An Empirical Evaluation of a Simple Energy Aware Scheduler for Mobile Grids

Alexander Perez Campos
Facultad de Ciencias Exactas
UNICEN
Tandil, Buenos Aires
Argentina
alexperezcampos@gmail.com

Juan Manuel Rodriguez
ISISTAN Research Institute
UNICEN-CONICET
Tandil, Buenos Aires
Argentina
juanmanuel.rodriguez@isistan.unicen.edu.ar

Alejandro Zunino
ISISTAN Research Institute
UNICEN-CONICET
Tandil, Buenos Aires
Argentina
alejandro.zunino@isistan.unicen.edu.ar

***Abstract*—Nowadays mobile devices are multi-core computers with considerable unused capabilities. Therefore, several researchers have considered harnessing the power of these battery-powered devices for distributed computing. Although their ever-growing capabilities, the fact that mobile devices run on battery poses a major challenge for applying traditional distributed computing techniques. Particularly, researchers aimed at using mobile devices as resources for executing computationally intensive task. Different job scheduling algorithms were proposed with this aim, but many of them require information that is unavailable or difficult to obtain in real-life environments, such as how much energy would require a job to be finished. In this context, Simple Energy Aware Scheduler (SEAS) is a scheduling technique for computational intensive Mobile Grids that only require easily accessible information. It was proposed in 2010 and it has been the base for a range of research work. Despite being described as easily implementable in real-life scenarios, SEAS and other SEAS-improvements works have always been evaluated using simulations. In this work, we present a distributed computing platform for mobile devices that support SEAS and empirical evaluation of the SEAS scheduler. The obtained result supports previous simulation results and by extension further validating other SEAS-based results.**

***Index Terms*—Mobile Grid, Energy Aware Scheduler, Job Scheduling, Mobile Device.**

I. Introduction

Mobile Grid computing has arisen as a consequence of the ever-growing mobile device capabilities [1], [2]. Nowadays, mobile devices are multi-core computers with several Gigabytes of RAM and storage. These characteristics give mobile devices the capability of executing different complex tasks [3], such as image processing, gaming, video streaming, and scientific computing. However, mobile devices are frequently underused, so researchers are attempting to take advantage of mobile devices for increasing the computational resources in different distributed computing environments, such as Grids or Clouds. A clear example of this is the port of the well-known BOINC platform for Android[1].

Although distributed computing is a well-established area, and there are many effective tools and techniques for performing distributed computing, integrating mobile devices poses new challenges that need to be addressed [2]. Mobility, on one hand, increases the probability of disconnection because mobile devices can move out of the wireless network area. Furthermore, wireless networks are slower and often more expensive than wired networks, such is the case of 3G/4G networks. On the other hand, mobile devices are battery dependent. Hence, overusing mobile devices can lead to battery depletion, which in turn results in reducing the available computational resources. Different works [4], [5], [6], [7], [8], [9] have aimed at providing resource scheduling schemes that take into account the energy consumption issue. Most of these works have been evaluated only through simulation. Furthermore, several of them require to know information that is not easy to collect or estimate in real-life scenario, such as how much energy requires a work to be executed in a particular node [5]. As a result, it is difficult to implement these schedulers for a real-life general-case scenario.

This work presents a distributed computing platform for mobile devices that supports Simple Energy Aware Scheduler (SEAS) [4]. SEAS is a computational-intensive job scheduler for mobile Grid that has been used as baseline for evaluating third-party scheduling approach [5] and as base for more complex scheduling techniques [10], [8], [9]. The main goals of this work are showing the feasibility of using SEAS in a platform, and empirically validating the SEAS, which has only been evaluated through simulation in previous works [4], [10], [8], [9]. Although simulation is a widely accepted practice in distributed system area, it is known that some times simulation might not result in accurate results for several reasons, such as incomplete models or badly parameterized ones [11]. Furthermore, in [10], the authors explicitly acknowledged that a limitation of the SEAS with Job-Stealing evaluation is that network energy consumption is not taken into account, and only [8] considers network energy consumption. The key contribution of this work is not only empirically validating the SEAS approach, but also showing that it can be integrated in a platform for real-life environments.

The rest of the paper is organized as follows. Section II outlines previous work in mobile distributed computing, focus-

[1]BOINC Android: https://play.google.com/store/apps/details?id=edu.berkeley.boinc

978-1-5386-3057-0/17/$31.00

ing on computational-intensive resource allocation when mobile devices are used as computational resources. Section III presents the SEAS approach and how it has been integrated into a platform that uses Android devices as mobile computing platform. Then, Section IV assess the effectiveness of SEAS when compared to traditional distributed job schedulers that are also supported by the presented platform. Finally, Section V concludes the paper highlight obtained results and future research lines.

II. Related work

Currently, there are many approaches for integrating mobile devices into distributed computing environments. Mobile devices can take advantage of distributed computing to offload its computation and reducing battery consumption [12], [13], [14]. This approach aims at using computational power provided by Cloud computing. Within this approach, there are different manners of offloading computations. Some approaches [12], [15] aim at offloading only the parts of the computation into the Cloud, being such parts usually CPU intensive. For instance, in a photo gallery application, the Cloud could be used for applying filters or effects to the pictures. On the other end, researchers [14] have proposed offloading whole applications and using mobile devices only as a remote front-end. In both cases, it is expected not only to reduce energy consumption, but also to improve application speed as Cloud servers' computational speed is usually higher than mobile device computational speed. Furthermore, Cloud computing can handle computational tasks that cannot be handled by a single mobile device.

In the literature, there are many works [4], [16], [17], [2], [10], [18], [5], [6], [7], [8], [9] addressing the issue of integrating mobile devices, as truly mobile battery-powered devices, into distributed environments as resource providers. These proposals are built over two types of network topology. The first topology is a network of self-organizing mobile devices that do no rely on a preexisting infrastructure. called Mobile Ad-hoc Networks (MANETs). In MANETs, mobile devices act as both hosts and routers [19]. Secondly, mobile devices can be connected to a network infrastructure by access-points. In this case, researchers have proposed two mobile Grid structures. The first is based on P2P technologies meaning that mobile devices in the Grid are self-organized in a network overlay. The second organization uses a central server to coordinate mobile devices connected to the mobile Grid [2]. In the latter, these servers might work as proxies between the Grid and the mobile devices, so that preexisting middlewares do not require to be aware of mobile devices characteristics.

Figure 1 outlines the proxy-based mobile Grid architecture. In this architecture, the Grid sees a set of mobile devices as a unique resource through the proxy. The proxy receives requests from the Grid and uses connected mobile devices for processing the request. In this scenario, the proxy has the responsibility of scheduling resources to execute the requests, which are called jobs. One of the main issues is defining a scheduling technique that takes into account mobile device characteristics. Several schedulers [2] have been proposed for maximizing the amount of job done with an energy budget.

In Section II-A several mobile Grid schedulers are discussed, presenting a general view of the state-of-the-art. Section II-B describes SEAS that is the main focus of this work and other SEAS-based schedulers. Finally, Section II-C outlines a real-live implementation of computing distributed systems using mobile devices.

A. Mobile Grid schedulers

In [16], [17], the authors propose schedulers for utility maximization in mobile Grids subject to energy constrains. Both works assume that jobs have a utility value. Hence, the main goal is not finishing the most jobs, but finishing the jobs that maximize the utility. Therefore, these scheduling techniques uses Lagrange multipliers for maximizing the utility function given energy constrains. The main difference between these works is how the schedulers model the utility functions and energy constrain functions. Despite these differences, both works require knowing how much energy would require to complete each job for each possible node in advance. These might be possible for very specific types of jobs and particular given devices, by profiling the jobs in the devices, but it is not possible for new jobs. There are many reasons that makes impossible to know energy consumption for an arbitrary job.

The authors of [18] propose a scheduler for wireless Grids that uses a non-cooperative game for assigning jobs to the mobile devices. As the schedulers mentioned above, the goal of this scheduler is to maximize the utility, but in this case each node competes for maximizing its gain. This negotiation is performed by different agents, namely the Matchmaker agent (MA) who matches jobs and potential executors (mobile devices), Negotiation agent (NA) who represents job executors, i.e., mobile devices, and Job allocation agent (JAA) who mediates between NAs and MA. Such agents run on nodes called resource brokers that can be seen as proxies. In this approach, jobs do not have an assigned utility, but a reserve price that is the maximum the user considers acceptable to pay for executing the job. Using this information, other information about the job, such as required power, memory and bandwidth, respectively, for job execution, and information about the mobile device, such as availability of battery power, memory and bandwidth, the MA selects who can perform each job. After that, the JAA initiates a negotiation between the different NAs representing suitable resources. Each NA have no information about other NAs, but knows job information. Hence, each NA can estimate the cost and benefit for executing a new job, but competes against other NAs offers. As a result of the negotiation each NA attempts to maximize its gain, while the whole system aim at minimizing the global price for executing a set of jobs.

In [6], the authors propose a scheduler whose main goal is to reduce energy consumption and job failure due to mobility. The authors assume that when a mobile device moves it might become unavailable because of a lack of coverage or that the amount of energy required for transmitting information,

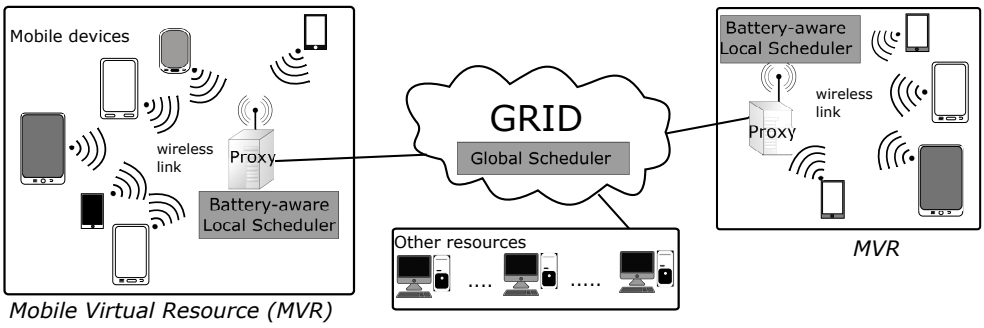

Figure 1. Mobile Grid architecture overview

e.g., job inputs/outputs, might vary. The latter is due to signal strength variation [20]. In order to predict mobile device locations, this work uses a Markov chain in which each mobile device is responsible for keeping historical information about its location. Using this information, it is possible to schedule jobs to mobile devices reducing the possibility of job failure and minimizing the energy consumption expected value. Authors also state that a job migration strategy is needed in case that mobile devices do not behave as expected. Such migration strategy is planned as a future work.

In [21] the authors analyze a wide range of scheduling algorithms for mobile Clouds based on a MANET architecture. In this Cloud, mobile devices act as both job executors and submitters. The evaluated strategies are extensions of traditional online (Minimum Execution Time and Minimum Completion Time) and batch (MinMin, MaxMin and Sufferage) scheduling heuristics. The main issue addressed by these scheduling strategies are related with communication concerns because the mobile Cloud is organized in an ad-hoc manner and this makes connections and routes to nodes very unreliable. To do so, the authors propose to take into account communication times and number of hops between nodes as part of the job execution time. Finally, this work outlines a novel heuristic, called MinHop, that only considers communication time when offloading tasks. As a result, the authors conclude that heuristics considering not only communication time and hops are more effective.

Mobile device services composition algorithm (MDSCA) [5] is another scheduler that considers mobile Grid scheduling as a market problem. Similarly to [16], [17], the authors present the scheduling problem as a maximization problem subject to restrictions that can be solve through Lagrange multipliers techniques. However, this approach relies on negotiation to solve the optimization problem. The main issue of this approach is that the required information is difficult to obtain in the general case. This work also provides evidence that the SEAS [4], which is the base for this work, achieves competitive results even when compared with more complex techniques that have access to more complete information. For instance, the execution success ratio of MDSCA is 91%, while the execution success ratio of SEAS is 90% according to their simulations [5].

Table I
SEAS REQUIRED INFORMATION

Notation	Meaning	Derived
pbc	Previous battery charge	No
pct	Previous battery event time	No
bc	Current battery charge	No
ct	Current time	No
$benchmark_i$	Device i speed	No
$number\ jobs_i$	Amount of jobs assigned to device i	No
dr	Discharge rate	Yes
rt	Remaining up-time at current dr	Yes
$estimatedUpTime_i$	Device i estimated up-time	Yes

B. SEAS-based Schedulers

SEAS [4] is a scheduling algorithm that requires information easily obtainable in a mobile device, namely its approximated processing speed, its current battery level and its workload. Since battery charge is informed in a discrete manner, battery charge arrives as an event. This can be observed in different mobile OS SDKs. For instance, in Android, battery charge is informed through Android intents[2], such intents are generated by the system, and the developers must register their listeners, called BroadcastReceiver, into the system.

For each device, SEAS needs to estimate the expected up-time for each mobile device. In order to perform such estimation, SEAS needs at least two battery events of a mobile device or an initial discharge rate. Expected up-time is re-estimated every time that a new battery event arrives. The first step is to estimate the discharge rate, as follows:

$$dr = \frac{pbc - bc}{ct - pct}$$

By assuming that discharge rate is constant, the remaining time (rt) can be calculated as:

[2]Android Monitoring Battery: https://developer.android.com/training/monitoring-device-state/battery-monitoring.html

$$rt = \frac{bc}{dr}$$

However, the discharge rate changes over the time [22], especially due to executing computations on the mobile devices [23]. In the original SEAS paper [4], it is stated that rt has a high variance, which might affect negatively the scheduling performance. To reduce such variation, the original SEAS uses Algorithm 1 to better estimate the remaining up-time. It basically averages previous estimations with the new one. The main issue is that previous estimations where made for the past, so they have to be corrected for the present. This can be observed in line 10 of the Algorithm 1.

When a new job arrives to the proxy, SEAS aims at balancing the amount or resources assigned for the job. To do so, SEAS estimates the amount of computational power that each mobile device can deliver, which is calculated multiplying the device speed, which is obtained executing a benchmark in the mobile device, by its estimated up time. This represents the computational resources available in the mobile device. Then, these resources are divided by the number of jobs assigned to that mobile devices, as follows:

$$resources\,per\,job_i = \frac{estimatedUpTime_i \times benchmark_i}{number\,jobs_i + 1}$$

SEAS selects the mobile device that have more resources per job, as a result the load in the Grid is distributed evenly considering not only the computational power, but the estimated up-time. Since this estimated up-time depends on the discharge rate and the current battery capacity, energy is considered. Moreover, SEAS indirectly considers CPU efficiency, as more efficient CPUs would deliver more up-time resulting in more computational capabilities.

SEAS was originally evaluated using energy profiles obtained from a wide range of notebooks and netbooks. In total, the experiment comprises eight simulations using different topology, from Grids of only 4 devices up to 400 devices. Furthermore, two kinds of jobs were considered during the evaluation, namely short jobs with high standard deviation in the execution times (8 minutes ±50%) and long jobs with low standard deviation in the execution time (24 minutes ±24%). SEAS was compared with a Round Robin scheduler and a Random scheduler. In the former comparison, SEAS finished from 1% up to 8.55% more jobs than the Round Robin scheduler, while outperformed the Random scheduler by finishing from 6% up to 11% mode jobs.

In [10], SEAS was extended using job stealing, that means that when a node finishes all its assigned jobs, it attempts to off-load other nodes by executing their jobs. This work uses SEAS for scheduling jobs when they first arrive to the Grid. When a node finishes all its jobs, it tries to steal jobs from other nodes. In order to select the node from whom to steal, three strategies were evaluated. Two of them use SEAS ranking, namely *Best Ranking Aware Stealing (BRAS)* and *Worst Ranking Aware Stealing (WRAS)*. *BRAS* selects the node whose resource per job is higher. The main goal is to increase the number of free nodes, which results in a high number of free nodes that can steal. In contrast, *WRAS* goal is to reduce the load of nodes that are most unlikely to finish their jobs. The last strategy is *Random Stealing (RS)*, i.e., selecting a random node who has jobs. *RS* is a well-known strategy in job scheduling [24], [25], [26]. These strategies can be combined with two strategies to determine the number of jobs to steal. Firstly, *Fixed Number* is a strategy that always steals the same number of jobs. Secondly, *Exponential* strategy duplicates the number of jobs that a node steals each time the node performs an steal. This means that the first time a node steals jobs, it steals 2^0 jobs, the second time 2^1, the third time 2^2 jobs, and so on. According to the results, *WRAS with Fixed Number* performed in most of the experimental scenarios, *BRAS with Fixes Number* was a closed second, and *WRAS with Exponential* was a closed third. The other combinations and SEAS alone were drastically outperformed. In this case, smartphones and tablets profiles were used.

Since job stealing might add extra time and energy consumption due to large-input job transmissions, other ranking techniques were analyzed in [8]. Three new ranking strategies are proposed, namely *Enhanced SEAS (E-SEAS)*, *Job Energy Aware Criterion (JEC)*, and the *Future Work Aware Criterion (FWC)*. E-SEAS is similar to SEAS, but uses the current battery level instead of the estimated up-time. This is due to the fact that, in the profiled Android devices, battery charge has a stable behavior. JEC uses a per-calculated factor that is obtained by dividing the maximum battery level (usually 100) by the number of benchmarks that can be executed on a full-charge. This factor relates battery level with available computational resources. The main drawback of this approach is that for each new device, this factor should be calculated, and it does not consider battery worn out. Finally, FWC uses job execution history to estimate job length in that device, the main problem of this approach is cold-start. Results shown that E-SEAS and JEC perform as well as SEAS with job stealing in most of the scenarios. Finally, job stealing was also applied to these techniques in [9]. This works add network energy consumption to the simulation. Such source of energy consumption was not considered in previous SEAS based works [4], [10], [8].

Although SEAS was designed aiming at been simple to implement, SEAS and all its extensions have been evaluated using simulation. This is an standard practice in the area as some of other approaches, such as [16], [18], would be difficult to be implemented in real life scenarios because the required information might be unavailable . Another issue which encourage simulation is that some approaches require too much historic information and benchmarking, such as location history [6] or JEC estimation [8]. This has resulted in that few works, such as [27], [28], are actually evaluated through real-life implementation, while the vast majority [4], [16], [10], [18], [6], [8], [9] are validated via simulation. Despite being a common practice, simulation does not replace real-life experiments as models might be incomplete or parameters

Algorithm 1 Battery time estimation

```
1: procedure UPDATESTIMATEDUPTIME(bc, ct, pbc, pct, historic_rt, historic_ct)
2:     dr ← (pbc − bc)/(ct − pct)
3:     rt ← bc/dr
4:     ADD(rt, historic_rt)                          ▷ Saves current rt into historic information
5:     ADD(ct, historic_ct)                          ▷ Saves current ct into historic information
6:     estimatedUpTime ← 0
7:     for i ← 0 to LEN(historic_rt) do
8:         art ←GET(historic_rt, i)
9:         act ←GET(historic_ct, i)
10:        estimatedUpTime ← estimatedUpTime + art − (ct − act)   ▷ art was estimated for act, it should be corrected
11:    end for
12:    estimatedUpTime ← estimatedUpTime/LEN(historic_rt)        ▷ Averaging estimations
13:    return estimatedUpTime
14: end procedure
```

could be wrongly set [11]. Hence, it cannot be said that SEAS has been completely validated.

C. Mobile distributed computing platforms

Recently, Computing While Charging (CWC) was presented in [28]. The authors have profiled fifteen users showing that most of the time mobile devices are charged overnight leaving them available during large periods of times. According to this work, using mobile devices over traditional PC for processing reduces energy consumption. In particular, a comparison between processing power and energy consumption of Intel Core 2 Duo and five smartphones shows that an enterprise with 100 employees can save up to U$S 300 in USA only by moving its computations to employees' mobile devices. Another important issue addressed by this work is who to load new job definitions in mobile devices, i.e., it presents a technically feasible mechanism to dynamically load and execute code. Otherwise, the application should be updated each time a new type of job is added. This is impractical because the size of the application increases as new job types are added, and there is no control of when a mobile device owner would update the installed application.

Mobile devices have also been considered as computational resources for distributed computing in real-life scenarios. One example of this is the BOINC port to Android[3]. The existence of such application is strong evidence that mobile devices can offer enough computational capabilities to be seriously considered as valuable resources for scientific computation. Despite taking advantage of Android mobile devices, this version of BOINC only uses such mobile devices when they are plugged to AC and running on WiFi. This makes BOINC and CWC similar in the vision of how to exploit mobile device capabilities. Since this version of BOINC considers mobile devices working in similar condition to fixed devices (PC or Servers), it does not cope with the issues resulting from mobile devices mobility and their limited energy supply.

[3]BOINC Android FAQ: http://boinc.berkeley.edu/wiki/Android_FAQ

Finally, DroidCluster [27] is an Android middleware for implementing cluster computing. The main goal of DroidCluster was to assess mobile device capability as computational resource providers without modifying the operating system. For the evaluation, a MPI version of the well-known LINPACK benchmark was used. To create the cluster, the authors used up to six LG P500, which are described as 2011 midrange mobile device. According to the reported results, a single mobile device achieved 5.81 MFlops, while the performance of the cluster was 29.04 MFlops when using six devices, i.e., a speed-up of about 5 times.

Briefly, there are many scheduling algorithms for mobile Grids that consider energy. However, they are not implemented in real-life scenarios. On the other hand, real-life implementation of mobile Grids or clusters disregard energy consumption. DroidCluster [27] disregards this because it is a usability analysis of mobile devices capabilities, while CWC [28] and BOINC only use mobile devices when they are charging. Therefore, the main contribution of this work is to propose and to assess an distributed computing platform for mobile devices that supports SEAS in a real setting.

III. SEAS PLATFORM FOR DISTRIBUTED COMPUTING

This section presents a platform for distributed computing that uses SEAS as its jobs scheduler. This platform was implemented for Android mobile devices. Android was selected because it is not only an open source mobile device operative system, but also one of most popular operating systems in the world. According to Google, there are more than 1 billion active Android users in the Play Store[4]. The platform was designed following SEAS assumption, so it has two main components. One of these components is the proxy that receives the jobs to execute and assigns them to a mobile device. The other component is the node applications, which is an Android application. This application is responsible of executing the assigned jobs and sending the results to the proxy.

[4]Google Play Developer console: https://developer.android.com/distribute/console/index.html

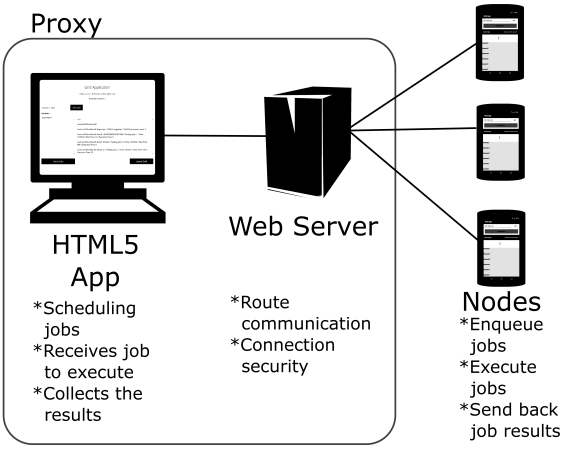

Figure 2. SEAS Implementation deployment

Figure 2 depicts a deployment of the SEAS implementation. To deploy the proxy, it is necessary to run a Web Server that acts as the communication interface between the mobile devices and the proxy logic. This Web server is implemented using Java Servlets and supports communication through WebSockets[5]. There are two main reasons for using WebSockets as communication protocols. Firstly, WebSocket technology is based on traditional Web technology, and by using standard ports and protocols it is less likely that network providers filter such communication. For instance, it supports HTTP proxy tunneling, and HTTP proxy technology is common place in enterprises. Furthermore, WebSockets support SSL using HTTPS, i.e., it can be secured easily in typical scenarios.

The proxy logic is currently implemented through a HMTL5 application. This implementation allows to easily control the Grid using a standard Web browser. Since the logic is implemented through HTML5, schedulers, including SEAS, are implemented in Javascript. This allows developers to add new schedulers in a simple manner. However, the logic can be easily implemented using other technology, for instance moving the HTML5 application to a REST service front-end, because there is a decoupling between the proxy logic and the communication module.

The Android application has a UI that allows mobile device owners connecting/disconnecting from the Grid proxy and knowing which job are currently in the mobile device queue. In order to inform CPU speed, the application uses BogoMIPS as reported by the Linux kernel in "`/proc/cpuinfo`". BogoMIPS stands for bogus MIPS and Linux calculates them at boot time to calibrate some timing loops. Although BogoMIPS might be an unreliable benchmark, it is inexpensive because Linux calculates it anyway. Furthermore, this benchmark has been used in other distributed systems for load balancing when heterogeneous resources are present [29]. This information is send to the server upon connection.

[5]RFC 6455 - The WebSocket Protocol: https://tools.ietf.org/html/rfc6455

To sense and report battery information, the application registers a BroadcastReceiver, which is an Android class intended to process Intents, to listen for battery intents (`Intent.ACTION_BATTERY_CHANGED`). In addition to work as events, Android Intents also might convey information. The expected information in an Intent depends on the type of the Intent. In the case of battery related Intents, the expected information includes battery level, its health status, the battery scale (maximum possible battery level, which usually is 100), whether the mobile device is charging, among other information. In this case, the application it is only interested in the battery level and the scale. Each time a new battery Intent is received, the information is sent to the proxy, so it can update the mobile device status.

Finally, regarding jobs, the current implementation only supports executing two types of jobs. This is because the current version of the platform does not implement a remote class loader, so adding new jobs requires to recompile, repackage, and reinstall the whole mobile application. The first kind of job is calculating the Fibonacci value for a number and the other is calculating its factorial. The Fibonacci is a CPU and memory intensive task because it calculates recursively the values, but puts them into a cache to avoid recalculation. As a result, several objects are stored in memory, which in turn result in more garbage collection. Notice that this cache is not share among tasks, so garbage is generated each time the task is executed. As a result, a fair amount of CPU and memory load is generated each time a Fibonacci job is executed, which is desirable for testing purposes. The implementation used makes factorial a CPU intensive task because it calculates the result recursively without using a caching result strategy.

Listing 1. Fibonacci implementation

```
public class Fibonacci {
  HashMap<Integer, Long> cache;
  public Fibonacci() {
    cache = new HashMap<>();
    cache.put(0, 0l);
    cache.put(1, 1l);
  }
  public Long calculate(Integer n) {
    Long cachedResult = cache.get(n);
    if (cachedResult != null)
      return cachedResult;
    else {
      Long result = calculate (n - 1) +
        calculate(n - 2);
      cache.put(n,result);
      return result;
    }
  }
}
```

Listing 2. Factorial implementation

```
public class Factorial {
  public Long calculate(long number) {
```

Table II
MOBILE DEVICES

Device	Processor	Memory	Battery
Samsung I5500	Single Core 600 MHz.	256 Mb.	1200 mAh
Samsung I9300	Quad-core 1.4 GHz Cortex-A9	1 Gb.	2100 mAh
Acer A100	Dual-core 1.0 GHz Cortex-A9	1 Gb.	1530 mAh
LG Optimus L9	Dual-core 1.0 GHz Cortex-A9	1 Gb.	2150 mAh

```
    if (number <= 1)
      return 1L;
    else
      return number *
        calculate(number - 1);
  }
}
```

Listing 1 depicts the Fibonacci implementation. Notice that upon creation, the Fibonacci class creates an associative cache and puts results for $fib(0)$ and $fib(1)$. Although it is not explicit, this requires using Integer and Long objects because HashMap require objects instead of primitive types.

Currently, the job model is inflexible for a real-life Grid. However, this can be easily solved by harnessing the power of DexClassLoader for dynamically loading new jobs. The potential of this approach has not only been discussed, but also implemented in CWC [28]. Furthermore, this work also provides a code-snippet showing how to implement such class loading mechanism.

IV. EMPIRICAL EVALUATION

To evaluate SEAS against traditional scheduling approaches, the proxy Web application was extended for submitting random jobs at random intervals. The goal is to generate random load in the Grid, evaluating how the different scheduling technique affects the energy consumption. In addition to SEAS, we evaluated Random and Round Robin scheduling algorithms. The goal of evaluating such traditional algorithms was to validate SEAS original results presented in [4], which were simulated. These results point out that SEAS uses more efficiently mobile devices than Round Robin or Random schedulers.

Table II presents the devices used to create the mobile Grid. For evaluating each scheduler, all the devices were fully charged, then connected to the mobile Grid, and the job submission process was started. The experiment finished when all mobile devices were disconnected from the Grid. The jobs were generated using the following logic:

1) Wait a random interval between 1 second and 5 seconds.
2) If no mobile device is connected, finish the experiment.
3) Decide randomly how many jobs to create between 1 and 10.
4) For each job:
   a) Select randomly the type (Fibonacci or factorial).
   b) Select the input, a random number between 1 and 60.
   c) Add it to a queue.
5) Send the jobs.

Notice that all random generations followed a uniform distribution, so no value was preferred over the others.

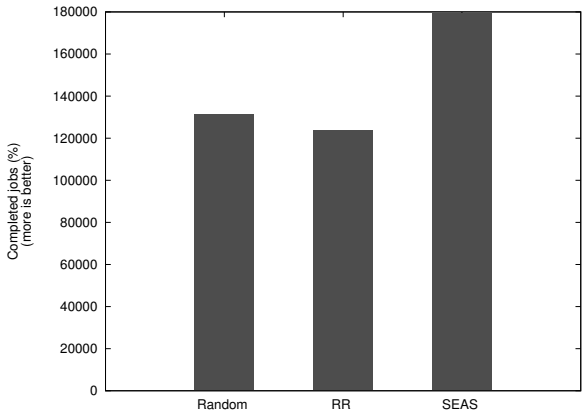

Figure 3. Finished jobs

Figure 3 presents the results obtained after executing the experiments. As expected, SEAS outperformed Round Robin and Random scheduler. In particular, the mobile Grid executed 26% more jobs using SEAS when compared with Random scheduling, and 31% when compared with Round Robin. This results where higher that the ones originally reported in [4]. However, there are several differences in this experiment that might account for these discrepancies. Firstly, in this work, smartphones and tablet were used instead of notebooks and netbooks. Secondly, the original SEAS simulator did not take into account energy consumption due to executing a task. This was introduced in the simulation presented in [10]. Thirdly, the battery recovery effect was not simulated. This effect is observed when available energy is less than the difference between battery charge and battery consumed. When a battery discharge rapidly, an internal imbalance makes some energy unavailable; it might become available if the energy requirements to the battery decreases, giving time to rebalance its internal charge [30], [31]. As a result, the battery level might increase. Since SEAS considers battery level, it might reduce the load on heavily used nodes given the battery time to experience this recovery effect.

Figure 4 depicts how the different schedulers distribute the jobs between different nodes. As expected, Round Robin and Random scheduler uniformly distributed the jobs, while SEAS have a clear preference for some device over others. As a result, the SEAS improve energy efficiency with regard to finished jobs.

V. CONCLUSION

The platform empirical evaluation presented in this work confirms that SEAS performed better than Random and Round

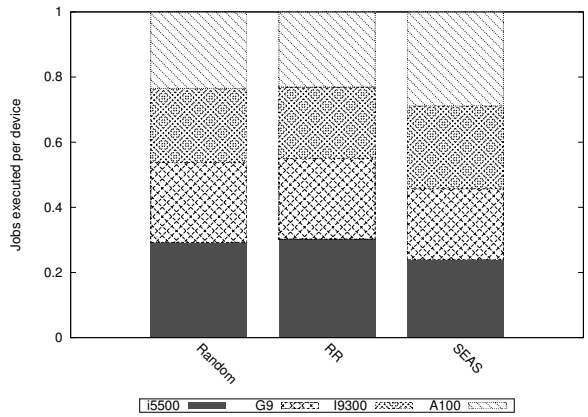

Figure 4. Relative executed jobs per device

Robin as expected. Such results enforce previous findings that were obtained through simulation. The main conclusion of this work is that simulation results are reliable enough for drawing general conclusion of how a mobile Grid scheduler will perform. Additionally, this work shows that implementing an energy aware scheduler for mobile Grid is feasible.

In addition, this work presents new research question for future works. Firstly, the experiments should be repeated with a larger Grid and other parameter for generating the jobs. In addition, more time-consuming jobs should be consider to make the experiments more comparable to experiments presented in other works [4], [10], [8]. Furthermore, testing the same scenarios through simulation and using SEAS implementation will allow to assess the simulator accuracy, and in turn test new schedulers through simulation techniques. Despite existing a SEAS implementation, simulation is a great tool for experiment because it does not require to have hundreds of devices for experimenting. Moreover, simulation gives the means for performing repeatable experiments.

Finally, we will further refine the implementation to make it usable in different scenarios. Firstly, the current application communication protocol is insecure, so support for a more robust and secure communication method must be designed and implemented. Other point to improve is the reported information. In future versions, the mobile application will report more information, such as wireless network signal strength, location, and current device usage to allow better predictions of mobile device availability, using techniques based on machine learning [32], [33]. Lastly, the mobile application requires the ability of loading jobs dynamically to be usable in real-life scenarios.

ACKNOWLEDGMENT

We also acknowledge the financial support provided by ANPCyT through PICT-2013-0464, and CONICET through PIP 11220120100185CO.

REFERENCES

[1] J. Aron, "Harness unused smartphone power for a computing boost," *New Scientist*, vol. 215, no. 2880, pp. 18 –, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262407912622556

[2] J. M. Rodriguez, A. Zunino, and M. Campo, "Introducing mobile devices into grid systems: a survey," *International Journal of Web and Grid Services*, vol. 7, no. 1, pp. 1–40, 2011.

[3] F. A. Silva, G. Zaicaner, E. Quesado, M. Dornelas, B. Silva, and P. Maciel, "Benchmark applications used in mobile cloud computing research: a systematic mapping study," *The Journal of Supercomputing*, vol. 72, no. 4, pp. 1431–1452, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11227-016-1674-2

[4] J. M. Rodriguez, A. Zunino, and M. Campo, "Mobile grid seas: Simple energy-aware scheduler," in *3rd High-Performance Computing Symposium. 39th JAIIO*, 2010.

[5] L. Chunlin and L. Layuan, "Exploiting composition of mobile devices for maximizing user qos under energy constraints in mobile grid," *Information Sciences*, vol. 279, pp. 654 – 670, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025514004654

[6] S. C. Shah, "Energy efficient and robust allocation of interdependent tasks on mobile ad hoc computational Grid," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 5, pp. 1226–1254, 2015, cPE-13-0007.R2. [Online]. Available: http://dx.doi.org/10.1002/cpe.3297

[7] S. W. Loke, K. Napier, A. Alali, N. Fernando, and W. Rahayu, "Mobile computations with surrounding devices: Proximity sensing and multilayered work stealing," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 2, pp. 22:1–22:25, Feb. 2015. [Online]. Available: http://doi.acm.org/10.1145/2656214

[8] M. Hirsch, J. M. Rodriguez, A. Zunino, and C. Mateos, "Battery-aware centralized schedulers for cpu-bound jobs in mobile grids," *Pervasive and Mobile Computing*, vol. 29, pp. 73 – 94, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119215001637

[9] M. Hirsch, J. M. Rodriguez, C. Mateos, and A. Zunino, "A two-phase energy-aware scheduling approach for CPU-intensive jobs in mobile Grids," *Journal of Grid Computing*, vol. 15, no. 1, pp. 55–80, 2017.

[10] J. Rodriguez, C. Mateos, and A. Zunino, "Energy-efficient job stealing for cpu-intensive processing in mobile devices," *Computing*, vol. 96, no. 2, pp. 87–117, 2 2014. [Online]. Available: http://dx.doi.org/10.1007/s00607-012-0245-5

[11] P. Velho, L. M. Schnorr, H. Casanova, and A. Legrand, "On the validity of flow-level tcp network models for grid and cloud simulations," *ACM Trans. Model. Comput. Simul.*, vol. 23, no. 4, pp. 23:1–23:26, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2517448

[12] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[13] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84 – 106, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001318

[14] X. Sun and N. Ansari, "Green cloudlet network: A distributed green mobile cloud network," *IEEE Network*, vol. 31, no. 1, pp. 64–70, January 2017.

[15] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, July 2012, pp. 784–791.

[16] C. Li and L. Li, "Tradeoffs between energy consumption and qos in mobile grid," *The Journal of Supercomputing*, vol. 55, pp. 367–399, 2011.

[17] ——, "A multi-agent-based model for service-oriented interaction in a mobile grid computing environment," *Pervasive and Mobile Computing*, vol. 7, no. 2, pp. 270 – 284, 2011.

[18] M. N. Birje, S. S. Manvi, and S. K. Das, "Reliable resources brokering scheme in wireless Grids based on non-cooperative bargaining game," *Journal of Network and Computer Applications*, vol. 39, pp. 266 – 279, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804513001677

[19] K. Chen, H. Shen, and H. Zhang, "Leveraging social networks for p2p content-based file sharing in disconnected manets," *IEEE Transactions on Mobile Computing*, vol. 13, no. 2, pp. 235–249, Feb 2014.

[20] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 317–328. [Online]. Available: http://doi.acm.org/10.1145/2348543.2348583

[21] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, pp. 1–28, 2015.

[22] W. X. Shen, C. C. Chan, E. W. C. Lo, and K. T. Chau, "Estimation of battery available capacity under variable discharge currents," *Journal of Power Sources*, vol. 103, no. 2, pp. 180 – 187, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/B6TH1-44V3JXV-2/2/77bf800f9c9901c16d550f67a4a31e6b

[23] A. Rodriguez, C. Mateos, and A. Zunino, "Improving scientific application execution on android mobile devices via code refactorings," *Software: Practice and Experience*, pp. n/a–n/a, 2016, spe.2419. [Online]. Available: http://dx.doi.org/10.1002/spe.2419

[24] R. V. Van Nieuwpoort, G. Wrzesińska, C. J. H. Jacobs, and H. E. Bal, "Satin: A high-level and efficient grid programming model," *ACM Trans. Program. Lang. Syst.*, vol. 32, no. 3, pp. 9:1–9:39, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1709093.1709096

[25] R. B. Rosinha, C. F. R. Geyer, and P. K. Vargas, "Wspe: a peer-to-peer grid programming environment," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 13, pp. 1709–1724, 2009. [Online]. Available: http://dx.doi.org/10.1002/cpe.1392

[26] B.-Y. Zhang, G.-W. Yang, and W.-M. Zheng, "Jcluster: an efficient java parallel environment on a large-scale heterogeneous cluster," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 12, pp. 1541–1557, 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.986

[27] F. Büsching, S. Schildt, and L. Wolf, "Droidcluster: Towards smartphone cluster computing – the streets are paved with potential computer clusters," in *2012 32nd International Conference on Distributed Computing Systems Workshops*, June 2012, pp. 114–117.

[28] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Cwc: A distributed computing infrastructure using smartphones," *IEEE Transactions on Mobile Computing*, vol. 14, no. 8, pp. 1587–1600, Aug 2015.

[29] C. A. Bohn and G. B. Lamont, "Load balancing for heterogeneous clusters of {PCs}," *Future Generation Computer Systems*, vol. 18, no. 3, pp. 389 – 400, 2002, cluster Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X01000589

[30] M. R. Jongerden and B. R. Haverkort, "Which battery model to use?" *IET Software*, vol. 3, no. 6, pp. 445–457, December 2009.

[31] C. K. Chau, F. Qin, S. Sayed, M. H. Wahab, and Y. Yang, "Harnessing battery recovery effect in wireless sensor networks: Experiments and analysis," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 7, pp. 1222–1232, September 2010.

[32] C. Rios, J. M. Rodriguez, D. Godoy, S. Schiaffino, and A. Zunino, "Usage pattern mining for smartphone use personalization," in *XIII Brazilian Symposium on Human Factors in Computer Systems*, 2014.

[33] J. M. Rodriguez, A. Zunino, A. Tommasel, and C. Mateos, *Encyclopedia of Information Science and Technology, Fourth Edition*. IGI Global, 2017, ch. Recurrent Neural Networks for Predicting Mobile Device State, p. In press.