Optimal Scalable Real-Time ECG Monitoring of Thousands of Concurrent Patients

K. Bajalcaliev[1], D. Mileski[12], P. Gushev[1], M. Gusev[12] and B. Jakimovski[2]
[1]Innovation Dooel, Skopje, North Macedonia
[2]Sts Cyril and Methodius University in Skopje, Faculty of Computer Science and Engineering, Skopje, North Macedonia
E-mail: kostadin.bajalcaliev@innovation.com.mk, dimitar.mileski@finki.ukim.mk,
pano.gushev@innovation.com.mk, marjan.gushev@finki.ukim.mk, boro.jakimovski@finki.ukim.mk

***Abstract*—This paper explores the transformation of electrocardiogram (ECG) monitoring from traditional offline to Real-Time analysis, enabled by high-speed mobile networks and affordable data plans. The transition to live monitoring presents challenges in data streaming and processing and the necessity of balancing immediacy with accuracy. We optimize two critical aspects of cloud architecture and scalability under the broader umbrella of cloud efficiency by evaluating the architecture's components and their contribution to overall efficiency. The focus is on accommodating over a thousand concurrent patients streaming ECG data while maintaining cost-effectiveness, constrained by Near Real-Time Round Trip Time (RTT) of $\leq$ 3 seconds, achieving a throughput of $\geq$ 333.333 (msgs/s).**

***Keywords*—cloud efficiency, scalable, cost-effective, near real-time, ECG**

I. Introduction

This paper addresses the challenge of real-time electrocardiogram (ECG) streaming for thousands of patients worldwide, emphasizing critical aspects such as cloud computing scalability, resource utilization optimization, and potential cost reduction for managing this high-frequency data flow, which involves sending ECG data every second for each of the thousands of patients.

The technologies facilitating continuous ECG signal monitoring include ECG sensors, Personal Area Networks (PAN) like Bluetooth, high-processing-power smart devices, and the emergence of smartphones with robust processing capabilities, along with cloud computing. Utilizing cloud computing techniques becomes essential for scaling the entire infrastructure to achieve real-time and continuous monitoring across thousands of patients.

The technologies supporting scalability in the cloud encompass containers, container management and orchestration systems, cloud-native services, and programming languages, distributed event streaming systems for high-performance data pipelines, high-performance in-memory data stores, and specific best practices derived from parallel and distributed processing. This paper focuses on technologies and practices that provide increased cloud scalability.

In prior implementations of Large-Scale Streaming ECG Monitoring for Thousands of Patients, with ECGs streamed every second [1], we observed a considerable expense associated with the Public Cloud [2]. One approach is to use FinOps [3] (Financial Operations) to reduce the cost of the Public Cloud that supports the businesses. In previous implementations, we used a public cloud and proprietary cloud-native services for the entire pipeline. In this paper, we employ a different approach. This paper experimentally identifies architectural components to determine the optimal scalable architecture. The experiments use one msg = 125 samples of ECG data, or 918 bytes.

In this paper, we set the following research questions:

- RQ1: What is the optimal scalable architecture that uses only VM and Cloud Storage that can support Real-Time ECG Monitoring of Thousands of Concurrent Patients, with RTT of $\leq$ 3 seconds, achieving a throughput of $\geq$ 333.333 (msgs/s)?
- RQ2: What is the optimal scalable architecture that minimizes the number of cloud services while maximizing resource utilization, with RTT of $\leq$ 3 seconds, achieving a throughput of $\geq$ 333.333 (msgs/s)?

This paper proposes a limitation by exclusively employing VM and Cloud storage. The VM hosts a serverfull architecture integrating open-source systems and programming languages recognized for their high performance.

Delays between sensor sampling and ECG transmission to the cloud (ranging from 10 to 60 seconds) hinder real-time monitoring capabilities and potentially compromise timely interventions. The high costs of utilizing managed services in public clouds to support a large user base are also significant. Some related work utilizes simple ECG processing techniques, which do not include beat and episode detection, classification, or real-time monitoring for thousands of patients worldwide. Our extensive ECG processing pipeline contributes to increased response times, adding milliseconds and seconds to the overall RTT.

We propose a high-performance system for real-time ECG data transmission and monitoring. By benchmarking combinations of system components with critical metrics, like response time and error percentage, we propose an optimal architecture that can efficiently handle thousands of patients. Our approach simplifies deployment by using a single VM and providing optional Cloud Object Storage.

The paper structure is as follows: Section II discusses the similarities and differences in related work. Section III presents the methods, including the system organization, used technologies, experiments, and the evaluation method-

ology. Results are presented in Section IV and discussed in Section V. Finally, Section VI provides conclusions and future work.

II. Related Work

The system proposed in [4] operates through HTTP client-server architecture. The ECG signal is transmitted from the Smartphone to the cloud using the MQTT protocol, which maintains a long-lived connection between the devices. Their solution incorporates doctors from sensors to smartphone apps to the cloud. Our paper focuses on the optimal architecture for processing signals, not the overall system. They tested the system for six patients, and there is no information about response time or throughput. Our approach streams data for thousands of patients.

In [5] Apache Storm topology, orchestrates the real-time processing. This system functions as a network of spouts and bolts within Storm. Apache Hadoop processes historical data (batch). The study reported average persisting times for ECG/EKG messages: 39 minutes and 30 seconds (±2 minutes and 40 seconds). The average message processing capacity for ECG/EKG messages achieves 126 messages per second (msgs/s) (throughput). In contrast, our paper focuses on ECG monitoring of thousands of real-time patients with an RTT of $\leq$ 3 seconds, achieving a throughput of $\geq$ 333.333 (msgs/s).

The dataset size of 3.2 GB, derived from five EDF files, was batch-processed. The multi-node Cloudwave implementation processed one ECG channel in 19.2 seconds, while the desktop took 62 seconds for the same task; processing one ECG channel in another scenario lasted approximately 25.2 seconds, and processing all four channels required about 94.2 seconds, whereas processing 36 segments of data took about 145.2 seconds, with the desktop's processing time unspecified [6]. It's important to note that our approach differs as we employ real-time processing and handle real-time streaming data from thousands of patients, not batch processing.

Other approaches include serverless computing on the federated cloud using lambda architecture [7], which is different from this approach that focuses on a single cloud provider.

III. Methods

This section encompasses system organization, experiments, and evaluation methodology. It explains the structuring and arrangement of components, determining how they interact and their roles within the system's architecture. Experiments include different combinations of system components and varying numbers of concurrent patients streaming ECG every second to determine the optimal scalable real-time ECG monitoring solution, utilizing evaluation metrics.

A. System organization

The core components for ECG processing are the Beat Detection and Classification (BDC) and the Data Processing Unit (DPU). Part of the DPU's functionality includes preprocessing, BDC invocation, postprocessing, and finalization tasks such as statistical analysis and report generation. It serves as the entity responsible for receiving incoming ECG data streams and processing, storing, and retrieving data locally or in cloud storage. The DPU utilizes patient tokens for authentication. BDC operates independently for arrhythmia detection. This module includes noise detection, data filtering, and identification of specific ECG features.

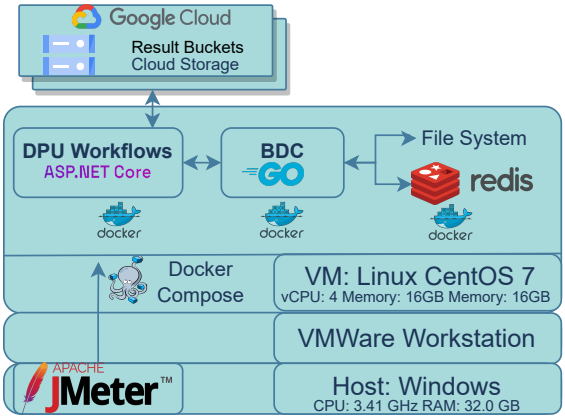


Fig. 1: System organization

The goal of this paper is not to evaluate the system's deployment across different cloud providers. The objective is to identify the optimal and scalable real-time ECG monitoring architecture suitable for thousands of patients, deployed within a single VM (Cloud Region: Skopje, Macedonia) and utilizing Cloud Storage (Cloud region: Europe-west1, Belgium). This research examines the impact of each architecture component (Fig 1) on the RTT.

The cloud solution deploys the Object Storage (Google Cloud Storage) as the only system component, leaving the deployment of object storage, virtual machine (VM) cloud instances, and storage regions on different cloud platforms for future work.

The workload generation and signal processing components are in the same local network on the same physical machine. Apache JMeter runs on a physical machine with a CPU of 3.41 GHz and RAM of 32.0 GB. A series of JMX test files collects the results in a JTL file (CSV format). It is important to emphasize that for JMeter to utilize the hardware resources efficiently, the jmeter.properties file needs the correct configuration, enabling it to send thousands of requests per second using a single-node Apache JMeter.

Linux CentOS 7 VM deploys the solution with the VMware Workstation virtualization software. The VM consists of vCPU with four cores, 16GB memory, and 100 GB disk. VMware supports the export of the VM to OVA (Open Virtualization Appliance) and OVF (Open Virtualization Format), as well as VM disks in VMDK format. Conversion tools can convert VMDK to VHD

TABLE I: Experiments Table

Experiment	Deployment	Patients	Duration (min)
E1	BDC Files	1, 10, 25, 50, 100, 150, 200, 500, 1000	10
E2	BDC Redis	1, 10, 25, 50, 100, 150, 200, 500, 1000	10
E3	DPU Workflows + BDC Redis	1, 10, 25, 50, 100, 150, 200, 500, 1000	10
E4	DPU Workflows + Cloud Storage + BDC Redis	1, 10, 25, 50, 100, 150, 200, 500, 1000	10

format, facilitating migration to different cloud platforms.

Docker containers deploy all components of the ECG processing pipeline, managing and orchestrating the containers with Docker Compose as a lightweight tool for multi-container applications on a single host. We do not utilize Kubernetes because we rely on a single host (VM). Kubernetes is a robust container orchestration platform for managing containerized applications at scale across a distributed cluster of hosts. Instead, we employ Docker Compose because it is exclusively for single-host setups. While Kubernetes offers the flexibility of managing single or multiple hosts, it is notably more complex than Docker Compose. The pipeline (Fig 1) uses a Docker Compose YAML file, specifying services, networks, volumes, environment variables, and other necessary configurations for the components of the ECG processing pipeline. The Docker Compose consists of the following services:

1) DPU Workflows: The DPU data processing unit service has a single Docker image to prepare and collect data from BDC, incorporating other ECG processing algorithms. The configuration parameters in DPU Workflows specify whether to invoke only BDC and whether patient data can be retrieved and stored from the cloud or local storage (Linux file system), which is crucial for conducting various experiments.

2) BDC: represents a proprietary ECG beat detection and classification service developed in the Go programming language to process patient data streams. Patient data streams generate 1-second ECG signals every second for each. During the initial activation, BDC collects data requiring warming up with a specific window size to start with proper detection and classification. Afterward, each request to BDC returns annotations of beats and ECG episodes and continues processing patient streams. Each patient stream uses a unique PatientToken to identify patients in DPU workflows, BDC, Redis, and storage.

3) Redis: integrates caching or storing intermediary data from the BDC service. A single-instance Redis deployment operates as a standalone server without clustering or replication mechanisms.

The experiments specify two types of storage: local storage (Linux file system) and Google Cloud Storage (Object Storage), which are configured by parameters in the request. DPU Workflows and BDC employ local storage, while DPU Workflows exclusively use cloud storage.

B. Experiments

Each deployment and patient scenario undergoes a test duration of 10 minutes (Table I). Five deployments define the experiments E1 to E5. Workloads from 1 to 1000 simultaneous patients specify test cases as patient scenarios. The total execution time for all experiments is 360 minutes or 6 hours for nine patient scenarios (test and five deployments. Table I presents the destination component for JMeter requests. JMeter directly sends requests to BDC in experiments E1 and E2. E3 (DPU Workflows + BDC Redis) and E4 (DPU Workflows + Cloud Storage + BDC Redis).

C. Evaluation methodology

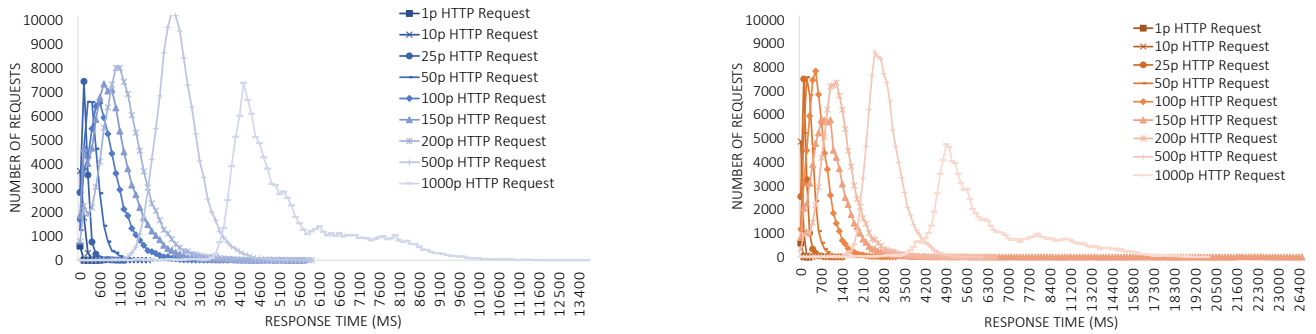
The experiment evaluation metrics are response time, throughput, and response error percentage.

- **Response Time Distribution**: Represents the number of requests distributed across different response time intervals (in milliseconds), providing insights into the distribution of response times across the application's workload.
- **Response Time Percentiles**: Offers a detailed breakdown of response times at different percentiles (e.g., 90th or 95th percentile), indicating the response time below which a certain percentage of requests fall.
- **Error percentage**: Indicates the percentage of requests that resulted in errors or failures, helping to assess the overall reliability and stability of the system.
- **Transactions per second**: Measures the system's throughput by quantifying the number of transactions processed per unit of time, providing valuable information about the system's capacity and ability to handle specific workloads.

IV. Results

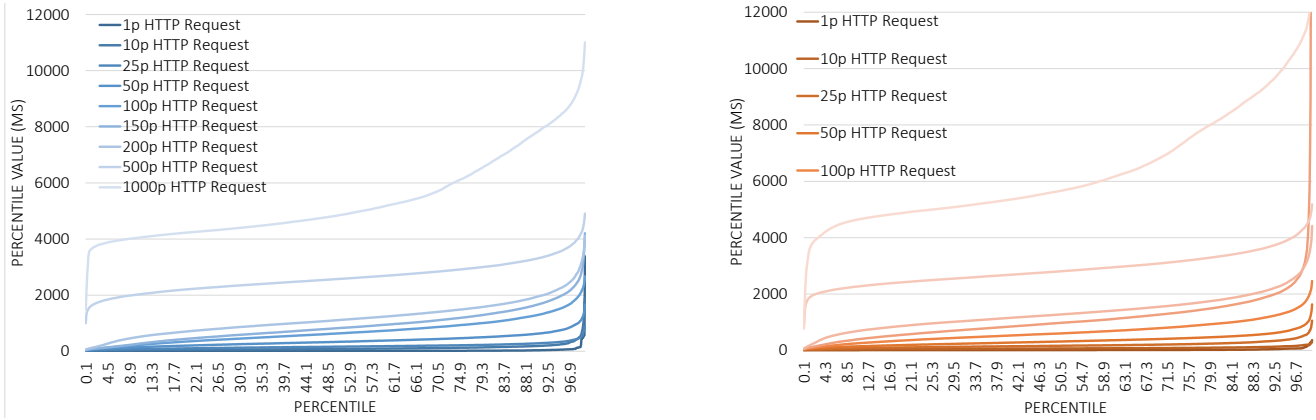
The results are grouped into BDC and DPU categories to measure how each component affects the system's overall performance. Fig. 2 and Fig. 6 display the Response Time distribution, Fig. 3 and Fig. 7 show Response Time percentiles, Fig. 4 and Fig. 8 illustrate Transactions per second, and Fig. 5 and Fig. 9 depict Error percentage.

Response time distribution on the x-axis represents response time, while the y-axis represents the number of requests with that response time. Response time percentiles on the x-axis indicate the percentile, and the y-axis displays response time in milliseconds. Transactions per second on the x-axis represent the experiment's duration in minutes, while the y-axis represents the number of transactions per second (throughput). Error percentage displays the percentage of all requests that returned an error (were not processed successfully).



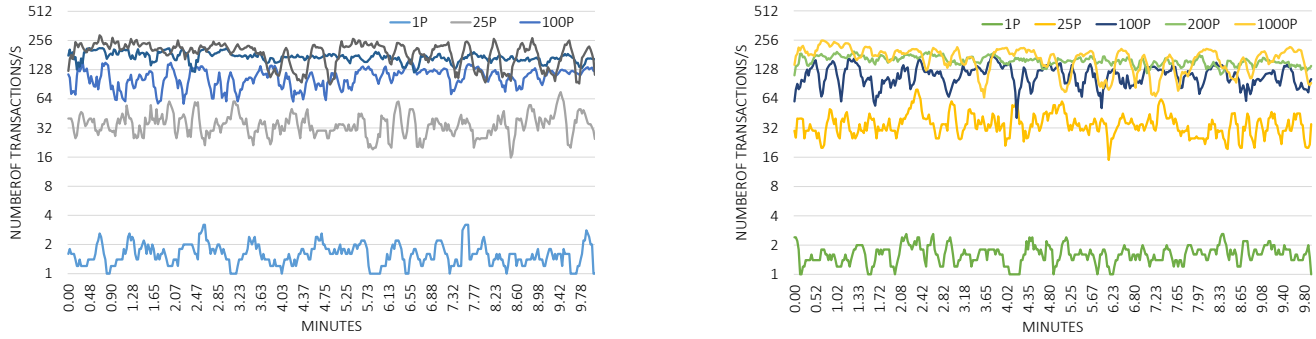
(a) Files (b) Redis

Fig. 2: BDC - Response time distribution



(a) Files (b) Redis

Fig. 3: BDC - Response time percentiles



(a) Files (b) Redis

Fig. 4: BDC - Transaction per second

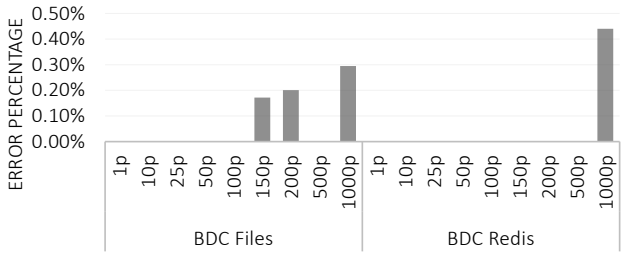


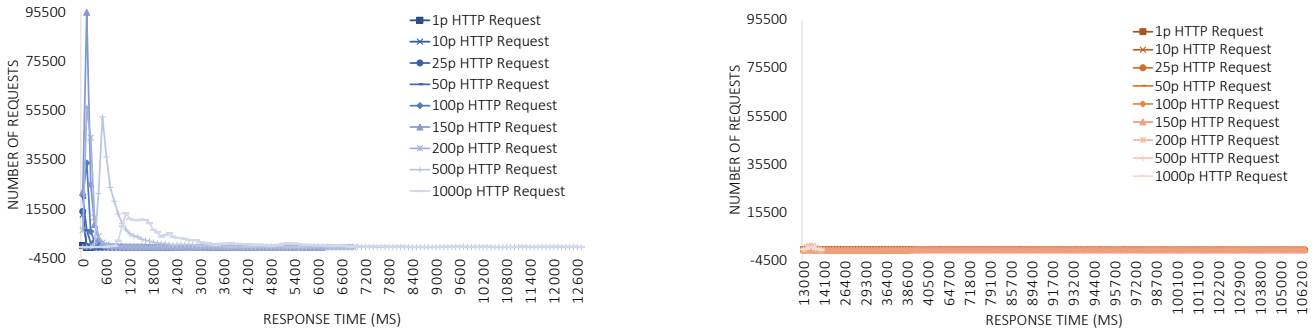
Fig. 5: BDC Files - Error percentage

V. Discussion

BDC integrates two operation modes: one with databases and the other with Redis.

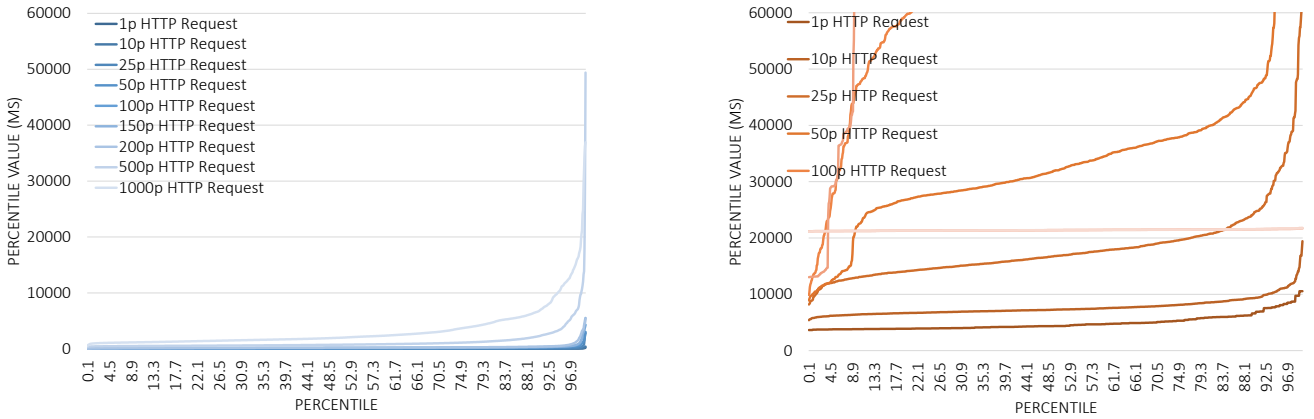
A. Analysis of E1

In the E1 experiment deploying BDC files, all requests below 200 patients (200 concurrent streams) are completed in less than 3 seconds. During the 10-minute experiment, 120,000 requests were sent for 200 patients at a rate of 200 requests per second. Only 0.07% of requests were completed between 3 to 6 seconds, while the others were under



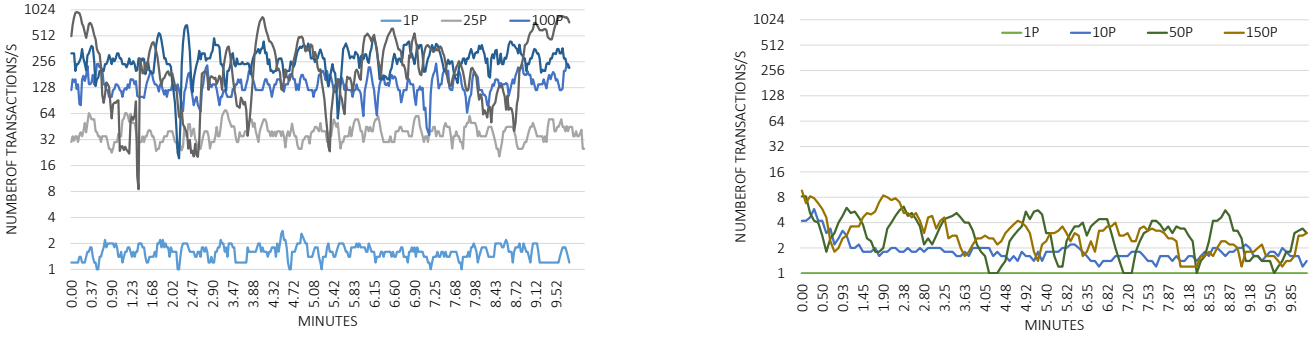
(a) DPU (b) DPU Cloud Storage

Fig. 6: DPU - Response time distribution



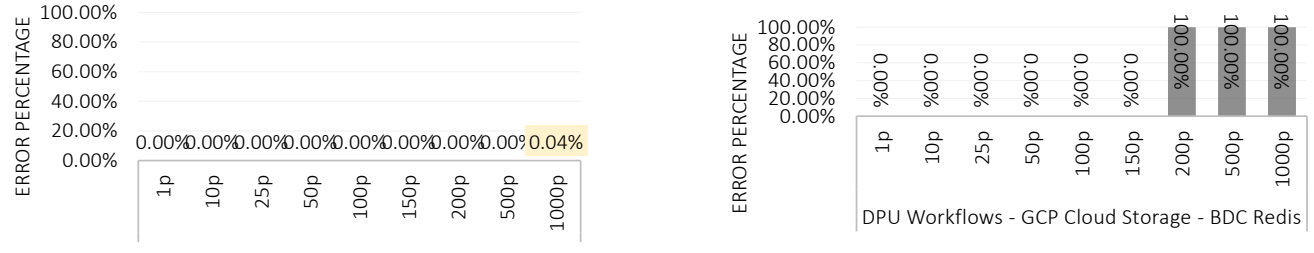
(a) DPU (b) DPU Cloud Storage

Fig. 7: DPU - Response time percentiles



(a) DPU (b) DPU Cloud Storage

Fig. 8: DPU - Transactions per second



(a) DPU (b) DPU Cloud Storage

Fig. 9: DPU - Error Percentage

3 seconds. For 500 patients (500 requests per second), the highest number of requests took 2500 ms, indicating that BDC Files satisfy Near Real-Time Processing ($\leq$ 3s) up to 500 patients. However, for 1000 patients, the most significant number of requests took 4200 ms, failing to meet the condition of $\leq$ 3s condition.

B. Analysis of E2

The competition response time for all requests of up to 200 patients was less than 3 seconds in the E2 experiment with BDC Redis deployment. Only 0.96% completed between 3 to 7 seconds, and all others under 3 seconds. Similar to E1 results (BDC Files), for 500 patients, the highest number of requests took 2500 ms, satisfying Near Real-Time Processing ($\leq$ 3s) up to 500 patients. However, for 1000 patients, the most significant number of requests took 5000 ms, not meeting the $\leq$ 3s condition.

The distribution of results is normal (Gaussian). Although the results show a symmetric distribution for 500 patients, it is a right-skewed or positively skewed distribution for 100 patients, indicated by a longer tail on the right side representing more data points with higher values than expected in a perfectly symmetrical distribution. Fig. 3 presents the response time in percentiles.

C. Analysis of errors

Regardless of whether it is BDC Files or BDC Redis, the error percentage is less than 0.5%. Errors appear in the E1 experiment (BDC Files) for more than 150 patients and in the E2 experiment (BDC Redis) for only 1000 patients. Converted to the number of requests that ended with an error, this equates to 0.44% of the total 120,000 requests for 1000 requests per second sent over 10 minutes, totaling 528 requests. Further, this error rate implies that 528 seconds (8.8 minutes) of ECG signal are not processed out of 120,000 requests (33 hours) by Redis and BDC. Cumulatively, BDC Files has an error percentage of 0.67%, and BDC Redis has 0.44%.

Errors in the BDC Redis deployment appear only for 1000 patients, which is negligible at 0.04%. A single factor contributing to errors in BDC Redis was the high volume of requests, rather than malformed data or requests requiring additional analysis.

D. Analysis of E3 and E4

Due to a lower error rate, Redis is preferable for the rest of the experiments E3 and E4 (Table I). In addition, Redis is a better scalability option, supporting a multi-node cluster to scale the system. Additionally, there is a lot of room for improvement for Redis, such as different in-memory database mechanisms like replication and sharding, numerous hosted Redis servers, and supporting a plain Linux file system.

Increasing the number of requests above 500 takes longer than 3 seconds, indicating that adding other components, such as DPU Workflows, will not satisfy the response times to be $\leq$ 3s.

E. Overall analysis

Evaluating the results within the scope of the research questions, we concluded the following:

RQ1: Results showed that employing BDC Redis, BDC Files, and DPU Workflows without Cloud Storage achieved an approximate throughput of 250 msg/s out of the targeted 333.333 msg/s, enabling the streaming of Real-Time ECG data for 750 patients per second out of the targeted 1000.

RQ2: Findings revealed that utilizing only VM cloud service improved performance compared to cloud storage. Cloud storage resulted in significantly lower throughput, equivalent to streaming ECG data for 24 patients per second out of the targeted 1000.

VI. Conclusion and Future Work

This research addressed designing an optimal scalable architecture for Real-Time ECG Monitoring of Thousands of Concurrent Patients using only Virtual Machines (VMs) and Cloud Storage. The study aimed to achieve a Round-Trip Time (RTT) of $\leq$ 3 seconds and a throughput of $\geq$ 333.333 msg/s by minimizing the number of cloud services while maximizing resource utilization. We achieved approximately 250 msg/s for BDC Redis and BDC Files, and (DPU Workflows - BDC Redis), effectively streaming Real-Time ECG data for 750 patients every second. The worst results occurred when using cloud storage, with about eight msg/s, equivalent to real-time streaming ECG data for 24 patients every second.

Future work for the optimal scalable real-time ECG monitoring system involves exploring deployment across different public cloud platforms, assessing various object storage solutions, scaling with multi-node Redis clusters, and integrating Kubernetes for container orchestration.

References

[1] M. Gusev, S. Ristov, A. Amza, A. Hohenegger, R. Prodan, D. Mileski, P. Gushev, and G. Temelkov, "Cardiohpc: Serverless approaches for real-time heart monitoring of thousands of patients," in *2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2022, pp. 76–83.

[2] D. Mileski and M. Gusev, "Finops in cloud-native near real-time serverless streaming solutions," in *2023 31st Telecommunications Forum (TELFOR)*. IEEE, 2023, pp. 1–4.

[3] J. Storment and M. Fuller, *Cloud FinOps*. " O'Reilly Media, Inc.", 2023.

[4] M. L. Sahu, M. Atulkar, M. K. Ahirwal, and A. Ahamad, "Iot-enabled cloud-based real-time remote ecg monitoring system," *Journal of medical engineering & technology*, vol. 45, no. 6, pp. 473–485, 2021.

[5] D. Chen, Y. Chen, B. N. Brownlow, P. P. Kanjamala, C. A. G. Arredondo, B. L. Radspinner, and M. A. Raveling, "Real-time or near real-time persisting daily healthcare data into hdfs and elasticsearch index inside a big data platform," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 595–606, 2016.

[6] S. S. Sahoo, C. Jayapandian, G. Garg, F. Kaffashi, S. Chung, A. Bozorgi, C.-H. Chen, K. Loparo, S. D. Lhatoo, and G.-Q. Zhang, "Heart beats in the cloud: distributed analysis of electrophysiological 'big data'using cloud computing for epilepsy clinical research," *Journal of the American Medical Informatics Association*, vol. 21, no. 2, pp. 263–271, 2014.

[7] S. Ristov, M. Gusev, A. Hohenegger, R. Prodan, D. Mileski, P. Gushev, and G. Temelkov, "Serverless ecg stream processing in federated clouds with lambda architecture," *IEEE Computer*, vol. to be published, 2023.