Final Report for Independent Study: CSCI Y790

under Dr. Geoffrey C. Fox

Research Topic: Serverless Computing

Name: Kumar Satyam
Email-Id: ksatyam@indiana.edu

Table of Contents

Table of Contents 2
Abstract 3
Introduction 3
Triggers 3
HTTP Trigger 4
Database Trigger 4
Object Storage 4
Trigger Comparison 4
Feature comparison among different cloud function function 5
IAAS vs FAAS Pricing information: 6
Language Comparison: 8
Use Cases 10
Bibliography 10

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. 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.

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 (Fox, Geoffrey C, Ishakian, Vatche , & Muthusamy) 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 their elasticity, scalability, flexibility, cost efficiency, concurrency and functionality and describe existing issues and restrictions. Use cases are followed to demonstrate its capacity to run large and distributed tasks including scientific computing applications.

Triggers

The serverless computing is a subset of event-driven computing which has a front-end event handler to invoke functions. We find that event types variously depends on application behaviors and purposes. For example, IoT device sends a notification when sensors detect new changes and the notification might be a trigger of other applications to process the sensor data. Serverless computing providers support various types of events including HTTP requests, object storage e.g. AWS S3, and a database e.g. IBM Cloudant thus as many actions as they can handle in order to answer back the event messages. Event handlers also called triggers either listen events and create a function invocation (push model) or collect changes at a regular interval to invoke a function (pull model). In this section, we measure trigger resolutions to see how sensitive it is and understand its capacity of concurrent event messages. We measured a latency of triggers between different serverless providers 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 an HTTP trigger and the IBM Cloudant trigger. For Google Cloud Function, we had triggers from HTTP and Google Cloud Storage. For Google Cloud Function do not offer database trigger, although a pub/sub messaging trigger is offered. For Azure Functions, we had triggers from HTTP and storage. The data for trigger result is in the report.

HTTP Trigger

Http trigger provides a simple format to invoke a function with various input types e.g. JSON and an asynchronous call is optional which is useful for dealing with concurrent requests without blocking. Note that the asynchronous option and concurrent request limit vary by providers and quotas by personal accounts. As per Figure 1 we see that Microsoft Azure has the highest number 142 of invocations per second whereas Google Functions shows the least throughput as they invoke very less number of functions per second.

Database Trigger

Database trigger invokes a function when there is an insertion/modification/deletion of a record in a table which behaves like a message queuing system. Google supports pub/sub trigger in their serverless platform and it might be exchangeable with a database trigger since Google Functions does not have a database trigger. We see the comparison of the database type of trigger with AWS DynamoDB and IBM Cloudant 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 graph, we see that performance of the AWS DynamoDB trigger surpasses the IBM Cloudant trigger.

Object Storage

Object storage is widely used to store and access data from various platforms and we find that the object storage trigger is supported in the most serverless providers. AWS S3 trigger performs better than the Google Cloud storage trigger. Note that we were not able to perform the object storage trigger for IBM cloud storage it does not offer a direct trigger to IBM Openwhisk as of now.

Trigger Comparison

In this section, we measured a trigger throughput to describe how many event messages that they can process at a time. Certain triggers e.g. Timer are not meant to deal with concurrent messages thus we chose three types of triggers across serverless providers; HTTP, database and object storage triggers. In Figure 1,we find that the median throughput of the HTTP trigger is 55.7 functions per seconds and the object storage has the 25.16 functions per seconds median throughput. The AWS Lambda Database trigger has throughput of 864.60 function per second which is approx. 32 time of object storage and approx. 15 times of HTTP trigger. It is understandable because the database trigger in AWS is configurable to add more database nodes in response to the number of event messages. The reason for this is that there is huge gap between the number of functions invoked per second among different trigger types. As of now we cannot compare Azure and Google Cloud database trigger as they do not have a direct trigger available to their respective functions.

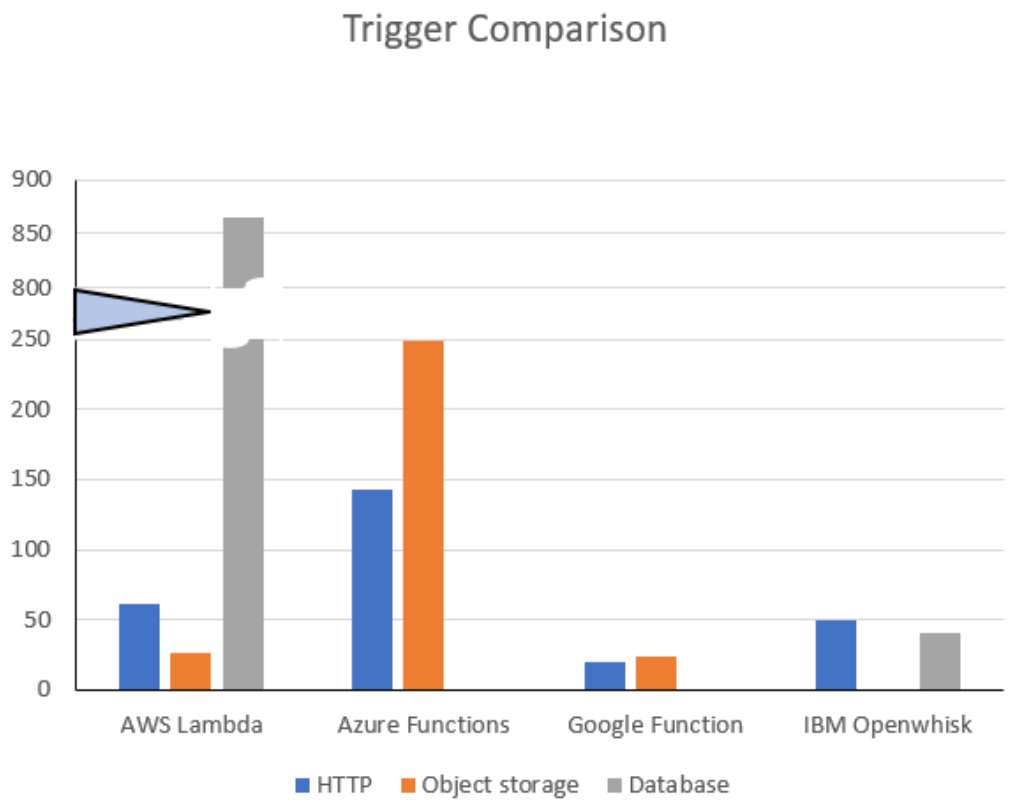

Figure 1: Trigger comparison among cloud providers

Feature comparison among different cloud functions

Table 1, AWS Lambda 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 terms 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.

Item	AWS Lambda	Azure Functions	Google Functions	IBM OpenWhisk
Runtime languague	Node.js, Python, Java, C#	C#, F#, Node.js, Java, PHP, Python	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,mCloudant)
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	Not applicable
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
Container 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	5 mins	5 mins
Code size limit	512 MB	16MB	100MB (compressed) for sources. 500MB (uncompressed) for sources plus modules	48 MB
API references	cli tool	.NET, python,node,java,ruby, rest	gcloud CLI tool, rest api, rpc api	cli tool

Table 1: Comparison of Features

IaaS vs FaaS Pricing information:

People who are new in the FaaS environment may get confused between functionality of IaaS and FaaS. The underlying concept states is that FaaS is supported by the containers which get started and killed many times during a cloud function lifecycle. We conducted an experiment by deploying in-house Apache Openwhisk. We found that the containers also maintain lifecycle that is instantiation, pause and killed. If a new function is invoked within a particular timeframe with respect to previous invocation, then the same container is invoked, which is currently in running state. This save the computation time. This is often term as hot start of function.

Next, if the container is ideal for a long time which means no function invocation (new function or the same old function) is happening within a certain timeframe (we analyzed 3 mins), the container goes to paused state for a fixed timeframe. If a function is invoked (different function or the same previous

function) within a specific period while the container is in paused state, then same container will get activated from paused to running state and then take the request of deploying and running the function. This may cost some more microseconds for the underlying container to come from paused to running state. But, if a good amount of time has passed by and no function got invoked while the container is paused state then the container is killed. If new function call happens after that then the new container will be instantiated and will serve the function invocation which may take more than the previous two cases. This is called cold start of function.

Whereas the IAAS works on the VMs. there is no concept of autoboot and auto killing of VMs. The disadvantage of VMs is that we need to keep them running irrespective of work is available or not. Most of the time the VMs are running ideally without doing any work and costing good amount of money to the customer. Also, the disadvantage is the compute resource like VMs are changed per hour, so if a work/process which gets invokes not very frequently and needs only seconds of compute time, VMs are not ideal scenarios and there FAAS has always has upper hand in this case.

We did a price comparison between IaaS and FaaS because it is unclear whether invoking FaaS is cost effective over running a VM instance. The charging unit is different, FaaS is based on 100 milliseconds per invocation and IaaS is based on an hour per a VM instance but when we break down the cost in a second, FaaS is not always cheaper than IaaS. We ran implementation of creation of binary trees of 18 depths and ran in different FaaS and IaaS platforms among AWS, IBM, Azure and Google cloud. We calculated the price per executing this binary tree creation program in each of the IaaS and FaaS.

Table 2 shows the execution time of the creating binary trees and the total cost with the rank ordered by cost effectiveness.

Platform	RAM	Cost/Sec	Elapsed Second	Total Cost (Rank)
AWS Lambda	3008 MB	$4.897e-5	20.3	$9.9409e-4 (6)
AWS EC2 (t2.micro)	1GiB	$3.2e-6	29.5	$9.439e-05 (3)
Azure Functions	192MB	$3e-6	71.5	$2.145e-4 (4)
Azure VM	1GiB	$3.05e-6	88.9	$2.71145e-4 (5)
Google Functions	2GB	$2.9e-5	34.5	$0.001 (7)
Google Compute (f1-micro)	600MB	$2.1e-6	19.2	$4.0319e-05 (1)
IBM OpenWhisk	128MB	$2.2125e-6	34.2	$7.5667e-05 (2)

Table 2: Building Binary Tree with cost awareness

Language Comparison:

A function in serverless computing has a writable temporary directory with a small size (e.g. 500MB) but it is useful for various purposes, for example, extra libraries, tools and intermediate data files can be stored while a function runs. We again ran a simple write/read function with a file size of 100MB and 400MB over 100 times in each serverless provider. The measured I/O performance toward a temporary directory is shown in Figure 2, Figure 3. Google Functions shows good performance in both writing and reading files because it consumes the temporary directory in memory. Reading speed in AWS Lambda is the most competitive among others although its writing speed is the opposite. IBM shows some variant, we may conduct other set of tests to get more accurate results in writing and reading files. The median speed of the test results is available in the Table 3.

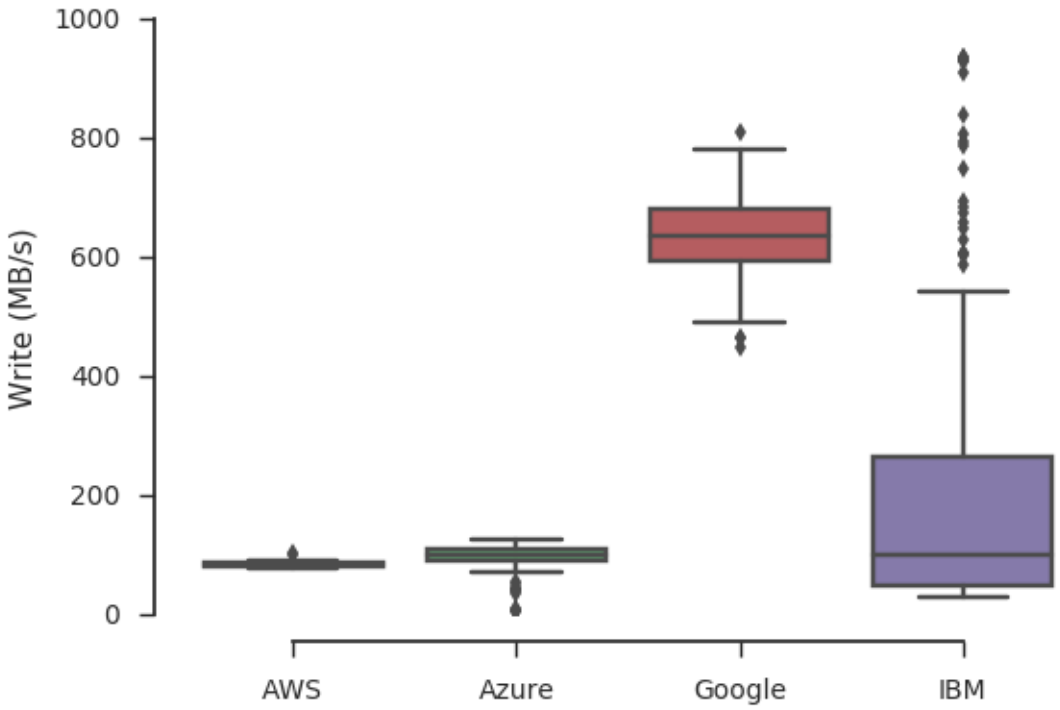

Figure 2 : Write Speed in Temp Directory

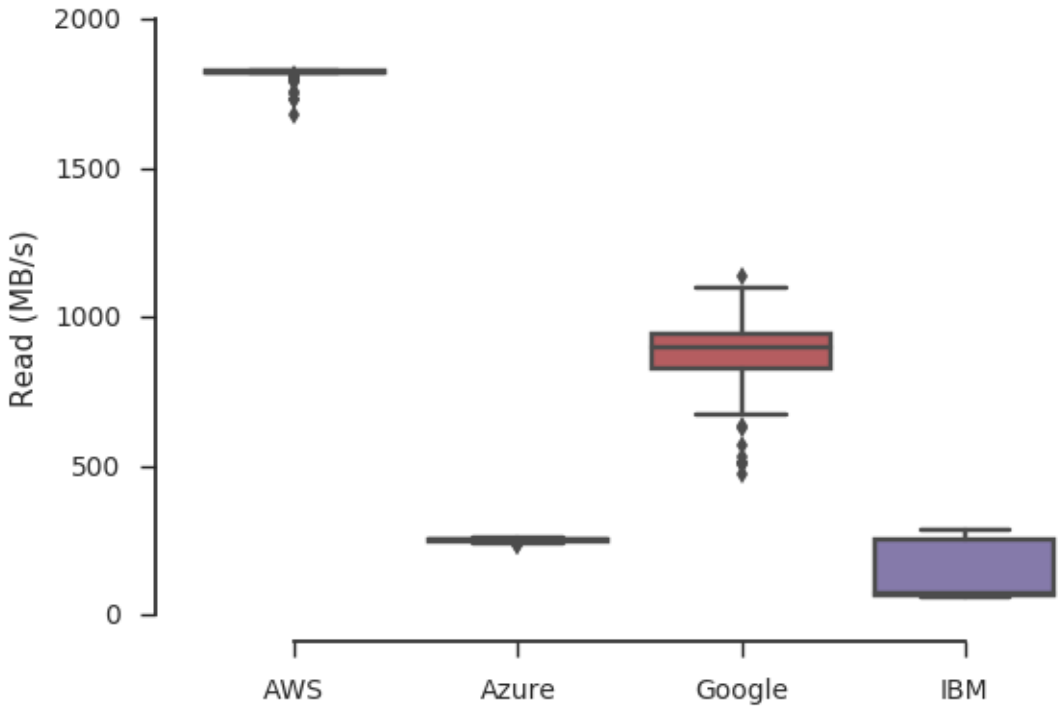

Figure 3 : Read Speed in Temp Directory

Provider	Write (MB/s)	Read (MB/s)
AWS	84.816877	1822.876337
Azure	99.601698	250.941013
Google	636.314804	897.581341
IBM	98.951235	73.074927

Table 3: Median Write/Read Speed

Use Cases

There are several areas where serverless can play an important role in research applications as well as in commercial cloud. Serverless map-reduce can be used to execute big data map-reduce jobs in a more tolerant and more cost-effective way (Sunil Mallya, 2017). Also, the commercial cloud backend logic can be run on serverless and persisting data in the cloud hosted database.

Image processing for CDN is also widely used by commercial purpose to process thumbnails of the images to client such as mobile and tablets which can be taken care by serverless. IOT is also one of demanding use case for serverless. IOT devices will trigger the lambda function using a rule.

For example, in case of a data-center, cooling facility is very important for proper functioning of servers. If the cooling of the datacenter is down, the sensors will call a lambda function which will contain the logic sending alerts to the support team. In near future, we will see a huge number of use-cases as serverless would be adopted as mainstream cloud based development.

Conclusion:

Adoption for serverless is happening at a very fast pace due to its event driven and simple programming management platform without thinking about the infrastructure/server. The performance and cost are the important factors behind the adoption. Many more research is expected to happen majorly driven by big cloud providers and universities. With this huge demand, we can expect a paradigm shift on how application would be developed in near future.

Bibliography

Fox, Geoffrey C, Ishakian, Vatche , & Muthusamy. (n.d.). Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research. 2017.

Sunil Mallya, A. (2017, 09/ 11). Retrieved from https://aws.amazon.com/blogs/compute/ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce