서버리스 컴퓨팅: 문제점과 해결책

고지원°, 엄태건, 전병곤
{alapha23, taegeonum}@gmail.com, bgchun@snu.ac.kr

Serverless Computing: Pitfalls and Solutions

Zhiyuan Gao, Taegeon Um, Byung-Gon Chun
Department of Computer Science and Engineering, Seoul National University

요 약

Serverless computing is becoming more prevalent in terms of its high elasticity and fine-grained resource billing. Serverless research topics include intermediate data storage optimizations (e.g., Locus, Pocket), programming paradigm (e.g., Pyren) and exploration into its auto-scaling potential (e.g., Sprocket, ExCamera). This paper aims to provide an overview about serverless computing — what are the benefits, what are the limitations. In addition to the pitfalls, we finally mention what has been mitigated about the limitations.

1 Introduction

Serverless computing is a cloud computing execution model that frees developers from server management and machine resources allocation. With the help of a serverless programming framework, applications are defined as a set of functions (e.g., Lambda handlers) with access to a common data store. At each customer request, one serverless worker is dispatched to execute a proprietary task. Every task has a timeout while resource billing is based on execution duration and memory usage per request. Programming models are designed by different vendors such as AWS Lambda Function, Azure Functions and IBM Cloud Functions. Usually, serverless applications are stateless, but regarding stateful applications, customers can store intermediate data on cloud storages.

Compared with virtual machines (VMs), serverless computing simplifies application development in two ways. First, serverless computing makes it easier to achieve auto-provisioning and auto-scaling. With VMs, auto-provisioning and auto-scaling relies on developers to implement but they are automatically achieved due to the benefit of serverless architecture. Second, serverless applications save the efforts of plenty of backend commitments needed by VMs-based services. Server management, handler isolation, load balancing and many other backend commitments are now handled by the cloud provider, meanwhile different customers only request a serverless worker from the common serverless pool.

As opposed to the offerings, serverless computing has its shortfalls. First, direct data transaction between serverless workers is not supported so a relay server is needed to ship intermediate data back and forth. Second, the slow initialization of serverless workers would add to the latency. Time-critical applications would have a longer latency due to the cold start of serverless workers. Third, long-running complex event processing applications benefit less from serverless computing since sequential program logic cannot run parallelly. Fourth, despite the built-in fault-torlerant mechanisms, failed serverless workers have a longer latency and can be a straggler in the whole program pipeline. Fifth, severless computing raises security concerns because traditional security layers and mechanisms need to be ported to support serverless environments. Finally, serverless computing makes it harder to utilize specified hardware (e.g., GPU, TPU or a large RAM).

In this paper, we will discuss the pros and cons of serverless computing and also state the existing solutions to address its limitations.

2 What serverless computing can do

This section illustrates the offerings of serverless computing, due to which hosting web services or executing computations are simplified, with respect of the benifits: auto-scaling, high availability, real-time metrics and error aggregation and fine-grained resource billing plan.

Auto-scaling Cloud services achieve high elasticity through resource overprovisioning, overbooking and overcommitment based on their relatively long start delay [1]. However, with a much shorter start delay [2], serverless workers can easily and automatically be scaled when load suddenly increases. In a warm start, it can take less than one second to start a cloud function. [3]

Automated High Availability Serverless functions have built-in availability and fault tolerance. Compute capacity is guaranteed by the cloud provider. For instance, every AWS region is a seperate geographic region, which contains several independent AWS availability zones. Although rare, a failure in one availability zone means failures of all instances in it. Developers can deploy resources across different availability zones to gain better availability.

Real-time metrics and error aggregation With help of serverless worker, we can easily monitor each serverless worker invoked. Memory issues, misconfiguration, timeout, runtime errors and exceptions can be recorded and reflected promptly with help of frameworks such as Dashbird, DATADOG, IOpipe and many others.

Resource Billing Plan Known as pay-by-use, users are not charged by idle handlers, which saving much overhead cost of container-based microservices. For AWS Lambda function, fine-grained resource billing allows more accurate calculation over resource use per user. The expenses are calculated in 100 ms increments. On the other hand, program logic with life span shorter than 100 ms is not benefited as much, (e.g., RPCs). One solution is to measure and control each serverless worker execution time to be longer than the cost calculation unit, saving cost from worker initialization and time rounding.

3 Limitations of serverless computing and solutions

In spite of the offerings, serverless computing has its setbacks and limitations. As we will see in this section, current serverless computing restricts the ability to work efficiently with distributed computing resources.

Intermediate data sharing Current serverless computing solutions are restricted in terms of communication between serverless workers. Serverless workers cannot communicate with each other by design. Furthermore, serverless workers are running on isolated VMs, separated from data. As a consequnece, developers are expected to ship data back and forth to the serverless workers and among the workers. As for the solutions, there are plenty of mechanisms to ship data around. Communication between serverless workers are commonly achieved by a relay server, which handles intermediate data from workers, coordinates with program scheduler [4] and sometimes implements fault-tolerance (e.g., Sprocket [2], Pocket [5]). Another solution is to communicate through a slow storage (e.g., AWS S3). Maintaining the state means writing the state to S3 and loading state back in subsequent calls. Access speed of existing solutions can be either too slow or too pricey, (e.g., S3, DynamoDB, ElastiCache Redis and Apache Crail). A plausible approach of the relay server is to raise job-attribute awareness in resource efficient scheduling. For instance, Locus [6] utilizes multi-tier storage to achieve resource-efficiency while Pocket [5] raises object-size awareness.

Cold start Slow initialization time of serverless workers can add to the overall latency, and it is especially harmful to time-critical tasks. Warming up a new serverless worker includes starting the container, loading the programming language runtime, downloading libraries and packages and many others. Newly initialized serverless workers will mostly likely face a cold start. Solutions are divided into two predominant kinds. On one hand, developers can use a large number of parallel serverless workers to keep the some serverless function trigger frequent enough to keep 'warm'. For instance, ExCamera [4] uses an identical independent parallel function to achieve warm start of every worker, and intermediate data is cached in its relay server. Even so, cold start latency is not prefectly resolved because: (1) there is a limit of workers in each AWS region, (2) ExCamera uses an identical Lambda function for all serverless workers, but not all applications could use an identical Lambda function, and (3) network bandwidth is shared by all workers in a VM. By default up to 20 workers of the same user reside on the same VM, so network of the same VM is shared by all the workers on it. On the another hand, there are optimizations in the serverless runtime to achieve faster cold starting time [7]. Those optimizations include: (1) speed up container lifecycle, (2) fork from initialized processes runtime to shorten the start time, and (3) multi-tier caching to reuse initialization work across serverless workers.

Fault-tolerance Cloud computing is pervasive but service outages still take place. Even though serverless functions have built-in fault-tolerance mechanisms implemented by the cloud vendor [8, 3], failed workers have a high risk to be stragglers among a large number of workers. In case there are downstream task dependencies, a straggling worker might stall the entire pipeline [9]. As a consequence, speculative execution [10], proactively cloning [9], worker pairing [2] are been applied to tackle this problem. Nevertheless, they all add complexity to the entire pipeline and consume more resources.

Security On one hand, serverless computing is raising security concerns. Traditional security protections need to be ported to support serverless environments. Traditional layers such as end-point protection cannot be deployed on serverless workers, because it is needed to build authentication into every function that gets accessed directly, even though serverless workers are ephemeral. Due to its ephemera, there is no session management built into serverless frameworks. For serverless computing, security monitoring becomes harder since existing solutions require a long-lived agent. On the other hand, serverless computing avoids some attacks due to its architecture by design. Many of the exploits today are targeted at vulnerable unpatch servers while serverless pushes the role to the cloud provider to keep servers patched. Additionally, DDoS becomes a billing issue thanks to elasticity of serverless computing. Nevertheless, serverless computing cannot resolve the application-level vulnerabilities inside the program logic. Static and dynamic security testing is still demanded.

Harder to utilized specified hardware Advantages by hardware specialization and GPU are not yet able to be utilized in serverless functions. Pervasive use of deep learning benefits from GPU, TPU or other hardware accelerators. Databases with large RAM based data structures such as SAP Hana also can not benefit from serverless computing. For instance, AWS Lambda function only offers 3 Gigabytes of RAM and is far from enough for application with high RAM demands. The lack of access to such hardware—along with appropriate pricing models limits the utility of serverless computing as a platform for software innovation [3].

4 Conclusion

Serverless computing differs from its predecessors in several essential ways: better autoscaling, fault-tolerance, security, strong isolation, platform flexibility and service ecosystem support. While some argue serverless computing is to some extent a rebranding of preceding offerings, we argue that standardization of programming model would help developers benefit from serverless computing. We could hopefully expect serverless to become more mainstream in enterprise applications and more integrated with other technologies, such as microservices and traditional application architectures.

5 Acknowledgments

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2015-0-00221, 다양한 분석을 고속 수행하는 단일화된 빅데이터 스택 개발).

참고 문헌

[1] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, (Denver, CO), USENIX Association, 2016.

[2] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, (New York, NY, USA), pp. 263–274, ACM, 2018.

[3] "Kubernetes and the path to cloud native," (Santa Clara, CA), USENIX Association, 2015.

[4] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, (Boston, MA), pp. 363–376, USENIX Association, 2017.

[5] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, (Carlsbad, CA), pp. 427–444, USENIX Association, 2018.

[6] Q. Pu, S. Venkataraman, and I. Stoica, "Shuffling, fast and slow: Scalable analytics on serverless infrastructure," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, (Boston, MA), pp. 193–206, USENIX Association, 2019.

[7] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "SOCK: Rapid task provisioning with serverless-optimized containers," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, (Boston, MA), pp. 57–70, USENIX Association, 2018.

[8] T. E. Anderson, D. E. Culler, and D. Patterson, "A case for now (networks of workstations)," *IEEE Micro*, vol. 15, pp. 54–64, Feb 1995.

[9] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, (Lombard, IL), pp. 185–198, USENIX, 2013.

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, (San Francisco, CA), pp. 137–150, 2004.