

Serverless Electrocardiogram Stream Processing in Federated Clouds With Lambda Architecture

Sashko Ristov^{ID}, University of Innsbruck

Marjan Gusev^{ID}, University Saints Cyril and Methodius

Armin Hohenegger and Radu Prodan^{ID}, University of Klagenfurt

Dimitar Mileski^{ID}, **Pano Gushev**^{ID}, and **Goran Temelkov**, Innovation Doel

In this article, we explore a novel architecture for distributing health monitoring computations over distributed cloud regions, both for constantly online patients and offline for several hours daily. We propose a conceptual architecture for a use-case example capable of processing thousands of simultaneous incoming streams with electrocardiogram signals.

Although telemedicine has emerged as an everyday necessity for health monitoring, still current solutions are built for a small number of patients or use sensors that do

not stream data with high velocity and volume. Current serverless cloud providers limit the concurrency within a single region, and we evaluate the performance of this solution across multiple cloud regions. The results indicate that our new solution can overcome the limitations of a single cloud for online and offline patients, thereby saving their lives in case of detected dangerous arrhythmia.

Digital Object Identifier 10.1109/MC.2023.3281873
Date of current version: 23 August 2023

This work is licensed under a Creative Commons Attribution-NonCommercial-No Derivatives 4.0 License. For more information, see <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

INTRODUCTION

Data statistics for 2020 and 2021 show that heart disease and stroke are the leading cause of death, followed by cancer and COVID-19. Every third person died due to cardiovascular problems.^{1,2} Telemedicine and health monitoring can mitigate the risk of death by detecting dangerous arrhythmia and preventing life-threatening conditions for patients outside the hospital, living in their homes, and working in their usual office environments. Typical symptoms appear 2 h before cardiac death occurs.³ However, this risk requires near real-time data processing and taking medical treatment within that period. Unfortunately, several challenges must be addressed, such as arrhythmia detection within seconds per patient and scaling the number of patients.

While most patients are online and their heart conditions can be monitored in real time, some patients may work without an Internet connection. Therefore, not all patients are online 24/7, which requires an architecture that allows real time for online patients and batch processing when offline patients connect online. Although serverless architectures⁴ are addressed in this article as a scalable and elastic cloud resource solution with these processing requirements, we still refer to several accompanied issues in practical implementations, such as storage data input/output limitations, network throughput, or the number of activated concurrent cloud instances. Telemedicine-related issues are mainly identified as big data challenges of many incoming data streams with large volumes and high velocities.

BACKGROUND

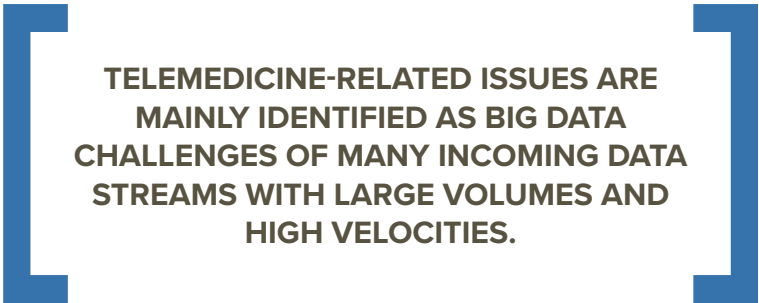
This section introduces the necessary concepts about serverless technologies

and lambda architecture for big data processing.

Serverless computing

Serverless computing offers developers a platform that shields the server usage, automatically scales without explicit provisioning, and charges only for the resources that are used while their code is running.⁵ Various serverless-based services are in a mature phase. Users

applications by serverless workflows, known in the literature as *function choreographies* (FCs).⁶ Many public cloud providers offer FC systems, such as AWS Step Functions, but they have several weaknesses. First, the cloud providers tend to lock the users into their clouds, and their FC systems allow them to build and run FCs, but mainly on their own FaaS platforms. Second, the FC systems generate additional costs



TELEMEDICINE-RELATED ISSUES ARE MAINLY IDENTIFIED AS BIG DATA CHALLENGES OF MANY INCOMING DATA STREAMS WITH LARGE VOLUMES AND HIGH VELOCITIES.

may dockerize their application and deploy it as a serverless container on, for example, Google Cloud Run or AWS App Runner. A lighter option is to develop the application by serverless functions using a function-as-a-service (FaaS) service, such as Google Cloud Functions or AWS Lambda. While the FaaS services are cheaper than the serverless containers for smaller loads, they are offered with many constraints and limitations, such as the maximum number of concurrent functions or the limited duration, memory, and input and output data size.

Serverless workflows or function choreographies

While the serverless functions introduce many advantages, still, due to the constraints, they are usually lightweight and short-running. To create serverless applications with increased complexity, developers may build their

for managing the states of the FC execution. Other FC systems, such as Google Workflows, support running FCs across various FaaS systems. However, their concurrency is limited, which restricts FC scalability. Other options are open source FC management systems, such as xAFCL,⁷ which abstract the application from the underlying FaaS systems, thereby supporting running FCs across federated FaaS.

Lambda architecture

Almost a decade ago, Waren and Marz⁸ introduced the lambda architecture pattern for big data processing, integrating batch and stream processing in a single architecture. Instead of querying the entire dataset, which may require enormous computing resources, they created the three-layered pattern comprising batch, serving, and speed layers. The batch and serving layers support arbitrary queries on all data with the

low latency updates tradeoff: that is, out-of-date queries by several hours. To overcome this tradeoff, the speed layer ensures that new data are integrated into queries as fast as needed by using incremental computation over newly received data.

TELEMEDICINE HEALTH MONITORING USE CASE

Wearable devices, such as electrocardiogram (ECG) sensors, allow the measurement of ECG stream data, and telemedicine offers patients near real-time heart health care. Recent technology solutions⁹ have proven that arrhythmia can be detected in a limited number of patients. The ViewECG system is defined as a *near real-time* processing system since it processes 30-s ECG stream data measurements. Additionally, the results are postponed depending on processing time.

The initial virtual machine-based architecture

The initial implementation of ViewECG was deployed on three virtual machines (VMs) (Figure 1). The main ap-

plication runs in the main VM, the access point for the remote user interface and web application interface (API) for external system communication. To keep the user credentials and authorization data, ViewECG uses the MSSQL database, including the statistical results and pointers to the medical data stored in the data processing unit (DPU). The DPU VM is the second VM, which accepts incoming ECG data streams, stores data in the local file system, and processes the incoming ECG stream data. Finally, the beat detection and classification (BDC) VM runs a separate module for BDC of arrhythmia.

Due to the low elasticity of VMs, this initial architecture is not scalable. Problems arise whenever the number of ECG stream files increases, or a high number of postponed requests arrive at the system when the offline patients send their ECG stream data as they appear online.

The serverless architecture

The new serverless architecture processes the ECG stream data by a sequence of four software modules implemented

as serverless functions executed in a pipeline one after another, including preprocessing (noise detection and data filtering), BDC, postprocessing (detecting specific ECG features), and finalization (statistical analysis and report preparation). Functions are developed with different programming languages (Java and C#).

Each file for near real-time processing consists of 3,750 ECG samples based on a 125-Hz sampling frequency. Since the size of these data are 23 kB, data are sent by value while invoking the serverless functions instead of storing the data as a file to the storage system and invoking the functions by reference to that file. We decided on this approach based on our previous work.¹⁰

In case of detecting a potential arrhythmia, the system notifies the doctor, and in case of arrhythmia, the doctor takes the appropriate medical measures.

ECG SERVERLESS PROCESSING CHALLENGES AND APPROACHES

While serverless computing offers high flexibility, scalability, and low

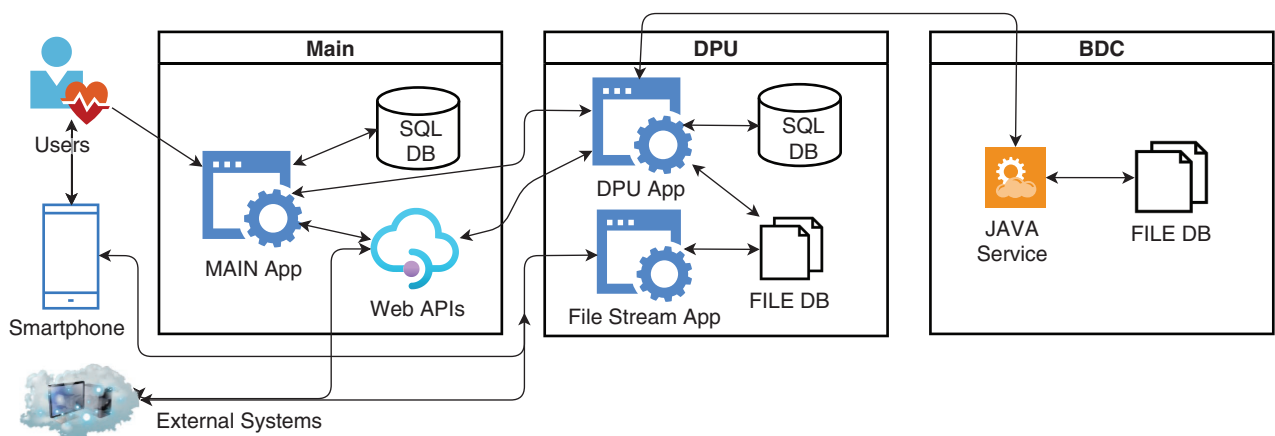


FIGURE 1. The initial VM-based architecture. API: application interface; BDC: beat detection and classification; DB: database; DPU: data processing unit.

development effort and maintenance, there are still several challenges that the research community should tackle.

Challenge 1: Near real-time processing for a scaled number of patients

Although cloud providers automatically scale the number of active serverless functions or containers, they still introduce a concurrency limit, which restricts scalability within one cloud region. For instance, AWS Lambda and the first generations of Google Cloud Functions limit the users to run a maximum of 1,000 functions simultaneously by default, which can be increased to 3,000 functions on demand. However, cloud providers schedule multiple functions on the same container to save resources and can vary the maximum number of parallel instances depending on dynamic memory allocation and virtual CPU use, or allow the users more instances upon special request. Consequently, the scalability is limited, which restricts the scale of telemedicine solutions to numerous patients.

Approach: Cloud federation to run functions across multiple cloud regions of various cloud providers.

Challenge 2: Low processing time for offline patients

In today's modern world, smartphones have become authentication devices working online through wireless or mobile networks. In some cases, patients may be offline due to low coverage or specifics on their working place, and collecting their ECG data up to hundreds of megabytes increases the overall processing time, causing late detection of arrhythmia.

Approach: Use high-performance computing (HPC) data parallelism

to speed up the processing for offline patients.

Challenge 3: Integration of heterogeneous software modules Cloud continuum¹¹ comprises a massively heterogeneous environment, both in hardware and software. This includes a variety of APIs, libraries, and software development kits for heterogeneous sensors, cloud providers, artificial intelligence (AI) services, etc. As a consequence, many modules of ECG systems are coded with different languages, which requires a high degree of interoperability between the modules.

Approach: On top of the serverless platform, use serverless workflows or function choreography that spawn numerous functions and orchestrate serverless functions developed with different programming languages.

CardioHPC approach with lambda architecture

The research is part of the CardioHPC project (https://www.ff4eurohpc.eu/en/experiments/2022031511530081/realtime_heart_monitoring_of_thousands_of_patients) aiming at finding an optimal environment to process thousands of concurrent ECGs. For the CardioHPC project based on simulation and experimental methods, we have developed a virtual patient generator web service to simulate parallel data streams representing ECG signals¹² from previously measured patients, where the user can specify up to 10,000 concurrent streams and files per minute with a default value of 2 (two 30-s files per minute). Besides files per minute and concurrent streams, the web application built as a user interface to this web service can specify the destination system

and the destination where the virtual patient generator will be hosted. The virtual patient generator is used in the evaluation of this research.

In our previous recent work,¹⁰ we analyzed several performance metrics that were input to this research. First, we determined that the near real-time scenario achieves maximum throughput for sending the ECG stream data every 30 s. On the other hand, batch processing results in the lowest latency to access storage and spawn start when the one-week file is split into files of 1-h ECG stream data. Nevertheless, at least 5-s additional latency was observed for FaaS to spawn 672 functions, each of which would process a 15-min ECG data stream or run 42 functions that process a 4-h ECG data stream. Therefore, we adapted the xAFCLFC system to be able to run serverless functions that are deployed on Google Cloud Run. Using this extension, in this article, we evaluated the tradeoff between the benefits of scalability across the regions and lower networking performance due to additional network latency and lower bandwidth between regions.

LAMBDA-BASED SYSTEM ARCHITECTURE FOR ECG STREAM DATA PROCESSING

To overcome challenges 1 and 2, we developed a new system architecture based on lambda architecture capable of conducting near real-time and batch ECG stream data processing for online and offline patients. The solution is deployed using the serverless Google Cloud Run service, which allows high scalability and elasticity with a real "pay-as-you-drink" pricing model. Finally, to overcome challenge 3, we used the serverless workflow management system⁷ to orchestrate serverless functions in a choreography

comprising functions written in an arbitrary programming language.

Figure 2 presents the lambda architecture for distributed ECG stream data processing. The client module is installed on a smartphone to submit the ECG stream from the last 30-s measurements. Since this short ECG stream is 23 kB, the ECG stream is sent by value to the stream, which invokes the pipeline of the four functions presented in Figure 3(a) to process the ECG stream in near real time. Similarly, the data forwarded between the functions are sent by value in JSON format. Scalability is achieved by scaling the serverless functions within the region and across multiple cloud regions, while elasticity is achieved by serverless functions.

However, when an offline patient connects online, the client module detects that the ECG stream is longer than 30 s and stores it on cloud storage. Assuming a patient connects online within 12 h, the ECG stream may grow to 30 MB. Such a large file raises two challenges. First, while invoking the near real-time pipeline with 23 KB may very rarely cause network transfer failures, invoking serverless functions with a payload of 30 MB over a WAN network often causes failures or huge latency due to TCP retransmissions.

To overcome this challenge, we invoke the pipeline by submitting a reference to the file. Second, processing big ECG stream data causes additional delays, which should be minimized to process the ECG stream, detect a possible arrhythmia, and react accordingly. Since detecting arrhythmia does not require processing the whole file, we run a workflow of several functions that split the large file into smaller files, each of 1 h, since our initial measurements revealed that processing the 1-h ECG files achieves the highest throughput.¹⁰ Nevertheless, the “Split” function can be reconfigured to split the file differently, based on, for example, the assigned memory or storage bandwidth. Splitting large files and parallel processing executes the workflow for up to 16 s.

The offline patients are processed by the workflow implementing the abstract function choreography language (AFCL),⁶ which is a YAML-based language and allows the orchestration of parallel loops with a dynamic loop iteration count. Figure 3(b) presents the target workflow in AFCL that passes data by reference between functions, which is then used by functions to access files on cloud storage, as well as to store the processed files back to the storage to be accessible to the

successor functions in the workflow. The function “countHours” is a helper function, which receives the file path and returns the hours as a collection that should be processed by the parallel loop “ParallelFor”. Within the parallel loop, the batch-processing workflow runs a sequence of five functions. We include another helper function “Split” inside the loop to speed up the splitting of the large file into multiple smaller files (e.g., per hour). Afterward, the same four functions are executed [Figure 3(b)]. Each of the four functions receives and returns file names (references).

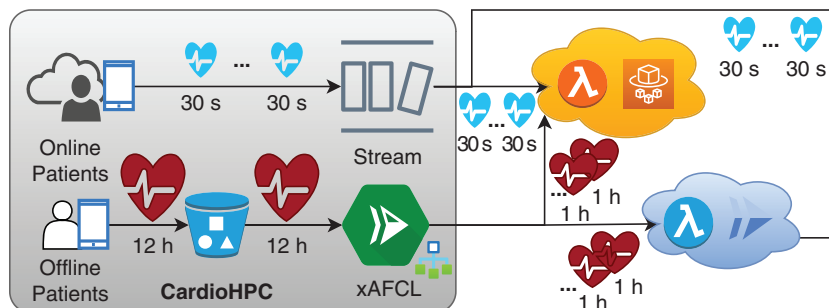


FIGURE 2. Lambda-based distributed ECG stream processing architecture.

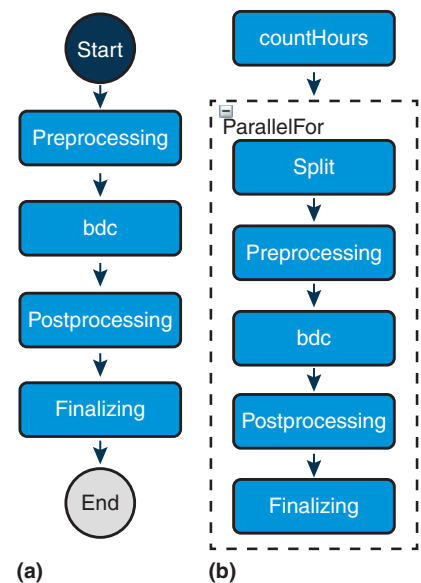


FIGURE 3. Two implementations of the serverless workflow. (a) Near real-time processing: that is, 30-s ECG data are passed as a value in JSON format between functions; (b) batch processing: that is, abstract function choreography language (AFCL) implementation, where data are stored on Google Cloud Storage, and references to files are passed between functions within JSON input data.

Both serverless workflows run the same cloud functions, which reduces system maintenance. Namely, the functions check whether the value of the input data “ecgDataFilePath” is null, in which case they retrieve data by value from the other data input ECG JSON object. Otherwise, the functions load data from the given “ecgDataFilePath” on storage, where the client smartphone application has already uploaded the file. The current implementation of the system architecture uses Google Cloud Storage.

REAL PRODUCT EVALUATION

We evaluated our system based on the lambda architecture both for online and offline patients. For the former, we evaluated challenge 1 by analyzing the achieved throughput in terms of patients per second by varying the number of requests. For each patient, an ECG stream of 30 s was submitted. On the other side, since challenge 2’s goal is to process the offline patients’ ECG data stream as fast as possible, we evaluated the makespan of the batch processing for various lengths of the ECG stream data.

Experiment setup in federated clouds

In our experiments, we deployed the three Google Cloud Run serverless functions—preprocessing, BDC, and post-processing—on four cloud regions of Google Cloud in London (United Kingdom), Frankfurt (Germany), St. Ghislain (Belgium), and Hamina (Finland), the BDC module developed with Java and the others with C#. All of these functions were used both for offline and online patients. We omitted the last function, finalizing, from the workflow because it is needed for statistical purposes and does not affect arrhythmia detection. For higher flexibility and lower cost of AWS

Lambda compared to Google Cloud Run, we deployed the helper functions “countHours” and “split” for the batch processing workflow as AWS Lambda functions in AWS Frankfurt. All functions access the same Google Object Storage in Frankfurt for offline patients, while for online patients, files are stored at each processing site and forwarded by value between containers and functions in the pipeline. Figure 4 presents the setup for processing both.

Online patients processing: Throughput

Evaluation plan for online patients.

When the first time hit the service, some of the requests returned an error code for the BDC module due to too many requests, which is a rather expected cold-start phenomenon¹³ for zero active instances when a large number of requests are generated per second. After this initial setup, we executed each experiment five times and calculated the average. The serverless setup specified the maximum number of active instances to be 1,000 and the concurrency parameter to be 1 to obtain the maximum parallel environment. The

evaluation for online patients includes analysis of the throughput as the number of processed requests per second for a given workload of requests per second.

Results for online patients. We specified five experiments with the following workload: 100, 200, 333, 433, and 533 requests per second, corresponding to 3,000, 6,000, 10,000, 13,000, and 16,000 patients that stream ECG files every 30 s. Each experiment was executed for 2 min in each Google Cloud region.

We observed that the average measured response time to fulfill a request for the whole workflow is 2.47 s, with a minimum of 1.06 s and a maximum of 9.63 s, which appear mainly in the cold start while activating a new instance (standard deviation of 2.74 s).

Figure 5 presents the achieved throughput for various workloads submitted by the virtual patient generator to the four evaluated Google Cloud regions. We observe that Google Cloud Run maintains handling all incoming requests, even for 533 requests per second, which leads to an achieved total throughput of 2,170 requests per second for all four cloud regions. Without losing generality, if one would exploit

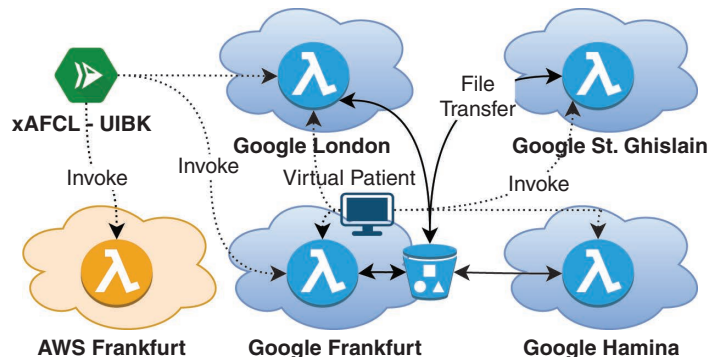


FIGURE 4. Experiment setup on one AWS and four Google cloud regions in Europe and used storage in Google Frankfurt.

all 12 Google cloud regions in Europe, the throughput of 6,510 requests per second may be achieved. This number can be even further increased if we increase the workload per cloud region and use other cloud providers. However, the former may increase the average runtime of the overall pipeline due to the concurrency overhead.^{7,14}

We also analyzed the number of instances activated for the different workloads in all evaluated cloud regions, presented in Figure 6. The cloud region St. Ghislain raised the highest number of active instances, revealing the highest throughput, while the cloud region Hamina had the lowest number of active instances. Note that this depends on the capacity and availability of the corresponding cloud region. All cloud

regions reached the desired throughput, and responses for each request were processed in an average of 2.47 s.

Offline patient data processing: Low makespan

Evaluation plan for offline patients. Offline patients were processed with the xAFCL serverless workflow management system.⁷ We evaluated offline patients by increasing the size of the ECG data stream. For this purpose, we created three different inputs of 1-h ECG files, as a baseline, and 2- and 12-h ECGs, which need to be split into files of 1-h ECG. We assumed that patients' maximum working time is 12 h, for which time period they can be offline. xAFCL runs at the University of Innsbruck,

Austria, and provides different latencies to each region. The batch workflow is executed on each cloud region with all three inputs. Each execution was repeated five times, and the last four executions were considered to avoid the cold-start effect of the first execution. Finally, we measured the makespan and evaluate speedup and efficiency of 2- and 12-h ECG streams compared to a 1-h ECG stream.

The makespan is represented in seconds and processing speed in minutes of ECG stream data per second for all three lengths of the ECG stream. Furthermore, we have additionally evaluated speedup as a number and efficiency in percentage for lengths of 2 and 12 h.

Results for offline patients. Figure 7 presents the makespan when running the workflow for offline patients in the four evaluated cloud regions with input files of 1, 2, and 12 h. We will refer to these experiments as *ECG(1)*, *ECG(2)*, and *ECG(12)*, respectively. We observed the fastest workflow processing in Frankfurt for all three ECG stream lengths, mainly due to two reasons. First, the network latency from the University of Innsbruck to Google Frankfurt is the smallest, leading to the smallest invocation delay. Second, the functions in Google Frankfurt access the co-located storage faster than functions from the other three cloud regions. On average, *ECG(2)* and *ECG(12)* run slower than *ECG(1)* by 1.45% and 17.32%, respectively, due to the longer execution of the function “split” for larger files. However, due to splitting and distributing 1-h chunks of ECG data streams on different functions that run concurrently, *ECG(12)* and *ECG(2)* process on average 50.15 and 9.72 min of ECG stream data per

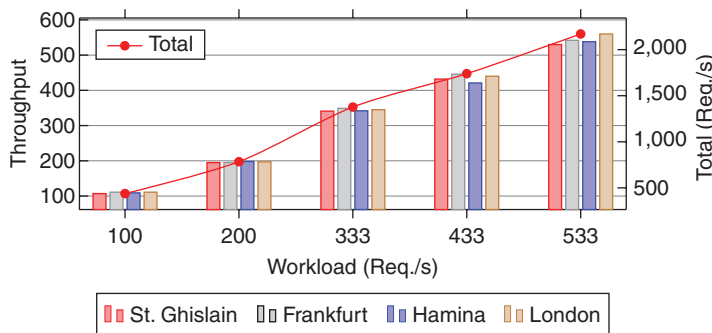


FIGURE 5. Achieved throughput for various workloads per cloud region and total for all four evaluated cloud regions.

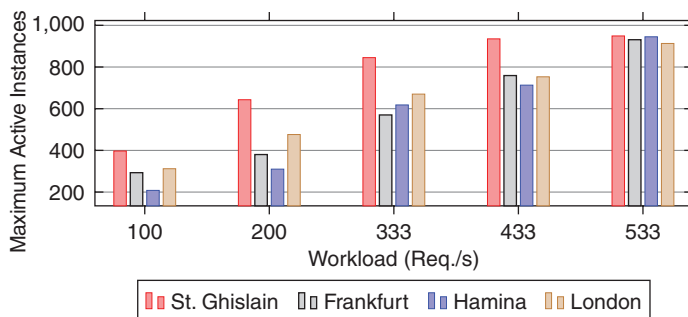


FIGURE 6. Maximum number of active instances of the near real-time processing.

second per cloud region, compared to ECG(1), which can process 4.89 min of ECG stream data per second on average per cloud region.

In Figure 8, we present the achieved speedup and efficiency in each region for the 2- and 12-h ECG streams compared to a 1-h ECG stream in different cloud regions. We observe that the achieved speedup was in the range of 1.84 in St. Ghislain to 2.13 in Frankfurt for ECG(2), up to 9.79 in Hamina and 10.69 in St. Ghislain for ECG(12). We observed two outliers for ECG(2) in Frankfurt and London, where a super-linear speedup of 2.13 and 2.05 was achieved compared to ECG(1).

DISCUSSION

Contributions

This manuscript brings several contributions:

- › The lambda architecture applied on a serverless platform allows both near real-time and batch processing of patients' ECG stream data.
- › Serverless containers seem like a promising platform for near real-time processing of 2,170 requests per second simultaneously and processes them within a few seconds.
- › With batch processing, we were able to process up to 50-min ECG stream data per second, achieving a speedup of 10.66× when running 12-h ECG stream data compared to 1 h.

Limitations of the proposed architecture

With our lambda-based serverless architecture supported with the xAFCL FC management system, we

achieved high throughput for near real-time processing of 30-s ECG stream data and speedup with parallelization of 12 h. Processing for both types is finished in a few seconds, which allows time to react in the case where a dangerous arrhythmia is detected.

Still, there are some limitations. While federated FaaS may overcome the concurrency limitation of a single region, it introduces higher network latency between regions. This was emphasized for larger ECG stream data of 12 h. Scaling the problem size may clog the bandwidth of the single storage, leading to additional delay in batch processing.

Another limitation may be additional costs due to the charges for the

outgoing traffic between regions. However, these additional charges may happen in case of a sudden workload when multiple patients appear online. Still, we believe that saving patient lives justifies these additional charges. Network and cloud providers may increase their awareness and reduce these charges.

Future directions and research challenges

The tradeoff between computation scalability and networking proximity may be improved by using the closer edge and fog environments, which may reduce data access time. With this approach, one could distribute data across regions and place computation closer to data, as recommended

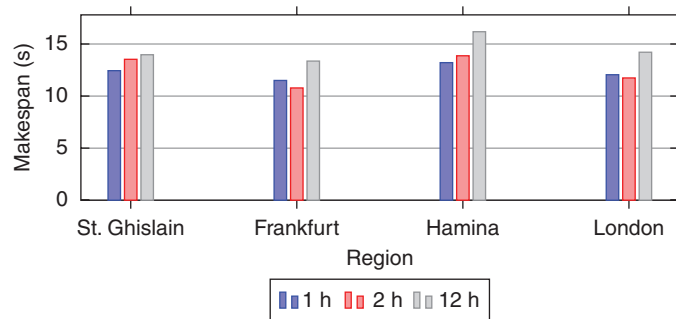


FIGURE 7. Makespan of the batch processing when workflow with different lengths in hours of ECG streams runs in different cloud regions.

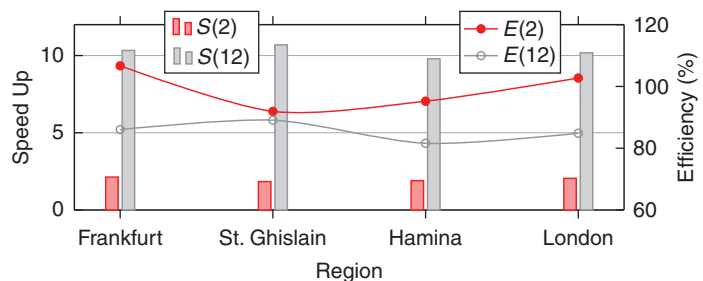



FIGURE 8. Speedup and efficiency of the workflows that run 2- and 12-h ECG streams compared to 1-h ECG stream in different cloud regions.

by Smith et al.¹⁵ with their FaaS functions and data orchestrator (FaDO) system. FaDO, accompanied by an xAFCL FC management system, may scale the batch processing with minimum data access delays within each region. Such a system would require a registry that will send information to the client's smartphone to store patient ECG stream data.

Moreover, using edge and fog may bring several benefits. Edge AI technologies¹⁶ combine hardware and software at the edge, which enables AI algorithms to run on patient smartphones. Many machine learning frameworks, such as TensorFlow or PyTorch, can train neural network architectures with federated learning on powerful edge devices. This is especially needed for offline patients using devices out of the edge of the Internet network, such as in the Dew Computing solution.¹⁷ When a dangerous arrhythmia is detected, their devices will inform the patient directly.

While the federated FaaS may bring scalability, it may increase the cost due to outgoing data transfer between regions, which is additionally charged. Therefore, novel techniques are needed to schedule functions and data location and minimize makespan and cost in federated FaaS. This is necessary for the cloud continuum, considering federated edge and fog layers.

Serverless computing accompanied by cloud federation is a flexible, scalable, and elastic platform suitable for telemedicine and health monitoring of patients with a high risk of cardiovascular diseases even outside of a hospital. Moreover, lambda architecture is necessary for patients that are offline for a longer period of the

day. As a result, ECG stream data can be processed for thousands of patients simultaneously within a few seconds. This mitigates the risk of death caused by heart disease and stroke. However, the tradeoff for high throughput and the near-real processing time is higher costs for data transfer when serverless functions and cloud storage are placed in different cloud regions for patients that are spending a longer part of the day offline. 

ACKNOWLEDGMENT

This research received funding from Land Tirol, under contract F.35499, and the European High Performance Computing Joint Undertaking, under Grant 951745 (FF4EuroHPC project and CardioHPC experiment). The experiment “CardioHPC—Improving DL-based Arrhythmia Classification Algorithm and Simulation of Real-Time Heart Monitoring of Thousands of Patients” has received funding from the European High-Performance Computing Joint Undertaking through the FF4EuroHPC project under Grant 951745. The joint undertaking receives support from the European Union's Horizon 2020 research and innovation program and Germany, Italy, Slovenia, France, and Spain.

REFERENCES

1. “The top 10 causes of death,” World Health Organization, Geneva, Switzerland, Dec. 2020. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>
2. “Leading causes of death,” National Center for Health Statistics, Hyattsville, MD, USA, Jul. 2021. [Online]. Available: <https://www.cdc.gov/nchs/data/nvsr/nvsr70/nvsr70-09-508.pdf>
3. D. Müller, R. Agrawal, and H. R. Arntz, “How sudden is sudden

cardiac death?,” *Circulation*, vol. 114, no. 11, pp. 1146–1150, 2006, doi: 10.1161/circulationha.106.616318.

4. S. Nastic et al., “A serverless real-time data analytics platform for edge computing,” *IEEE Internet Comput.*, vol. 21, no. 4, pp. 64–71, Jul. 2017, doi: 10.1109/MIC.2017.2911430.
5. P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing,” *Commun. ACM*, vol. 62, no. 12, pp. 44–54, Nov. 2019, doi: 10.1145/3368454.
6. S. Ristov, S. Pedratscher, and T. Fahringer, “AFCL: An abstract function choreography language for serverless workflow specification,” *Future Gener. Comput. Syst.*, vol. 114, pp. 368–382, Jan. 2021, doi: 10.1016/j.future.2020.08.012.
7. S. Ristov, S. Pedratscher, and T. Fahringer, “xAFCL: Run scalable function choreographies across multiple FaaS systems,” *IEEE Trans. Services Comput.*, vol. 16, no. 1, pp. 711–723, 2023, doi: 10.1109/TSC.2021.3128137.
8. J. Warren and N. Marz, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. New York, NY, USA: Simon and Schuster, 2015.
9. D. Innovation. “A platform for ECG monitoring and reporting tools.” ViewECG. Accessed: May 21, 2023. [Online]. Available: <https://viewecg.com/>
10. M. Gusev et al., “CardioHPC: Serverless approaches for real-time heart monitoring of thousands of patients,” in *Proc. IEEE/ACM Workshop Workflows Support Large-Scale Sci. (WORKS)*, 2022, pp. 76–83, doi: 10.1109/WORKS56498.2022.00015.
11. P. Beckman et al., “Harnessing the computing continuum for programming our world,” in *Fog Computing: Theory and Practice*. Hoboken, NJ, USA: Wiley, 2020, pp. 215–230.
12. D. Mileski and M. Gusev, “Serverless FaaS scalability evaluation: An

ABOUT THE AUTHORS

SASHKO RISTOV is an assistant professor at the University of Innsbruck, 6020 Innsbruck, Austria. His research interests include performance modeling and optimization of parallel and distributed systems, particularly workflow applications and serverless computing. Ristov received a Ph.D. in computer science from Saints Cyril and Methodius University, Skopje, North Macedonia. Contact him at sashko.ristov@uibk.ac.at.

MARJAN GUSEV is a professor at the Saints Cyril and Methodius University, 1000 Skopje, North Macedonia. His research interests include eHealth solutions, cloud computing, and the Internet of Things. Gusev received a Ph.D. from the University of Ljubljana, Ljubljana, Slovenia. He is a Senior Member of IEEE. Contact him at marjan.gushev@finki.ukim.mk.

ARMIN HOHENEGGER is an M.Sc. student in computer science at the University of Innsbruck, 6020 Innsbruck, Austria. His research interests include machine learning approaches in parallel and distributed systems. Hohenegger received a B.Sc. from the University of Innsbruck, Austria. Contact him at Armin.Hohenegger@student.uibk.ac.at.

RADU PRODAN is a professor in distributed systems at the Institute of Information Technology, Klagenfurt University, 9020 Klagenfurt am Wörthersee, Austria. His research interests include performance and resource management tools for parallel and

distributed systems. Prodan received a Ph.D. from the Vienna University of Technology. Contact him at radu.prodan@aau.at.

DIMITAR MILESKI is a senior software developer at Innovation Doel, 1000 Skopje, North Macedonia. His research interests include serverless computing. Mileski received an M.Sc. in high-performance computing serverless systems from the Saints Cyril and Methodius University, Skopje, North Macedonia. He is a Student Member of IEEE. Contact him at dimitar.mileski@innovation.com.mk.

PANO GUSHEV is a chief technical officer of Innovation Doel, 1000 Skopje, North Macedonia, and an expert in implementing cloud-based applications. His research interests include cloud-based eHealth solutions. Gushev received an M.Sc. from the Saints Cyril and Methodius University, Skopje, North Macedonia. Contact him at pano.gushev@innovation.com.mk.

GORAN TEMELKOV is a senior software developer at Innovation Doel, 1000 Skopje, North Macedonia, working on implementing specific signal processing and data science eHealth cloud-based solutions. His research interests include cloud-based eHealth solutions. Temelkov received a B.Sc. from Saints Cyril and Methodius University, Skopje, North Macedonia. Contact him at goran.temelkov@innovation.com.mk.

- ECG signal processing use case," in *Proc. 45th IEEE Jubilee Int. Conv. Inf., Commun. Electron. Technol. (MIPRO)*, 2022, pp. 853–858, doi: 10.23919/MIPRO55190.2022.9803568.
- W. Lloyd, M. Vu, B. Zhang, O. David, and G. Leavesley, "Improving application migration to serverless computing platforms: Latency mitigation with keep-alive workloads," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion (UCC Companion)*, 2018, pp. 195–200, doi: 10.1109/UCC-Companion.2018.00056.
 - J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López, "Serverless data analytics in the IBM cloud," in *Proc. 19th Int. Middleware Conf. Ind.*, Rennes, France: ACM, 2018, pp. 1–8, doi: 10.1145/3284028.3284029.
 - C. P. Smith, A. Jindal, M. Chadha, M. Gerndt, and S. Benedict, "FaDO: Faas functions and data orchestrator for multiple serverless edge-cloud clusters," in *Proc. IEEE 6th Int. Conf. Fog Edge Comput. (ICFEC)*, 2022, pp. 17–25, doi: 10.1109/ICFEC54809.2022.00010.
 - R. Singh and S. S. Gill, "Edge AI: A survey," *Internet Things Cyber-Physical Syst.*, vol. 3, pp. 71–92, Mar. 2023, doi: 10.1016/j.iotcps.2023.02.004.
 - M. Gusev, "What makes Dew computing more than Edge computing for Internet of Things," in *Proc. IEEE 45th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, 2021, pp. 1795–1800, doi: 10.1109/COMPSAC51774.2021.00269.