Serverless FaaS Scalability Evaluation: An ECG Signal Processing Use Case

D. Mileski* and M. Gusev*
* Sts Cyril and Methodius University in Skopje, Faculty of Computer Science and Engineering, Skopje, North Macedonia
E-mail: dimitar.mileski@ieee.org, marjan.gushev@finki.ukim.mk

***Abstract*—The serverless approach provides a completely new way of developing cloud services, service scalability, and elasticity by utilizing the container-level virtualization and abstraction. In this paper, we present a use case of an application that processes streaming electrocardiograms by implementing the Function as a Service approach. Several experiments are conducted to evaluate the scalability and elasticity performance by checking the following hypothesis: The system will be highly scalable keeping the same response time and throughput, for up to 7000 data streams. To check the validity of the hypothesis we will provide an experimental research on the following research questions to analyze the performance behaviour of response and throughput with the increased load of parallel data streams testing the following cases: A) Function that will generate a variable number of data streams, B) Functions that are invoked sequentially, and C) Functions that are invoked in parallel. Our use case proved that system scales horizontally and satisfies all requests. When the workload has linearly increased the system has linearly increased throughput and results in a similar response time for each individual request.**

Keywords— FaaS; serverless; ECG; cloud computing; public cloud; cloud storage; cloud pub/sub

I. Introduction

The power of abstraction and virtualization enables new technologies and software development paradigms. Function as a Service (FaaS) is a serverless implementation, which allows for scalability at the function level and a payment model that depends on the memory used by the function and the number of requests to the function.

Serverless is on-demand computing, which is automatically scaled and billed only for the time the code is running and only for resources used, avoiding the need to pay for idle servers, closest to the original idea of cloud computing as a utility [1]. The abstraction hides how servers are used and maintained, which makes developing easier [1]. This became possible with the development of containers, microservices [2], and Event-Driven architectures [3]. Serverless is considered as the developer control over the cloud infrastructure [2], filling the large gap between PaaS and SaaS [3]. The way serverless services work brings challenges like quality-of-service (QoS), monitoring, scaling, and fault-tolerance [1]. Serverless can be efficiently used as a real-time data analytics platform for edge [4] or dew computing [5].

Function-as-a-Service (FaaS) is a serverless service where the computational unit is a function, with a main components triggers to events or HTTP requests [1]. FaaS should be small, short-lived, stateless, on-demand service with a single responsibility, small input and output after a short amount of time, which makes them easily scalable. FaaS are platform-agnostic, operational concerns are delegated to the platform and context-agnostic, unaware why and how it is used [3]. FaaS can be configured for their memory size, number of CPUs, max execution time, min and max number of function instances, execution environment (programming language runtime).

This paper presents a prototype application that processes electrocardiogram (ECG) signals using serverless technologies, more precisely FaaS. The advantage of implementing this approach to process streaming medical signals, or any other type of IoT device, is to develop services without taking care to manage the resources (virtual machines or the containers). This allows the developer to focus on the input, code, and output, managing only the connections to other cloud services.

Google Cloud Functions (GCF) [6] is a serverless FaaS product from Google. Elasticity and scaling from zero to "infinity" [1] are probably the most important. Scalability is achieved only if the response time remains approximately the same with an increase or decrease in the workload. Elasticity refers to the ability of a cloud function to automatically expand or compress the resources on a sudden-up and down in the requirement so that the workload can be managed efficiently. Fine-grained elasticity in serverless platforms is useful for on-demand applications like creating image thumbnails [7] or processing streaming events [8]. Elasticity also plays an important role in data analytics workloads [8].

GCF support autoscaling by setting maximum bounds of the number of created instances in response to the request load. Sudden traffic spikes may cause the limit to be temporarily exceeded.

GCFs are triggered by events. In the experimental part we will use HTTP, Cloud Storage, and Pub/Sub events. Publisher/Subscriber (Pub/Sub) is a popular communication paradigm allowing for interaction in a decoupled fashion by publishing their messages on logical channels and receiving the messages from the subscribed channels [9]. A pub/sub middleware offers three main types of decoupling which make it particularly suitable for large-scale IoT deployments: (1) Message producers (publishers) and consumers (subscribers) are decoupled in time, i.e., they do not have to be connected at the same time; (2) Messages are not explicitly addressed to a specific consumer but

This is accepted version. Published version DOI: 10.23919/MIPRO55190.2022.9803568

to a symbolic address (channel, topic); (3) Messaging is asynchronous, non-blocking [10]. In the topic-based scheme, the symbolic channel addresses are topics, usually in the form of strings, i.e., producers publish to and consumers subscribe to topics. Messages are only delivered to matching subscribers. Topics may be organized hierarchically, i.e., a topic may be a subtopic of another topic. Subscriptions on a parent topic will then usually also match all subtopics [10]. Google Cloud Pub/Sub [11] is Google's Cloud Service that integrates well with other services such as Cloud Functions and Cloud Storage. Cloud Pub/Sub has Topics in which Publishers send messages and Subscribers subscribe and receive those messages.

In this paper, we evaluate the scalability of FaaS with ECG signals collected on the Google Cloud Storage, as an Object Storage [12], which allows triggering of GCF and then persistence of the returned results. This paper contains methods and performance evaluation of the developed serverless solution for processing ECG signals. The research is based on simulation and experimental methods applied to the application, ECG data streams are simulated by realizing a virtual patient generator aiming at the creation of parallel data streams with small data chunks representing real ECG signals. Further on, this is used as input to two functions that process the incoming data files.

Evaluation of the scalability will be focused on checking the following hypothesis: The system will be highly scalable keeping the same response time and throughput, for up to 7000 data streams. To check the validity of this hypothesis, we will provide an experimental research on the performance behaviour of response and throughput with the increased load of parallel data streams testing the following cases:

- Function to generate a large number of data streams,
- Functions that are invoked sequentially, and
- Functions that are invoked in parallel.

The paper follows the next structure. Related work and analysis of the state-of-the-art are presented in Section II. System architecture, Experiments, and Evaluation methodology are described in Section III illustrating the approach that will achieve system scalability, and how system response time, throughput, and active instances will be measured. Results are evaluated and discussed in Section IV. Finally, Section VI presents the Conclusions and future work directions.

II. State of the art

A lot of research has been conducted focusing on response time, throughput, instance creation time, number of active instances per number of requests, cold starts and deployment time of GCF. This is an overview of the related work about GCF performance metrics.

Malla and Chritensen [13] compare Google cloud's FaaS (Cloud Functions) with its IaaS (Compute Engine) in terms of cost and performance for a parallel task, concluding that FaaS can be 14% to 40% less expensive than IaaS for the same level of performance, including the setup overhead time. Their experiment includes 861 independent tasks without communication between the results from the tasks, giving rise to an embarrassingly parallel HPC workload, and 1.65 times faster IaaS than FaaS in terms of raw computation performance.

The very first function initiation introduces a cold start, and then the instance stays alive to be reused for subsequent requests. The empirical data show a high variance of idle periods [14], depending on the instance type and implemented programming language with variations up to 5 times difference. Instances are recycled after 15 minutes of inactivity. GCF CPU resources are allocated proportionally to the memory. So, in theory, larger instances could start faster, without a significant speed-up as the instance size grows [14]. Setting a minimum number of instances can avoid application cold starts and reduce application latency. Cloud Functions attempts to keep function instances idle for an unspecified amount of time after handling a request. Setting a minimum number of instances incurs cost [15].

Cloud Run is a serverless managed compute platform based on request or event-triggered functions in containers. Tests with thousands of different variables over a large set of data [16] reveals that the average API response time is decreased by 30% when changing from Cloud Functions to Cloud Run. The performance and cost of FaaS are influenced by network latency and retrieval times from object storage [17], concluded after investigation of the object retrieval times from regional buckets with Google Cloud Storage and AWS S3. Optimization of the FaaS functions for the underlying processor architectures is very important [18] and can improve performance by 18.2 times and save costs by 76.8% on average, achieving a maximum speedup of 1.79 times by tuning the function to the instruction set of the underlying processor architecture.

Continuous development of FaaS changes the performance of different Cloud providers over time, initiating the importance of developing a measurement platform for serverless systems and performing continuous measurements for better performance characterization [19], launching more than 50,000 function instances.

Although CPU shared for cloud functions is proportional to the memory allocated claims AWS and Google, Malawski et al. [20] explain that they observe linear performance growth with the memory size in the case of AWS, although sometimes slightly slower, while for Google Cloud Functions the performance is proportional to the memory allocated, but sometimes much faster. They don't observe any correlation between the function size and performance for Azure Functions and IBM Functions. When it comes to benchmarks, AWS achieves higher scores in Linpack (over 30 GFlops) whereas GCF tops at 17 GFlops [20], and also, the day of the week and time of the day is not a significant performance dependency of the functions in different cloud providers. The computational

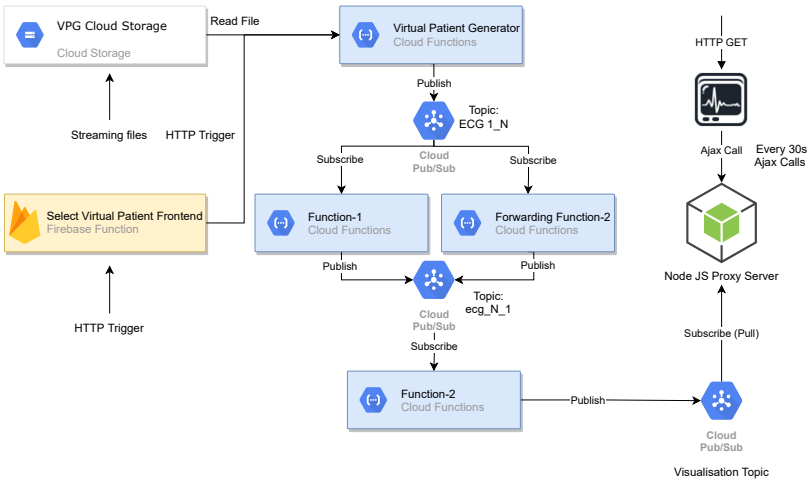

Fig. 1: ECG Serverless Processing Architecture

performance of a cloud function is proportional to the function size for AWS Lambda and Google Cloud Functions, with the exception of about 5% cases when Google function runs faster than expected [21], notifying that IBM and Azure computational performance of a cloud function is not proportional to function size. Analyzing the Network performance (throughput) of a cloud function, AWS and Google are proportional to function size, [21], without conducting experiments on IBM and Azure, finding a constant overhead that does not depend on cloud function size.

Application server instances are reused between calls and are recycled at regular intervals, and their instance lifetime differs between providers [21]. Heterogenous hardware is used for the execution of Functions [21]. McGrath and Brenner [22] observed Linear scalability for AWS Lambda and highest throughput of the commercial platforms, at 15 concurrent requests, sub-linear scaling for Google Cloud Functions and extremely variable performance of Azure Functions.

A novel serverless platform IBM-PyWren was used for executing massively parallel tasks in the IBM Cloud, including a real MapReduce presenting the potential of serverless platforms for distributed computing with speedups $>$ 100X [23]. A cost comparison of monolithic architecture, a microservice architecture operated by the cloud customer and a microservice architecture operated by the cloud provider shows that the monthly costs are advantageous for architecture operated by the cloud provider. [24]. Wagner and Sood [25] observed one year of operating cost for different approaches: AWS Spot Instance Cost 744$, AWS On-Demand Instance Cost 2519$, GCP Preemptible Instance Cost 1079$, GCP On-demand Instance Cost 3153 $, AWS Lambda Cost 186$. Today's serverless state of the art shows that we can not derive a general rule that serverless services are cheaper than VM or containers services, it all depends on the specific use case in which you want to implement serverless services.

III. Methods

A. Solution Architecture

Fig. 1 presents the architecture of the experiment realized as a serverless solution.

1) VPG Cloud Storage: This is the Bucket in Google Cloud Storage [26] for our experiment, where ECG data files are collected.

2) Virtual Patient Frontend: The user interface to select virtual patients is configured by two parameters:

- CS (concurrent streams), which can be changed in the experiments from $x = 5, 10, 20, 30, \ldots, 7000$, and
- FPM (files per minute) with a default value of 2,

Serverless functions are used in this work [2], Firebase Functions [27] to implement the frontend creating HTTP GET requests to trigger the serverless function Select Virtual Patient Frontend, and HTTP Post is sent to the Virtual Patient Generator function to read files from VPG Cloud Storage.

3) Virtual Patient Generator (VPG): This software module generates and streams ECG files. The input of VPG is permanent cloud storage that contains a database of 44 ECG files. Each ECG file contains $FS = 225000$ integers, written in a textual form one per row. Each integer is a 10-bit integer with values between 0 and 1023. This represents a 30-minute ECG with a sampling frequency of 125 Hz. Upon an HTTP trigger VPG starts generating x parallel data streams, where each data stream is associated with one of the predefined ECG files from the MITDB database. If $FPM = 2$, then each data stream generates 2 files per minute, such that immediately takes the first 3750 numbers from the selected file, and writes a file in the bucket 1, after 30 seconds (determined as 1 minute/FPM), it takes the next 3750 numbers for the selected file and writes in the bucket 1, and continues until it generates all files.

For example, let the selected files from the database are 100.ecg and 101.ecg. Let CS=2 so these two files will be

used as input in the VPG. Let FPM=2, meaning that VPG will take 100.ecg and generate 100-01.ecg with the first 3750 integers from the 100.ecg and write them in bucket 1. Then after 60 seconds/(CS+FPM)= 60/4 sec = 15 seconds it will take the first 3750 integers from file 101.ecg and write them in a file 101-01 in bucket 1. After next 15 seconds it will take the next 3750 integers from 100.ecg and write them in a file 100-02.ecg in bucket 1, repeating the action until finished.

4) Cloud Pub/Sub: Our experiment specifies Cloud Functions as Publishers. Subscribers as Cloud Functions and Node JS [28] as Proxy Server. We developed Google Cloud Pub/Sub solution with 3 topics ECG 1_N, ECG N_1, and Visualization Topic which implements a 1:1 Publish/Subscribe model.

5) Function 1 and Function 2: Two functions are developed that process ECG data. Cloud Pub/Sub supports One-to-many (fan-out), Many-to-one (fan-in), Many-to-many models. To implement Many-to-one we need to have more publishers who will send messages on a given topic. In order to make some of the messages from VPG go directly to Function-2, and some go to Function-1 and then to Function-2, the Forwarding Function 2 has been implemented, thus achieving that Function-1 and Forwarding Function 2 will be Publishers on a topic to which Function-2 can be subscribed.

6) ECG Visualisation: Custom Javascript Framework makes AJAX [29] calls at a predefined time to the lastMessageData endpoint of NodeJS Proxy Server to retrieve the data needed for visualization.

B. Experiments

Input into the experiment are ECG files created by VPG as data chunks of 3750 integers for each patient. Since the input file is a 30-minute ECG (with 225,000 integers in the range between 0 and 1023), the VPG will create 60 data chunks, where each data chunk represents a 30-second ECG record. These data chunks are then processed by Function-1 and Function-2.

An initial user interface is created by a web application to allow specification of the number of concurrent streams CS and the rate of sending these files as a number of files per minute FPM. The default values are FPM=2 for setting 30-second ECG data chunks and CS=2 for setting 2 patients that will be concurrently processed.

The test cases include variations of the number of streams in the range from 10 to 7000, with a fixed 2 FPM.

The experiment performs by the following procedure. During the first execution with the workload - 10 ECG files are taken twice from the first time to avoid a cold start, and the measurement starts from the second execution. Then each subsequent execution 10, 20, 30, ...500...6000, 7000 Workload - Number of ECG files is done once with a break of 5 to 10 minutes between executions.

This allows function instances to be idle and the request to be processed without a cold start. The autoscaling option allows starting new instances to satisfy the increased workload. Note that the new instances will have a cold start, but those instances that have already been started and are idle will execute the requests as they arrive. Furthermore, these functions that were idle will process several requests while waiting for the cold start of the new functions.

C. Evaluation methodology

Evaluation metrics address measuring the response time, throughput and cost.

The response time is measured only for Function-1 and Function-2 (expressed in milliseconds). Since each function executes CS*FM*60 times the average response times are T_{F1} and T_{F2} for the functions F1 and F2 respectively. Total Response Time is the sum of the average response times of F1 and F2, that is $T_{tot} = T_{F1} + T_{F2}$.

Throughput is only measured for Function 1 and Function 2 by calculating how many times the function is called per second and how many of the calls ended with the status code ok and how many with error. Google Cloud Monitoring Service [30] dashboards and metrics explorer are used to measure and evaluate parameters.

IV. Results

Fig. 2 presents a performance evaluation of scalability by displaying the response time, on a chart where the x-axis shows the workload as a number of files per minute, and y-axis the average response time of all processed functions.

Response time for F1 is in the range between 58 and 163 ms, with a standard deviation of 30 ms. The high variance of these values was due to the cold start of new F1 instances activated for the increased workload.

Functions F1 and F2 required a cold start in the range between 1000 and 1700 ms. Our experimental production system includes instances of idle functions able to immediately serve requests, which increase the throughput and respond not exceeding an upper limit of 250 and 380 ms respectively for F1 and F2 functions.

Fig. 3 presents scalability performance evaluation by displaying the throughput measurements, as a number of calls to a function that ended with an OK status code since no functions requests ended with an error status code. We observe that the throughput for F2 function is greater than F1, which is due to the more complex nature of the implemented algorithm.

F2 is subscribed to ecg_N_1 which is a Cloud Pub/Sub Topic that receives messages from function F1 and forwarding to F2. Cloud Pub/Sub delivers each published message at least once for each subscription [31]. Approximately If N files are pushed to the function F1, then $2N$ files will be pushed to the function F2. The system scales well, for the different workloads of the same function but also for different workloads of the two different functions,

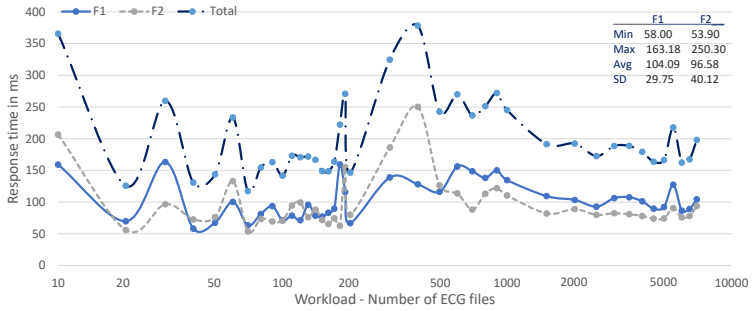

Fig. 2: Performance evaluation of scalability - response time measurements

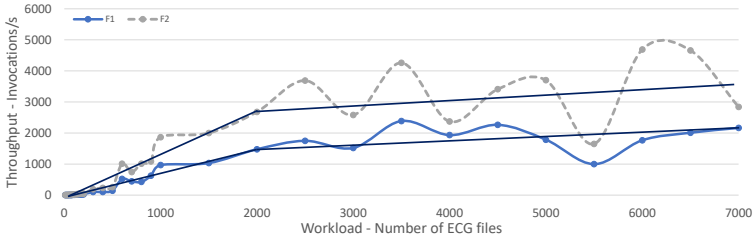

Fig. 3: Performance evaluation of scalability - throughput measurements

average response times are similar (T_{F1} = 104ms, T_{F2} = 97ms).

V. Discussion

Fig. 4 presents a performance evaluation of scalability by displaying the max active instances of the serverless functions VPG, F1, and F2.

The upper limit of active instances per cloud function is 3000. For all functions in this paper, the lower limit of instances was set to 0 and the upper limit was set to 3000. You can avoid cold starts for your application and reduce application latency by setting a minimum number of instances. Minimum instances will be kept running at idle, ready to serve requests. Instances kept running in this way do incur billing costs [32]. In our experiment, this was not necessary because during each execution the requests were processed by the idle instances of the previous execution and by the new instances which were started with the increased workload.

During one execution the number of instances increased and decreased as the requests came. Figure 4 shows the maximum number of instances that were active for a particular Workload - The number of ECG files. We can notice that VPG has some linearity of max active instances, while in F1 and F2 we have a saturation of max active instances. VPG represents HTTP triggered Cloud Function, while F1 and F2 are Cloud Pub/Sub Triggered. We make HTTP Post requests with the Javascript library Axios. F1 and F2 are Push subscribers, so they start when a push message arrives. There are Pub/Sub quotas and limits in the official documentation from Google Cloud [15]. It is necessary to research whether there is a causality between quotas of Cloud Pub/Sub and the maximum number of instances that are instantiated especially for the part over 2000 workload.

VI. Conclusion

This paper presents the advantages of FaaS in terms of elasticity. FaaS can be used for whole system implementation or for individual services that need to be elastic. FaaS can be cheap for services that have a small number of requests and that will not run for very long. FaaS is elastic for services that need to support sudden spikes in requests. The system scales horizontally and satisfies all requests. We have proved that with the increased workload, the system has a similar response time for each of the requests. When the workload is linearly increased the system has linearly increased throughput and has a similar response time for each individual request with a Standard Deviation from 30 to 40ms.

FaaS can be an elastic and inexpensive solution only if the services to be implemented are developed and tuned properly according to the recommendations for FaaS development. Our future plans address realization of an experiment that simulates an ECG monitoring center for 10.000 simultaneous ECG data streams.

Acknowledgement

This research is supported from the European High- Performance Computing Joint Undertaking (JU) under grant agreement No 951745, in particular, the FF4EUROHPC CALL-2, for the project entitled “CardioHPC Improving DL- based Arrhythmia Classification Algorithm and Simulation of Real-Time Heart Monitoring of Thousands of Patients”.

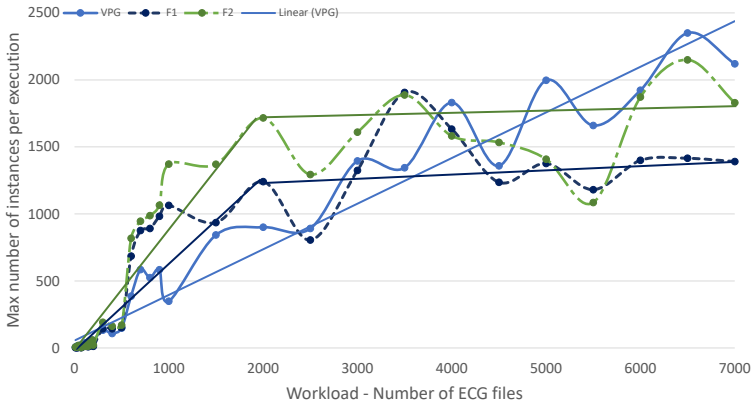

Fig. 4: Performance evaluation of scalability - Maximum number of active instances for different number of ECG files

REFERENCES

[1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Communications of the ACM*, vol. 62, no. 12, pp. 44–54, 2019.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," in *Research advances in cloud computing*. Springer, 2017, pp. 1–20.

[3] E. Van Eyk, A. Iosup, S. Seif, and M. Thömmes, "The spec cloud group's research vision on faas and serverless architectures," in *Proceedings of the 2nd International Workshop on Serverless Computing*, 2017, pp. 1–4.

[4] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017.

[5] M. Gusev, "Serverless and deviceless dew computing: Founding an infrastructureless computing," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2021, pp. 1814–1818.

[6] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, and D. A. Patterson, "What serverless computing is and should become: The next phase of cloud computing," *Communications of the ACM*, vol. 64, no. 5, pp. 76–84, 2021.

[7] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 363–376.

[8] Q. Pu, S. Venkataraman, and I. Stoica, "Shuffling, fast and slow: Scalable analytics on serverless infrastructure," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 193–206.

[9] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics," in *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, 2007, pp. 109–118.

[10] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting iot platform requirements with open pub/sub solutions," *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 41–52, 2017.

[11] S. Krishnan and J. L. U. Gonzalez, "Google cloud pub/sub," in *Building Your Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 277–292.

[12] E. Bisong, "An overview of google cloud platform services," *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 7–10, 2019.

[13] S. Malla and K. Christensen, "Hpc in the cloud: Performance comparison of function as a service (faas) vs infrastructure as a service (iaas)," *Internet Technology Letters*, vol. 3, no. 1, 2020.

[14] A. M. Shilkov, "Cold starts in google cloud functions." [Online]. Available: https://mikhail.io/serverless/coldstarts/gcp/

[15] Google, "Pub/sub quotas and limits; cloud pub/sub documentation; google cloud." [Online]. Available: https://cloud.google.com/pubsub/quotas

[16] R. Gulbrandsen, "We cut our average api response time by 30% when changing from cloud functions to cloud run." [Online]. Available: https://unloc.app/en/magazine/we-cut-our-average-api-response-time-by-30-when-changing-from-cloud-functions-

[17] LightStep Research, "How google cloud storage processes cloud functions 4x faster than amazon web services," Jul 2019. [Online]. Available: https://research.lightstep.com/reports/google-cloud-storage#abstract

[18] M. Chadha, A. Jindal, and M. Gerndt, "Architecture-specific performance optimization of compute-intensive faas functions," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 478–483.

[19] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (ATC 18)*, 2018, pp. 133–146.

[20] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *European Conference on Parallel Processing*. Springer, 2017, pp. 415–426.

[21] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 23, p. e4792, 2018.

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

[23] J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López, "Serverless data analytics in the ibm cloud," in *Proceedings of the 19th International Middleware Conference Industry*, 2018, pp. 1–8.

[24] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano *et al.*, "Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures," *Service Oriented Computing and Applications*, vol. 11, no. 2, pp. 233–247, 2017.

[25] B. Wagner and A. Sood, "Economics of resilient cloud services," in *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2016, pp. 368–374.

[26] J. Ju, J. Wu, J. Fu, Z. Lin, and J. Zhang, "A survey on cloud storage." *J. Comput.*, vol. 6, no. 8, pp. 1764–1771, 2011.

[27] L. Moroney, "Cloud functions for firebase," in *The Definitive Guide to Firebase*. Springer, 2017, pp. 139–161.

[28] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[29] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

[30] H. J. Syed, A. Gani, R. W. Ahmad, M. K. Khan, and A. I. A. Ahmed, "Cloud monitoring: A review, taxonomy, and open research issues," *Journal of Network and Computer Applications*, vol. 98, pp. 11–26, 2017.

[31] Google, "Subscriber overview; cloud pub/sub; google cloud." [Online]. Available: https://cloud.google.com/pubsub/docs/subscriber

[32] Google Cloud Functions Documentation, "Using minimum instances." [Online]. Available: https://cloud.google.com/functions/docs/configuring/min-instances