Serverless technologies comparison
Large Systems project

Sean Liao and Mar Badias

December 21, 2019

Abstract

Write abstrat here

1 Introduction

Public clouds are growing, and with it comes the latest push into serverless offerings. These come in many forms depending on the abstraction level, but they can largely be grouped into: long-lived containers, short-lived containers, functions as a service.

Long-lived containers or Platform as a Service (PaaS) represent the first generation of serverless technologies. These can be full-fledged, stateful applications, packaged in containers. The clouds will take these and run them for you on VMs. Auto-scaling and load balancing is usually offered, but fast startup times are not guaranteed. These should be considered an alternative UI to the underlying VMs, which will be reflected in the pricing model (charge for underlying VMs). Examples: AWS Elastic Container service, GCP App Engine, Azure App Services, Alibaba Container Service.

Functions as a Service (FaaS), currently the highest level of abstraction and the second generation of serverless technologies. Developers provide their application code for the clouds to compile, package, deploy, and run. These are short-lived and stateless, an instance may be started for every request and killed after it completes. Billing is only for the time it is running serving a request. Examples: AWS Lambda, GCP Cloud Functions, Azure Functions, Alibaba Function Compute, IBM Cloud Functions, Zeit Now.

Short-lived containers or Containers as a Service (CaaS). Is the newest technologies short-lived, stateless runtimes and a similar billing model. Where they differ from FaaS is that they introduce containers, giving developers control of the execution environment, allowing them to run languages or runtimes unsupported by FaaS. Examples: AWS Fargate, GCP Cloud Run, Azure Container Instance, Alibaba Elastic Container Instance.

insert why people use serverless.

Our reaserch will measure and analiyse its CPU performance, overhead and specially their capacity to scale when the workload increases. Scaling entails creating more instances, known as cold starts, and assigning new resources between them without influencing others' instances performance.

2 Related work

Serverless technologies have been subject of previous research. We can find good examples of PaaS analysis in [1] and [2], where the reaserchers analize Google App Enginee performance with CPU-intensive applications. CaaS have also been a topic of study, comparing them to long live containers or performance test among of different provides. We can find examples of this in [3] and [4]. Comparison between Faas and short-lived containers has also been analized but from a functionality point of view but oversimplificating it and not taking into account the performance [5][6][7].

FaaS on its own also has been subject of previous research. An excellent example is [8] where multiple serverless providers are continuously been benchmarked. Another example of a comparison of Faas providers is [9]. In [**?**] the researches dicuss the advantages of using cloud services and AWS Lambada for systems that require higher resilance. Finally, is necessary to mention [**?**], which fouses in its performance evaluation and also benchmarking the data transfer to storage and its lifetime of the major cloud functions providers.

Aldought serverless performance has been heavily analized, to the best of our knowledge their scaling and its consequences to performance and overhead have not been comprehensively analysed yet, which motivates our research.

3 Research question

- **How do different platforms compare when with sudden spikes in traffic?**

Specifically we will be looking at:

- **Does scaling out affect the performance of the services?** When scaling out to meet demand, serverless platforms should isolate instances to provide a consistent level of performance.
- **Does scaling out affect platform overhead** Everything has a cost, what we hope to see is that scaling out does not severely degrade the platform's performance, in terms of instance management.

4 Methods

Given the amount of services of this type in the current market we defined a selection criteria based on the most popular features [10] of serverless computing: dynamically managed runtimes, globally reachable HTTP(S) endpoint, pay only for usage and autoscaling.

Having defined these characteristics, we selected the products that provide them offered by the top 5 cloud providers as of 2019 [11]: Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, Alibaba Cloud, and IBM Cloud. Additionally, we also included Zeit, a startup in the FaaS space popular for its streamlined experience. The products selected can be found in Table 1.
As mentioned in section 1, the the pricing model for PaaS (charge for underlying VMs) does not fit in the 'Pay only for usage' defined in our selection criteria. However GCP App Engine offers the possiblitily to scale your applications to 0 when it is not been used so it fits into our selection criteria.

Our choice of workload was an image resizing service implemented in Python, a real world [12] usecase and an often used example in what FaaS solutions are good for. This is a more CPU intensive workload that doesn't rely on external services.

Type	Platform	Product	Web Framework	Instance Size	Pyhon runtime	Location of the DC
FaaS	AWS	Lambda	Custom	128 MB	3.8	London, UK
FaaS	GCP	Functions	Flask	128 MB	3.7	St. Ghislain, BE
FaaS	Azure	Functions	Custom	128 MB	3.7	NL
FaaS	IBM Cloud	Functions	OpenWhisk	128 MB	3.7	London, UK
FaaS	Alibaba Cloud	Function Compute	Custom	128 MB	3.6	Frankfurt, DE
FaaS	Zeit	Now	Python http.server	128 MB	3.6	Brussels, BE
CaaS	GCP	Cloud Run	Python http.server	128 MB	3.8	St. Ghislain, BE
PaaS	GCP	App Engine	Flask	128 MB	3.8	St. Ghislain, BE

Table 1: Serverless services that will be tested in this reaserch.

We aimed to use the same code (excluding API adapters) for all platforms, running on the highest version of Python 3 available. We used the datacenters closest to Amsterdam, with machine types configured with 128MB of memory where available.
Based on prior research machine size should not affect platform overhead.

Our server code accepts HTTP requests with an image, resizes it, and responds with the resized image. It also returns the time spent processing the image, and a unique identifier generated on startup. Our client code has a 2 phase testing cycle run every hour: phase 1 (which will be referred single test) sends 10 requests sequentially, phase 2 (which will be referred multi test) starts 50 workers in parallel to each send 10 requests sequentially. For each request we recorded the total roundtrip time, in addition to the metadata returned by the server, We aimed to use a stable set of images with similar distributions of file sizes for both phases.

5 Results

We ran our test over the course of 4 days (including weekdays and weekends). In table 2 and table 3 we can find a summary of the multi and single phases of the test performance and the data gathered, which will be discussed in section 6.

Looking into the success rate of our requests, we see some transient errors whose effect we consider negligible. However, Alibaba Cloud had a significantly elevated error rate, which we will discuss later. It is also important to mention that we did not observe any variations based on the time of day (see figure 1).

Service	**Total request**	**Success rate**	**Cold starts ratio**	**Parallel instances**	**StDev (parallel inst.)**
AWS Lambda	47000	1	0.097	50.402	3.858
GCP Functions	46972	0.999	0.062	36.348	6.231
Azure Functions	46997	1	0.005	3.598	0.680
IBM Cloud Functions	47000	1	0.097	50.685	5.256
Alibaba Cloud Function Compute	43501	0.926	0.169	81.043	4.427
Zeit Now	46995	1	0.08	41.663	4.962
GCP Cloud Run	47000	1	81.163	33.575	
GCP App Engine	47000	1	0.049	29.402	5.985

Table 2: Results multi test.

Service	**Total request**	**Success rate**	**Cold starts ratio**	**Parallel instances**	**StDev (parallel inst.)**
AWS Lambda	940	1	0.098	1.022	0.209
GCP Functions	940	1	0.027	1.022	0.209
Azure Functions	940	1	0.099	1.022	0.209
IBM Cloud Functions	940	1	0.099	1.022	0.209
Alibaba Cloud Function Compute	940	1	0.098	1.022	0.209
Zeit Now	940	1	0.098	1.022	0.209
GCP Cloud Run	940	1	0.098	1.011	0.104
GCP App Engine	940	1	0.001	1.011	0.104

Table 3: Results single test.

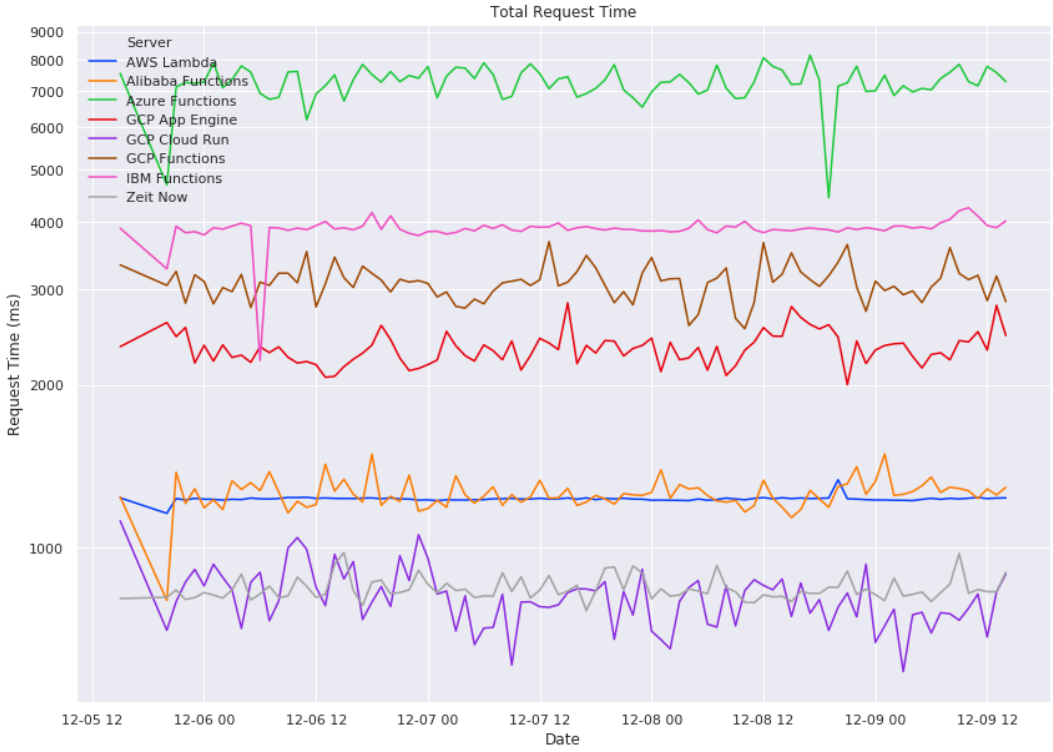

Figure 1: Performance over time of all services tested.

5.1 Compute performance

In Figure 2 we see the cumulative frequencies for the image processing time as measured by the server, split by concurrency level and cold/warm starts.

5.2 Overhead

In Figure 3 we see the cumulative frequencies for the overhead time (total roundtrip time - server processing time) split by concurrency level and cold/warm starts. With all else being equal, we atrribute the difference to time spent conducting a cold start.

6 Discussion

6.1 Scaling

From Table 2 we observed elevated failure rates for Alibaba Function Compute. Their documentation [13] states they have a limit of 50 instances per service, and even though our client places a strict limit of 50 inflight requests at any point in time, their scheduler might not be keeping up in either reusing an instance or cleaning it up before the next request arrives.

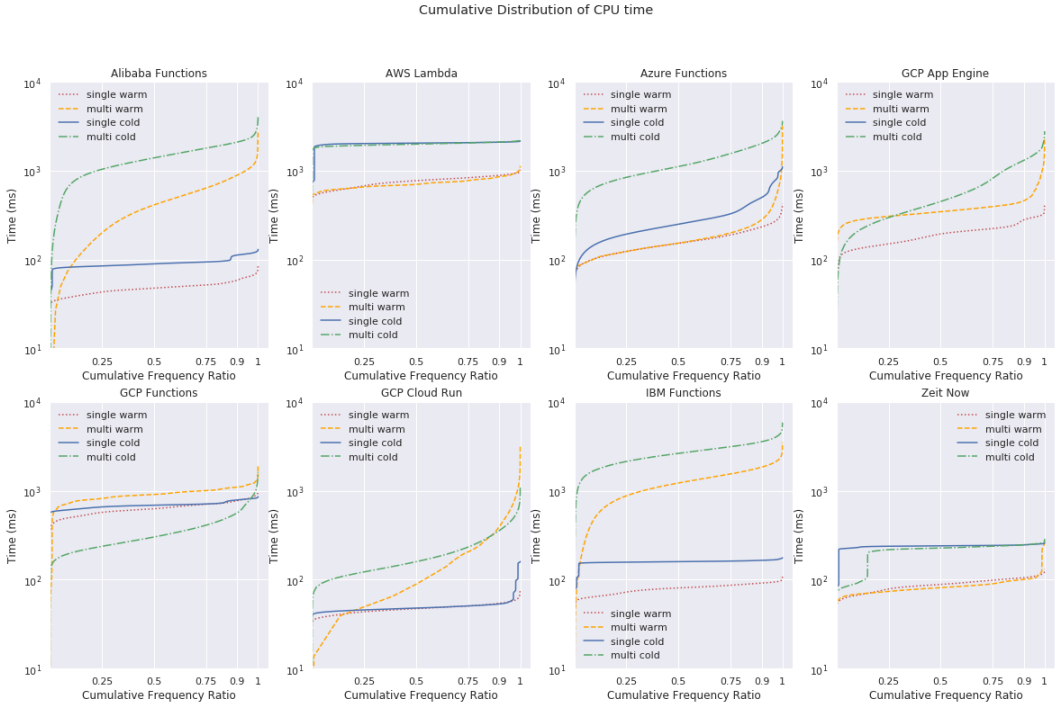

Figure 2: Acumulative distribution of CPU time.

Also from Table 2 we can see the average number of unique instances per test cycle. Ideally, this number would be 50, to match the number of parallel requests we are sending.
On the low end, Azure Functions is notable for only starting 4 instances. This may be from a documented limitation [14] of only starting at most 1 instance per second with HTTP triggers. Their load balancers still hold the requests in queue, but they appear to be reluctant to start new instances even as sufficient time has passed.
At the high end, we configured GCP Cloud Run to accept up to 5 concurrent requests per instance, expecting it to utilise the available memory and CPU more efficiently as this was one of its main selling points. However, their scheduler also took CPU utilitzation into account, limiting the usefulness of setting a higher concurrency level with our CPU intensive task. In hindsight, setting the concurrency level to 1 would have been a fairer comparison.
From Figure 2 we can see for AWS, Azure, Zeit, and GCP Functions, we saw good isolation of workloads, they performed similarly for warm requests at both concurrency levels. GCP Cloud Run saw an expected degradation in performance from handling multiple requests, While Alibaba and IBM were serverely impacted.

6.2 Warm vs Cold Starts

An often cited concern of using truly on-demand compute services is the cold start, when the platforms have to start up a new instance to handle increases in load. This is such a concern that some some platforms have specific features built to keep your instances

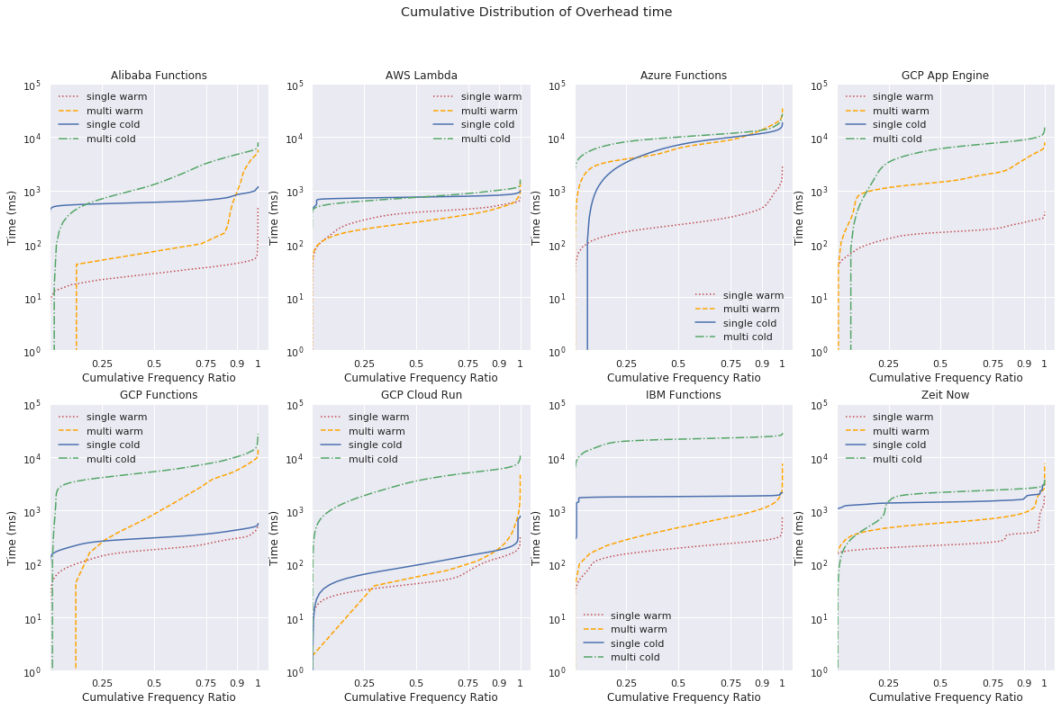

Figure 3: Acumulative distribution of overhead time.

warm, such as AWS Provisioned Concurrency [15] and Azure Functions Premium Plan. Our results in Figure 3 show that, as expected, in most cases, single concurrency cold starts are slower than warm starts and the start times are highly consistent. Cold start times are much more variable at higher concurrency levels, with the exception of AWS Lambda which appears unaffected. We suspect this could be due to having the system image on disk, which is reusable as long as instances are on the same machine, vs needing to retrieve the image over a network, but we don't have a solid way of testing this.
What is unexpected is in Figure 2, that that cold starts affect CPU performance. These instances consistently perform worse on a cold start, even as they are reused for future requests. Further testing would be needed to identify the root cause of this.

6.3 Technologies

GCP App Engine is the oldest technology being compared. Under the hood it appears to be an NGINX reverse proxying Python apps through Gunicorn. Its scaling is controlled by CPU utilization, and in our case a heavy workload pushes that up and limits the concurrent requests a single instance can handle. While it can scale to zero instances, each instance has a high (15 minute) startup fee [16] resulting in higher costs for short spikes in traffic.
The current generation of FaaS are built on a wide variety of technologies. AWS and GCP have publicly stated that they use their own VMs, Firecracker and gVisor respectively, to isolate workloads. The results from Figure 3 show they provide a highly consistent

environment even under load. Zeit Now uses AWS datacenters [17] for their serverless offering. They behave similarly to AWS Lambda and we believe that is what they use, but with the largest machine type. Alibaba Functions, Azure Functions, and IBM Functions all appear to be implemented on containers, as either their developer tooling or documentation hint at the possiblitily of customizing the runtime in not very well documentated ways. Both Alibaba and IBM exhibit similar characteristics of degraded performance under load, possibliy from poor isolation between workloads.
GCP Cloud Run is a fully managed container runtime implementing the Kubernetes Serving API, and the only service we tested that exposed the Docker runtime directly. As expected, single concurrency performance is consistent but drops when more requests are routed to a single instance. As the industry moves to converge on Kubernetes as a common interface and runtime, we expect more fully integrated products in this space in the future. While other platforms provide container runtimes or hosted Kubernetes, they are at a lower level in the stack and don't provide the full functionality we were looking for.

7 Conclusion

References

[1] Prodan R. Sperk M. Ostermann S. Evaluating High-Performance Computing on Google App Engine. IEEE Software (Volume: 29, Issue: 2, March-April 2012).

[2] M. Wojcik P. BubakM H. Malawski, M. Kuzniar. How to Use Google App Engine for Free Computing. IEEE Internet Computing (Volume: 17, pp. 50-59, 2013).

[3] Michael Wittig. ECS vs. Fargate: What's the difference? `https://cloudonaut.io/ecs-vs-fargate-whats-the-difference/`, 2019.

[4] Y.C. Tay ; Kumar Gaurav ; Pavan Karkun. A Performance Comparison of Containers and Virtual Machines in Workload Migration Context. `https://ieeexplore.ieee.org/document/7979796`.

[5] Mike Chan. Containers vs. Serverless: Which Should You Use, and When? `https://www.thorntech.com/2018/08/containers-vs-serverless/`.

[6] Philipp Muns. Serverless (FaaS) vs. Containers - when to pick which? `https://serverless.com/blog/serverless-faas-vs-containers/`.

[7] Chad Arimura. Functions vs Containers. `https://medium.com/oracledevs/containers-vs-functions-51c879216b97`.

[8] Bernd Strehl. Serverless Benchmark. `https://serverless-benchmark.com/`.

[9] Maciej Malawski; Kamil Figiela; Adam Gajek; Adam Zima. Benchmarking Heterogeneous Cloud Functions. `https://www.icsr.agh.edu.pl/~malawski/CloudFunctionsHeteroPar17InformalProceedings.pdf`.

[10] CloudFlare. What Is Serverless Computing? — Serverless Definition. https://www.cloudflare.com/learning/serverless/what-is-serverless/.

[11] Larry Dignan. Top cloud providers 2019: AWS, Microsoft Azure, Google Cloud; IBM makes hybrid move; Salesforce dominates SaaS. https://www.zdnet.com/article/top-cloud-providers-2019-aws-microsoft-azure-google-cloud-ibm-makes-hybrid-move-salesf

[12] Amazon Web Services. Square Enix Case Study. https://aws.amazon.com/solutions/case-studies/square-enix/.

[13] Alibaba Cloud Documentation. Limits. https://www.alibabacloud.com/help/doc-detail/51907.htm.

[14] Microsoft Azure Documentation. Azure Functions scale and hosting. https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale.

[15] AWS Lambda. AWS Lambda announces Provisioned Concurrency. https://aws.amazon.com/about-aws/whats-new/2019/12/aws-lambda-announces-provisioned-concurrency/.

[16] Google Cloud Guides. How Instances are Managed. https://cloud.google.com/appengine/docs/standard/python/how-instances-are-managed.

[17] Zeit Now. Regions and Providers. https://zeit.co/docs/v2/network/regions-and-providers/.