Serverless Function Review Report

JeongWan Gho

Abstract

This is a review report of "*Benchmarking, Analysis, and Optimization of Serverless Function Snapshots*" [1] and "*FaaSnap: FaaS Made Fast Using Snapshot-based VMs*" [2]. Both articles try to reduce the cold-start delay of serverless applications. OpenWhisk [3] was the most popular open-source framework for serverless platforms before Firecracker [4]. OpenWhisk is a framework based on container technology. Firecracker tried to make the workloads of virtual machines lighter. These lighter virtual machines are called as microVMs. To control the microVMs, firecracker virtualized the creation and management of multi-tenant container and function-based services in serverless operational models. One of the key features that Firecracker provides is the snapshot of microVMs. Without a snapshot, there exists a cold-start delay for booting a virtual machine and initializing the environment for running the serverless program. With the snapshot technique, the total memory state of the running serverless function is saved as a file on the disk. This saved snapshot file will be loaded when the serverless function is needed to be executed. The snapshot technique reduced the cold-start delay greatly, but there is still much to be optimized. "*Benchmarking, Analysis, and Optimization of Serverless Function Snapshots*" [1] and "*FaaSnap: FaaS Made Fast Using Snapshot-based VMs*" [2] both suggest a new methodology to optimize cold-start delay based on the snapshot technique. "*Benchmarking, Analysis, and Optimization of Serverless Function Snapshots*" [1] suggest REAP; Record And Prefetch. "*FaaSnap: FaaS Made Fast Using Snapshot-based VMs*" [2] suggests FaaSnap which is a snapshot-based platform that improves cold-start performance by reducing the guest VM page fault.

1 Introduction

The traditional infrastructure for providing IT services is on-premises. On-premises software is usually installed and operated on a single huge server computer which is owned by the company which is also an owner of the IT services. However, the operation of infrastructure is a heavy burden and requires difficult skills. Most IT companies do not have enough computer engineers to operate the infrastructure, and they wanted to assign the infrastructure labor to companies which are specialized in infrastructure. Clients of infra-specialized companies are the actual service companies. Service companies develop and maintain programs to earn money. Most of the employees of service companies are developers and usually lack at skills to operate infrastructures to run programs they developed. Infra-specialized companies take orders from the service companies and replace the role to manage the infrastructure.

The early stage of infra-specialized companies was server hosting companies. Server hosting companies purchase bare-metal hardware computers and set the infrastructure from the bottom. Setting the infrastructure includes installing an operating system to a bare-metal machine, making a network environment for internet connection, preparing for storage backup, and enhancing security. Service companies lease these managed computers to run their programs. The lease contract is based on the actual machine and the charging unit is longer than a month.

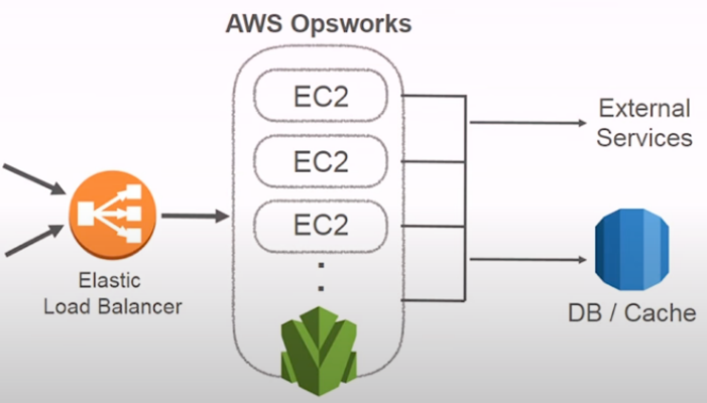
In 2006, Amazon established the Amazon Web Service. AWS provided computing power based on the cloud instance. Amazon elastic compute cloud (EC2) [5] is an infra platform that offers processor, memory, storage, networking, and operating system for running workloads of IT services. EC2 instance is not a physical computer but a virtual machine running over Amazon's data center. The key feature that the software developers were so happy about is that EC2 instance offers auto-scaling without the request of the software developers. The CPU utilization and memory utilization are monitored by Amazon Cloud Watch [6]. If the utilization surpasses a certain threshold, AWS automatically scales up the instance to provide better computing power. Before the cloud innovation, the developers should manually monitor the utilization of computer resources and should manually request for the change of instance to the infra-specialized company. These manual monitoring and requests caused serious night shifts, but developers were liberated after the cloud innovation based on the auto-scaling of instances.

Recently serverless is emerging as a beloved innovation in the cloud area. The traditional monolithic software architecture tends to be replaced by the micro-service software architecture. Due to the micro-service architecture, the size of a single service becomes smaller and smaller. Traditional scaling up of a single powerful instance is no longer appropriate for these small services. The need for a smaller unit of computing power was requested from the industry, and the container was used as a computing unit for serverless platforms such as OpenWhisk. Amazon created a new serverless platform, Firecracker, which provides microVM to handle these small and decoupled micro-services. The actual execution time and required computing resources are very small for each micro-service. The booting of machine and the initialization of the environment becomes a new bottleneck because the booting and initialization time becomes relatively bigger than the actual execution time. To reduce the cold-start delay, Firecracker offers a feature that enables snapshotting a microVM. The snapshotted state of microVM can now be saved as a file on disk and can be loaded to memory. "*Benchmarking, Analysis and Optimization of Serverless Function Snapshots*" [1] suggested recording the working set files and then prefetching the working set file only to the memory. This is very effective for reducing cold-start delay because the actual size that should be loaded on memory is greatly reduced. "*FaaSnap: FaaS Made Fast Using Snapshot-based VMs*" [2] suggested concurrent paging, per-region mapping, and usage of loading set. These three techniques greatly reduced the guest VM page fault and reduced cold-start delay consequently.

2 Background

To understand what kind of expectation is rising for serverless technology from the industry, micro-service architecture needs to be considered. Micro-service architecture came out as a replacement for monolithic software architecture. The micro-services can be defined as follows; small autonomous services that work together, modeled around a business domain. To handle the micro-services, the following five features can be suggested for the convenience of software development. First, each micro-service should be scaled independently. Second, each micro-service should be deployed independently. Third, each micro-service should have independent versions. Fourth, each micro-service should be developed independently. Finally, each micro-service should be tested and monitored independently. The comparison between instance and serverless infra based on these 5 features will show the reason why serverless functionality becomes so popular among recent developers.

2.1 Instance-based cloud architecture



The term business logic or domain logic is the essence of the functionality that the program intends to provide. Most of the business logic relates to how data is created, updated, and selected from the storage. The storage can be provided as external services from external companies, or the storage can be provided internally by storage-related AWS services such as Amazon relational database service [7] or DynamoDB [8]. The aggregation of business logic is usually developed as a single program and run on a single machine. In the instance-based cloud architecture, each software code that runs on the EC2 instance is identical. EC2 instances that have identical functionality are grouped and managed by AWS Opsworks [9]. Instance-based cloud architecture responds to the different scales of requests by scaling up or scaling out these identical EC2 instances. After EC2 instances are successfully scaled up or scaled out, the workload is distributed through the Elastic Load Balancing [10] to each EC2 instance.

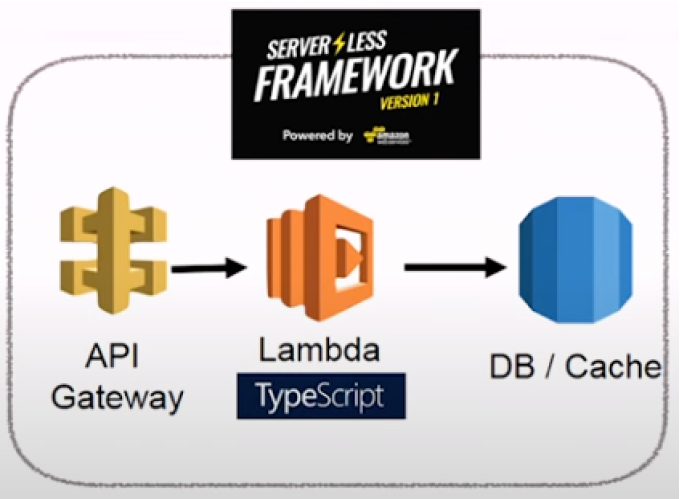
When Amazon showed off the Elastic Compute Cloud, Amazon came out with the concept of “credit” as a charging unit for the cloud infra. Credits are consumed when the cloud infra is utilized. According to the billing that customer chose, the credit increases by a uniform amount during the unit time. If the CPU utilization is low, the increasing speed of the credit is faster

than the speed of consumption. The credit increases until the upper limit of the billing system. If the CPU utilization is high, the consumption speed of the credit is faster than the increasing speed. If the credit is exhausted, the customer can have a choice between three options. First, customers can endure the temporal downgrade of the EC2 instance. This downgrade can cause temporal lag in the service processing or even can cancel the processing. Second, customers can scale up the EC2 instance to prevent the deprivation of computing power by changing the billing option to a more expensive option. The scaling up of the EC2 instance will provide more credits in unit time and thus let the EC2 instance handle the increased credit consumption. Third, customers can scale out EC2 instances by purchasing new instances. The third option increases the number of EC2 instances, not the computing power of a single EC2 instance. Therefore, appropriate load balancing is required.

In the traditional monolithic software architecture, the scale-up option is preferred over the scale-out option. This is because load balancing is a difficult task for the majority of developers. To support the scale-out option, AWS came out with the Elastic Load Balancer which automatically distributes the workload to the scaled-out EC2 instances. Amazon also supports auto-scaling through the combination of AWS Cloud Watch [6] and Opsworks [9]. Due to the convenience of automation, scale-out is no longer a difficult burden for developers. Nowadays, scale-out is considered a better way than the scale-up option because of independent runtime. If the scale-up fails, this can lead to the total shutdown of the whole service. On the other hand, scale-out does not affect the functionality of old instances even though the creation of new instance is failed.

Since the billing and credit system is based on the utilization of computing power, the auto-scaling triggers are also based on the utilization of CPU and memory. During the time when the scale-up was the more considerable option than the scale-out, utilization was an appropriate option for auto-scaling. Unlike scaling up a single EC2 instance, measuring the utilization of scaled-out multiple EC2 instances became a new challenge. This challenge became more difficult due to the rise of the micro-service architecture trend. Now, developers want to separate a huge monolithic service into micro-services and run micro-services in an independent runtime environment. Traditional EC2 instance and utilization-based billing systems become inefficient to handle these micro-services. To overcome this inefficiency, serverless-based cloud architecture starts to rise as a great substitute for traditional instance-based cloud architecture.

2.2 Serverless-based cloud architecture



AWS Lambda [11] is a serverless, event-driven compute service based on Firecracker serverless VMM. When HTTP API is requested, API Gateway [12] triggers events so that AWS Lambda can process the business logic through the computing power of microVM. As the unit of computing power is changed from EC2 instance to Lambda microVM, the billing unit also changed from resource utilization to millisecond of microVM activation. In the serverless cloud architecture, scaling is no longer considered by developers. Since the major cause of the heavy workload is the drastic increase in the number of HTTP requests, scaling can be easily solved by assigning an independent microVM to each HTTP request.

Serverless also provides innovation in the software development and the testing of the software. Before the serverless technology, developers have to manage three types of servers infra: production, stage, and development. The production server is the real server that provides computing power for real customers. The stage server provides an environment for testing the software before the new version of the software is applied to the actual market. The development server is an environment for developers to develop a new version of the software. The problem with managing these 3 types of servers is that the version of 3 types can conflict due to unexpected accidents. If the version conflict problem becomes too messy, the recommended solution is the creation of new stage servers and development servers. This kind of accident does not occur often but can become heavy work for developers if the versioning accident occurs. Serverless technology liberates developers from the fear of unstable testing and development environments, and therefore serverless technology is gaining more and more popularity in the industry.

However, serverless architecture brought a new challenge called cold-start delay. Since each business logic run on a microVM is so small, booting and initializing the runtime environment is becoming of a new bottleneck. One of the features that Firecracker provides to handle the cold-start delay is snapshotting. REAP and FaaSnap provides the state-of-art methodology for snapshotting optimization.

3 Methodology for snapshot

Snapshotting methodology reduces cold-start delay by reducing the booting time of the machine and the initialization of the environment. The state of a fully booted function is snapshotted, and the created snapshot image is stored as a file on disk. The stored image file is loaded to the memory and reduces the total time needed until the actual invocation of the function. Based on the snapshotting, REAP and FaaSnap suggests a more optimized methodology for faster function invocation. REAP is a lightweight software mechanism in serverless infrastructures suggested by the article "*Benchmarking, Analysis, and Optimization of Serverless Function Snapshots*" [1]. And FaaSnap is a snapshot-loading mechanism that uses complementary optimizations for better cold-start performance suggested by the article "*FaaSnap: FaaS Made Fast Using Snapshot-based VMs*" [2].

3.1 REAP

The abbreviation REAP stands for "REcord And Prefetch". The key idea of REAP is the usage of the working set file instead of a full snapshot image. REAP mechanism consists of two phases: the record phase and the prefetch phase. During the record phase, working set files and trace files are recorded. The working set file is a copy of accessed guest memory pages of the fully booted microVM whereas the trace file has the offsets of the original pages inside the guest memory file. During the prefetch phase, recorded working set files and trace files are prefetched and loaded to memory. The benefit of REAP mechanism is the small size of the working set file compared to the full snapshot image. Since the file size of the working set is smaller than the full snapshot file, the possibility of page fault decreases. The reduction of page faults leads to reduced disk reading and enables faster function invocation consequently.

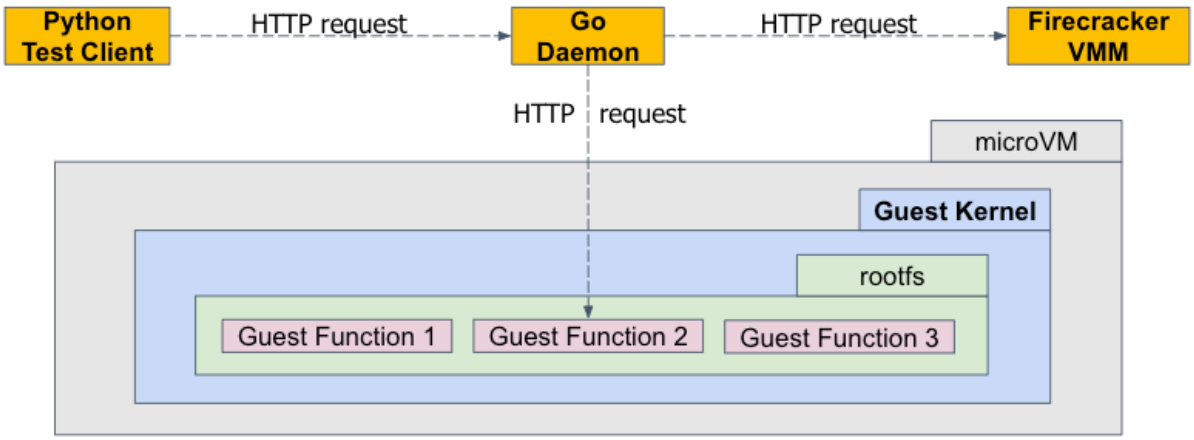
3.2 FaaSnap

FaaSnap also tried to optimize snapshotting for a faster cold-start delay. For more efficient snapshotting, FaaSnap combines multiple techniques such as concurrent paging, working set group, host page recording, per-region memory mapping, and loading set. The first technique suggested by FaaSnap is concurrent paging. Concurrent paging starts the VM concurrently while prefetching the pages from the disk. This concurrency is possible through the goroutine syntax which is supported by the Go programming language. The second technique suggested by FaaSnap is a working set group. Through the working set group technique, working set pages are grouped by their access order. The number of the working set group is used as an order for loading, and this loading order is more effective compared to the readahead mechanism of Linux. The third technique suggested by FaaSnap is a host page recording. FaaSnap uses the mincore syscall to scan the present bits in the page table entries. This scan detects if guest pages are in the host page cache. By recording the results of mincore syscall,

FaaSnap can be tolerant of changes in the working set. The fourth technique suggested by FaaSnap is per-region memory mapping. FaaSnap maps the pages that are non-zero to the guest memory file to handle the semantic gap problem between the host and the guest. This is much more efficient mapping than the memory mapping of the entire guest memory file. The fifth technique suggested by FaaSnap is the loading set. The loading set is the working set pages excluding the zero pages. Consequently, the loading set file is more compact than the full working set file. This compactness enables efficient loading of files to memory with less page fault.

4 Components of FaaSnap

FaaSnap provides a framework to run and test serverless functions with different snapshot methodologies. The framework consists of 5 components: python test client script, go daemon web application server, customized Firecracker virtual machine manager, customized Linux guest kernel, and pre-built root file system which contains serverless guest functions.

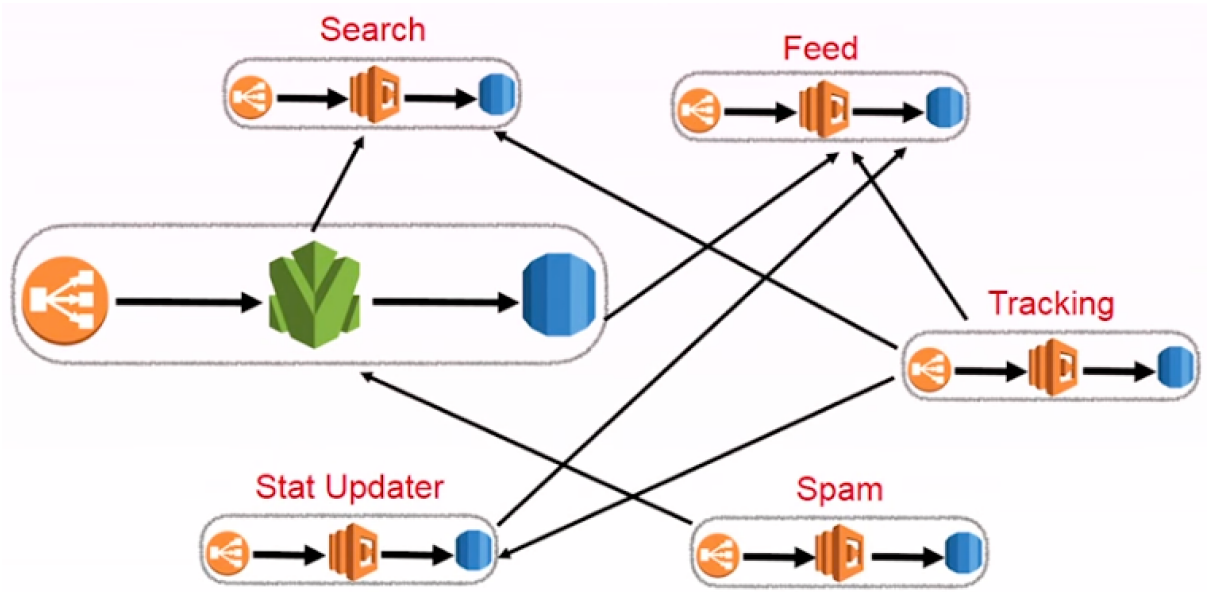


Python test client script is the test scenario which contains information about the invocation order of serverless functions and the invocation option of serverless functions. Snapshotting methodology is considered one of the invocation options of serverless functions. The test scenario of the python client is passed to go daemon through HTTP request.

Go daemon is a web application server that is implemented by Go programming language. Go daemon handles test scenarios requested by the python test client. Go daemon creates microVM through HTTP requests to the Firecracker VMM. After the creation of microVM, a monolithic root file system containing serverless functions is loaded on the microVM with the Linux guest kernel. Serverless functions can be invoked when the creation of microVM and a load of the root file system is finished. This fully booted serverless function is snapshotted by Go daemon with various snapshotting methodologies. Go daemon also evaluates the cold-start delay and invocation delay while executing these test scenarios.

5 Future Work

FaaSnap is a framework based on the monolithic root file system with the sequential test scenario. Some test scenarios test a single serverless function requiring high CPU computing power and memory usage. However, recent developers in the industry are trained to separate every huge monolithic service into micro-services requiring low CPU computing power and memory usage. Serverless functions in the real industry are developed in micro-service architecture and are invoked in asynchronous scenarios. Thus, the scalability of the serverless framework can be considered a more important factor than the efficient utilization of CPU and memory resources. To test the asynchronous serverless invocation on a large scale, a new test framework that supports heterogenous function calls needs to be developed. This heterogeneous function call can be supported through parallel processing like multi-threading or coroutine. Python supports multi-process and asynchronous libraries but is not fast enough to run the large-scale invocation scenario efficiently. Therefore, the new test framework should be developed using a high-performance programming language such as Go.



Scalability is an important factor that should be tested and researched in the serverless field. Another important factor that should be considered during the development of serverless applications is the dependency flow of each serverless function. When a single monolithic application is separated into multiple micro-services, dependencies between micro-services can occur. This dependency flow is a very useful clue for end-to-end application debugging. If one serverless function has a high dependency on another, the other serverless function can be a bottleneck or cause the malfunction. The bottleneck of a particular serverless function is a serious problem because it can increase the execution time of multiple serverless functions, leading to extra charging and a bad user experience. Unfortunately, tools for analyzing the dependency flow of each serverless function are not good enough as the auto-scaling feature of serverless infra. Developers in the real-world industry usually draw the dependency flow of serverless functions by their own hands, but this is becoming heavy labor as the aggregated

application becomes more complex. Thus, the methodology for analyzing the dependency flow of the serverless function can be a good further work.

6 Conclusion

Serverless cloud architecture is getting more and more popular for industry developers compared to traditional instance-based cloud architecture. This popularity is due to the paradigm shift from a monolithic application to micro-services. Since the actual execution time of a micro-service is too short, the booting time of microVM and the initialization time for the runtime environment becomes a new bottleneck. This challenge is called cold-start delay and Firecracker VMM tried to reduce cold-start delay by snapshotting. REAP and FaaSnap suggest optimization techniques for the snapshotting methodology. However, the test scenario is still based on monolithic software and does not consider asynchronous scalability. For further work, the heterogenous function invocations on a large scale need to be considered.

References

[1]
Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021.
Benchmarking, Analysis, and Optimization of Serverless Function Snapshots.
In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS'20).
pp. 559– 572.

[2]
Lixiang Ao, George Porter, Geoffrey M. Voelker. 2022.
FaaSnap: FaaS made fast using snapshot-based VMs.
In *Proceedings of the 17th European Conference on Computer Systems* (EuroSys '22).
pp. 730-746

[3]
Apache OpenWhisk. (n.d.).
Open Source Serverless Cloud Platform.
https://openwhisk.apache.org

[4]
Firecracker. (n.d.).
Secure and fast microVMs for serverless computing.
https://firecracker-microvm.github.io

[5]
AWS. (n.d.).
Amazon Elastic Compute Cloud. User Guide for Linux Instances.
https://docs.aws.amazon.com/pdfs/AWSEC2/latest/UserGuide/ec2-ug.pdf

[6]
AWS. (n.d.).
Amazon CloudWatch. User Guide.
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/acw-ug.pdf

[7]
AWS. (n.d.).
Amazon Relational Database Service. User Guide.
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/rds-ug.pdf

[8]
AWS. (n.d.).
Amazon DynamoDB. Developer Guide.
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/dynamodb-dg.pdf

[9]
AWS. (n.d.).
AWS OpsWorks. User Guide.
https://docs.aws.amazon.com/opsworks/latest/userguide/opsworks-ug.pdf

[10]
AWS. (n.d.).
Elastic Load Balancing. User Guide.
https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/elb-ug.pdf

[11]
AWS. (n.d.).
AWS Lambda. Developer Guide.
https://docs.aws.amazon.com/lambda/latest/dg/lambda-dg.pdf

[12]
AWS. (n.d.).
Amazon API Gateway. Developer Guide.
https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-dg.pdf