Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

LETTER

# HPC in the Cloud: Performance Comparison of Function as a Service (FaaS) vs Infrastructure as a Service (IaaS)

Sulav Malla | Ken Christensen

Department of Computer Science and Engineering, University of South Florida, Tampa, Florida, USA

**Correspondence**
Sulav Malla, 4202 East Fowler Avenue, ENG 030, Tampa, Florida 33620. Email: sulavmalla@mail.usf.edu

High performance computing (HPC) in the cloud has been shown to be a viable alternative to on-premise clusters, especially for loosely coupled or embarrassingly parallel jobs. The traditional approach for a user would be to use the cloud provider's IaaS to provision virtual machines (VM) and use it in a similar manner to an on-premise cluster. A new paradigm of serverless cloud computing, primarily offered as FaaS, allows a user to execute code in the cloud without any system administration overhead. A comparison is needed to better understand the cost versus performance trade-off between FaaS and IaaS such that cloud users can decide which approach is suitable for them, however, such a study is lacking. In this paper, we compare Google cloud's FaaS (Cloud Functions) with its IaaS (Compute Engine) in terms of cost and performance for an embarrassingly parallel task. We find that FaaS can be 14% to 40% less expensive than IaaS for the same level of performance. However, performance of FaaS exhibits higher variation since the number of CPUs allocated (scalability) depends on the cloud provider.

**KEYWORDS:**
Cloud computing, serverless computing, high performance computing, Function as a Service, Infrastructure as a Service, Google cloud platform

## 1 | INTRODUCTION

High performance computing (HPC) typically requires a specialized cluster built specifically to serve scientific batch workloads. HPC users can also leverage the computing offered by the cloud to supplement on-premise clusters during peak usage[1] and various studies on the feasibility of HPC in the cloud have concluded that embarrassingly parallel jobs are a good candidate for the cloud[1,2]. In the Infrastructure as a Service (IaaS) model, HPC users can provision on-demand virtual machines (VM) in the cloud without having to spend significant time procuring and setting up physical servers. However, in IaaS, HPC users are responsible for all the system administration tasks, such as, maintaining and updating VMs, scaling VMs according to the workload, and monitoring and fault tolerance of VMs. The time and effort required to setup the virtual resources can be greater than the actual time and effort spent doing the computations[3]. Serverless computing is a new paradigm of cloud computing where the cloud provider handles all of the server administration tasks, relieving the users from tedious resource management responsibility altogether. An example of serverless computing is Function as a Service (FaaS), which allows a user to write code in programming language of their choice and run it on the cloud without having to first provision a VM. Elasticity, availability, scalability, and fault tolerance, are transparently provided by the cloud provider. FaaS based implementations can reduce complexity and maintenance overhead[4] for HPC users, which can lead to wider adoption of cloud computing for HPC workloads[3]. FaaS is a true pay-as-you-go model where users pay only for the amount of time their code is executed rather than paying for the amount of time VMs are provisioned (even when idle), as in the case of IaaS. Furthermore, FaaS also benefits the cloud provider as well since they have a more granular control over the service provided.

Studies have been published in recent years[3,5,6], that have explored and evaluated the performance of using the FaaS approach for HPC workloads. While these studies demonstrate that FaaS is easy-to-use, inexpensive, and suitable for HPC workload, the effectiveness of FaaS over the conventional IaaS approach have not been quantified. Some HPC cloud users might be constrained by cost while others may be willing to pay a premium for better performance. Hence, a detailed comparison of cost and performance between FaaS and IaaS is necessary, and our paper is a step in this direction. The main contribution of this paper is:

- We perform a preliminary comparison, in terms of cost and performance, between using FaaS and IaaS approaches for an embarrassingly parallel HPC workload.

## 2 | RELATED WORK

The Magellan report[7], funded by the U.S. Department of Energy, was one of the first comprehensive feasibility study of the use of cloud computing for HPC workload. The report, back in 2011, found that cloud computing was expensive, performed very poorly for tightly coupled applications, and required considerable system administration overhead, compared to their on-premise HPC clusters. Marathe et al.[2] challenged this position saying that, while on-premise clusters may be faster than cloud VMs in terms of raw computation and inter-node communication latency, on-premise clusters have significant queue wait time which should not be ignored. They argued that the turnaround time (time from job submission to job completion) could be better in the cloud due to practically no queue wait times for the jobs. Much has changed since these studies. The types and number of services offered by cloud computing along with their adoption has seen a meteoric rise. It is estimated that more than 50% of global enterprise using some form of cloud computing today will fully adopt it by 2021[8].

Serverless computing, similar to fog computing, edge computing[9], and vehicular cloud computing[10], presents a fundamental shift in the cloud computing domain. It transfers all the system administration responsibilities from the user to the cloud provider[9] while still offering all the benefits of cloud computing. Jonas et al.[11] compare this shift to serverless computing as analogous to the transition from an assembly-level programming to a high-level programming for programming languages. Lynn et al.[4] reviews seven different enterprise serverless cloud computing including, AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions, to conclude that serverless computing can be applied to a wide range of use cases. Spillner et al.[5] successfully demonstrated that FaaS model of cloud computing can be used for different HPC batch workloads, such as, calculating the value of $\pi$, image face detection, password cracking, and weather forecasting. Similarly, Lloyd et al.[12] investigated the performance variation of FaaS along various dimension using a synthetic workload, while Kijak et al.[13] outlined challenges of using FaaS for HPC workloads. Niu et al.[3] (an extension of Kumanov et al.[6]) and Lee et al.[14] showed how FaaS provides a simple and scalable, yet inexpensive, solution for embarrassingly parallel HPC jobs. All previous works have evaluated IaaS or FaaS models of cloud computing in isolation. To the best of our knowledge, our work is the first to compare IaaS to FaaS in terms of cost and performance for an HPC workload.

## 3 | METHODOLOGY

An embarrassingly parallel HPC workload will generally have multiple tasks that can run independently. In this section, we describe the FaaS and IaaS services offered by Google cloud platform that can be used to run such independent tasks.

### 3.1 | Google Compute Engine

The Google Cloud Platform (GCP) offers the traditional IaaS primarily as *Google Compute Engine* (GCE). GCE are VMs that users can provision by specifying its type, for example, general-purpose, memory-optimized, or compute-optimized, each of which is backed by a different hardware type. Users can also choose the number of CPU (a single hardware hyper-thread) and amount of memory for the VM instance from a pre-determined range. For our experiments, we use the general-purpose n1-standard-64 VMs with 64 CPUs and 240GB memory. Several VM instances can be combined to form a computing cluster for which we used SLURM[15] to coordinate and schedule jobs to the cluster. For example, if we need 512 CPUs for an experiment, we would build a SLURM cluster of eight n1-standard-64 VMs. We would also need two additional low end VMs (for example, n1-standard-2), one as a controller node and another as a login node, the cost of which is negligible compared to the cluster. When an HPC job is submitted, SLURM schedules the tasks to available CPUs. If the number of tasks is greater than the CPUs

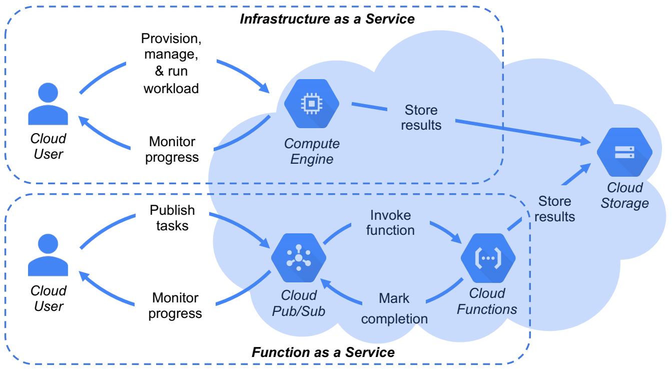

**FIGURE 1** Block diagram showing the different services of Google cloud platform when used as Infrastructure or Function as a Service.

available, the task is queued until a CPU is freed. The output of the computation is uploaded to the *Cloud Storage* by each VM, as shown in Figure 1, since it is the most scalable and reliable storage options available.

### 3.2 | Google Cloud Function

GCP offers FaaS in the form of *Google Cloud Function* (GCF). In GCF, the user writes code in their language of choice (JavaScript, Python, and Go are supported as of this writing) and deploys it in the cloud. The cloud function is executed in response to a triggering event such as a HTTP request, an API call, or specific cloud events chosen by the user. We found *Cloud Pub/Sub*, a message passing service, to be a scalable solution to invoke hundreds of concurrent functions almost instantaneously. The message we pass can be considered as parameters passed into the function. For example, for each task, we pass a message with the name of input files, the name of the output file, and task number to invoke a function. The function performs the computation on those input files (which may be in Cloud Storage or local disk), store the results to Cloud Storage, and send back a message via Cloud Pub/Sub to the user signaling the task completion. This workflow is illustrated in Figure 1. *Function instances* run in an isolated executable environment and currently have the following limits: function memory up to 2048MB, execution duration up to 9 minutes, and one CPU up to 2.4 GHz. Users can choose function memory as well as execution timeout during deployment (CPU frequency increases with increasing function memory) and are charged only for the time function instances are executing. Compared to GCE, we cannot define the number of CPUs in GCF. We simply invoke a function for each task and let GCP (the cloud provider) allocate appropriate function instances and carry out the computation. We neither provision, nor manage the computing resources as done in the IaaS approach, which further simplifies our workflow.

## 4 | EXPERIMENT AND RESULTS

We have an embarrassingly parallel HPC workload, as described below, that we run for different GCE and GCF configurations. We repeat each experiment 10 times and report the average (with 95% confidence interval for bar graphs).

### 4.1 | HPC workload

The HPC workload we choose for evaluation is an all-against-all pairwise comparison of human proteins using a dynamic programming algorithm to compute similarity between two protein sequences, as implemented by Kumanov et al.[6] and Niu et al.[3]. The dataset consists of 20,336 unique human protein sequences split into 41 files (each with about 500 proteins). Pairwise comparison of these 41 files, including comparison with itself, results in 861 tasks in total. Each task involves taking two input files and comparing the protein sequence similarity between each protein sequence in one file to every protein sequence in the other file. The resulting protein sequence similarity scores are written to the output file and uploaded to Cloud Storage. The 861 tasks are independent of each other since no result communication is required between the tasks, giving rise to an embarrassingly parallel HPC workload. Interested readers can find the code, dataset, and implementation details in the GitHub repository by Kumanov et al.[6] (https://github.com/BioDepot/TaskPerform_AWSLambda), from which we adapted for Google cloud platform.

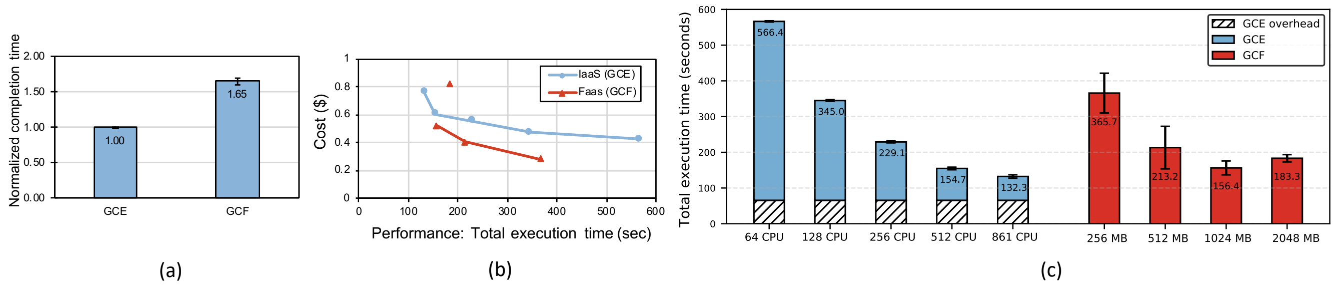

**FIGURE 2** (a) Normalized single task completion times (average of 10 runs) on Google Compute Engine (GCE) and Google Cloud Function (GCF). (b) Performance and cost comparison between Infrastructure as a Service (GCE) and Function as a Service (GCF). (c) Performance comparison between Infrastructure as a Service (GCE) and Function as a Service (GCF).

## 4.2 | Raw computation

To compare the single node raw computing power, we ran one task on a single CPU (1 CPU with 2048 MB memory) for both GCE and GCF. Ignoring any setup overhead time, we measure the time from the start to end of the computation. The computation took 22.4 seconds on GCE while the same single task computation, under similar setting, took 36.9 seconds on GCF. As shown in Figure 2 (a), we found that a single node of GCE is 1.65 times faster than that of GCF. Furthermore, performance variation is higher in GCF compared to that of GCE, denoted by wider confidence intervals in Figure 2 (a). The difference in performance is primarily due to a difference in CPU platform, CPU clock frequency, and virtualization technology used for GCE and GCF.

## 4.3 | Total execution time

We measure the *total execution time* as the time from when a job (with 861 tasks) is submitted to the time when all of the 861 tasks are completed. We repeated the entire experiment with 64, 128, 512, and 1024 number of CPU for GCE and with 256 MB, 512 MB, 1024 MB, and 2048 MB function memory for GCF (since this is the only parameter we can vary).

**Setup overheads**: While GCF function invocation (equivalent to job submission) using Cloud Pub/Sub happened almost instantaneously, it took some time for the function instances to actually materialize and start the computation. Function instances are assigned and scaled dynamically by the cloud provider and the user has no control over it. This function instance overhead time (from function invocation to the start of function instance) ranged from a couple of seconds to more than 30 seconds, and different function instance experienced different overhead time. Hence, the total execution time for GCF intrinsically captures this dynamic setup overhead time for each task. On the other hand, in the case of GCE, once we provision the VMs, the job starts executing as soon as it is submitted. However, GCE also has some setup overhead time when we initially request VMs from the cloud before we can start using them. For a fair comparison of performance between GCE and GCF we must also consider the setup overhead time for GCE. We found this setup overhead to be 65.5 seconds, on average, and was consistent regardless of the number of VMs requested. We add this fixed overhead time to the GCE total execution time for a fair comparison with GCF.

Total execution time of completing 861 tasks for different GCE and GCF configuration is shown in Figure 2 (c). For GCE, we varied the number of CPU allocated, from 64 CPUs up to the highest allocation of one CPU per task (we actually provisioned 1024 CPUs but only 861 were used since we had only 861 tasks). The computation times almost halve as the number of CPU is doubled, typical of an embarrassingly parallel workload. In the case of GCF, we invoke 861 functions (a function for each task) and varied the function memory allocation during deployment from 256 MB to 2048 MB. The total execution times generally decreased as the function memory was increased, as shown in Figure 2 (c), since the CPU frequency increases for higher memory request. To our surprise, the GCF performance decreases when we moved from 1024 MB to 2048 MB. Upon further investigation, we found that the number of function instance (equivalent to CPU allocated for the experiment) varied between runs. We tagged each function instance in an experiment with a universally unique identifier (UUID) by writing to its temporary file system. Ideally, we would have wanted 861 function instances to process the 861 tasks in parallel. However, we found that the number of unique function instances was less than 861 and some tasks were being queued only to be started once an executing task is complete (i.e., function instances were being reused). We observed that about 600 function instances were generally allocated for the GCF experiments. However, for the 2048 MB experiment, the number of function instances allocated dropped significantly to about 400. This caused the execution time to increase although CPU frequency and memory increased.

**TABLE 1** Cost breakdown for GCE and GCF.

| | **GCE** | **GCF** |
|---|---|---|
| CPU | \$0.031611 per CPU-hour | \$0.036 per GHz-hour |
| Memory | \$0.004237 per GB-hour | \$0.009 per GB-hour |

## 4.4 | Cost calculation

Looking at the billing invoice for our experiments, we found that we were primarily charged for computation (CPU and memory), while the storage and message passing fees were negligible or even free due to free usage limits. Hence, we only take the computation cost into account for comparison purpose, as shown in Table 1. For example, for GCE with 64 CPU which takes 500.94 seconds (without setup overhead) to complete the job, the total cost is calculated as 64 CPU * (\$0.031611 per CPU-hour + 3.75 GB per CPU * \$0.004237 per GB-hour) * (500.94 seconds / 3600 seconds per hour) = **\$0.423**. Similarly, for GCF with 1024 MB function memory which takes 31441.95 seconds of total function execution time (sum of each function execution), the total cost is calculated as (1.4 GHz * \$0.036 per GHz-hour + 1 GB * \$0.009 per GB-hour) * (31441.95 seconds / 3600 seconds per hour) = **\$0.519**. Figure 2 (b) shows the cost versus performance for each of our experiment. In the case of GCE, the cost would have remained constant if the execution times exactly halved on every CPU doubling, but the slightly higher execution times causes the cost to rise steadily. For GCF, the cost increases with memory allocation because the execution times do not decrease in proportion to the increase in price. The outlier point for GCF in Figure 2 (b) is the 2048 MB experiment described above. It is interesting to note that, even though computation with GCF seems expensive compared to GCE as shown in Table 1, we found the actual cost of running the experiments to be less expensive for GCF as shown in Figure 2 (b). This counter-intuitive result is due to the fact that GCF is a true pay-as-you-go model, i.e., we are charged only when function instances are running (some finish earlier than others) compared to GCE where we pay for all the provisioned CPU for the entire length of the experiment.

Major findings from our experiments are summarized below:

- **GCF is less expensive than GCE for similar performance**: GCE with 128 CPU has similar performance with that of GCF with 256 MB memory, however, GCF is 40% less expensive. Similar observation can be made between GCE with 256 CPU and GCF with 512 MB memory (26% less expensive) as well as between GCE with 512 CPU and GCF with 1024 MB memory (14% less expensive).
- **GCF performance variation is high**: The total execution time for GCF varied in between runs, as evident from greater confidence intervals in Figure 2 (a). This fluctuation is due to the variation in the count, setup overhead, and raw computation speed of function instances between experiment runs.
- **GCE achieves the highest level of performance**: Highest performance is achieved when we have a CPU per task, all executing in parallel. This is easily achieved in GCE, but not in GCF as we trade-off lack of customization (no control) for simplicity (no system administration overhead).

## 5 | SUMMARY AND FUTURE WORK

In summary, we compared the cost and performance of GCE (IaaS) with that of GCF (FaaS) for a particular human protein similarity comparison HPC workload. Our preliminary results showed that even though IaaS can be 1.65 times faster than FaaS in terms of raw computation performance, FaaS can cost 14% to 40% less than IaaS for similar performance, when we take setup overhead time into account. In the future, we would like to extend and further validate our work with other different HPC workloads and with different cloud providers. FaaS and serverless computing in general are relatively new cloud computing services and limited research exists in this area. It would be interesting to see serverless computing being used in other fields such as Internet-of-Things, fog computing, and artificial intelligence, each of which have their own set of challenges.

## ACKNOWLEDGMENTS

This material is based upon work supported by Google Cloud. We would like to thank the anonymous reviewers for their valuable feedback in improving this paper.

## References

1. Netto MAS, Calheiros RN, Rodrigues ER, Cunha RLF, Buyya R. HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Comput. Surv.* 2018; 51(1): 8:1–8:29.
2. Marathe A, Harris R, Lowenthal DK, et al. A comparative study of high-performance computing on the cloud. In: Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing. ; 2013: 239–250.
3. Niu X, Kumanov D, Hung LH, Lloyd W, Yeung KY. Leveraging serverless computing to improve performance for sequence comparison. In: Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics. ACM; 2019: 683–687.
4. Lynn T, Rosati P, Lejeune A, Emeakaroha V. A preliminary review of enterprise serverless cloud computing (Function-as-a-Service) platforms. In: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). ; 2017: 162–169.
5. Spillner J, Mateos C, Monge DA. FaaSter, better, cheaper: The prospect of serverless scientific computing and HPC. In: High Performance Computing. Springer International Publishing; 2018: 154–168.
6. Kumanov D, Hung LH, Lloyd W, Yeung KY. Serverless computing provides on-demand high performance computing for biomedical research. *arXiv preprint arXiv:1807.11659* 2018.
7. U.S. Department of Energy . The Magellan Report on Cloud Computing for Science. tech. rep., Office of Advanced Scientific Computing Research (ASCR), Washington, DC; 2011.
8. Maamar Z, Baker T, Sellami M, Asim M, Ugljanin E, Faci N. Cloud vs edge: Who serves the Internet-of-Things better?. *Internet Technology Letters* 2018; 1(5): e66.
9. Gill SS, Tuli S, Xu M, et al. Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges. *Internet of Things* 2019; 8: 100118.
10. Tseng L, Mantica J, DeAntonis J, Higuchi T, Altintas O. Peer-assisted content delivery network by vehicular clouds: Algorithm and evaluation. *Internet Technology Letters* 2019; 2(4): e103.
11. Jonas E, Schleier-Smith J, Sreekanti V, et al. Cloud programming simplified: A Berkeley view on serverless computing. Tech. Rep. UCB/EECS-2019-3, EECS Department, University of California, Berkeley; 2019.
12. Lloyd W, Ramesh S, Chinthalapati S, Ly L, Pallickara S. Serverless computing: An investigation of factors influencing microservice performance. In: 2018 IEEE International Conference on Cloud Engineering (IC2E). ; 2018: 159–169.
13. Kijak J, Martyna P, Pawlik M, Balis B, Malawski M. Challenges for scheduling scientific workflows on cloud functions. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). ; 2018: 460–467.
14. Lee BD, Timony MA, Ruiz P. DNAvisualization.org: A serverless web tool for DNA sequence visualization. *Nucleic Acids Research* 2019; 47(W1): W20–W25.
15. SchedMD . SLURM Workload Manager. 2019. https://slurm.schedmd.com. Accessed September 23, 2019.