Leveraging Smartphones for Distributed Global Navigation Satellite System Post-Processing

A. Ivanovski*, V. Zdraveski*, M. Gusev*, M. Kostoska*
* Ss. Cyril and Methodius University in Skopje, Faculty of Computer Science and Engineering, Skopje, North Macedonia,
E-mail: aleksandar.ivanovski.1@students.finki.ukim.mk,
{vladimir.zdraveski, marjan.gushev, magdalena.kostoska }@finki.ukim.mk

***Abstract*—Smartphones play an important role in today's society. Although their performance increases rapidly, we are witnesses of their small average daily usage. In this paper, we focus on executing post-processing tasks required for the growing amount of daily generated GNSS data. The research hypothesis is to check whether we can take an advantage to use smartphones while they are idle for such kind of post-processing.**

We have realized a prototype model and measured the performance as a proof of concept use-case to provide evidence if such an implementation is effective.

***Index Terms*—smartphones, grid computing, GNSS post-processing**

I. Introduction

Smartphones became an essential part of our everyday life, and which imposed rapid increase in their performance. The latest PassMark cross-platform benchmark [1], lists smartphones which score better than many mid-range CPUs. And that list keeps growing, as new smartphones are being launched. But even if they are essential part, they are not used all the time. Statista [2] estimates that the average daily usage of smartphones is 3 hours and 15 minutes, i.e. the smartphones are idle 87% of the time.

As our society is becoming more and more data-driven, the need of terrain mapping increases. Countries and institutions are developing and extending geographic information systems, autonomous vehicles are on the rise and they rely on precise geolocation, many other economic sectors try to incorporate Global Navigation Satellite System (GNSS) data into their workflows. In order for GNSS data to become useful, they need post-processing which is time and resource consuming task.

The research hypothesis analyzed in this paper is to find whether smartphones can be used when idle to provide processing of raw GNSS data.

In order to check the hypothesis validity, we develop a use-case system which connects the generators of raw data with idle smartphones. The starting point is identification of core building blocks of such a system, followed by a specification of abstractions and with guidance of software engineering approaches, and finally, a an implementation is being proposed.

The current effort put into the optimization of GNSS data post-processing relies on implementing a local parallel system using common multi-threading concepts and the amount of vertical scalability is approaching the upper boundary. Our approach exceeds those limits by providing massive parallel systems, with horizontal scalability approaching millions of CPUs.

This paper is organized as follows, An overview of the related work is presented in Section II. Section III defines the system and the corresponding architecture, followed by description of the system implementation in Section IV. The results from the experiments are are presented and elaborated In Section V. The final Section VI elaborates the conclusions and future work.

II. Related Work

Research on using mobile devices for distributed computing was conducted in 2004 [3] and resulted with a joint proposal to use PDA-s for grid computing.

More recent research has been realized in 2012 [4] to develop a task scheduler by developing a cross-platform benchmark, feasibility study and proposal of abstract implementation.

An implementation for distributed computing on Android based smartphones was proposed in 2017 [5].

The challenges in implementing such a system was analyzed in [6] with a comprehensive discussion on the usage and availability trends of smartphones.

Several proposed implementations of smartphone-based grid systems were analyzed in [7] along with elaboration of challenges.

This topic has been discussed by many researchers and companies around the world [8] and over the years, many implementations at a higher-abstract level have been proposed. [9] There is even a working Python-based framework called Misico [10].

Kakooei and Tabatabaei [11] demonstrated the results of parallel GNSS data post-processing and demonstrated 15% improvement compared against classical serial approaches.

Chui et al. in their paper [12] demonstrated significant results from the parallelization of GNSS data post-processing.

III. Architecture of a smartphone-based GNSS post-processing

The main actors along with their roles and responsibilities into the system are identified, and the actions taken by each of them are covered within a use-case scenario. The goal of this scenario is to provide guidance for the transformation from raw into processed data.

A. Functional description

Actors, roles and responsibilities are illustrated in Fig. 1.

Clients: Producers of raw GNSS data while conducting measurements with specialized equipment. In order to make use of these data, post-processing has to be done which is time and performance consuming task. Optimization could be achieved by leveraging the proposed system.

Post-processing provider: The entity which receives the raw data produced by the clients. Responsible for orchestrating the post-processing, and managing the pool of workers. *(abbr. provider)*

Smartphone workers: Managed devices which process batches of the raw data, and run software provided and managed by the Post-processing provider.

B. Primary use case

The client produces raw GNSS data while doing it's regular business activities, and sends that data to the post-processing provider. When the provider receives the data, they are identified into the system and ready for processing. The provider utilizes it's reducer to split the data into smaller batches, each batch is send to a worker from the pool of available workers. Each worker upon reception of the data, starts the processing algorithms which are embedded in the previously mentioned software provided by the Post-processing provider.

Data processing is centered around linear algebra, more preciesly matrix algebra and linear transformations and invloves floating point arithmetics.

For even greater efficiency, each worker uses multi-threading.

Using device GPUs was considered, but due to the fact that in general the processing capabilities of smartphone GPUs is not significant, and after in-depth analysis a CPU approach with multi-threading was chosen.

As the workers complete the tasks, they send the processed data back to the provider, who is keeping track of the overall process, and appends each completed batch to the data for the particular job. When the workers finish, the data is send to the appropriate client. If a worker fails, the provider will allocate that worker's batch or batches to a new worker from the pool - this mechanism provides fail-safe environment and prevents inconsistencies.

C. Software Architecture

The abstract software solution architecture is presented in Fig. 2.

The **Clients** are entities which have interface to the system for sending raw data, receiving processed data and are being charged for the services by a contract.

The **Post-processing provider** is an entity which consists of the following modules:

- Client management module, which keeps track of all clients, their new or completed tasks and keeps track of several metrics which are needed for billing and analytics.
- Mapper module, which splits a client's raw data into smaller batches, and assigns a worker for each batch.
- Reducer module, which receives processed data from a worker, appends it to the overall processed data. And when all the data is processed sends it back to the client.
- Smartphone workers pool, which keeps track of all the available workers, sends them batches of data, gets the processed data from the workers, keeps track of several metrics required for analytics and compensation.

The **smartphone worker** is an entity - more precisely a smartphone, which runs a mobile application from the provider. The application serves as an endpoint to the provider, for receiving raw and sending processed data, utilizes the device's multi-threading environment to process raw data. There is another part from the application which is not strongly related to the core computing process, but is important for the business process, that part collects various metrics which are required for compensating the worker for their resources, and for other analytics.

IV. Implementation

Google Cloud Platform [13] and Firebase [14], are going to be used for this proposed concrete implementation.

The **Post-processing provider** is going to be a web application, deployed on Google Cloud Platform with a central role in the system. Upon receival of the client's data via web interface, the app is going to split the raw data into batches, and upload each batch to Firebase Cloud Storage. The app manages the batches and their identifiers in a NoSQL Database - Firebase Firestore, and using the Firebase Cloud Messaging sends message to the desired workers which are registered for providing processing power, and identified with FCM Tokens.

A Firebase Cloud Messaging Token is an unique identifier for each device registered to a certain Firebase Cloud Messaging Topic. Each message contains the assigned batch identifier, which is going to be used by the Smartphone-Worker to pull the batch from Firebase Cloud Storage. After each worker finishes with the processing, the app merges all the batches which are already uploaded to Firebase Cloud Storage, into one file, available for the client.

The **Smartphone workers** are Android or iOS based devices, which have installed a native application, developed in Dart and Flutter [15]. The app allows the user to register his or hers device for accepting post-processing tasks. After registration the app waits to be assigned a task, i.e. a receive a cloud message with the batch identifier. Upon receival of a message, the app pulls the batch with the given identifier from Firebase Cloud Storage, and starts the processing task. Each processing task is using isolates, which enable local multi-threading on Android and iOS. When a worker finishes the processing, the app uploads the now processed data file to Firebase Cloud Storage.

Flutter uses the APIs provided by Andorid and iOS SDKs to keep track of the state in which the device is, and only idle devices will receive processing tasks.

The **Clients** access the Post-Processing provider via web based interface. Clients can upload their raw data, monitor the

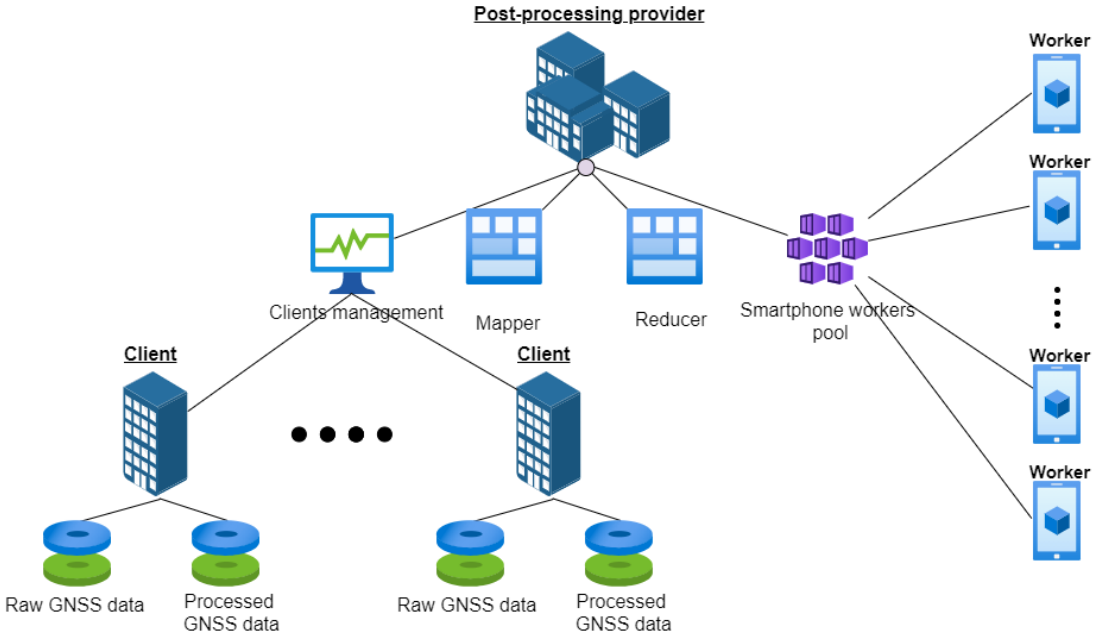

Fig. 1. Actors and their modules

TABLE I
SUMMARY OF THE RESULTS

Attempt Number	Number of Workers	Internet Connection	Start to Finish Time	Download Time	Upload Time	Processing Time	Utilization
1	6	40/20 (D/U) Mbit/s FTTx	6122.45 ms	1170.33 ms	4040.11 ms	908.43 ms	0.94
2	6	30/15 (D/U) Mbit/s FTTx	7054.45 ms	1162.38 ms	5959.93 ms	2.71 ms	0.96
3	6	30/16 (D/U) Mbit/s Telekom 4G	9866.48 ms	3890.29 ms	6939.14 ms	3.82 ms	0.97
4	12	20/10 (D/U) Mbit/s Broadband	14368.02 ms	5514.60 ms	8703.76 ms	89.94 ms	0.58
5	12	20/10 (D/U) Mbit/s Broadband	15428.62 ms	7003.46 ms	8257.99 ms	92.40 ms	0.52
6	12	20/10 (D/U) Mbit/s Broadband	14857.33 ms	6288.05 ms	8432.25 ms	82.51 ms	0.63
		Average:	**11282.89 ms**	**4171.52 ms**	**7055.53 ms**	**196.64 ms**	**0.77**

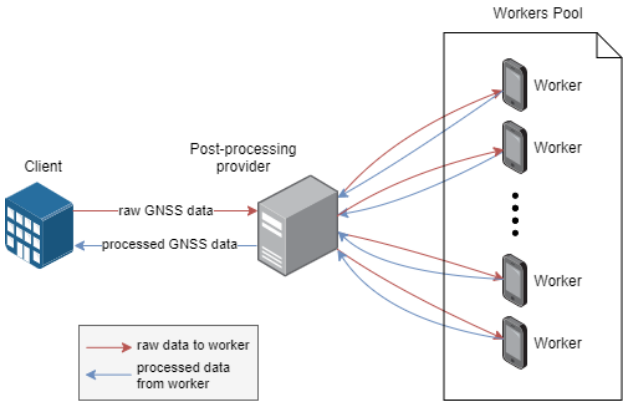

Fig. 2. Abstract Solution Architecture

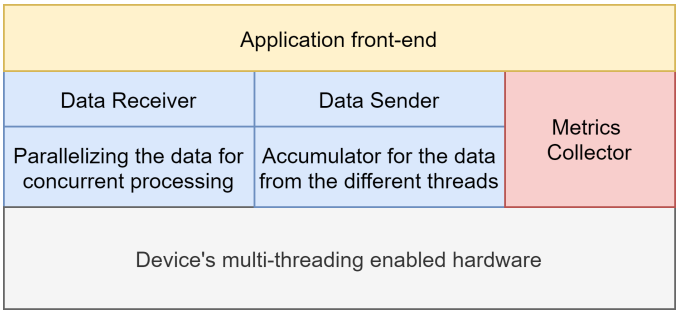

Fig. 3. Software architecture of the mobile application

progress of the processing, and when complete, download the processed data.

Fig. 4 shows the UML activity diagram of the implemented use-case.

Each of the described components, include smaller modules, such as billing, analytics, usage history, task monitoring, etc. These software modules are not explained in details since they are beyond of the scope of this paper.

V. RESULTS

For obtaining the initial results, many attempts were made, on various devices and internet connections. The data used for this experiment is a real dataset provided by a mapping company based in Strumica, with 6000 raw-data points. Due to business policies the data is anonymized, but that doesn't

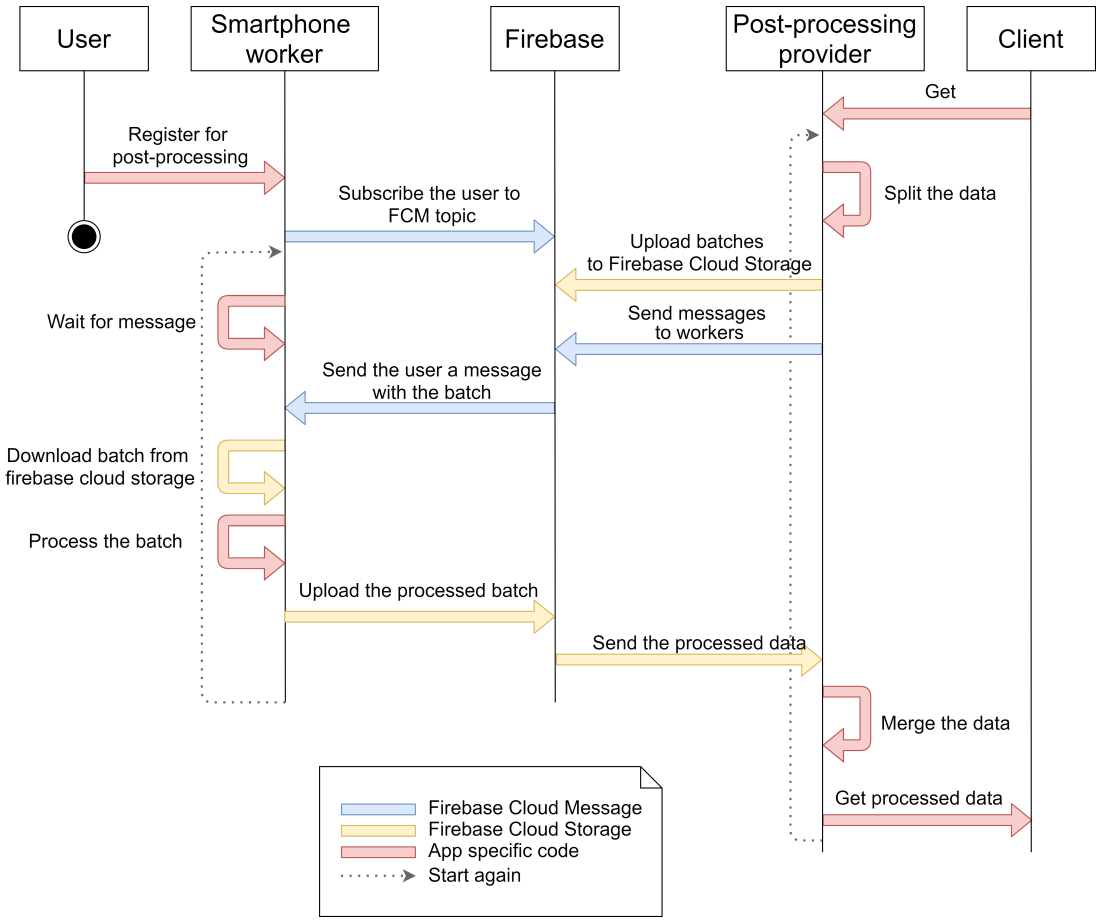

Fig. 4. UML activity diagram of the implemented use-case

effect the results. The company processed the data for around 3 minutes using proprietary software, and a North Macedonia's post-processing provider offers post-processing times of 100 ms per raw-point, i.e. 10 minutes total for this data, these metrics were obtained from the publicly available data and with domain experts which use their services on daily basis, the internal architecture of their post-processing system is proprietary. Using cloud based scalable infrastructure such as Google Cloud Platform or Amazon Web Services can be considered but the costs of such approach would conflict with our primary objective, i.e. leveraging idle hardware, instead of employing active hardware.

Using the proof of concept implementation, the average start to finish time was: 11282.89ms, shown in Fig. 8, with 4171.52ms average download time (required for pulling the raw-data batches from Firebase Storage) as shown in Fig. 6, and 7055.53ms average upload time (for uploading the processed data to Firebase Storage) as shown in Fig. 7.

The utilization is shown in Fig. 5, it was influenced by per worker file I/O, but our implementation provided an average of 77%.

The above elaborated summary statistics were obtained from the experimentation phase which results are shown in I. When concluding experiments the quantity and diversity of smartphones proposed limiting factor. In Attempt Number 1 older LG smartphones were used in which the Flutter app couldn't leverage the device's multi-threading capabilities and the processing time is enormously higher compared to other attempts, on first glance it may seem as an outlier, but this result shows the significance of using multi-threading on a worker level.

This proof of concept showed that the post-processing of GNSS data, can be significantly improved, in this case we have improvement of 5317% compared with traditional post-processing providers, and 1595% improvement compared to in-house data post-processing.

Due to the test environment in which experiments were conducted, the cloud platform always requires more time on the first attempt to start up the services, but in a production environment with clients and workers this will not be an issue.

The metrics from our approach, can be compared to the other approaches with different amount of raw data. The processing time needed with traditional post-processing providers and in-house data processing is proportional with the amount of data.

The results in Table II, indicate that our approach outperforms the others, due to it's horizontal scalability. The other approaches are linearly dependent on the amount of data, and

TABLE II
COMPARISON OF OUR APPROACH VERSUS EXISTING METHODS WITH RESPECT TO THE AMOUNT OF DATA

Number of points in raw data	**Processing Time required**		
	Post-processing providers	In-house processing	Our approach
6000	600000 ms	180000 ms	11289 ms
15000	1500000 ms	450000 ms	28222.5 ms
30000	3000000 ms	900000 ms	56445 ms
50000	5000000 ms	1500000 ms	94075 ms

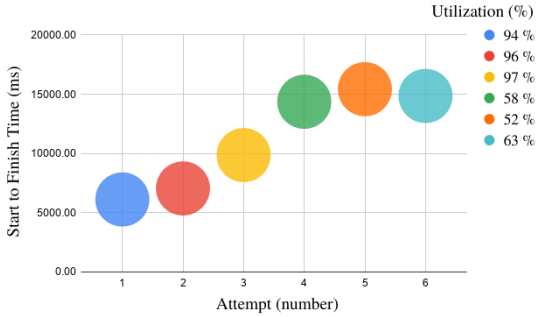

Fig. 5. Start to Finish Time per attempt with Utilization

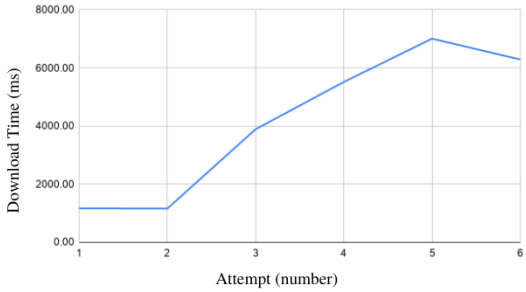

Fig. 6. Download Time per attempt

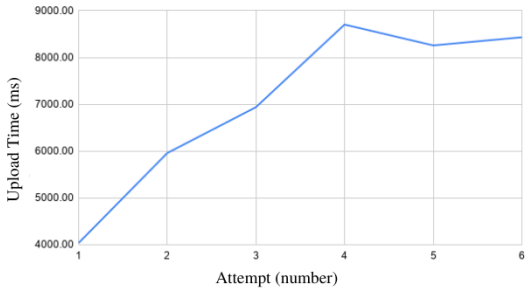

Fig. 7. Upload Time per attempt

that dependency introduces a bottleneck in processing time. These results are shown in Fig. 9.

VI. CONCLUSION AND FUTURE WORK

This paper demonstrates that using idle smartphones to provide post-processing services of raw GNSS data can give huge improvements of the processing time.

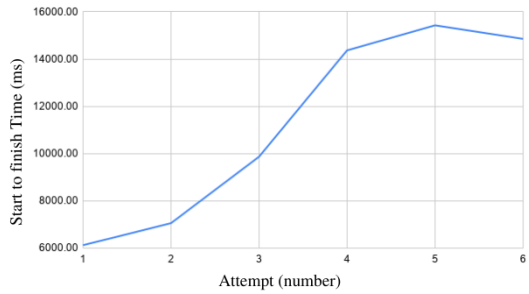

Fig. 8. Start to Finish Time per attempt

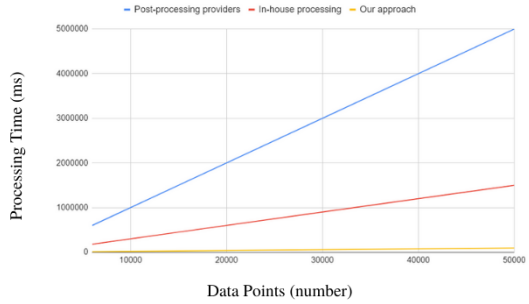

Fig. 9. Comparison between approaches and amounts of data

Major aspect that should be taken under consideration is the power consumption produced from performing the calculations. During the research and experimentation slight increase in battery life was noticed and the heat production was stable and no major fluctuation were noticed., but in order to draw general conclusion experimentation on vast number of devices from different vendors should be performed.

Due to the structure of the data and the increasing speeds of networks, especially the rise of 5G, the upload and download times do not pose a bottleneck of the system.

Our research process evolved from abstractions to design of details, and building proof of concept implementation which provided performance measures necessary to prove the hypothesis. The overall outcome can be mutually beneficial to industry and the general population.

This proof of concept implementation can be further improved and optimized, and the experiment can be carried on a number of smartphones ranging in hundreds or even thousand of devices.

REFERENCES

[1] PassMark® Software Pty Ltd, "Cross-platform CPU performance - most common desktop & laptop CPUs vs common mobile devices," 2020 (accessed November 8, 2020). [Online]. Available: https://www.cpubenchmark.net/cross-platform.html

[2] Statistica, "U.S. daily media usage time via mobile 2018-2022," 2020. [Online]. Available: https://www.statista.com/statistics/469983/time-spent-mobile-media-type-usa/

[3] D. C. Chu and M. Humphrey, "Mobile ogsi. net: Grid computing on mobile devices," in *Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE, 2004, pp. 182–191.

[4] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Computing while charging: Building a distributed computing infrastructure using smartphones," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 193–204.

[5] K. D. Chethan K, Chiranthan H R, "A distributed computing infrastructure using smart phone," in *International Advanced Research Journal in Science, Engineering and Technology*, 2017.

[6] S. C. Shah, "Recent advances in mobile grid and cloud computing," *Intelligent Automation & Soft Computing*, pp. 1–13, 2017.

[7] B. Schaffner, J. Sawin, and J. M. Myre, "Smartphones as alternative cloud computing engines: Benefits and trade-offs," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2018, pp. 244–250.

[8] J. M. Rodríguez, C. Mateos, and A. Zunino, "Are smartphones really useful for scientific computing?" in *International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability*. Springer, 2011, pp. 38–47.

[9] V. Gharat, A. Chaudhari, J. Gill, and S. Tripathi, "Grid computing in smartphones," *Int. J. Res. Sci. Innov.-IJRSI*, vol. 3, no. 2, pp. 76–84, 2016.

[10] D. G. Adam Dou, Vana Kalogeraki, "Misico: A mobile mapreduce framework." [Online]. Available: http://alumni.cs.ucr.edu/ jdou/misco/

[11] M. Kakooei and A. Tabatabaei, "A fast parallel gps acquisition algorithm based on hybrid gpu and multi-core cpu," *Wireless Personal Communications*, vol. 104, no. 4, pp. 1355–1366, 2019.

[12] Y. Cui, Z. Lu, H. Lu, J. Li, Y. Wang, and L. Huang, "A parallel processing strategy of large gnss data based on precise point positioning model," in *China Satellite Navigation Conference (CSNC) 2015 Proceedings: Volume III*. Springer, 2015, pp. 139–147.

[13] "Google cloud platform," Google. [Online]. Available: https://cloud.google.com

[14] "Firebase," Google. [Online]. Available: https://firebase.google.com

[15] "Flutter," Google. [Online]. Available: https://flutter.dev