Serverless Clusters: The Missing Piece for Interactive Batch Applications?

[Position Paper]

Ingo Müller[1] Rodrigo Bruno[1] Ana Klimovic[2] John Wilkes[2] Eric Sedlar[3] Gustavo Alonso[1]

[1]{ingo.mueller,rodrigo.bruno,alonso}@inf.ethz.ch
Systems Group, Department of Computer Science, ETH Zurich

[2]{anakli,johnwilkes}@google.com
Google Inc., Mountain View, CA

[3]eric.sedlar@oracle.com
Oracle Labs

ABSTRACT

The massive, instantaneous parallelism of serverless functions has created a lot of excitement for interactive batch applications. We argue that *functions* are in fact the wrong abstraction for this use case. We call instead for another type of infrastructure, "serverless clusters," and discuss what is missing to make them a reality.

1 INTRODUCTION

Parallelization has emerged as one of the main techniques to increase performance after the end of CPU frequency scaling. Thanks to the elasticity of the cloud, massive degrees of parallelism are available to and affordable by essentially anybody. Recently, *serverless functions* (or Function-as-a-Service), have pushed elasticity into previously unchartered territory: These services allow to get an extremely high degree of parallelism (in the order of several thousand) within a very short amount of time (typically a few seconds) and are billed at sub-second granularity. Furthermore, functions are typically programmed in a high-level language such as JavaScript, Python, or Java and do not require any administration or maintenance from the user. Serverless functions thus have the potential to make massive parallelization available to the masses.

Due to this potential, an increasing number of workloads has been proposed for taking advantage of serverless functions. In particular, functions have been used for batch applications, which run in short but intense *bursts* of related invocations. Examples include distributed make [5, 13], sorting [11, 13, 17, 19], video encoding [4, 5, 6], image and video classification [4, 5, 13], unit tests [5], as well as MapReduce-style [11, 12, 15, 17, 19] and SQL-style analytics [15, 16, 17]. The massive parallelism offered by functions often reduces the end-to-end running time from hours to a few minutes or a few dozen seconds—making them interactive where they previously were not. Furthermore, functions "scale to zero"—they have no economic or operational overhead even for very sporadic use.

2 FUNCTIONS ARE NOT DESIGNED FOR JOBS

However, we claim that functions are actually not a good fit for the applications mentioned above[1] as they were designed for a fundamentally different use case. The problem functions solve is *request serving*—a long-running service that processes a stream of independent requests, whose volume may change rapidly and unpredictably and thus needs fast auto-scaling of resources. For example, AWS Lambda is advertised [2] "to execute code in response to triggers" and "to handle web, mobile, [...] and [...] API requests." In contrast, the applications mentioned above run massively parallel *jobs*—a set of dependent tasks that run to completion and are best solved by closely-coupled workers running at the same time. For this set-up, the serverless functions architecture lacks a number of fundamental features: Different function invocations have no means of direct communication; there is no API for batch invocation; there is no mechanism to know which functions are currently running; and there is not even a guarantee how many *concurrent* invocations there will be at any given point in time. We argue that the serverless functions architecture does not provide these features *by design* as they are not required to serve independent requests.

Previous work on using serverless functions for batch applications thus mainly consists of working around the limitations of serverless functions (including our own [15]). For example, a large number of authors have proposed methods that enable different function invocations to communicate through some external service [11, 12, 13, 15, 16, 17, 19]. The solutions either consist of additional system components (which severely limits elasticity and thus defeats the purpose of functions) or sophisticated communication mechanisms that are purpose-built for serverless functions, potentially only for those of a particular cloud provider. Similarly, we suspect that many of the proposed systems *assume* that serverless functions are executed concurrently and that their results or side-effects are available to other concurrent invocations immediately even though cloud providers do not *guarantee* that. These systems could thus stop working from one day to the next if cloud providers change the way they schedule invocations. While these projects make a good case for interactive applications that need extremely elastic cloud infrastructure, we argue that they are actually fighting the technology they run on and are thus not likely to find broad and sustained adoption.

3 VMS ARE NOT SERVERLESS

In contrast, virtual machines (VM) or containers do not suffer from the shortcomings of serverless functions mentioned above. They represent the de-facto standard for packaging applications in a way that they can be run on any physical infrastructure. A large number of VMs can be easily batch-deployed, through orchestration tools like Kubernetes or provider-specific APIs, and each of them can be addressed directly through the standard network stack. In other words, VMs are more versatile and mature for providing massive parallel resources than functions.

[1] We are not the only ones to be skeptical [9].

SPMA '20, April 27, 2020, Heraklion, Greece

https://doi.org/10.3929/ethz-b-000405616

However, serverless functions have two huge advantages compared to the current VM and container stacks, which explains the excitement around them in spite of their short-comings: i) extremely low startup time, and ii) lower operational complexity. Amazon's and Google's median invocation latency of "cold" functions has been measured to be in the order of 200 ms [20]. In contrast, the deployment times of VMs in the cloud currently range around 20 s–60 s [1] and, interestingly, startup times for managed general-purpose containers are not significantly lower [8]. In order to amortize this startup effort, the cloud providers also have a much higher minimum billing time for VMs or containers than for functions: for example, 100 ms in AWS Lambda vs. 1 min in both AWS EC2 (VMs) and AWS Fargate (containers). What VMs and containers thus lack is *burstability*—low-latency deployment at large scale with correspondingly low billing granularity.

Containers and VMs might also expose more system complexity to the user. While functions essentially only consist of high-level code with few configuration knobs, the user has to provide container or VM images with up-to-date and correctly configured software and set up the network among the different components of the application. Furthermore, she might need to choose the number and type of VM instances, which may have a significant impact on the performance and price of her application. Finally, in order to avoid the long delay of per-job start-ups, the user might instead keep a cluster of VMs running, which increases operational complexity further and leads to potentially high under-utilization and thus overall higher cost. In particular, for sporadic usage, this complexity has the risk to be too high to be worth doing.

4 SERVERLESS CLUSTERS

To fill the gap in current cloud computing offerings, we call for a new type of cloud infrastructure, "serverless clusters," which combines features from VMs or containers and serverless functions. With this type of service, users submit descriptions of *jobs*, which define a set of workers along with their type, size, number, configuration, and addressable name, and the maximum running time. The service sets up the workers, including the allocation of the necessary resources, the starting of the workers' runtimes, and the set-up of the network between them. The set-up process takes a few seconds at most—this is, at the same time the biggest challenge and, as we argue below, within reach. When the job has completed or reached the given timeout, the service stops the resources and charges the user with second-granularity or finer, ideally from the first second.

We think that serverless clusters perfectly fulfill the needs of interactive batch applications. They offer the benefits of serverless functions, but without their limitations for batch applications. In a way, they turn the cloud into an infinitely-sized supercomputer with interactive scheduling latency accessible by anyone. In the talk, we will illustrate the potential gains of serverless clusters based on the extensive measurements and cost estimates from our previous work [15].

Note that, while some existing services come close to the idea, cloud providers do not offer serverless clusters just yet. For example, Google's *Cloud Dataproc* allows to run Hadoop and Spark jobs on ephemeral, "job-scoped" clusters, but with an advertised cluster start-up time of 90 s [7], which is two orders of magnitude higher than that of functions. Google's *Batch* and Amazon's *AWS Batch* run jobs on long-running, user-specific clusters, which are likely to be heavily under-utilized in interactive scenarios. *AWS Step Functions* allow to invoke serverless functions in batches, which has low start-up time and usage-based billing, but still suffers from the lack of direct communication as discussed above. Finally, *Google Cloud Run* offers serverless containers, but since they are designed for request serving like functions, they also lack the same features (batch-invocation, addressability, etc).

5 OPEN RESEARCH QUESTIONS

While we argue for the attractiveness of serverless clusters, we think that they are not yet feasible due to a number of open questions and challenges, including the following:

Job programming abstraction. The serverless functions abstraction provides a clean and simple interface to process a stream of independent events. A similar abstraction should be provided for jobs that allows workers to communicate—whether they are deployed as high-level code (similar to Amazon Lambda) or as containers (similar to Google Cloud Run). A related question is what the job submission interface should look like: Are existing solutions such as Kubernetes, YARN, or Slurm suitable or do we need something new?

Interactive cluster start-up time. Datacenter infrastructure is currently not optimized for deploying a large number of ephemeral resources in a short amount of time. The problem does not seem to be the start-up times of the traditional VM and container stacks itself: optimized hypervisors are able to start VMs in the order of 100 ms [3], Linux can be made to boot in 1 s–2 s [18], and specialized kernels in as little as 6 ms [14]. However, orchestrated deployments (using Kubernetes, for example) can easily take up to several dozens of seconds to deploy a set of VMs or containers. There is hence a need to revisit cluster management in light of the requirement of serverless clusters. One possible avenue for research is to use specialized hardware for the control plane, which could help speed up this process.

Infrastructure overhead. Guaranteeing interactive start-up time increases infrastructure overhead significantly. For serverless functions, the cloud providers need to maintain a significant amount of infrastructure that handles a high rate of function invocations within tight latency bounds. This includes keeping around idle resources to handle load spikes and caching function environments to speed up future invocations. This is in contrast with the short execution time of functions, which increases the relative overhead of this additional infrastructure. Serverless clusters are likely to be even more demanding as they require the short-term availability of a large number of resources *concurrently*.

Accelerators for interactive jobs. Once general-purpose VMs can be rented for extremely short bursts, it seems feasible to extend the serverless cluster model to specialized instance types with hardware accelerators such as GPUs and FPGAs in order to increase compute efficiency. However, these devices often have little to no support for virtualization or multi-tenancy. Our group has several ongoing projects [10], but more research is required in this direction.

REFERENCES

[1] Samiha Islam Abrita, Moumita Sarker, Faheem Abrar, and Muhammad Abdullah Adnan. "Benchmarking VM Startup Time in the Cloud." In: *Bench*. 2019. DOI: 10.1007/978-3-030-32813-9_6.

[2] Amazon Web Services. *AWS Lambda – Serverless Compute*. URL: https://aws.amazon.com/lambda/ (visited on 02/03/2020).

[3] Inc Amazon Web Services. *Firecracker – Secure and fast microVMs for serverless computing*. 2019. URL: https://firecracker-microvm.github.io/ (visited on 02/13/2020).

[4] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. "Sprocket: A Serverless Video Processing Framework." In: *SoCC*. 2018. DOI: 10.1145/3267809.3267815.

[5] Sadjad Fouladi, Shuvo Chatterjee, and Francisco Romero. "From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers." In: *USENIX ATC*. 2019.

[6] Sadjad Fouladi et al. "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads." In: *NSDI*. 2017.

[7] Google. *Dataproc – Cloud-native Apache Hadoop & Apache Spark*. 2020. URL: https://cloud.google.com/dataproc (visited on 02/13/2020).

[8] Tyler Harter, Brandon Salmon, Rose Liu, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. "Slacker: Fast Distribution with Lazy Docker Containers Slacker: Fast Distribution with Lazy Docker Containers." In: *FAST*. 2016.

[9] Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. "Serverless Computing: One Step Forward, Two Steps Back." In: *CIDR*. 2019.

[10] Zsolt Istvan, Gustavo Alonso, and Ankit Singla. "Providing Multi-tenant Services with FPGAs: Case Study on a Key-Value Store." In: *FPL*. 2018. DOI: 10.1109/FPL.2018.00029.

[11] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. "Occupy the Cloud: Distributed Computing for the 99%." In: *SoCC*. 2017. DOI: 10.1145/3127479.3128601.

[12] Youngbin Kim and Jimmy Lin. "Serverless Data Analytics with Flint." In: *CLOUD*. 2018. DOI: 10.1109/CLOUD.2018.00063.

[13] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. "Pocket: Elastic Ephemeral Storage for Serverless Analytics." In: *OSDI*. 2018.

[14] Waldek Kozaczuk. *Making OSv Run on Firecracker - OSv Blog*. URL: http://blog.osv.io/blog/2019/04/19/making-OSv-run-on-firecraker/ (visited on 02/12/2020).

[15] Ingo Müller, Renato Marroquín, and Gustavo Alonso. "Lambada: Interactive Data Analytics on Cold Data using Serverless Cloud Infrastructure." In: *SIGMOD*. 2020. DOI: 10.1145/3318464.3389758.

[16] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. "Starling: A Scalable Query Engine on Cloud Function Services." In: (2019). arXiv: 1911.11727.

[17] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. "Shuffling , Fast and Slow: Scalable Analytics on Serverless Infrastructure." In: *NSDI*. 2019.

[18] Kaveh Razavi, Gerrit Van Der Kolk, and Thilo Kielmann. "Prebaked μvMs: Scalable, Instant VM Startup for IaaS Clouds." In: *ICDCS*. 2015. DOI: 10.1109/ICDCS.2015.33.

[19] Josep Sampé, Gil Vernik, Marc Sánchez-Artigas, and Pedro García-López. "Serverless data analytics in the IBM cloud." In: *Middleware Industry*. 2018. DOI: 10.1145/3284028.3284029.

[20] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. "Peeking Behind the Curtains of Serverless Platforms." In: *USENIX ATC*. 2018.