Characterizing Bottlenecks in Scheduling Microservices on Serverless Platforms

Jashwant Raj Gunasekaran*, Prashanth Thinakaran*, Nachiappan Chidamabaram Nachiappan*, Ram Srivatsa Kannan†, Mahmut Taylan Kandemir*, Chita R. Das*

*The Pennsylvania State University †University of Michigan, Ann Arbor
{jashwant, prashanth, kandemir, das}@cse.psu.edu, nachi@alumni.psu.edu, ramsri@umich.edu

***Abstract*—Datacenters are witnessing an increasing trend in adopting microservice-based architecture for application design, which consists of a combination of different microservices. Typically these applications are short-lived and are administered with strict Service Level Objective (SLO) requirements. Traditional virtual machine (VM) based provisioning for such applications not only suffers from long latency when provisioning resources (as VMs tend to take a few minutes to start up), but also places an additional overhead of server management and provisioning on the users. This led to the adoption of *serverless functions*, where applications are composed as functions and hosted in containers. However, state-of-the-art schedulers employed in serverless platforms tend to look at microservice-based applications similar to conventional monolithic black-box applications. To detect all the inefficiencies, we characterize the end-to-end life cycle of these microservice-based applications in this work. Our findings show that the applications suffer from poor scheduling of microservices due to reactive container provisioning during workload fluctuations, thereby resulting in either in SLO violations or colossal container over-provisioning, in turn leading to poor resource utilization. We also find that there is an ample amount of slack available at each stage of application execution, which can potentially be leveraged to improve the overall application performance.**

***Keywords*-serverless-functions; resource-management; containers; scheduling;**

I. Introduction

The advent of public clouds in the last decade has fueled the proliferation of microservice-based applications. These applications are usually comprised of multiple microservices[1] (called a microservice-chain) [1]. This trend is becoming increasingly pervasive with large cloud providers, such as Amazon [2], Facebook [3], and Netflix [4], having already adopted the microservices application model [5]. Further, the popularity of deep learning (DNN) based models and their multi-faceted application to different domains has lured application developers to train and host DNN models [6] as microservice-based applications. Figure 1 shows the overall framework of how a micro-service chain is deployed in the public cloud. Typically, these applications demand a strict SLO with tight response latencies, which is usually under `1000 ms` [7]. Hence, mitigating end-to-end latency of a microservice-chain is quintessential to provide a satisfactory user experience especially for interactive applications.

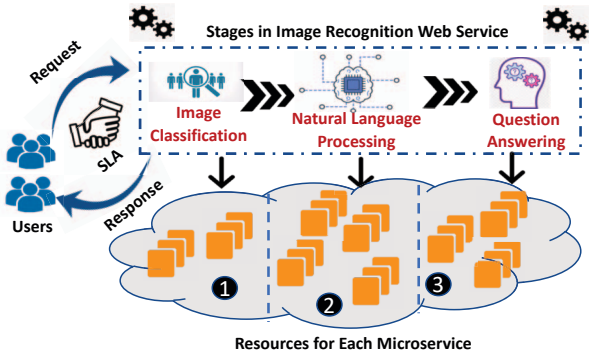

Figure 1: An overview of the framework used to host Microservice Chains in public cloud.

The SLOs for such web-services are typically bound by two factors (i) the resource provisioning latency, and (ii) the actual application execution time. The majority of these applications usually execute within a few milliseconds [6], [8], however, the provisioning times vary from a few seconds to minutes depending on the type of resources being procured. As opposed to using virtual machines (VMs), *serverless functions* such as AWS Lambda [9], Azure functions [10], IBM OpenWhisk [11] etc., have been adopted by application developers to mitigate provisioning latencies, as well as to abstract away the need for the users to provision or manage resources. Serverless computing provides several advantages such as (i) independent function hosting on containers, (ii) simplified programming, and (iii) automatic instance provisioning and elastic scaling for functions. However, they introduce a new set of challenges in terms of scheduling and resource management for the providers [12]. This opens up new research avenues especially when *deploying microservice-chains in serverless platforms* with strict SLO requirements along with optimal resource utilization.

Existing schedulers always spawn a new container for every incoming request if existing containers are busy [13]. This inherently leads to container over-provisioning while handling a large number of requests. Furthermore, they are unaware of the asymmetry between overall application execution time [14] and user-response latency (a.k.a *slack*), thus

[1]A microservice is the smallest granularity of an application performing an independent function.

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00195

leading to sub-optimal request placements on containers. This, in turn, leads to SLO violations, despite having over-provisioned containers. These schedulers are also unaware of the execution times of every microservice in an application. This leads to poor container allocation per microservice, thereby increasing the SLO violations. Additionally, when there are large dynamic variations [15] in request arrival rate, these schedulers fail to adapt rapidly as they cannot predict the request rate.

II. Characterization

To better understand the above-mentioned challenges, we characterize the end-to-end life cycle of microservice-based applications and briefly summarize our findings below.

- *Effects of Cold Starts:* We quantify the implications of cold-starts by executing a plethora of image recognition web-services (which uses ML-inference functions) on AWS Lambda. Our results show that cold-starts for different services leads to a large disparity between the container provisioning time when compared to application execution time. For instance, turn-around time of certain inference jobs were `2000 ms` to `7000 ms`, while the actual inference execution time was only `300 ms`. If SLO for the application is set at `1000 ms`, this leads to SLO violations due to uncertain provisioning delays whenever there are cold starts. Based on our observation, the existing schedulers make poor request placement decisions causing SLO violations and are also agnostic of inter-stage slack. Intuitively, by taking advantage of available slack at each stage, one can smartly queue requests at currently-busy containers to ensure optimal container provisioning, while minimizing the overall SLO violations due to cold starts.
- *Exploiting Stage-wise Application Slack:* Apart from different execution times and SLO requirements, applications also have unequal execution times for every individual stage. We profiled the runtime of well-known microservice chains from *Djinn&Tonic* [8] benchmark suite and observed that each stage within the application has widely varying execution times and more than 50% of the execution time is dominated by just one stage in the chain. Uniformly scaling containers for every stage is not ideal, as the longest-running stage would bottleneck the subsequent stages. In such scenarios, it becomes essential to have stage-wise container allocation policies based on each stage's execution times.
- *Effects of Container Scaleout:* Though request queuing helps in reducing the SLO violations when there is plenty of slack available, it still cannot help in hiding the cold-start latencies encountered when starting up new containers. Spawning new containers are inevitable especially when there is dynamism in request rate, such as during a request surge. To address these scaleout problem, we deduce that proactive container provisioning policy can mitigate SLO violations by spawning containers in advance. While cold-start latencies can be reduced by OS-level optimizations, the only way to hide them entirely is by proactive spawning of containers

III. Conclusions and Future Work

In this work, we identify the bottlenecks of existing schedulers employed by serverless platform providers especially when hosting microservice chains. We discuss our preliminary findings of (i) SLO violations due to cold starts, (ii) need for exploiting inter-stage application slack, and (iii) need for stage-wise container allocation policies. Our future work includes building a comprehensive system that leverages on our preliminary observations by building a system that eliminates the above-mentioned inefficiencies. We plan to evaluate our system using both real-system and large scale simulations of real-world traces and production serverless workloads.

Acknowledgments

This research was partially supported by NSF grants #1931531, #1629129, #1763681, #1629915, #1908793, #1526750 and we thank NSF Chameleon Cloud project CH-819640 for their generous compute grant.

References

[1] H. Yang, Q. Chen, M. Riaz, Z. Luan, L. Tang, and J. Mars, "Powerchief: Intelligent power allocation for multi-stage applications to improve responsiveness on power constrained cmp," in *Computer Architecture News*, 2017.
[2] A. E. C. Cloud, "Amazon web services," *Retrieved November*, 2011.
[3] A. Sriraman, A. Dhanotia, and T. F. Wenisch, "Softsku: Optimizing server architectures for microservice diversity@ scale," in *ISCA*, 2019.
[4] T. Mauro, "Adopting microservices at netflix: Lessons for architectural design," *https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices*, 2015.
[5] M. Villamizar, O. Garces, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in *CCGrid*, 2016.
[6] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *ASPLOS*, 2015.
[7] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services with Resource Efficiency," in *USENIX Middleware Conference*, 2017.
[8] J. Hauswald, Y. Kang *et al.*, "DjiNN&Tonic: DNN as a service and its implications for future warehouse scale computers," in *ISCA*, 2015.
[9] "AWS Lambda." https://aws.amazon.com/lambda/.
[10] "Microsoft Azure Serverless Functions," https://azure.microsoft.com/en-us/services/functions/.
[11] "IBM Serverless Functions," https://www.ibm.com/cloud/functions.
[12] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, "Centralized Core-granular Scheduling for Serverless Functions," in *SoCC*, 2019.
[13] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *ATC*, 2018.
[14] R. S. Kannan, L. Subramanian *et al.*, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *EuroSys*, 2019.
[15] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *ATC*, 2019.