Serverless Comparison

Hyungro Lee, Kumar Satyam and Geoffrey C. Fox
School of Informatics,
Computing and Engineering
Indiana University
Bloomington, Indiana 47408

***Abstract*—Serverless computing also called Function-as-a-Service (FaaS) provides a small runtime container to execute lines of codes without management of infrastructure which is more like simpler version of PaaS. Amazon, Google, Microsoft and IBM offer serverless computing with various features but some limitations exist. We intend to generate a comparison with benchmarking results therefore our report becomes a guideline of further research on serverless computing. We also investigate existing platforms to see if it can be used to perform large distributed computation and apply to big data analytics. This report provides comparisons towards 1) elasticity, 2) scalability, 3) flexibility, 4) cost efficiency, 5) concurrency and 6) functionality.**

***Keywords*—FaaS, Serverless, AWS, GCE, Azure, IBM OpenWhisk**

I. Introduction

Serverless computing is a first commercial cloud service that uses 100 milliseconds as a charging metric compared to traditional cloud services using an hourly charge metric. Serverless is a miss-leading terminology because it runs on a physical server but it succeeded in emphasizing no infrastructure configuration requirement to manage compute workload. Geoffrey et al [1] defines serverless computing among other existing solutions i.e. Function-as-a-Service (FaaS) and Event-Driven Computing. We also understand that serverless evolved recently because container technology allows to create a namespace for the workload within a minute and certain restrictions e.g. 300 seconds timeout increase overall resource utilization from the provider perspective. In the following sections, we simply investigate current serverless platforms in terms of its elasticity, scalability, flexibility, cost efficiency, concurrency and functionality and describe existing issues and restrictions including suggestions. Use cases are followed to demonstrate its capacity to run large and distributed tasks including scientific computing applications.

II. Results

In the form of the first report, we show progress with the current results regarding to computation, elasticity, price and frontend request handler, the trigger. We may add more results and adjust current results in the final paper with taking into all considerations and feedback.

A. CPU Flops

Serverless platform allocates compute resources based on the amount of requests which shows up to a peak double-precision floating point performance of 70 TFLOPs when 3,000 concurrent function invocation requested on Amazon Lambda. Figure 1 is the first result of invoking 3000 functions on Serverless Functions which indicates proportional between the number of functions and the aggregated GFLOPs. AWS Lambda outpaces the other competitors which generates almost 7x faster compute speed. Azure Functions, IBM OpenWhisk, and Google Functions are in either a beta service level or an early stage of development therefore we expect that the allocated compute resource will be more comparable when the service is fully mature.

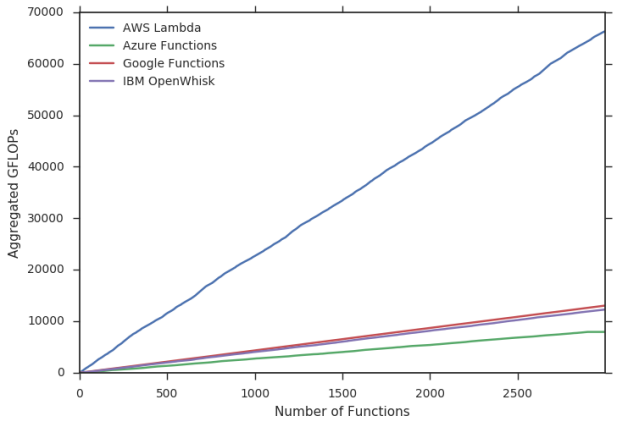

Fig. 1: Serverless Functions with 3000 parallel execution

B. Elasticity

Figure 2 shows AWS Lambda elastic provisioning for dynamic workload changes over time. The blue dot indicates a function execution time which normally takes less than two seconds but there are additional fifteen seconds for initializing environments and installing packages when the function is newly created. You find that there is a blue dot group at the beginning with about 15 function execution time because compute resource is being increased for the first request. The second group of the blue dots is shown in between 100 and 200 elapsed time where increased workload given to a function and the additional compute resources are provisioned to deal with more workload than the current function can handle. The maximum workload is given in between 400 and 500 elapsed time but we do find no additional compute resources is created. It explains that the increased amount of compute resources are kept for a certain amount of time to provide enough capacity about the future resource demands. We observe that Azure functions and AWS lambda terminate a process after 30 minutes idle timeout.

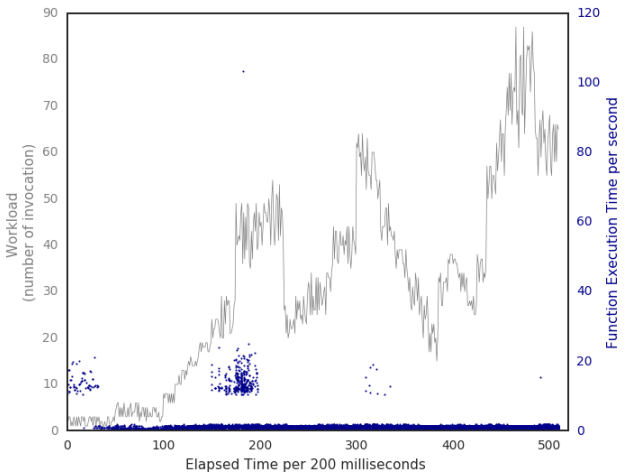

Fig. 2: AWS Lambda Provisioning for Workload

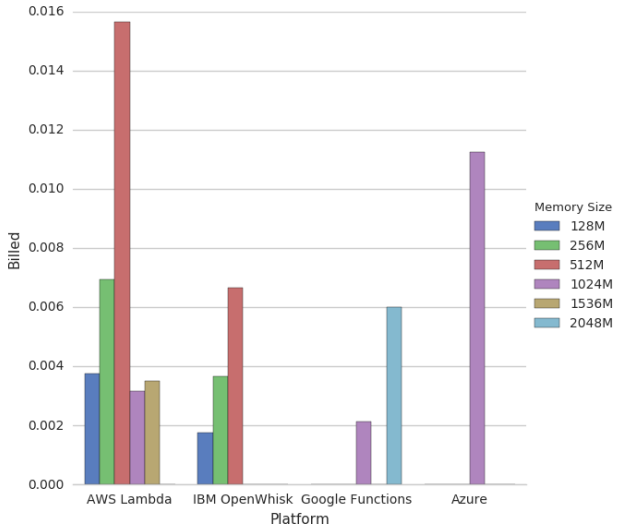

Fig. 3: Price Comparison

C. Value

Commercial serverless providers introduced a millisecond charge metric for a serverless function and we ran a same program to measure actual costs among different providers with memory size options. Note that the program we used is a matrix multiplication which generates 2-dimensions of two 1024 matrices and the charge is calculated with the amount of allocated memory. Figure 3 shows that IBM OpenWhisk with 128 memory option is the most inexpensive choice and choosing a small size of memory does not save bills regarding to AWS Lambda. Note that some bars are missing because there is no option in some of providers. Google Function also failed to run the program between 128 M and 512 M memory options.

III. Trigger Comparison

The serverless platforms that we have seen is a subset of event-driven computing and the front-end event handler is the most important use case for serverless technology which is also called a trigger. Serverless computing providers support different type of events including HTTP, object storage e.g. AWS S3, and a database e.g. IBM Cloudant. We find that a trigger has an important role in serverless computing because functions are invoked by triggers. We also find that each trigger has different capacity to deal with large number of concurrent requests and we did latency comparison of HTTP and database triggers between different cloud platforms such as AWS Lambda, IBM OpenWhisk, Google Cloud Functions and Microsoft Azure Functions.We executed the same function across different cloud function. We first started with AWS Lambda. We tested AWS Lambda with triggers from HTTP API gateway, DynamoDB and S3 as well. For IBM OpenWhisk, we tested a HTTP trigger and the IBM Cloudant trigger. For Google Cloud Function, we had triggers from HTTP, Google Cloud Storage.They do not offer database trigger, although a pub/sub messaging trigger is offered.For Azure Functions, we had triggers from HTTP. The numbers are given in the link.

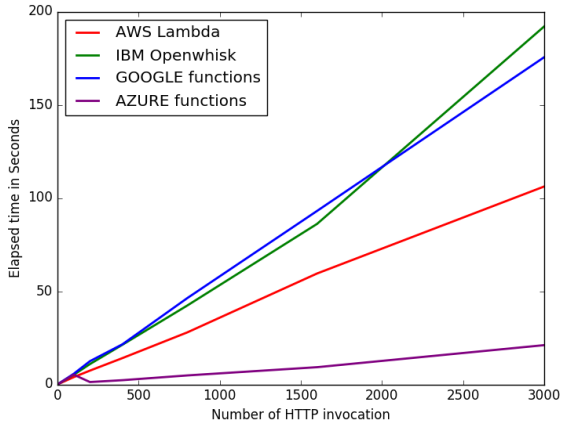

Fig. 4: HTTP Trigger Comparison

A. HTTP Trigger

We have highlighted the HTTP trigger which is available in all the cloud platforms. As per Figure 4 we see that the time taken to invoke IBM OpenWhisk function and Google functions from HTTP endpoint trigger is the highest where as the Azure Function takes the least. Also, all cloud vendors show a linear pattern of function invocation when the parallel HTTP requests increases. We don't see any degradation of performance in handling massive requests up to 3000 concurrent invocations. We can conclude that the increase in invocation does not affect the performance.

B. Database Trigger

In the Figure 5, we did a comparison of the database type of trigger. AWS DynamoDB and IBM Cloudant are tested as a direct trigger to their respective vendors' functions. As of now we cannot compare Azure and Google Cloud as they do not have a direct trigger available to their respective functions. As per the bar chart, the throughput of the AWS DynamoDB

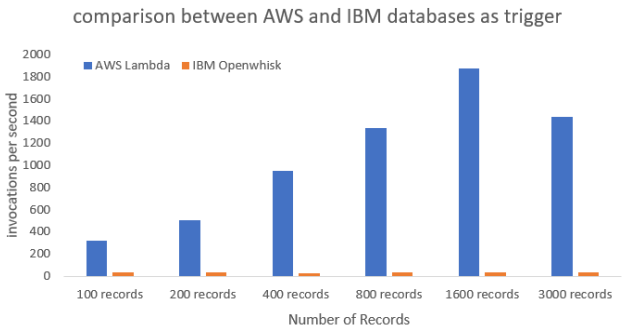

Fig. 5: Database Trigger Comparison

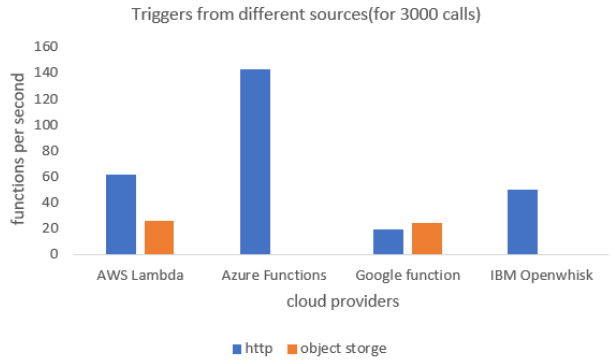

Fig. 6: Throughput of Triggers

trigger and the IBM Cloudant trigger are shown with the increased number of parallel writes to those databases. As per the graph, we see that performance of the AWS DynamoDB trigger surpasses the IBM Cloudant trigger in each number of records. We observe that AWS DynamoDB to Lambda indicates a decrease in performance between 2600 and 3000 records whereas the IBM Cloudant trigger to Openwhisk maintains linearity in spite of increasing the number of writes to Cloudant.

Figure 6 shows throughput of each trigger per provider. We compared HTTP trigger and object storage trigger to show which trigger performs better among the providers. The Azure HTTP trigger shows the best performance whereas the Google HTTP is the least capable of processing requests among the four providers. For object storage, AWS S3 trigger performs better than the Google Cloud storage trigger but we still find that HTTP trigger is a more reliable choice in processing multiple requests. Note that we were able to perform the object storage trigger for AWS and Google cloud storage only as the other providers do not offer a direct database trigger as of now. This will help the reader of the paper to decide which trigger work faster among all the four providers.

IV. Feature Comparison Among Different Cloud Function

The feature comparison would be helpful to the new users of serverless computing and will help the readers of this paper understand the underlying system level information of the serverless platform. As per the Table I, AWS Lambda

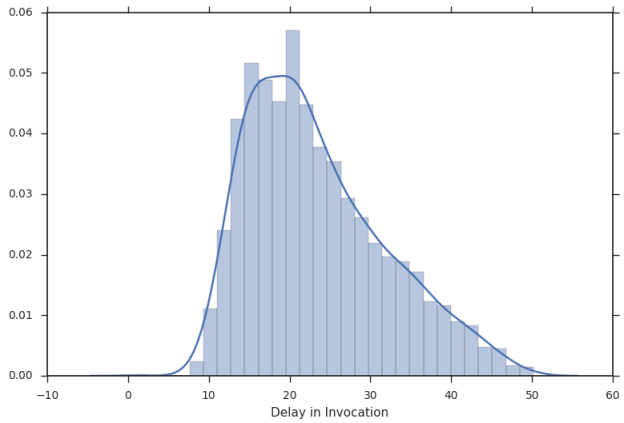

Fig. 7: Cold Start Delay in seconds invoking Azure Functions over 3000 parallel execution

offers a wide range of trigger endpoints compared to the other cloud providers.We also see that the cost of usage of serverless function is based on two metrics. First, the number of invocation of serverless functions. Second, the time taken by a serverless function to execute and complete paired with an amount of memory in size of gigabytes allocated. Invocation to the serverless functions is really cost effective in all serverless providers if an application is executable with certain restrictions that serverless computing has. All providers have similar pricing tables but IBM openWhisk does not charge the number of invocations whereas the other providers do charge. Google upscales in terms of memory as it provides maximum of 2 GB of memory to run a serverless function. Google also outperforms in terms of providing maximum execution timeout of 9 minutes which would be helpful for long running jobs. IBM OpenWhisk has the container which can provided the best clock speed of 2100 *4 MHz.

V. WIP

The following sections are currently under development. We will add more thoughts and perform experiments to enrich final results of this work.

A. Latency for Cold Start vs Warm Start

We measured delay for invoking functions over 3000 function calls on Azure in Figure 7. Other platforms will be compared.

VI. Guidelines of Serverless

This section describes current restrictions and certain factors of serverless computing for better understanding how to use and how to migrate any current applications if it satisfies requirements. Runtime languages, memory restrictions, and timeout limitations are addressed. The current implementation of serverless platform allows you to choose programming languages such as node.js and Python but more languages would be added soon with extra libraries.

Item	AWS Lambda	Azure Functions	Google Functions	IBM OpenWhisk
Runtime language	node.js, Python, Java, C#	C#, F#, node.js, Java, PHP	node.js	node.js, Python, Java, C#, Swift, PHP, Docker
Trigger	18 triggers (i.e. S3, DynamoDB)	6 triggers (i.e. Blob, Cosmos DB)	3 triggers (i.e. HTTP, Pub/Sub)	3 triggers(i.e. HTTP,Cloudant)
Price per Memory	$0.0000166/GB-s	$0.000016/GB-s	$0.00000165/GB-s	$0.000017/GB-s
Price per Execution	$0.2 per 1M	$0.2 per 1M	$0.4 per 1M	n/a
Free Tier	First 1 M Exec	First 1 M Exec	First 2 M Exec	Free Exec / 40,000GB-s
Maximum Memory	1536MB	1536MB	2048MB	512MB
OS	Linux ip-10-13-100-130 4.9.43-17.39.amzn1.x86_64	Windows NT	Debian GNU/Linux 8 (jessie)	Alpine Linux;14.04.1-Ubuntu
Max CPU	2900.05 MHz,1 core	1.4GHZ	2200 MHz, 2 Processor	4 cpu cores,2100.070 MHz
Temp Directory	512 MB (/tmp)	500 MB (%Local%)		
Execution Timeout	5 mins	5 mins	9 mins	5 mins
Logging limit				10 MB
Code size limit				48 MB
API references	cli tool	.NET, pyhton,node,java,ruby, rest	gcloud CLI tool, rest api, rpc api	cli tool

TABLE I: Feature Comparison

A. *Restriction*

It is worth to mention the existing restrictions and discuss why it is necessary and how to change for building better platforms and achieving cost efficiency. The following items are discussed:

- runtime languages
- timeout
- memory size
- number of invocation
- number of functions

B. *Suggestion*

We also find that this section might be useful to discuss what type of workloads and applications are beneficial in running on Serverless platforms. The following items are broadly discussed so far:

- increasing timeout
- no limitation on runtime choice
- options choosing server types for particular workloads e.g. gpu enabled
- common library packages e.g. numpy, scipy, matplotlib, tensorflow, caffe

VII. Use Cases of Serverless

This section demonstrates a few examples in utilizing serverless platforms to perform a large computation in a sense of parallel job execution. Big Data examples, hadoop word count, sort and machine learning training can be included.

VIII. Discussion

There are overlaps and similarities between serverless and the other existing services, for example, Azure Batch is a job scheduling service with an automated deployment for a computing environment. AWS Beanstalk [2] is deploying a web service with automated resource provisioning.

IX. Related Work

PyWren [3] is introduced in achieving about 40 TFLOPs using 2800 AWS lambda invocations. It is necessary to show a similar computing power among other serverless providers. McGrath et al [4] showed latency comparisons among the commercial serverless providers but other aspects are not fully investigated such as trigger/binding performance and failure rate. OpenLambda [5] is the first paper addressing serverless platforms for web application since then there are changes that need to be mentioned. Kubeless [6] is a new serverless framework using Kubernetes, a container framework.

References

[1] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and function-as-a-service (faas) in industry and research," *arXiv preprint arXiv:1708.08028*, 2017.

[2] A. Amazon, "Elastic beanstalk," 2013.

[3] E. Jonas, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," *arXiv preprint arXiv:1702.04024*, 2017.

[4] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 405–410.

[5] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," *Elastic*, vol. 60, p. 80, 2016.

[6] Kubeless, "A kubernetes native serverless framework," 2017.