Hybrid Elastic ARM&Cloud HPC Collaborative Platform for generic tasks

David Petrocelli [1 2 *], Armando De Giusti [3 4] and Marcelo Naiouf [3]

[1] PhD Student at Computer Science School, La Plata National University, 50 and 120, La Plata, Argentina
[2] Professor and Researcher at Lujan National University, 5 and 7 routes, Luján, Argentina
[3] Instituto de Investigación en Informática LIDI (III-LIDI), Computer Science School, La Plata National University - CIC-PBA, 50 and 120, Argentina
[4] CONICET - National Council of Scientific and Technical Research, Argentina
dmpetrocelli@gmail.com, degiusti@lidi.info.unlp.edu.ar, mnaiouf@lidi.info.unlp.edu.ar
* Corresponding Author

Abstract. Compute-heavy workloads are currently run on Hybrid HPC structures using x86 CPUs and GPUs from Intel, AMD, or NVidia, which have extremely high energy and financial costs. However, thanks to the incredible progress made on CPUs and GPUs based on the ARM architecture and their ubiquity in today's mobile devices, it's possible to conceive of a low-cost solution for our world's data processing needs.

Every year ARM-based mobile devices become more powerful, efficient, and come in ever smaller packages with ever growing storage. At the same time, smartphones waste these capabilities at night while they're charging. This represents billions of idle devices whose processing power is not being utilized.

For that reason, the objective of this paper is to evaluate and develop a hybrid, distributed, scalable, and redundant platform that allows for the utilization of these idle devices through a cloud-based administration service. The system would allow for massive improvements in terms of efficiency and cost for compute-heavy workload. During the evaluation phase, we were able to establish savings in power and cost significant enough to justify exploring it as a serious alternative to traditional computing architectures.

Keywords: Smartphone, Distributed Computing, Cloud Computing, Mobile Computing, Collaborative Computing, Android, ARM, HPC.

1 Introduction

Since their inception, x86 CPUs (Intel/AMD) and their corresponding GPUs (Intel/AMD/NVidia) were developed to solve complex problems without taking into account their energy consumption. It was only in the 21st century that, with the exponential growth of data centers, the semiconductor giants began to worry and seriously tackle their chips' TDP, optimize their architectures for higher IPC, and lower their

The final authenticated version is available online at http://doi.org/10.1007/978-3-030-27713-0

power consumption [1][2][3]. If we analyze the current context, the power and cooling bills are two of the biggest costs for data centers [4][5]. It is for this reason that energy consumption and efficiency are two key issues when designing any computing system nowadays and it is acknowledged that accomplishing compute benchmarks must not be done so by brute force but rather by optimizing resources and architectures, keeping in mind the high financial and environmental cost of the energy required by large data centers.

Unlike the x86 processors, ARM-based chips were conceived with power efficiency as key priority from their inception, since these were oriented to the mobile devices and micro devices, which are for the most part battery-powered. Although the raw power of ARM chips is lower than that of their x86 counterparts, they are much more efficient power-wise [6][7][8]. At the same time, ARM-based devices vastly outnumber traditional x86 computers, so while their individual computer power might be lower, there is a very large install base with long idle periods while charging that, if properly managed, could become massive distributed data centers consuming only a fraction of the energy for the same computing power as their traditional counterparts. This distributed workload during idle time principle has been used before in x86 for collaborative initiatives such as SETI@Home and Folding@Home which use the computing power of computers that people left turned on during idle period to construct large virtualized data centers. Currently some investigators have converted this collaborative model to work on mobile devices [9][10][11][12].

After analyzing the methodology, structures, and results obtained in those case studies, our team built and evaluated a collaborative platform for HPC based on ARM-based mobile devices, following the footsteps of the previous study [8]. The platform receives data fragments and instructions from its clients via de cloud portal, generates tasks and distributes them to the worker nodes (mobile devices) for them to apply their massively parallelized computing power towards the function requested by the client. Once the task is complete, all worker nodes return their processed fragments to the central nodes and the final processed data is stored until the client requests it. In this paper in particular the implemented task was video compression using the FFMpeg library, which allows for different profiles and compression configurations to be requested by the client, as defined by the previous study [8]. The platform was evaluated through a series of performance and power usage metrics both on ARM well as x86 chips. This allowed us to make a comparative analysis between the architectures and demonstrate that it is completely feasible to offload compute heavy workloads to ARM architectures. It also allowed us to compare the power usage for the same task on ARM-based chips compared to their x86 counterparts. An analysis of a cloud-based x86 architecture (IaaS) was also performed with the objective of offering an estimation of the costs which could be recuperated if those tasks were to be migrated to the collaborative computing platform.

In order to clearly describe the different aspects of the work, the rest of the paper is organized in the following manner: Section 2 details the implemented protocol, technologies used, and functions developed for it. Section 3 describes the testing model and metrics used. Based on that section as well as the data collected during the experiment,

section 4 presents an analysis of the results. Finally, the last chapter will focus on the conclusion and future work to further explore the topic.

2 Distributed Architecture for HPC on ARM

The developed architecture implemented a hybrid model composed of (a) cloud-based resources, and (b) mobile devices. In Fig. 1 a functional diagram is presented.
The cloud resources (a) carry out the following functions:

- RESTful web server for task reception and delivery
- Admin and task management system
- Database storing statistics (time used per task and power usage)

The mobile devices (b) process the task fragments they receive in a collaborative and distributed fashion. Each node has access to the application (apk) that allows it to access the network and includes the video compression logic.

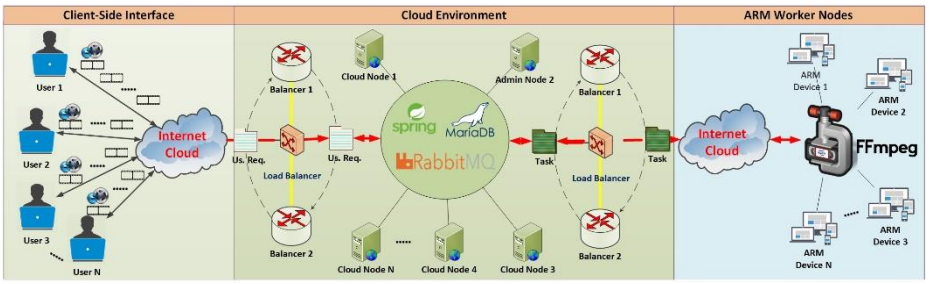

Fig. 1. Functional Diagram of the Collaborating Computing Network

2.1 Cloud Nodes

RESTful web servers for task reception and delivery. A RESTful web app was developed through the SPRING Java framework so that both clients and nodes can communicate with the web servers in the network. A RESTController entity was developed for the purpose of establishing the connection with the distributed queuing system (RabbitMQ), with the statistics database (MariaDB), and translating the HTTP requests from the users into specific functions as determined by the roles they possess. All communication between clients, worker nodes, and servers is done through HTTP messages encoded using the JSON format, as are the task objects (Messages).

Admin and task management system. In order to construct an admin and task management system that was persistent, redundant, and fault-tolerant, the RabbitMQ middleware was used as a message queue-based persistence module and was integrated with the RESTful Java platform. With this tool, we implemented persistent FIFO

queues and they were configured to offer high cluster availability. The messages storage uses the exchange format (JSON).

At the same time, the REST Java server implements a manual configuration model for the content of the general message queue whence worker nodes obtain their tasks. In the event of a server error, client-side issues, or execution timeout, the admin thread moved the task back from pending to available and places it back on the queue so that another worker can process it.

Database with statistics data storage (time and power usage). The web server also includes a link to the MariaDB database through a CRUD schema using the DAO design pattern. The database must register each executed task, which worker node executed it, the type of task, the processing time (milliseconds), and the power usage (watts). This allows us to access that information for the purpose of evaluating different architectures and derive their efficiency from the comparative analysis between their performance and power usage

2.2 Mobile Worker Nodes

The application for the mobile nodes was developed on the native Android environment (Android Studio) and an apk was generated based on a Java codebase. The application is basically a series of modules that allow for the execution of the compute-heavy tasks to be performed without negatively impacting the OS.

The application was developed using the MVC pattern, based on activities with Java on Android Studio. A given activity defines a view that allows for the interaction with the user and visualization of the state changes made by the controller and on the flipside generates an independent worker thread which is in charge of the compute-heavy workload. This thread will continuously iterate the following process: first it connects to the REST server and downloads a task that needs to be processed, then it takes the parameters from the JSON message and applies them to the compression of the source video with the FFMpeg library which is integrated in the app as defined in the message. Once the video has been compressed, it generates a new JSON message and sends it along with the compressed video back to the server. Once the cycle is finished, it attempts to obtain another message from the queue and repeats the aforementioned process.

It should be noted that in order to evaluate and compare the different platforms, a worker node was developed in Java for the x86 architecture with the same functions as those implemented on the Android app but without a GUI since the management is done through command line.

The codebase and all the tools used in this project are available on GitHub at the following URL: https://github.com/dpetrocelli/distributedProcessingThesis

3 Test model and metrics

With the goal of evaluating the performance of our prototype, we ran compute tasks composed of video compression with different sets of parameters on both types of architectures: ARM-based devices running Android and Intel x86-based devices. We aim to obtain the performance and power usage numbers for executing this task on these devices. For this purpose, the following steps were taken: 3.1) Selection of source videos; 3.2) Configuration of compression profiles and duration of video fragments; 3.3) Definition of the testing devices (ARM/x86); 3.4) Definition of the metrics to be measured; 3.5) Construction of the custom power usage meters.

3.1 Selection of source videos

In order to generate a test that could saturate devices of both architectures, we analyzed previous works and configurations [13][14] and used them as a reference for this test. Based on that we selected three source videos with characteristics that were relevant for both platforms (codecs used, bitrate, compression level, frame size, aspect ratio, bits per pixel, etc.). The overview of the most important properties of each video are detailed in Table 1.

Table 1. Principal characteristics of the source videos used for compression tests

Audiovisual Test Material											
File Name	Duration	Size	Size Screen	Format	Bitrate	Comp. Prof.	Comp. lvl	FPS	Audio Codec	Kbps Audio	Sampling	Channel
3dmark-4k-120fps.mkv	2m 35 segs	487 MB	3840x2160	AVC x264	27545 Kbps	high	@L6	120	Vorbis	160	48000	2
bbb-sunflower-2160p.mp4	10 m 34 segs	605 MB	3840x2160	AVC x264	8000 Kbps	high	@L5.1	60	aac	160	48000	6
bigbuckbunny-1500.mp4	9 m 56 segs	109 MB	1080x608	AVC x264	1080 Kbps	main	@L3.1	24	aac	128	48000	2

3.2 Configuration of compression profiles and video fragment duration

The chosen codec AVC h.264 [15] (x264 library on FFMpeg) is widely supported on current devices and video streaming platforms such as Youtube, Vimeo, Netflix, etc. The encoding profiles of x264 that were selected to run the tests mentioned in this activity are detailed below in Table 2.

Table 2. Description of the properties of compression profiles used in FFMpeg

x264 Codec	Size	Video C.	Bitrate V.	Lvl	FPS	Preset	Bframes	BF/GOP	BF Ref.	Audio C.	Bitrate A.	Sampling	Audio Ch.
High Profile													
4K	4096x2160	libx264	15600	L@5.1	60	very slow	6	3	2	ac3	512	48000	6
1080 Full HD	1920x1080	libx264	3900	L@4.1	30	slow	6	3	2	ac3	320	48000	6
Main Profile													
720 HD	1280x720	libx264	2000	L@4.1	25	medium	3	3	1	aac	320	44100	2
480	852x480	libx264	900	L@3.1	25	fast	3	3	1	aac	256	44100	2

Four profiles were selected for the two higher compression levels available on the x264 codec (main, high); these range from simple, low compression profiles, to complex, high compression ones. As such, the encoding and compression performed will be the same as the most common configurations found on video streaming platforms;

which allows the public to access content in different qualities in order to match their device compatibility and connection bandwidth [16][17].

In addition, each video is also split into 3 and 1 second fragments for each compression level. These are the recommended values for video streaming services [18][19].

3.3 Definition of testing equipment (ARM/x86)

In order to evaluate the performance demands of the compression tasks, a mix of x86 and ARM-based resources were used; these were chosen based on their compute power. Both architectures have a high-end cluster (x86-i7-16gb / ARM Samsung S7), and a mid-tier cluster (x86-i5-8gb / ARM Samsung A5 2017), all of them fully support encoding/decoding of x264 videos [20]. The relevant characteristics of the equipment are listed in Table 3.

Table 3. Principal characteristics of the equipment used for compression tests

x86/ARM Cluster				
Cluster Name	Processor	Memory	Disk	GPU
x86-Clusteri7	Intel i7-4770-8x3.4Ghz	16 GB-DDR3	500GB-7200RPM	Intel HD Graphics 4600
x86-Clusteri5	Intel i5-2400-4x3.1Ghz	8 GB-DDR3	500GB-7200RPM	AMD Radeon HD 7670-480x800Mhz
ARM-ClusterS7	Exynos 8890-4x2.3 GHz+4x1.6 GHz	4 GB-LPDDR4	16 GB-SD IntStor	Mali-T880-12x650MHz
ARM-ClusterA5	Exynos 7880-8x1.9 GHz	3 GB-LPDDR4	16 GB-SD IntStor	Mali-T830-2x600Mhz

3.4 Definition of measured metrics

The following metrics were defined for evaluating the performance and power usage of the different architectures (x86/ARM) while performing compute-heavy tasks:

Time Metrics. Processing time per task for each architecture (IndTime).

- Minimum (MinIndTime) / Maximum (MaxIndTime) / Average (AVGIndTime)

All metrics were measured in milliseconds.

Power Usage Metrics. Register power usage metrics per task for each architecture

- Minimum (MinWattCons) / Maximum (MaxWattCons) / Average (AVGWattCons)

All metrics were measured in watts.
It is worth mentioning that custom tools were developed for the purpose of measuring these power usage metrics. These tools are detailed in Subsection (3.5).

Effectivity Score. The effectivity score is derived from the following metrics: the average task execution time (AVGIndTime) and power usage (AVGWattConsum) for each architecture as detailed below:

$$TimeRatio = AVGIndTime\ (ARM) \div AVGIndTime\ (x86)$$

(Determines how many times slower the ARM device is)

$$PowerUsageRatio = AVGWattCons\ (x86) \div AVGWattsCons\ (ARM)$$

(Determines how many times less power the ARM device uses)

$$EffectivityScore = PowerUsageRatio \div TimeRatio$$

If *EffectivityScore* = 1, equilibrium between architectures
If *EffectivityScore* <1, the effectivity score is better on X86
If *EffectivityScore* > 1, the effectivity score is better on ARM

3.5 Construction of power usage devices.

x86 Architecture. The x86 devices run on direct current provided by their power supply, which is fed from a standard 220V AC mains outlet. In order to measure their power usage, a non-invasive device was built from standard micro components which consists of a current clamp wired to an Arduino controller through a headphone jack connector and a multimeter for obtaining the grid voltage.

ARM Android. While mobile devices also run on direct current, they obtain it directly from their built-in battery. Due to this, it was necessary to build a software-based power usage meter. After researching the matter [21] [22], the power usage meter protocol was integrated into the ARM worker apk through an independent thread which only runs while a task cycle is underway and does not interfere with it. This thread polls the battery sensor information from the OS every 1000 milliseconds.

4 Experimentation and Analysis

Having defined the testing model and metrics to be measured, the experiment was executed. The analysis of the results is detailed below.

4.1 x86 Architecture Analysis:

Power Usage Analysis: Taking into account the execution of the compression tasks for both architectures, the power usage of all their components (except for the monitor) required on average were: 115 Watts for the Intel i7 (37% higher than the CPU's TDP: 84 Watts) and 183 Watts for the Intel i5, (93% higher than the CPU's TDP: 95 Watts). This increase is due to the i5 having a GPU (TDP: 66W) which aided in the assigned tasks. For this reason, the i5 cluster consumes, on average, 59% more than the i7 for all compression tasks.

Performance Analysis: Both architectures correctly completed the compression tasks (4k, HD, 720p, 480p) in the fragment configurations used (1 and 3 seconds) since their processing capabilities and memory (including swap) are comparatively large. The 4k profile, both in its 1 and 3 second fragments and with the chosen compression presets, is the most complex task given that it presents a large jump in processing time compared to the other presets; this can be observed in Fig. 2. This is basically due to the bitrate of the source video, which is 4 or 5 higher than the rest, as well as the use of the highest

quality profile available in the library (x264 high L@5.1) and its more in-depth precision requirement (preset: very slow). At the same time, it's clear that the i7 architecture has, on average, a 40% performance lead on its competition. Based on this it can be deduced that, the higher the task complexity, the better the i7 architecture will perform compared to the i5 architecture.

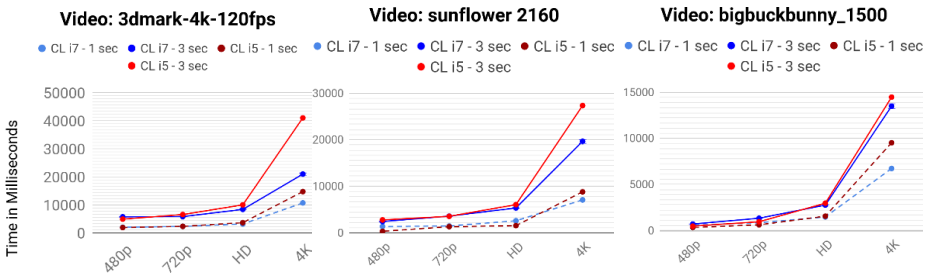

Fig. 2. x86 compression test results

x86 Platform Choice for Comparison between Architectures. The x86-i7 cluster was chosen as the one to be used for the comparison between the x86 and ARM architectures based on the performance results detailed above and the fact that the x86-i7 cluster uses on average 60% less power compared to the x86-i5 cluster.

Cost Analysis: Definition of the Cloud Architecture. For the cloud architecture, "General Purpose Av2" virtual machine instances (IaaS) on Microsoft Azure were chosen; specifically the "Standard_A8_v2" model, which contains 8 Virtual CPUs, 16GB of RAM and 80GB of SSD storage. It is the most similar configuration to the local i7 cluster while also being the cheapest one. Working on a Linux environment, the operating cost is 0.4$/hour (Official Azure pricing in April 2019) [23].

In order to validate the performance and costs, the same 3-video compression test suite was executed on the cloud VM but only applying the most complex profile (4K) in the smallest fragment size (1 second), with the purpose of narrowing down the results to one task in particular. The result of this testing showed that, on average, the cloud VM is 130% slower than the i7 cluster, as shown in Fig. 3.

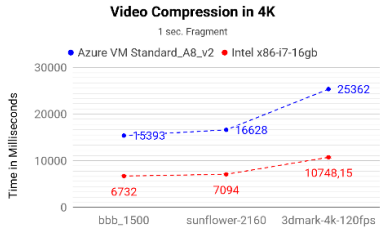

Fig. 3. x86 vs Cloud IaaS compression test results

Taking into account the requirement of compressing a 1-hour video split into 1-second fragments at the most complex profile (4k), the data from the previous analysis results

in cost figures of $10.15/h for the first video, $6.65/h for the second one, and $6.15/h for the third one as shown on Table 4. It should be noted that this is the estimated cost for just the processing time as it ignores any other costs arising from network usage, redundancy, traffic rules, etc. which all add further cost to cloud solutions [24].

Table 4. Azure IaaS Cost Estimates

VM: Azure Standard_A8_v2		VM Cost/hour ($)	0,4
Video Compression: 4k Video Fragment: 1 second	Comp. Time in ms per 1 Video sec.	Comp. Time in hour per 1 video hour	1 video hour Comp. Cost ($)
3dmark-4k-120fps.mkv	25362	25,362	10,14
bbb_sunflower_2160.mp4	16628	16,628	6,65
bigbuckbunny_1500.mp4	15393	15,393	6,16

Furthermore, if the first video is taken as a reference (3dmark - 487MB - 2min 35s) and its file size is extrapolated to what it would be with a 1-hour duration, that would result in a file size of 11GB. According to a study [25] from encoding.com, one of the most extensive compression and streaming cloud-based platforms on the web, they received requests to process and compress more than 6.6PB of data between 2017 and 2018. Although the relationship is not entirely linear, handling these requests through the use of idle devices would result in significant energy as well as costs savings.

4.2 ARM Architecture Analysis:

Power Usage Analysis. For video compression, the test results show that the Samsung S7 cluster consumes, on average, 3,15W while the Samsung A5 2017 cluster consumes, on average, 2,25W. This means that the high end platform consumes on average 40% more than the mid-range platform.

Performance Analysis. The S7 cluster is, on average, 39% faster than the A5 2017 for all the compression tasks, which matches the expected result based on their specifications. It can be concluded then that if we pair the performance and power usage metrics, they are pretty much equal in terms of performance/power.

Both platforms successfully processed the one-second video fragment tasks in all four profiles. However, when attempting the three-second video fragment tasks, the two most complex tasks failed in all four profiles (sunflower_2160 and 3dmark-4k-120fps) and the simplest one (bigbuckbunny_1500) was successfully processed in all its four profiles. The one-second video fragment task execution times are detailed in Fig. 4 on the left side. The compression performance of the three-second video fragment tasks on the different architectures is only available for the video bigbuckbunny_1500, since the other two failed to process, and can be observed in Fig. 4 on the right side.

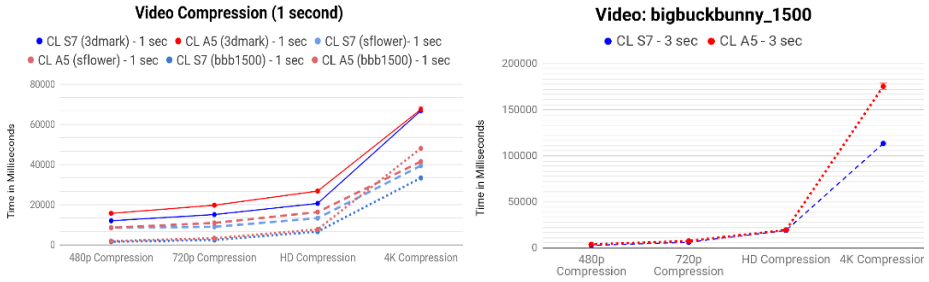

Fig. 4. ARM compression test results

The cause for failure in the three-second video fragment tasks for the most complex videos is due to the devices' inability to hold the necessary amount of data for processing that task in memory since the mobiles clusters have much less ram than their x86 counterparts (3GB and 4GB against 8GB and 16GB) and completely lack swap memory as that feature is not supported on Android.

Therefore, it is recommended that, for complex compression tasks, the video fragment size should be limited to one-second in order to fully take advantage of the mid and high-end mobile devices available in the market today

4.3 Comparative Analysis:

The performance and power usage results of the i7-x86 architecture were compared through the effectivity index against the two mobile device clusters (Samsung S7 and Samsung A5 2017), only taking into account the one-second video compression tasks that the ARM devices were able to complete successfully.

The ARM architecture, both on mid and high-end devices is between 5 and 16 times more effective compared to the x86 architecture depending upon the complexity of the task and source video. Generally, the less complex the task and source video are, the more effective ARM is. This can be observed in fig. 5

Furthermore, the mid-range cluster was able to process the tasks in a similar timespan to that of the high-end cluster but with a proportionally lower power usage. Thus its effectivity score is, on average, 19% higher for the bigbuckbunny-1500 video, 28% higher for sunflower-2160, and 12% higher for 3dmark-4k-120fps.

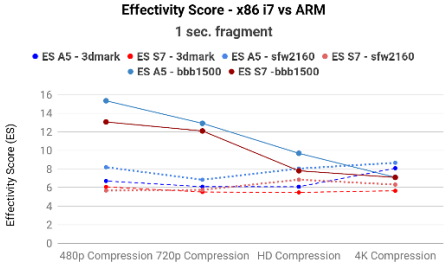

Fig. 5. Effectivity comparison of the platforms

5. Conclusions

A distributed collaborative platform was developed for the purpose of carrying out compute heavy tasks on mobile devices while they're idle and charging their batteries. It was determined through experimentation and evaluation that mid and high-end Android devices are capable of performing this type of tasks with a competitive power and cost advantage over traditional and cloud architectures. Furthermore, it was also demonstrated that the developed platform is sufficiently robust to withstand an assortment of tasks and challenges often faced by this type of applications. Lastly, it was determined that the mid-range nodes with the highest efficacy scores might not be the fastest but will always be more competitive than their more powerful counterparts owing to their lower power usage and cost.

6 Future Work

Based on the work done in this paper, the next steps to be undertaken to extend the functionalities and analysis of the platform are:

- Implement a scaling protocol for the cloud management nodes so that they can quickly respond (scale up or down) to the platform demand, as defined in [21].
- Implement other functions applicable to the massively parallel processing paradigm such as Distributed IR, machine learning, signal processing, physics simulations, pattern recognition in image and video, data clustering, etc. which allow for further testing and analysis of the capabilities of the devices and to improve the platform
- Implement a classifier of tasks and worker nodes and intelligently allocate them according to the complexity requirements and capabilities.
- Implement Credit-based incentives model for contributing idle device time to the platform.

References

1. Intelligent Machines. Intel: Chips Will Have to Sacrifice Speed Gains for Energy Saving, https://bit.ly/1PERUYu, last accessed 2019/04/01

2. Muthu Pandi, K. & Somasundaram, K. Energy efficient in virtual infrastructure and green cloud computing: A review. Indian J. Sci. Technol. 9, (2016).
3. Zaib, S. J., Hassan, R. U. & Khan, O. F. Green Computing and Initiatives. International Journal of Computer Science and Mobile Computing, Vol.6 Issue.7, 49–55 (2017).
4. Kania-Richmond, A., Menard, M. B., Barberree, B. & Mohring, M. A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. J. Bodyw. Mov. Ther. 21, 274–283 (2017).
5. Eficiencia eléctrica para Centros de Datos, https://bit.ly/2I7y3Vl, last accessed 2019/04/01
6. Blem, E., Menon, J. & Sankar K. A Detailed Analysis of Contemporary ARM and x86 Architectures. High Perform. Comput. Archit. (HPCA), 2013 IEEE 19th Int. Symp. (2013).
7. Pang, B. Energy Consumption Analysis of ARM- based System. Aalto University School of Science Degree Programme of Mobile Computing, p. 68 (2011)
8. Petrocelli, D., De Giusti, A. E. & Naiouf, M. Procesamiento distribuido y paralelo de bajo costo basado en cloud & movil. XXIII Congreso Argentino de Ciencias de la Computación, XVIII Workshop de Procesamiento Distribuido y Paralelo (WPDP), 216–225 (2017).
9. Arslan, M. Y. et al. Computing while charging: Building a Distributed Computing Infrastructure Using Smartphones. 8th Int. Conf. Emerg. Netw. Exp. Technol. 193–204 (2012).
10. Gharat, V., Chaudhari, A., Gill, J. & Tripathi, S. Grid Computing In Smartphones. International Journal of Research and Scientific Innovation-IJRSI vol.3 issue 2, pp.76-84 (2016).
11. Miguel Castanheira Sanches, P. Distributed Computing in a Cloud of Mobile Phones. Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa FCT: DI - Dissertações de Mestrado, 41599 (2017).
12. Sriraman K R, Lead Architect.: Grid Computing on Mobile Devices - A Point of View. IEEE Xplore Conference: Grid Computing. Proceedings. IEEE/ACM Int. Workshop on (2004)
13. Lee, B. D. Empirical analysis of video partitioning methods for distributed HEVC encoding. Int. J. Multimed. Ubiquitous Eng. 10, 81–90 (2015).
14. Adriana Garcia, Hari Kalva, Borko Furht. A study of transcoding on cloud environments for video content delivery. MCMC '10 Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing Pages 13-18. Firenze, Italy - October 29. (2010)
15. Linux Encoding - x264 FFmpeg Options Guide, https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping, last accessed 2019/04/01
16. Weiser, C. Video Streaming. Media & Methods 38(4), 10–14 (2002).
17. New possibilities within video surveillance (White Paper), https://bit.ly/2Vg9iJQ, last accessed 2019/04/01
18. Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length, https://bitmovin.com/mpeg-dash-hls-segment-length/, last accessed 2019/04/01
19. Choosing the Optimal Segment Duration, https://bit.ly/2FLeMWe, last accessed 2019/04/01
20. Video Encoding Settings for H.264 Excellence, https://bit.ly/1yuCXwp, last accessed 2019/04/01.
21. Tiwana, B. et al. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis 105 (2010)
22. Li, D., Hao, S., Gui, J. & Halfond, W. G. J. An empirical study of the energy consumption of android applications. Proc.- 30th Int. Conf. Softw. Maint. Evol. ICSME 121–130 (2014).
23. Configure and estimate the costs for Azure products. https://bit.ly/2UwqLk8, last accessed 2019/04/01
24. Pricing Schemes in Cloud Computing: An Overview. Mazrekaj A Shabani I Sejdiu B. International Journal of Advanced Computer Science and Applications, vol 7 (2016).
25. 2018 Global Media Formats Report, https://bit.ly/2HXfSxn, last accessed 2019/04/01