarse-grained functions, it is possible to reduce the overhead to useful levels [10], but the general case remains an open challenge.

We identify three main categories of performance overhead for FaaS platforms. The *provisioning overhead* is caused by deploying the cloud function on-demand, and is likely the dominant overhead. The FaaS platform needs to deploy the cloud function prior to its (first) use, and possibly to provision the underlying resources (e.g., a container or VM) prior to deployment. In an unoptimized context, provisioning can take seconds (containers) to minutes (VMs), and additional time to deploy the function (e.g., startup and network setup). The *request overhead* appears while handling requests: routing the request to a matching function, executing the function possibly without warm-up (e.g., cold JIT-compiled code), transferring data to and from the requester, and handling additional metadata such as logging and telemetry. The *function lifecycle management and scheduling overhead* is due to management and scheduling processes in the FaaS platform (see also Section 2.3).

Several FaaS projects aim to reduce the performance overhead through basic mechanisms. Fission[1] and OpenWhisk[2] attempt to avoid full (*cold*) function-deployments for each request by reusing deployed functions for subsequent requests (*hot starts*). Additionally, Fission has a pool of 'generic', always-running containers, into which it injects the function during provisioning. Another approach creates new resource abstractions for serverless functions, to avoid the OS overheads of VMs and containers [11].

Image-resizing example: The platform deploys a pool of image-resizing functions, to process the camera-feed at 30Hz. This leads to large overheads only when first executing the function.

2.2 Performance Isolation

Ensuring performance guarantees while fully utilizing the physical machines is important for FaaS operators. FaaS platforms commonly deploy (*consolidate*) multiple functions on the same physical machine, which improves resource utilization, and possibly reduces the provisioning and lifecycle overheads (see Section 2.1). As a trade-off, consolidation can make the performance of a function be influenced by other functions running on the same machine. This raises the challenge of achieving *performance isolation* [12], which is a key step in ensuring performance guarantees under high utilization.

Achieving performance isolation is an ongoing research topic. Even for state-of-the-art platforms using Linux containers, the *load-profile* of the application, be it CPU-, memory-, or IO-intensive, leads to different levels of performance isolation across containers [23]. Unfortunately, traditional and especially data-intensive applications can exhibit widely varying load-profiles.

With FaaS, the application is disaggregated into simple, single-task functions, each of which likely exhibiting a dominant load-profile. This gives new opportunities to schedule (see Section 2.3), but also requires additional research to understand the performance isolation of FaaS functions, for example, starting from the metrics proposed by Krebs et al. [12] for cloud environments. Once the influencing factors for performance isolation of FaaS functions are known, novel approaches, for example using machine learning, can be used to predict the performance isolation of different placements.

Image-resizing example: The image-resizing function keeps the image in memory during processing, which is memory-intensive unless the resizing logic is complex. A scheduling algorithm aware of performance isolation would avoid deploying the function on the same machine as other memory-intensive functions.

2.3 Scheduling Policies

On a FaaS platform, function executions are triggered in response to events. A *scheduling policy* determines where a specific function (*task*) should run. The resource (e.g., a container or VM) to which the function is mapped can have a significant impact on performance, depending on available resources, workflow deadlines, location of input data and code, load balancing or QoS requirements, co-located functions (see Section 2.2), etc. In general, function scheduling is a hard problem with multiple constraints, and possibly conflicting goals. Additionally, it needs to be solved online and thus avoid scheduling overheads (see Section 2.1).

Beyond meeting basic task requirements (e.g., ensuring the needed amount of memory), a smart scheduler can: (i) reduce the operational costs without compromising performance (see Section 2.5); (ii) reduce the provisioning overhead and in particular cold-starts (see Section 2.1); (iii) improve data locality in the multi-function workflow corresponding to a complex application; (iv) decide which resources to use, for example, between running a function in a cloud datacenter or at the *edge* [20]; (v) make near-optimal decisions despite using only partial information.

Smart scheduling, primarily *operator-level* and automating user-level decisions, is a largely unexplored area for FaaS. Beyond simple load-balancing by FaaS schedulers, existing approaches include limited production-ready capabilities, e.g., container-reuse in Fission, orchestration of complex workflows in Fission. Although some recent work provides useful new capabilities, such as caching of Python packages to reduce lifecycle overhead [17], and identifying the sources of performance-variability [13], FaaS schedulers cannot yet leverage these capabilities and their inter-play needs further study. The area of scheduling across datacenter and edge resources is also understudied [16]. We envision the community can leverage the large body of work of scheduling for Web services and for net-/work-flows, and elastic scheduling in Big Data frameworks (e.g., [5, 14, 18, 24]).

Image-resizing example: Depending on the use case, for example for video surveillance, a delay of up to a second can be tolerated. A technique like "delay scheduling" [25] could improve data or code locality, and reduce network pressure, at the cost of a small delay.

2.4 Performance Prediction

Predicting the performance of individual cloud functions is another important building block in the management processes of both users and providers of FaaS platforms. Platform providers could use predictions to preemptively adjust the allocated resources to the incoming load. Users could use predictions to re-flow their applications, without potentially time- and cost-intensive testing of each configuration.

For traditional software systems, performance models enable accurate performance predictions [19]. Applying these approaches

[1] https://github.com/fission/fission
[2] https://openwhisk.apache.org/

to FaaS comes with new challenges, including the information gap and the function-specific performance. The information gap means the FaaS user is unaware of the hardware resources the functions are executed on, while the FaaS provider has no information about the implementation details of the (*black-box*) function. The performance modeling needs of FaaS are function-specific. As for traditional applications, the input (size, structure, and content) of a function influences its performance, but for FaaS there are significantly more functions to model and predict. This raises new challenges of finding relevant models at the scale and with the input-dependency required by FaaS applications, and of propagating the calibration data of these models throughout a complex application-level performance model, to achieve accurate performance predictions.

The information gap is an active research topic in cloud computing, and IaaS approaches could be adapted for FaaS platforms; for example, soft reservations incentivize users to share performance predictions with the providers [15]. Similarly, techniques developed for general software systems could be leveraged by FaaS, after careful research; for example, classic and modern performance modeling in software systems [19]. It is unclear how techniques developed for *white-box* components or for automated dependency detection (e.g., as proposed in [3]) could be used in the FaaS context.

Image-resizing example: The resource demand of the function is correlated with the size of the input image. Modeling this correlation would allow predicting the performance of the image-resizing function for different image sizes.

2.5 Engineering for Cost-Performance

FaaS customers have a strong incentive to optimize *simultaneously* performance and operational expenditure (OPEX costs) [4]. The granular billing characteristic of serverless deployments, where costs are directly related to business-critical functionality, leads to fine-grained decisions. The pricing model for functions seems currently calibrated for a moderate number of requests per second. As soon as this metric exceeds the moderate threshold, on-demand VMs or containers become more cost-effective than FaaS and the customer should stop FaaS deployment. With more complex pricing models, the most cost-effective alternative may vary during peak and low usage hours, and the decision becomes complex.

Other than educating users about the how the differences in pricing schemes may affect their total operational costs [4], we are not aware of any research on how these decisions can be facilitated or even automated for the user. Similarly to section 2.4, we envision that the first steps could leverage techniques proposed for other types of cloud settings. These include combining on-demand, and spot or reserved VMs, to minimize costs while providing certain guarantees, like always-on [6] or highly available [21] services, and meeting job deadlines [8].

Image-resizing example: A hybrid cloud platform or broker could (dynamically) switch between different cloud providers to schedule the cloud function on the cloud with the best cost-performance balance at that moment in time.

2.6 Evaluating and Comparing FaaS Platforms

The field of serverless computing currently lacks a systematic, objective benchmark of FaaS platforms. A comprehensive benchmark, such as the recently published SPEC Cloud IaaS benchmark [1], would help users compare the various cloud providers in terms of performance, costs, reliability and other aspects. This lack of a systematic benchmark is most pressing, already holding back the adoption of the serverless paradigm for production and for performance-critical use cases [2]. Although several industry attempts to benchmark FaaS providers exist [2], they lack systematic approaches and academic rigor.

We identify three main causes for the lack of an adequate serverless benchmark. First, FaaS platforms are inherently complex systems. Compared to the IaaS and PaaS cloud models, FaaS includes more operational logic, such as autoscaling mechanisms, resource management, and function lifecycle management. Additionally, there are difficult to quantify aspects, such as differences in 'development speed' of applications on the different FaaS platforms. Even more components have to collaborate to execute a complex workflow. These layers and dependencies introduce interfering variables to the benchmark. Toward a systematic benchmark, we need to identify the common components, and the potentially interfering factors that influence the performance of a FaaS platform [9]. A reference architecture would help with these aspects, and is a work-in-progress described in our proposed Roadmap (see Section 3).

Second, due to the immaturity of the paradigm, there is a lack of representative workloads traces. The authors of OpenLambda [7] used the calls made to the web server of the Gmail web application, but this remains far from a representative workload for evaluating FaaS platforms; in contrast, the *image-resizing example* is much more data- and compute-intensive.

Last, many of the prevailing FaaS platforms are proprietary, closed-source implementations. The lack of introspection into these implementations makes it challenging to identify both the relevant and interfering factors. Additionally, the only way of evaluating these FaaS platforms is online, in a public cloud environment, which adds the complexity of getting consistent results from these volatile environments and an element of cost. How to reliably evaluate components in a closed-source, online cloud environment is an active field of research, whose emerging best-practices could be used in future serverless benchmarks.

3 ROADMAP

The SPEC RG Cloud Group focuses on a broad understanding of performance in cloud settings. For FaaS platforms, our roadmap focuses primarily on the challenge of evaluating FaaS platforms (see Section 2.6). The two main milestones towards this goal for 2018 are a validated reference architecture for FaaS platforms, followed by a systematic benchmark for FaaS platforms.

3.1 Reference Architecture

Establishing a comprehensive reference architecture for FaaS platforms will provide the community with a valuable tool: consistent terminology, a set of typical components, low-level details. This facilitates approaching the challenges identified in Section 2.

Based on the analysis of the major open-source and of a handful of closed-source FaaS platforms so far, we have created the preliminary reference architecture depicted in Figure 1. As suggested by [22], this reference architecture is based on the notions of *business logic* derived from the use-case of the cloud user, instead of

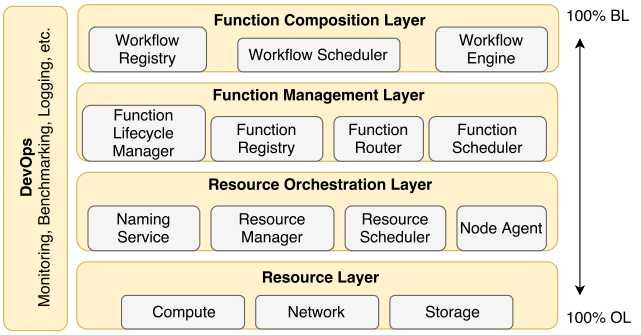

Figure 1: FaaS Reference Architecture (Work-in-Progress) ordered from business logic (BL) to operational logic (OL).

operational logic that focuses on the QoS of the application (e.g., autoscaling and resource management). The *Resource Layer* represents the available resources within a cloud, possibly virtualized as VMs or containers. These resources are managed by the *Resource Orchestration Layer*, which is often implemented by IaaS orchestration services (i.e., Kubernetes). On top of these orchestrated resources, the cloud function abstraction is managed by the *Function Management Layer*. It includes components to fetch the needed cloud functions from repositories (*function registry*), to ensure cloud functions are deployed on selected resources (*function scheduler*), to map incoming requests to deployed functions (*function router*), and to ensure that functions are elastically scaled and cleaned up when needed (*function lifecycle manager*). Similar to how the resources at the *resource layer* are orchestrated by a higher-level layer, cloud functions are treated as lower-level service-resources by the *function composition layer*, and composed into more complex workflows. Akin to traditional architectures of workflow management systems, this layer consists of a store of workflows *(workflow registry)*, a component to track and manage active workflow invocations (*workflow engine*), and a scheduler to decide on the execution order of the workflow invocation (*workflow scheduler*). Finally, there are cross-cutting (DevOps) concerns at each layer, e.g., monitoring.

To validate this reference architecture, we have already matched its components with real-world FaaS platforms such as OpenWhisk and Fission, and envision similar results for other FaaS platforms.

3.2 Systematic Benchmark

Using the reference architecture as a stepping stone, we aim to develop a systematic, industry-wide benchmark of both closed-source vendor-based and open-source FaaS platforms. This benchmark allows users, especially those with large business-critical, production workloads, to understand the impact of workload characteristics, and the strengths and weaknesses of the different FaaS offerings. Benchmarking open-source FaaS platforms would also allow researchers to investigate what approaches in implementations perform well, opening new design and tuning opportunities.

As a next step, we envision expanding the set of test-workloads used in the benchmark, beyond the canonical image-resizing example. Here, we specifically envision a collection of at least five use cases, each representing the differing workload characteristics of relevant domains, such as IoT, stream/data processing, latency-critical applications, and scientific computing. This collection of use cases, combined with the collection of real-world workload traces, will provide the basis for developing a systematic benchmark.

4 CONCLUSION

The emerging FaaS model holds good promise for future cloud applications, but raises new performance-related challenges that can hamper its adoption. In this work, we identify six performance-related challenges for the serverless domain and plot a roadmap for alleviating these challenges. Our roadmap is two-pronged: a reference architecture to stimulate common understanding, and to develop a serverless benchmark guided by FaaS use-cases.

Last[3], through this vision paper we launch a call to action: join an expanded SPEC RG Cloud Group in tackling these challenges.

REFERENCES

[1] S. Bast, M. Silva, and N. Wakou. 2017. SPEC Cloud IaaS 2016 Benchmark. In *ACM/SPEC ICPE*.
[2] M. Billock. 2017 (accessed January 6, 2018). Serverless Performance Shootout. http://blog.backand.com/serverless-shootout/. (2017 (accessed January 6, 2018)).
[3] F. Brosig, N. Huber, and S. Kounev. 2011. Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems. In *IEEE/ACM ASE*.
[4] A. Eivy. 2017. Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing* 4, 2 (2017).
[5] B. Ghit, N. Yigitbasi, A. Iosup, and D. Epema. 2014. Balanced resource allocations across multiple dynamic MapReduce clusters. In *ACM SIGMETRICS*.
[6] X. He, P. Shenoy, R. Sitaraman, and D. Irwin. 2015. Cutting the cost of hosting online services using cloud spot markets. In *HPDC*.
[7] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. 2016. Serverless Computation with OpenLambda. In *USENIX HotCloud*.
[8] H. Huang, L. Wang, B. Tak, L. Wang, and C. Tang. 2013. CAP3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances. In *IEEE CLOUD*.
[9] R. Jain. 1990. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.
[10] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *ACM SoCC*.
[11] R. Koller and D. Williams. 2017. Will Serverless End the Dominance of Linux in the Cloud?. In *HotOS 2017*.
[12] R. Krebs, C. Momm, and S. Kounev. 2014. Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. *Elsevier SciCo* (2014).
[13] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. 2018. Serverless Computing: An Investigation of Factors Influencing Microservice Performance. In *IEEE IC2E, to appear*.
[14] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, and A. Greenberg. 2011. Join-Idle-Queue: A Novel Load Balancing Algorithm for Dynamically Scalable Web Services. *Perf. Eval.* 68, 11 (2011).
[15] S. Mohammadi, S. Kounev, Juan-Verdejo, and B. Surajbali. 2013. Soft Reservations: Uncertainty-Aware Resource Reservations in IaaS Environments. In *BMSD*.
[16] S. Mortazavi, M. Salehe, C. Simoes, C. Phillips, and E. de Lara. 2017. CloudPath: A Multi-Tier Cloud Computing Framework. In *ACM/IEEE Symp. Edge Comp. (SEC)*.
[17] E. Oakes, L. Yang, K. Houck, T. Harter, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. 2017. Pipsqueak: Lean Lambdas with Large Libraries. In *ICDCS-W (WoSC)*.
[18] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. 1998. Locality-aware Request Distribution in Cluster-based Network Servers. *SIGOPS Oper. Syst. Rev.* 32, 5 (1998).
[19] R. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann. 2016. *Modeling and simulating software architectures: The Palladio approach*. MIT Press.
[20] M. Satya. 2017. The emergence of edge computing. *Computer* 50, 1 (2017).
[21] S. Shen, K. Deng, A. Iosup, and D. Epema. 2013. Scheduling Jobs in the Cloud Using On-Demand and Reserved Instances. In *Euro-Par*.
[22] E. van Eyk, A. Iosup, S. Seif, and M. Thömmes. 2017. The SPEC cloud group's research vision on FaaS and serverless architectures. In *WoSC held with ACM/IFIP/Usenix Middleware*.
[23] M. Xavier, I. De Oliveira, F. Rossi, R. Dos Passos, K. Matteussi, and C. De Rose. 2015. A performance isolation analysis of disk-intensive workloads on container-based clouds. In *IEEE Euromicro PDP*.
[24] Q. Xie, M. Pundir, Y. Lu, C. Abad, and R. Campbell. 2017. Pandas: Robust Locality-Aware Scheduling With Stochastic Delay Optimality. *IEEE/ACM TON* 25:2 (2017).
[25] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. 2010. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *ACM EuroSys*.

[3]This work is also supported by the Dutch projects Vidi MagnaData and Commit.