Performance Analysis of ECG Processing on Mobile Devices

Monika Simjanoska[1][0000−0002−5028−3841], Bojana Koteska[1][0000−0001−6118−9044], Marjan Gusev[1][0000−0003−0351−9783], and Ana Madevska Bogdanova[1][0000−0002−0906−3548]

Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University
Rugjer Boshkovik 16, PO Box 393, 1000 Skopje, Macedonia
{monika.simjanoska,bojana.koteska,
marjan.gushev,ana.madevska.bogdanova}@finki.ukim.mk

Abstract. Recently, there is a great focus on preventive medicine and consequently, lots of telemedicine applications have been developed for real-time health monitoring. The monitoring requires a set of biosensors and tablets, smartphones or other mobile devices. The biosensors continuously scan and transmit data and the mobile devices have the capacity to process the data and at the same time to save the battery as much as possible. In this paper, we assess the performance of an algorithm for ECG derived heart rate and respiratory rate. Three different scenarios are inspected encompassing local, remote Linux server, and remote MatLab©server processing of the locally collected ECG signals by using the biosensors attached to a person. We set a hypothesis that processing locally instead of remotely is worse in terms of time and power consumption. Thus, developing a web service for processing will be best in both time and battery consumption. A total of 1297 signals have been processed. The results show that the mobile device is appropriate for processing ECG files up to approximately 5MB size. The processing of larger files spends too many resources when compared to the remote processing solution. Additionally, we have proved that splitting large files in chunks up to 4 MB per chunk resulted in faster processing for the local scenario.

Keywords: ECG · Algorithm · Signal processing · Mobile Platform.

1 Introduction

Preventive rather than curative medicine has brought small laboratories into people's own devices. Biosensors are no longer high-cost equipment available only to the hospital units or to the great research centers. Instead, there is a trend of developing smaller units to measure the persons' vital parameters without degrading their comfortability. Those biosensors are able to capture the persons' signals from various sources such as heart electrical activity (ECG), heart sounds (PCG), oxygen saturation level (PPG), brain electrical activity

(EEG), etc. The signals are continuously transmitted and consequently need to be processed in real-time to extract reliable data of the current health condition.

ECG is one of the most popular physiological signals that can accurately provide an insight into the individual's heart rate, respiratory rate, and even blood pressure (mostly combined with other signals). This ability makes the ECG to be widely used in cases of rapid diagnosis when no other equipment is available, meaning it is suitable to use a mobile device as a processing unit [10]. However, the up-to-date mobile devices still lack the storage and processing power of the "big" computers and therefore, we must take into account the limits of the storage, processing, and battery when creating algorithms for real-time ECG signals processing.

Usually, the biosensors do not have their own memory and these data are sent to a portable electronic device located close to the sensor (for ex. mobile phone or tablet). A mobile device can receive real-time data from several biosensors attached to a patient and are used to monitor the health condition. This is a typical case of measuring vital parameters when a patient wears a portable biosensor that monitors ECG. The final goal is to monitor the health condition and to save the phone battery as much as possible.

In order to provide an accurate and fast algorithm performance preserving a long battery life, the huge dilemma is where to execute the algorithm: locally on the mobile device or on a remote server. For example, if a patient wants to check the heart rate and respiratory rate after every 30 minutes, the dilemma is: to send the ECG data from the customer's phone to a remote server and execute the algorithm remotely or to execute the algorithm on the customer's phone.

We set a hypothesis that local processing performs worse than remote Linux server and remote MatLab©processing in terms of storage, computation, and consequently battery consumption. It is expected that developing a web service will provide faster processing time; however, the effect of the server response times on the energy consumption of mobile clients [16] still has to be investigated. In order to confirm the hypothesis, we conducted an experiment to evaluate the algorithm's performance and battery consumption of the mobile device.

Three scenarios are included: local processing on a mobile device, own remote Linux-hosted web service, and a remote MatLab©-hosted service.

Further on, we set up a research problem to find out the most time efficient and battery efficient way for real-time processing of an ECG signal and calculating ECG derived heart rate and respiratory rate.

The experiment includes testing of various ECG signal lengths from three different databases relied on different ECG sources in terms of sampling frequency, including Cooking Hacks e-Health Sensor Platform [11], Savvy Sensor Platform [19] and Physionet [23].

The rest of the paper is organized as follows. Related work in the area of ECG processing algorithms designed for portable devices is given in Section 2. Section 3 presents the testing methodology, plan and infrastructure. The results from the experiments are described in Section 4. Section 5 is dedicated to a discussion of the outcome and time processing trade-off, comparison of the results

and analysis which environment provides the best processing time and longest battery life. The conclusion and future work are specified in Section 6.

2 Related Work

Parak and Havlik stress the importance of developing an approach that saves computing time, aiming to achieve suitable microprocessor implementation [17].

Elgendi at al. [7] recommend a first derivate of the flittered ECG signal for long-term processing of ECG recordings and ECG large databases since it is highly numerically efficient for QRS detection phase.

Elgendi presented a mobile phone application for QRS complex identification out of large recorded ECG signals and achieves a processing time of 2.2 seconds for an ECG signal of 130 minutes [6].

A novel mobile-friendly approach for detection of QRS complexes from a high fidelity ECG real-time monitoring is provided in [2]. The authors used continuous wavelet transform which resulted in lesser computation time compared to Pan-Tompkins algorithm. Li at al. [14] proposed a low power and efficient QRS processor for real-time ECG monitoring based on harr wavelet transform and optimized FIR filter structure. Power consumption is only 410 nW in 1 V voltage supply.

Lahiri et al. proposed an algorithm for real-time compressing ECG waveforms and send them through a mobile phone. They compare their algorithm performance with several existing compression algorithms and concluded that their algorithm can achieve good compression with low distortion on the receiving side [13]. Gia at al. [8] analyzed ECG signals in smart gateways with wavelet transform mechanism. Their experimental results showed that fog computing helped to achieve more than 90% bandwidth efficiency and low-latency real-time response at the edge of the network.

Mazomenos et al. [15] introduced a low-complexity algorithm for fiducial points from ECG designed for mobile healthcare applications. They have applied discrete wavelet transformation as a principal analysis method. Their algorithm was applied to 477 ECG recordings and achieved an ideal trade-off between computational complexity and performance.

Ibaida et al. [12] proposed a compression algorithm to minimize the traffic going to and coming from cloud ECG compression. Their technique achieved a higher compression ratio of 40 with lower Percentage Residual Difference (PRD) Value less than 1%. Wang et al. [22] developed a cloud-based system for performing massive ECG data analysis. Their system showed up to 38x performance improvement and 142x energy efficiency improvement compared to commercial systems. A cloud system providing real-time ECG data acquisition, transmission and analyzing over the Internet was presented in [24]. Evaluation of this system showed that it has good performance in terms of real-time, scalability and latency.

A low-cost wireless ECG data acquisition system capable of acquiring up to 22 hours of continuous ECG data is described in [1]. The authors use Arduino

Uno and Bluetooth serial communication to real-time ECG data transfer. Their system is portable and battery powered. Wand et al. [21] proposed a mobile-cloud computational solution for ECG telemonitoring. Their approach enhances the mobile medical monitoring in terms of accuracy, execution and energy efficiency.

Pineda et al. [18] presented a portable system that enables the acquisition, processing, storage, and transmission of ECG signal. Their system can record ECG data over 120 hours in local mode, up to 10 days of ECG data could be stored in the phone in the cellular mode. For the remote signal visualization, the average delay of the packets was less than 1.73ms, with a mean power consumption of 0.48 w/h, using a battery of 2 A/h.

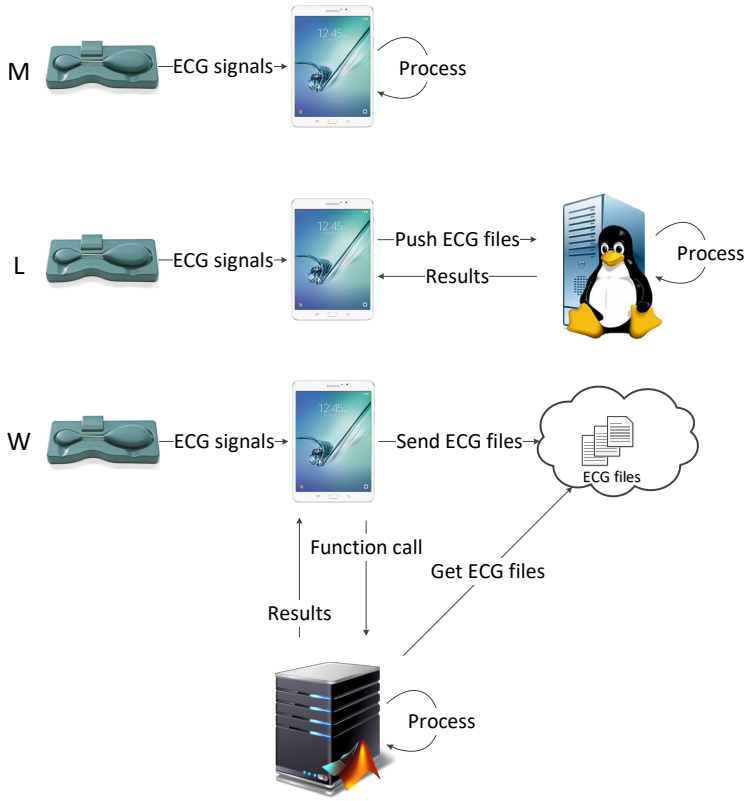

Fig. 1. The three architectural scenarios to conduct the testing

3 Testing Methodology

This section presents the testing environments, test cases, and test data.

3.1 Test Goal

The test goal is to determine the most efficient scenario in terms of performance (execution time) and resource saving (battery) to execute an ECG-derived heart rate and respiratory rate algorithm.

We implemented the ECG-derived heart rate and respiratory rate algorithm relying on the idea presented by Ding et al. [5]. The algorithm accepts two parameters: a file path where the ECG signal is stored and the sampling frequency at which the signal is collected (this option is necessary since various biosensors work with different sampling rates).

The pseudo code of the algorithm is presented in Fig. 2.

```
1: Input: signal, frequency
2: Output: HR, num_respirations, RR
3: function H3R(signal[], frequency)
4:     num_of_peaks = Pan_Tompkins (signal, frequency)
5:     seconds = round(length(signal)/freq)
6:     HR = 1/(seconds/num_of_peaks) * 60
7:     for i = 1 to num_of_peaks-1 do
8:         k = kurtosis(signal[peak[i] : peak[i + 1])
9:     smoothed_k = smooth(k, rlowess)
10:    RR_num_of_peaks = findpeaks(smoothed_k)
11:    num_respirations = length(RR_num_of_peaks)
12:    RR = 1/(seconds/breaths) * 60
```

Fig. 2. A pseudo code of the ECG-derived heart rate and respiratory rate algorithm

3.2 Test Scenarios

Figure 1 presents the testing environment and the architecture of three scenarios, according to the location where the processing is realized.

- *Local mobile-only scenario (M)*, where the algorithm is executed on the Android mobile device. The algorithm is implemented as a C source code by using the Android Native Development Kit (NDK).
- *Remote Linux server scenario (L)* where a shell script is used to transfer the locally collected ECG signals stored on the mobile device to a Linux server to execute the algorithm on the server. In this scenario, the C algorithm source code is compiled and executed on the Linux machine. The shell script is executed on the Android phone by using an Android terminal application.

– *Remote MatLab©web service scenario (W)*, where the processing is performed on the recently developed MatLab©Mobile application for Android and connected it to a Windows server with MatLab©installed. The locally collected ECG signals stored on the mobile device have been automatically synced to the server and accessed by the MatLab©based version of the algorithm. In this scenario, we used the originally developed algorithm's MatLab©version.

3.3 Testing Environment

We use Samsung Galaxy Tab S2 - tablet with OS Android 6.0 (Marshmallow) as a testing environment for the experiments. This tablet has 3GB RAM, 64GB internal memory, Octa-core (4x1.9 GHz & 4x1.3 GHz) processor and non-removable Li-Ion 5870 mAh battery.

In the scenario M, the algorithm is executed on the tablet, with a specially developed Android application. The algorithm C source code is added to the Android application by compiling it into a native library using Android Development Studio that Gradle can package with the APK file. Then, in the Java code, the functions in the native library can be called through the Java Native Interface (JNI). Our application only makes a call to the algorithm function by providing two parameters: the ECG recording file path from the tablet storage and the frequency (sampling rate at which the ECG signal is collected).

The scenario L is based on sending the ECG recordings to a remote machine in a *local computer network* for the further processing. We have created a shell script that connects to the remote server, transfers the recording files to a remote server for further processing and stores the file transfer time stamp. In order to calculate the CPU time used by each process, we sum up the User and Sys time. The remote server is a Linux machine (OS-CentOS 7.2) with octa-core processor and 8GB RAM. In order to execute the shell script on the Android tablet, we used the Android Terminal Emulator application which provides access to the Android's built-in Linux command line shell.

The scenario W uses computation offload by sending ECG recordings to a remote machine for the further processing. In this scenario, we used the Android MatLab©Mobile application [20] and connect it to a Windows server where MatLab©software is being installed. The windows server (OS Windows 7) has a quad-core processor (1.80GHz) with 4GB RAM. We use Tonido Server personal cloud [4] which allows us to transfer the files from the tablet to the remote server. The ECG recordings stored on the tablet are automatically synced to the server by the Android Tonido application [3]. Then, they are accessed by the MatLab©based version of the algorithm running on the server. In this scenario, we used the originally developed algorithm's MatLab©version.

Tablet battery consumption is evaluated by using the Android GSam Battery Monitor application [9]. The battery is fully charged at the beginning of each test scenario.

3.4 Test Cases

To cover variable lengths and sources of ECG, we used three different databases and a total of 1297 ECG recordings divided in the following categories:

1. *Short term measurements - small files.* A total of 1191 ECG recordings with varying length from 10 to 90 seconds at a sampling rate of 125Hz are created by using the Cooking Hacks e-Health Sensor Platform. This platform allows Arduino and Raspberry Pi to interact with the shield and be used for biometric and medical applications. The ECG Sensor in this set is a single-lead ECG which uses three ECG electrodes. These basic leads acquire enough information for rhythm-monitoring, but are insufficient for determination of ST elevation (since there is no lead that gives information about the anterior wall) [11].
2. *Medium term measurements - medium file length.* A total of 22 ECG recordings with varying length from 300 to 800 seconds at 125Hz are created by using the Savvy Sensor Platform. SAVVY ECG is a portable medical device for continuous and accurate monitoring of heart rhythm. It is a comprehensive and reliable solution for patients with cardiovascular diseases, athletes, the elderly and for people with an active lifestyle who want to monitor their own health [19].
3. *Large term measurements - large file length.* A total of 84 ECG recordings with varying length from 2860 to 5170 seconds at 360Hz are obtained from the Massachusetts General Hospital-Marquette Foundation Hemodynamic and Electrocardiographic Database (MGH/MF) Physionet database [23].

Our test cases include files of different size from all databases: starting from 4KB up to 52KB from the first database, 59KB up to 9,571KB from the second database, and 8655KB up to 17273KB from the third database. A total of 1.65 GB data have been processed in each experiment.

3.5 Test Data

The measured data in the experiment is the execution time and battery consumption. Further on, we calculate the average algorithm time efficiency per file size and total algorithm time efficiency for all files.

In order to calculate the average algorithm time efficiency, we split the files into categories according to the file size and find the average algorithm performance behavior. The total algorithm time efficiency is the sum of the algorithm time performances for all 1297 files. The average algorithm time efficiency and total algorithm time efficiency are calculated separately for each testing scenario.

The total execution time is defined by (1) where T_i is the execution time for each file i and n is the total number of files.

$$T_{total} = \sum_{i=1}^{n} T_i \tag{1}$$

T_{avg} is average execution time for each unique file size in KB, defined by (2), where k is the number of files of equal file size s.

$$T_{avg}(s) = \frac{\sum_{i=1}^{k} T_i(s)}{k} \tag{2}$$

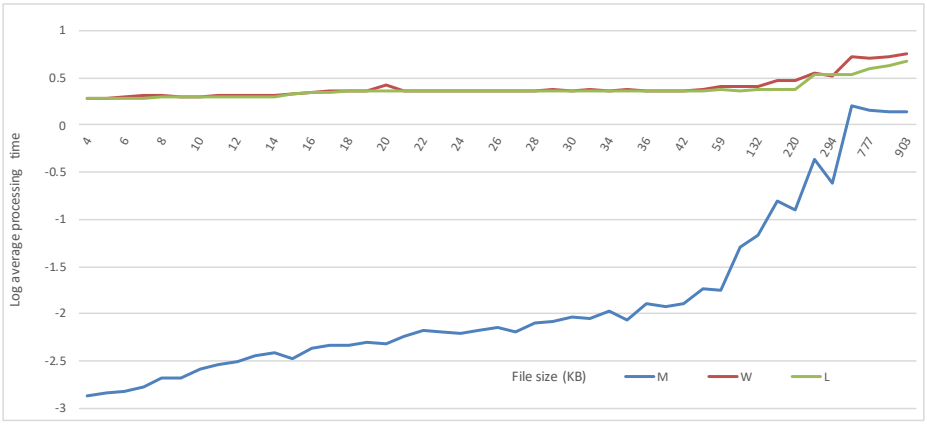

Fig. 3. Average execution times (small files category).

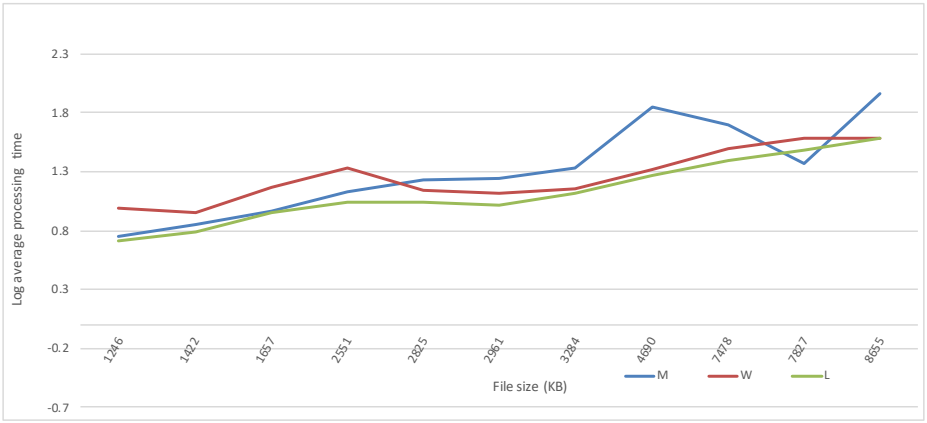

Fig. 4. Average execution times (medium files category).

Note that the average times T_{avgM}, T_{avgL} and T_{avgW} are calculated for each scenario M, L, and W correspondingly.

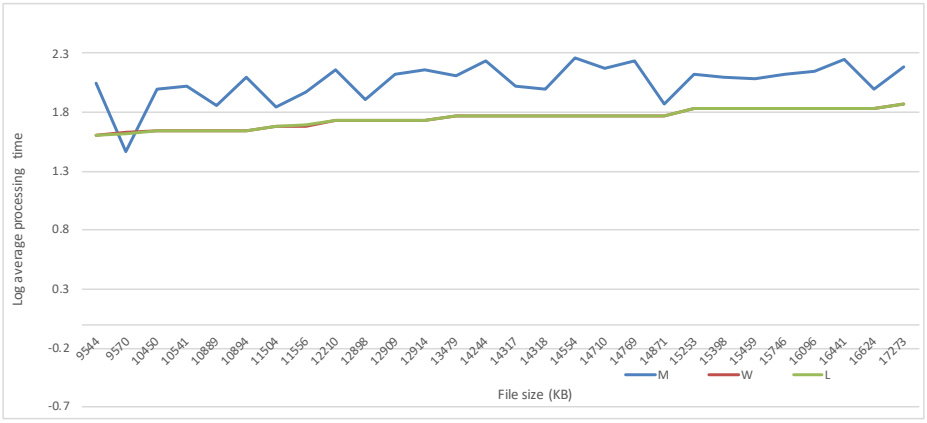

Fig. 5. Average execution times (large files category).

We assume the scenario M is the referent one and compare the results obtained for L and W scenarios. The relative speedup RP is a ratio of average execution times for each pair of test scenarios (3) by calculating RP_{ML} and RP_{MW} correspondingly.

$$RP_{ML} = \frac{T_{avgM}(s)}{T_{avgL}(s)} \qquad RP_{MW} = \frac{T_{avgM}(s)}{T_{avgW}(s)} \tag{3}$$

Additionally, we calculate the speedup RP_{LW} by (4) to compare the scenarios L and W.

$$RP_{LW} = \frac{T_{avgL}(s)}{T_{avgW}(s)} \tag{4}$$

The total battery consumption is calculated for 1297 processed files per each of the M, L and W scenario. The total battery consumption is defined by (5) where B_i is execution time for each file i and n is the total number of files.

$$B_{total} = \sum_{i=1}^{n} B_i \tag{5}$$

4 Experimental Results

This section presents the results obtained from the defined testing scenarios.

4.1 Total performances

A total of 1.65GB data is processed and the summary results are provided in Table 1.

Table 1. Summary of the execution time and battery consumption.

Scenario	T_{total} (sec)	B_{total} in (%)
M	10551.06	91
W	7715.07	36
L	7524.89	30

The first column presents each scenario, the second shows total execution time per scenario, and the third represents the total battery consumption. The average execution time has been calculated for each of the distinct file sizes ranging from 4KB - 17MB.

Analyzing the results, the scenario M is the worst case where the total time for processing the ECG files is 10551 seconds and 91% of the device battery is spent. Even though the processing times for scenarios W and L do not differ significantly (W performs 2.5% worse than L); however, the battery consumption at W case is ∼16% worse when compared to the L scenario.

One can conclude that the ECG data transfer to a remote execution of the algorithm on the Linux server to be the fastest and most efficient scenario in terms of battery consumption.

4.2 Dependence on file size

We have inspected the execution times in all three scenarios considering the size variability of the ECG files. To represent the very small execution times versus the huge values, we use a 10-base logarithmic normalization of the average execution times per distinct ECG file size. Considering the file sizes available for our research, we can clearly distinguish three categories which we refer to: a small files category containing files less than 1MB in size, a medium files category with sizes $\geq$ 1MB and $\leq$ 9MB, and a large files category with files $\geq$ 10MB. Figures 3, 4 and 5 present the normalized average execution times (y-axis) for each file size in each category (x-axis) correspondingly for the M, L and W scenarios.

The figure clearly shows that the M scenario performance follows a logarithmic increasing trend. Although it performs extremely better for file sizes up to 5MB, it is worse for the larger files sizes. Considering the L and W scenarios, it can be perceived that they both follow the same performance trends proportional to the rise of the file size. Even though the scenario M is ∼1.3 times faster than the scenarios W and L for files up to 5MB, it is ∼2 times slower for large files and considering the absolute values of the difference in times to be measured in hundreds and thousands of seconds. The scenario M is not acceptable for unlimited ECG files sizes in terms of processing time.

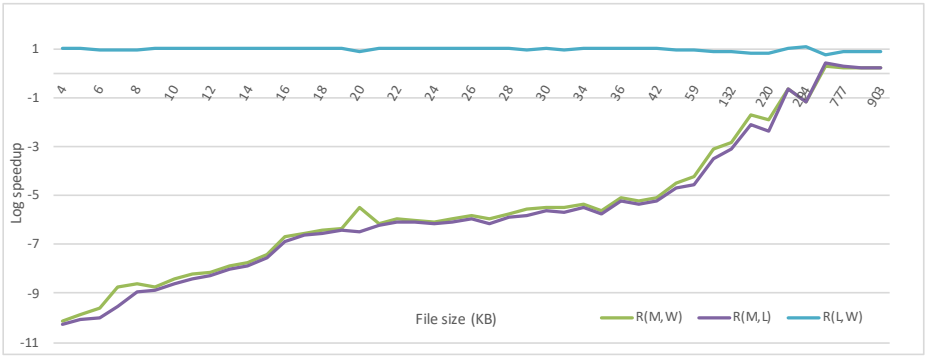

Fig. 6. Relative speedup (small files category).

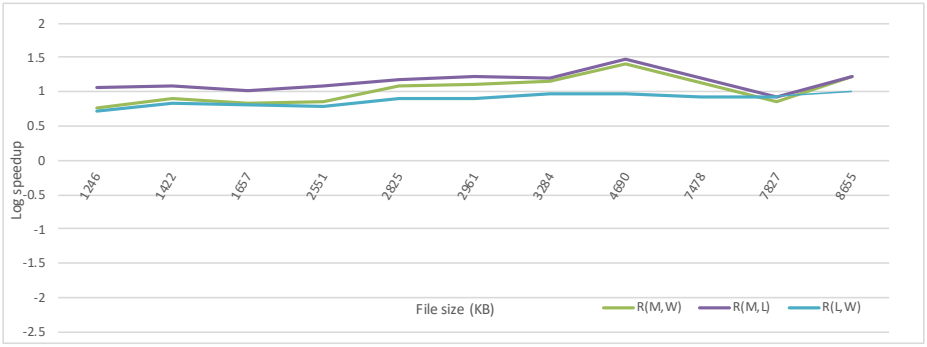

Fig. 7. Relative speedup (medium files category).

Fig. 8. Relative speedup (large files category).

5 Discussion

5.1 Speedup analysis

Figures 6, 7 and 8 present the relative speedup for all pairs of scenarios correspondingly. To represent the high range of the values we also use a logarithm scale with base 10.

Table 2. Comparison of processing times considering chunks.

File size	# chunks	Chunk size	T_M (sec)	T_W (sec)	T_L (sec)
5MB	1	5MB	25.78	25.32	24.18
5MB	2	2.5MB	17.98	36.80	35.14
5MB	5.000	1KB	0.31	2505.90	2393.20
1GB	1	1GB	80954.00	290.47	277.40
1GB	2	500MB	56466.00	422.18	403.18
1GB	200	5MB	5156.80	5063.70	4835.80
1GB	250	4MB	4592.10	5711.50	5454.40
1GB	1.000.000	1KB	61.66	501190.00	478630.00
60GB	1	60GB	40785000.00	1914.10	1828.00
60GB	2	30GB	28448000.00	2782.10	2656.90
60GB	12.000	5MB	309410.00	303820.00	290150.00
60GB	15.000	4MB	275530.00	342690.00	327270.00
60GB	60.000.000	1KB	3699.60	30071000.00	28718000.00

One can observe that there is a linear speedup of M vs W and M vs L for file sizes approximately up to 5MB large. Above this size value, the speedup is stabilized and non-increasing. Analyzing the relative speedup of L vs. W scenario, we can see that W is never better than L and the value is approximately 0, except for small range where L scenario is better, leading to a conclusion that there is no significant difference between the execution times.

5.2 File size dependence

Table 2 shows the processing times for the three scenarios would take 40785000 (M), 1914 (W) and 1828 (L) seconds. The processing times if we have divided the file in 2 chunks of 30GB each, would decrease to 28448000 seconds in the M scenario. However, it would increase to 2782 and 2656 seconds in W and L scenarios correspondingly. Considering 5 MB to be the threshold at which the processing times for all the three scenarios are expected to be nearly equal, dividing a 60GB file in 12000 chunks of 5MB each has resulted in the expected outcome of nearly same processing times. In order to prove that if the chunk size is less than 5MB we have split the large file into 15000 chunks of 4MB each and obtained fastest processing time for the M scenario. Even more, considering the smallest chunk size to be 1KB, the results also showed that the processing time is best in favor of the M scenario. Testing also for 1GB and 5MB file sizes we can conclude the following:

1. If there is no Internet connection available, files must be processed locally on the mobile device (scenario M) and should be split into chunks of the smallest size of ∼1KB;
2. If an Internet connection is available, files larger than 5MB should be sent to the remote web service (scenario L);

3. If an Internet connection is available, files less than 5MB should be split into chunks of the smallest size of ∼1KB and processed locally on the mobile device (scenario M).

6 Conclusion and Future Work

In this paper, we have analyzed the performance of ECG processing on mobile devices, including smartphones, tablets or similar devices.

The analysis of the performance includes response time, battery consumption and resilience analysis of an algorithm for ECG derived heart rate and respiratory rate. To make relevant conclusions we have realized three experiments for processing ECG files with various lengths, each of them providing a different execution environment. In the first experiment, files were processed locally, on a portable device. In the second experiment, files were processed remotely using a web service, and in the third files were processed remotely on the MatLab ©server.

The target goal was to analyze how to choose a more performance-effective and battery-effective solution. The hypothesis that the mobile-only solution performs worse than offloading solutions is valid only for files larger than 5MB. However, if a large file is split in chunks, then the mobile-only solution performs better.

If the files were split into chunks which size is less than 5MB then processing them locally is also acceptable even for large files. This conclusion does not apply to battery consumption, and in these cases, one needs to offload the computation in order to preserve the battery.

We have modeled the performance behavior and compared the experimental and theoretical results. Considering the actual measurements, the theoretical (predicted) response time results have shown to follow the trend of the real response times.

The experiment results solve the customer's dilemma which option provides better performance and is more battery-effective, when processing ECG files with various size.

We will continue to realize the same and similar experiments with different mobile devices.

References

1. Ahamed, M.A., Asraf-Ul-Ahad, M., Sohag, M.H.A., Ahmad, M.: Development of low cost wireless ECG data acquisition system. In: Advances in Electrical Engineering (ICAEE), 2015 International Conference on. pp. 72–75. IEEE (2015)
2. Amiri, A.M., Mankodiya, K., et al.: m-: An efficient QRS dQRSetection algorithm for mobile health applications. In: E-health Networking, Application & Services (HealthCom), 2015 17th International Conference on. pp. 673–676. IEEE (2015)
3. CodeLathe Technologies Inc: Tonido file access share sync, https://play.google.com/store/apps/details?id=com.tonido.android

4. CodeLathe Technologies Inc: Tonido server software, https://www.tonido.com/
5. Ding, S., Zhu, X., Chen, W., Wei, D.: Derivation of respiratory signal from single-channel ECGs based on source statistics. International Journal of Bioelectromagnetism **6**(2), 43–49 (2004)
6. Elgendi, M.: Fast QRS detection with an optimized knowledge-based method: Evaluation on 11 standard ECG databases. PloS one **8**(9), e73557 (2013)
7. Elgendi, M., Eskofier, B., Dokos, S., Abbott, D.: Revisiting QRS detection methodologies for portable, wearable, battery-operated, and wireless ECG systems. PloS one **9**(1), e84018 (2014)
8. Gia, T.N., Jiang, M., Rahmani, A.M., Westerlund, T., Liljeberg, P., Tenhunen, H.: Fog computing in healthcare internet of things: A case study on ECG feature extraction. In: Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on. pp. 356–363. IEEE (2015)
9. GSam Labs: Gsam battery monitor, https://play.google.com/store/apps/details?id=com.gsamlabs.bbm&hl=en
10. Gusev, M., Stojmenski, A., Guseva, A.: Ecgalert: A heart attack alerting system. In: International Conference on ICT Innovations. pp. 27–36. Springer (2017)
11. Hacks, C.: e-health sensor platform v2. 0 for arduino and raspberry pi [biometric/medical applications] (2013)
12. Ibaida, A., Al-Shammary, D., Khalil, I.: Cloud enabled fractal based ECG compression in wireless body sensor networks. Future Generation Computer Systems **35**, 91–101 (2014)
13. Lahiri, K., Mandal, A., Sinha, S., Mukherjee, A.: Wireless personal health ECG monitoring application. In: Wireless Personal Multimedia Communications (WPMC), 2014 International Symposium on. pp. 618–623. IEEE (2014)
14. Li, P., Jiang, H., Yang, W., Liu, M., Zhang, X., Hu, X., Pang, B., Yao, Z., Chen, H.: A 410-nw efficient QRS processor for mobile ECG monitoring in 0.18-μm cmos. In: Biomedical Circuits and Systems Conference (BioCAS), 2016 IEEE. pp. 14–17. IEEE (2016)
15. Mazomenos, E.B., Biswas, D., Acharyya, A., Chen, T., Maharatna, K., Rosengarten, J., Morgan, J., Curzen, N.: A low-complexity ECG feature extraction algorithm for mobile healthcare applications. IEEE journal of biomedical and health informatics **17**(2), 459–469 (2013)
16. Miettinen, A.P., Nurminen, J.K.: Energy efficiency of mobile clients in cloud computing. HotCloud **10**, 4–4 (2010)
17. Parák, J., Havlík, J.: ECG signal processing and heart rate frequency detection methods. Proceedings of Technical Computing Prague **8**, 2011 (2011)
18. Pineda-López, F.M., Martínez-Fernández, A., Rojo-Álvarez, J.L., Blanco-Velasco, M.: Design and optimization of an ECG/holter hybrid system for mobile systems based on dspic. In: Computing in Cardiology Conference (CinC), 2014. pp. 453–456. IEEE (2014)
19. Saving: Monitor the activity of your heart, http://www.savvy.si
20. The MathWorks, Inc.: Matlab mobile, https://play.google.com/store/apps/details?id=com.mathworks.matlabmobile&hl=en
21. Wang, X., Gui, Q., Liu, B., Jin, Z., Chen, Y.: Enabling smart personalized healthcare: a hybrid mobile-cloud approach for ECG telemonitoring. IEEE journal of biomedical and health informatics **18**(3), 739–745 (2014)

22. Wang, X., Zhu, Y., Ha, Y., Qiu, M., Huang, T.: An fpga-based cloud system for massive ECG data analysis. IEEE Transactions on Circuits and Systems II: Express Briefs (2016)
23. Welch, J., Ford, P., Teplick, R., Rubsamen, R.: The massachusetts general hospital-marquette foundation hemodynamic and electrocardiographic database–comprehensive collection of critical care waveforms. Clinical Monitoring **7**(1), 96–97 (1991)
24. Zhou, S., Zhu, Y., Wang, C., Gu, X., Yin, J., Jiang, J., Rong, G.: An fpga-assisted cloud framework for massive ECG signal processing. In: Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on. pp. 208–213. IEEE (2014)