Simple Energy Aware Scheduler: An Empirical Evaluation

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

and

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

and

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

Abstract

Mobile devices have evolved from single purpose devices, such as mobile phones, into general purpose multi-core computers with considerable unused capabilities. Therefore, several researchers have considered harnessing the power of these battery-powered devices for distributed computing. Despite their ever-growing capabilities, using battery as power source for mobile devices represents a major challenge for applying traditional distributed computing techniques. Particularly, researchers aimed at using mobile devices as resources for executing computationally intensive tasks. 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. This evaluation followed the methodology of the original SEAS evaluation, in which Random and Round Robin schedulers were used as baselines. Although the original evaluation was performed by simulation using notebooks profile instead of smartphones and tablets, results confirms that SEAS outperforms the baseline schedulers.

Keywords: Mobile Grid, Energy Aware Scheduler, Job Scheduling, Mobile Device.

1 Introduction

Using mobile devices as resource for distributed computing is a direct result of the mobile devices ever-increasing capabilities [1]. 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, 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 [1]. 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 [2, 3, 4, 5, 6, 7] 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.

Another issue of several of the proposed schedulers [8, 3, 6] for mobile Grids is that they require to know information that is not easy to collect or estimate in real-life scenarios, such as how much energy requires a work to be executed in a particular node, or how long it would take to execute that job in a given node. Although this information can be assumed in a simulation, these schedulers are difficult to implement for a real-life general-case scenario. Additionally, effectively simulating battery behavior is complex because there are different physical phenomena to emulate. Ideally, a battery contains a certain amount of power and the device can use all the battery capacity before going off-line. However, the device might not be able to consume all the available energy. If the device consumes energy too fast, an imbalance in the battery reduces the amount of energy that the device can access, but if the device reduces its rate of consumption, the battery has an opportunity to rebalance its state and offer more energy. As a result, it seems as if the battery recharges itself in what is known as the recovery effect [9]. To our knowledge, there is no simulation for mobile Grids that considers such effect.

This work presents a distributed computing platform for mobile devices that supports Simple Energy Aware Scheduler (SEAS) [2]. SEAS is a computational-intensive job scheduler for mobile Grid that has been used as baseline for evaluating third-party scheduling approach [3], called Mobile device services composition algorithm. Despite its simplicity, SEAS performs competitively when compared against the later and other schedulers, such as [10, 11]. In addition, SEAS is the base of several more complex scheduling techniques [12, 6, 7]. The main goals of this work are showing the feasibility of using SEAS in a platform. Therefore, empirically validating SEAS, which has only been evaluated through simulation in previous works [2, 12, 6, 7], which partially validate these algorithms. Although simulation is a widely accepted practice in distributed systems, it is known that sometimes simulation might not result in accurate results for several reasons, such as incomplete models or badly parameterized ones [13]. In addition to limitations in battery models, in [12], the authors explicitly acknowledged that a limitation of SEAS with Job-Stealing evaluation is that network energy consumption is not taken into account. Although the later phenomenon is added in the model presented in [6], simulation might not effectively consider all the possible situations. Hence, one key contribution of this work is empirically validating the SEAS approach. Besides, the other key contribution is showing that SEAS can be integrated in a platform for real-life environments.

This work is based on previous attempts to harness mobile device for distributed computing, such as [14, 15, 16] or BOINC for Android. Although these works are platforms for performing distributed computing using mobile devices, they are not aimed at using energy aware schedulers. Moreover, BOINC for Android and [16] only use mobile devices while they are charging. As a result, reducing energy consumption is not as important as when computing on battery, which is the scenario discussed in [2]. As a result, the presented platform allowsharnessing mobile devices in a manner that differs significantly from previous works.

The rest of the paper is organized as follows. Section 2 outlines previous work in mobile distributed computing, focusing on computational-intensive resource allocation when mobile devices are used as computational resources. Section 3 presents the SEAS approach and its integration into a platform that uses Android devices as mobile computing platform. Then, Section 4 assesses the effectiveness of SEAS when compared to traditional distributed job schedulers that are also supported by the presented platform. Finally, Section 5 concludes the paper, highlights obtained results and describes future research lines.

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

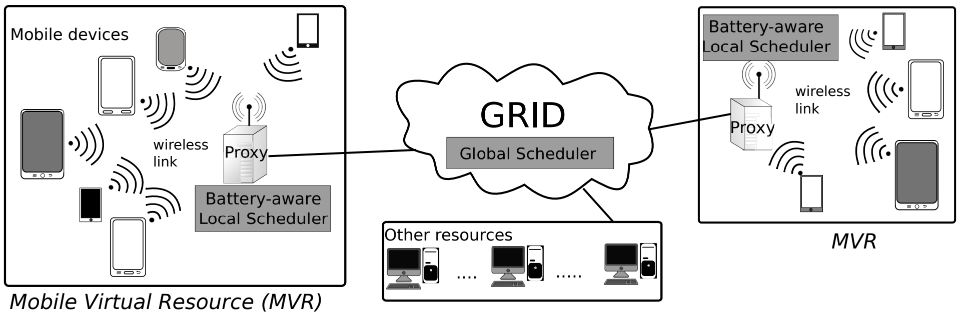


Figure 1: Mobile Grid architecture overview

2 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 their computations and reducing battery consumption [17, 18]. This approach aims at using computational power provided by Cloud computing. Within this approach, there are different manners of offloading computations. Some approaches [19] 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 [18] 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 [2, 8, 1, 12, 20, 3, 4, 5, 6, 7] 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 not rely on a preexisting infrastructure, called Mobile Ad-hoc Networks (MANETs). In MANETs, mobile devices act as both hosts and routers [21]. 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 [1]. In the latter, these servers might work as proxies between the Grid and the mobile devices, so that preexisting middlewares do not require being 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 [1] have been proposed for maximizing the amount of job done with an energy budget.

Since MANETs only depend on the mobile devices connected wirelessly among them, MANETs are ideal for scenarios where infrastructure based networks are unreliable or not available [22]. Examples are military or emergency scenarios. However, in most scenarios, infrastructure based networks can be assumed. Furthermore, wireless network technologies, such as Mobile IP, 4G, and IEEE 802.11, take into account mobile device mobility. Additionally, they are constantly evolving and incrementing their speed and reliability [23, 24]. Hence, most of the works in mobile Grid [1] use infrastructure based networks and are proxy-based Grids.

In Section 2.1 several mobile Grid schedulers are discussed, presenting a general view of the state-of-the-art. Section 2.2 describes SEAS that is the main focus of this work and other SEAS-based schedulers. Finally, Section 2.3 outlines a real-live implementation of computing distributed systems using mobile devices.

2.1 Mobile Grid schedulers

In [8], 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 use 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 completing each job for each possible node in advance. This 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 the general case. There are many reasons that makes impossible to know energy consumption for an arbitrary job.

The authors of [20] 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 who matches jobs and potential executors (mobile devices), Negotiation agent who represents job executors, i.e., mobile devices, and Job allocation agent who mediates between Negotiation agents and the Matchmaker agent. 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 Matchmaker agent selects who can perform each job. After that, the Job allocation agent initiates a negotiation between the different Negotiation agents representing suitable resources. Each Negotiation agent have no information about other Negotiation agents, but knows job information. Hence, each Negotiation agent can estimate the cost and benefit for executing a new job, but competes against other Negotiation agents offers. As a result of the negotiation each Negotiation agent attempts to maximize its gain, while the whole system aims at minimizing the global price for executing a set of jobs.

In [4], 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, e.g., job inputs/outputs, might vary. The latter is due to signal strength variation [25]. 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 [26] 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 issues 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.

The Hybrid Ant Colony algorithm based Application Scheduling [27] is an Ant-Colony-based scheduler algorithm for applications in a local mobile Cloud. This work aims at maximizing the profit while running applications in a resource constrained environment. The profit for executing a given applications is known. Additionally, applications are not atomic and partially executing the application results in a partial profit, which is directly proportional to the percentage of completion. In addition, this approach needs to know how many resources are needed for executing an application in a particular device. This is a drawback of the algorithm as, in the general case, it is not possible to know how many resources it would take to execute an application in a given node. Finally, this approach has only been evaluated through simulation.

The Adaptive Probabilistic Scheduler [28] is a decentralized two-phase scheduler for Mobile Cloud Computing. The phases are the *resource discovery phase* and the *adaptive probabilistic scheduling phase*. In the discovery phase participant nodes periodically exchange information of their computing capacity, power and queue length. In the scheduling phase, a source node assigns tasks to nearby collaborators. When a task is summited, the submitter also must provide the required amount of computation, size of data to be transferred, and time constraints. Moreover, time of computation and communication on every potential processing node is known and used to generate a set of candidate nodes that best meet task time constraints. Each node of such set is assigned with a probability based on the relative energy consumption of tasks.

Mobile device services composition algorithm [3] is another scheduler that considers mobile Grid scheduling as a market problem. Similarly to [8], 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

Table 1: 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

the required information is difficult to obtain in the general case. This work also provides evidence that the SEAS [2], 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 the algorithm is 91%, while the execution success ratio of SEAS is 90% according to their simulations [3].

2.2 SEAS-based Schedulers

SEAS was introduced in [2]. It is a scheduling algorithm designed to only require easily obtainable information because either it is provided by most mobile operating systems APIs or it can be calculated from the mobile device state. In particular, SEAS requires the following information from the mobile devices:

- approximated processing speed, which can be estimated using benchmarks.
- current battery level, which can be accessed using standard APIs in all mobile device platforms.
- workload, which is the number of unfinished assigned jobs.

Since battery charge is informed in a discrete manner, battery charge changes arrive as discrete events. This behavior can be observed in different mobile OS. 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.

Table 1 outlines SEAS required information for scheduling each job. For performing job scheduling, SEAS needs to estimate the expected up-time for each mobile device. In order to perform such estimation, SEAS uses the last two battery events of a mobile device to calculate the discharge rate.When there are no two events available SEAS assumes 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:

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

However, the discharge rate changes over the time [29], especially due to executing computations on the mobile devices [30]. In the original SEAS paper [2], it is stated that rt has a high variance, which might affect negatively the scheduling performance. To reduce such variation, the original SEAS uses the average of previous remaining time corrected to current time. Algorithm 1 depicts how the corrected remaining time is calculated. Line 10 of Algorithm 1 is where the historical remaining time is corrected to account the pass of time since it was originally calculated.

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

Algorithm 1 Battery time estimation

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

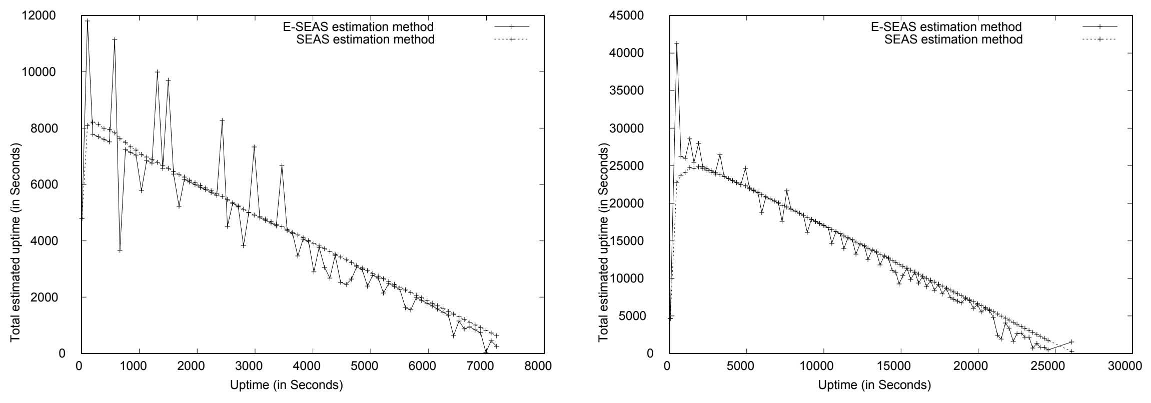
Each time a new job arrives to the proxy, SEAS goal is to balance the amount or resources assigned for the job. To do so, SEAS estimates the amount of computational power that each mobile device can deliver, which is calculated multiplying the device speed. This, in turn, is obtained executing a benchmark in the mobile device, multiplied 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 those mobile devices, as follows:

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

SEAS selects the mobile device that has 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.

The original SEAS [2] was evaluated using energy profiles obtained from a wide range of notebooks and netbooks. For such simulated evaluation, authors defined eight different simulations. Simulation scenarios were specified using a wide range of topologies, ranging 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% more jobs.

The first SEAS extension was presented in [12]. In this extension job stealing was added for increasing node balance. Job stealing means that when a node finishes all its assigned jobs, it will attempt to off-load other nodes by executing their jobs. This work uses SEAS original strategy for scheduling jobs when they first arrive to the Grid. However, when a node finishes all its jobs, instead of remaining idle waiting for new jobs, it tries to steal jobs from other nodes. For selecting the node from whom to steal jobs, three strategies were evaluated. Two of them were based on 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 stealOn the other hand, *WRAS* aims at decreasing the work load of nodes that have on average fewer resources per job. The third strategy is *Random Stealing (RS)*, i.e., selecting a random node that has jobs. *RS* is a well-known strategy in job scheduling [31, 32]. These strategies can be combined with two policies 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 a 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 simulations, *WRAS with Fixed Number* outperformed other combinations 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. Unlike the original SEAS study [2], smartphones and tablets profiles were used for the simulations in this study. Although job stealing adds energy consumption steaming from



(a) Notebook Intel Core 2 Duo. 4 Gb RAM DDR2 (b) Table Acer A100

Figure 2: Batteries Estimation Comparison

network transference, this factor was not simulated in [12].

To reduce the impact of network transference, new raking strategies for job scheduling where proposed in [6]. In particular, this work outlines 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. SEAS strategy for estimating the remaining uptime was appropriated for notebooks and netbooks, but we found that smartphones and tablets have a more stable battery consumption rate when we profiled them. This can be observed in Figure 2 that depicts SEAS and E-SEAS battery estimation for both a notebook and a tables. Figure 2a presents the battery estimation for one of the notebooks used for the original SEAS study, while Figure 2b outlines the prediction for an Acer A100 that was used for this study.

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 [7]. This works add network energy consumption to the simulation. Such source of energy consumption was not considered in previous SEAS based works [2, 12, 6].

Although SEAS was designed aiming at been simple to implement, SEAS and all its extensions have been evaluated using simulation. This is a standard practice in the area as some of other approaches, such as [8, 20], would be difficult to be implemented in real life scenarios because the required information might be unavailable. Another issue which encourages simulation is that some approaches require too much historic information and benchmarking, such as location history [4] or JEC estimation [6]. This has resulted in that few works, such as [14, 16], are actually evaluated through real-life implementation, while the vast majority [2, 8, 12, 20, 4, 6, 7] are validated via simulation. Despite being a common practice, simulation does not replace real-life experiments as models might be incomplete or parameters could be wrongly set [13]. Hence, it cannot be said that SEAS has been completely validated.

2.3 Mobile distributed computing platforms

DroidCluster [14] is a prototype for building clusters with Android mobile devices. This work proposes to build a local cluster integrated by Android mobile devices connected through a single WiFi access point. Authors argue that adding more access point would result in an unacceptable latency, while theoretically a single access point can handle up to 1042 mobile devices. The communication protocol for the proposed cluster is MPI, which allows to easily porting existing cluster applications. The main goal of this study was to evaluate how well a mobile device cluster capability scales when mobile devices are added. The empirical evaluation used the MPI version of the Linpack benchmark to measure the cluster megaflops. According to the results, the cluster scaled reasonably well with up to 6 nodes despite WiFi drawback. In particular, the 6-node cluster achieved a 75% of the performance predicted by an ideal, but unrealistic (according to the authors), linear scaling. Authors conclude that they expect to see mobile device based distributed computing environments because mobile devices might provide a better “computation per watt” ratio than stationary computers.

In [15], the authors discuss another MPI-based cluster environment integrated by Android mobile devices. Unlike DroidCluster, the focus is to use checkpointing techniques for handling mobile device mobility, and unexpected connection/disconnection. Although Android applications can be written in Java, authors implemented the application also in Android native C to ensure CPU native performance. Regarding checkpoint/restart, authors opted for Distributed MultiThreaded CheckPointing[3] that is a well-known open-source non-intrusive checkpointing technology [33]. Authors evaluate not only the speedup of adding up to 4 nodes, but also checkpointing overhead. According to their results, the checkpointing impact is different for different applications. In some applications, checkpointing has no perceivable overhead (less than 1%), while for others the overhead is up to 500% of the time required for running the application without checkpointing. Authors further study the impact of checkpointing and inter-process communication in [34]. This new study also provided evidence that relocating processes according to their affinity can be used to reduce runtime by up to 11%. This is because when two processes that need to communicate between them run within the same node they can use UNIX domain sockets for the communication, which is lightweight compared to TPC/IP.

Recently, Computing While Charging (CWC) was presented in [16]. 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 PCs 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. This work also addresses the issue of sending code for execute new kinds of tasks or updating existing task code definition in runtime. 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[4]. 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.

Briefly, there are many scheduling algorithms for mobile distributed computing that consider energy. However, they are not implemented in real-life scenarios. On the other hand, real-life implementations of mobile Grids or clusters disregards energy consumption. DroidCluster [14] is an example because it is focused on analyzing mobile devices capabilities, while CWC [16] and BOINC only use mobile devices when they are charging. Therefore, the main contribution of this work is to propose and to assess a distributed computing platform for mobile devices that supports SEAS in a real setting.

3 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 because Android is not only an open source mobile device operating system, but also one of most popular operating systems in the world. There are more than 1 billion active Android users in the Play Store[5] according to Google reports. Furthermore, Android provides access to all the information required by SEAS. The platform was designed following SEAS original architecture that identifies two main roles, namely the proxy, and the nodes. The proxy receives the jobs to execute, and assigns them to a node, i.e., a mobile device. The node is responsible of executing the jobs and returning their results to the proxy. Each node executes an Android application which is the only part of the platform that actually runs on the mobile devices. This application is responsible of executing the assigned jobs and sending the results to the proxy.

Figure 3 outlines the deployment of the proposed distributed system. The proxy implementation is divided into two components: a Web Server that acts as the communication interface between the mobile devices and the proxy logic, and an HTML5 application that allows users to submit applications and implements the scheduling logic. The Web server is implemented using Java Servlets and supports communication

[3]DMTCP: `http://dmtcp.sourceforge.net/index.html`

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

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

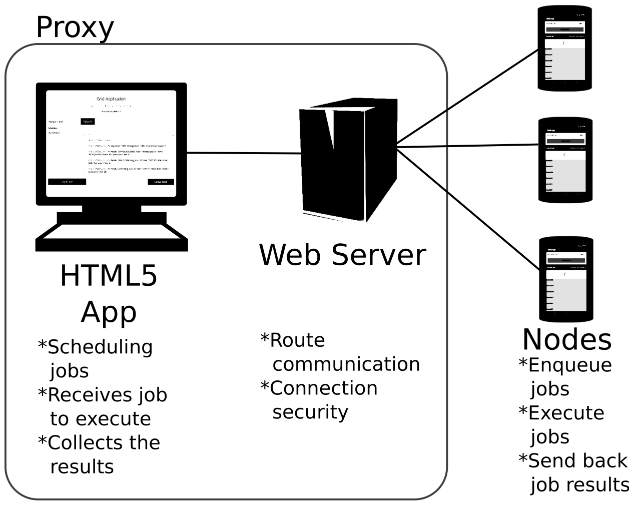


Figure 3: SEAS Implementation deployment

through WebSockets[6]. There are two main reasons for using WebSockets as communication protocol. 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 Web Server has two main responsibilities. First, it allows the Mobile Grid user to access the HTML5 application for managing the deployment. Second, it relays communication between the nodes and the HTML5 application.

As previously stated, the proxy logic is currently implemented through an HMTL5 application. This implementation allows to easily controlling 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 without the need of recompiling the proxy or redeploying it. 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. Currently, the implementation supports only one HTML5 application running for each Web Server, but it is possible to extend the current architecture to support several HTML5 applications per server. This would mean that several proxies might run using a single access point.

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. Figure 4 depicts the Android application UI. 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 [35]. Moreover, other benchmarks, such as Linpack or SciMark 2, can be used because benchmark can be run once and the value store for future reference. Notice that a mobile device performance is unlikely to change because hardware cannot be updated; only software updates might modify the device performance. Hence, benchmark results should seldom be updated, so they can be stored and access in future executions.

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 jobs that are already preloaded in the application because the current version of the platform does not implement a remote class loader, so adding new jobs requires recompiling, repackaging, and reinstalling the whole mobile application. For evaluation purposes we have defined only two kinds of jobs. The first kind of job is calculating the

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

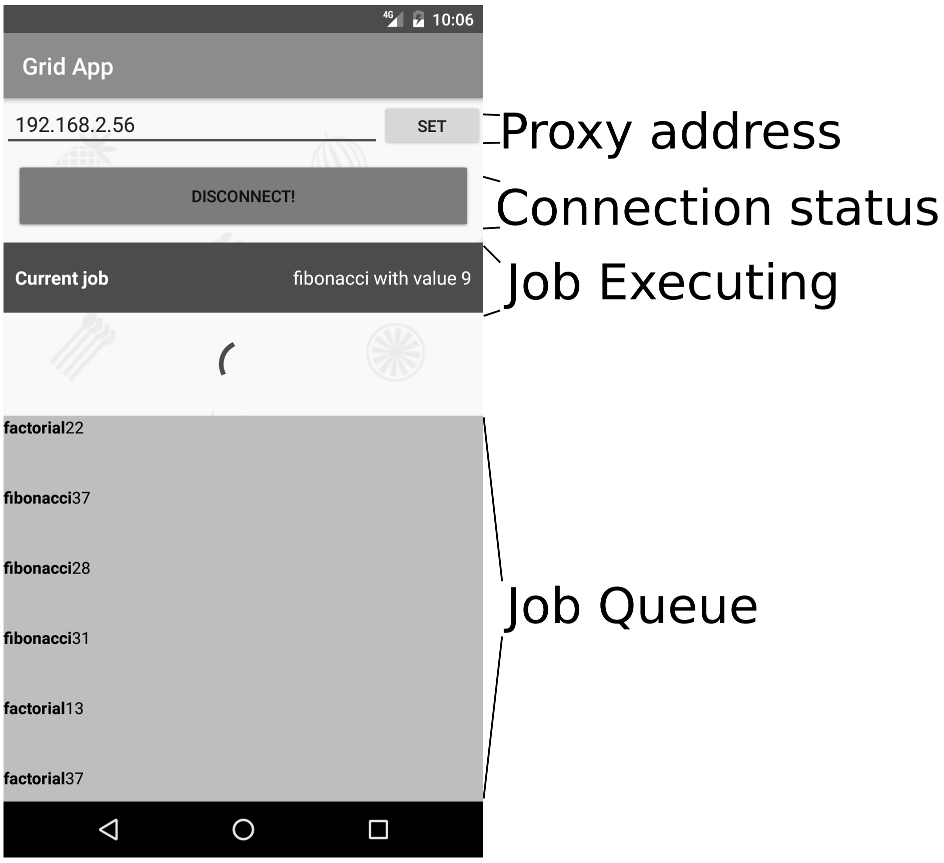


Figure 4: Android Application UI

Fibonacci value for a number and the other is calculating its factorial. The code to calculate the Fibonacci value is CPU and memory intensive 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 garbage collection. Notice that this cache is not shared 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

```java
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;
    }
  }
}
```

Table 2: Mobile devices

Device	Processor	Memory	BogoMips	Battery
Samsung I5500	Single Core 600 MHz.	256 Mb.	599.65	1200 mAh
Samsung I9300	Quad-core 1.4 GHz Cortex-A9	1 Gb.	2786.91	2100 mAh
Acer A100	Dual-core 1.0 GHz Cortex-A9	1 Gb.	1987.37	1530 mAh
LG Optimus L9	Dual-core 1.0 GHz Cortex-A9	1 Gb.	1592.38	2150 mAh

Listing 2: Factorial implementation

```
public class Factorial {
  public Long calculate(long number) {
    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 [16]. Furthermore, this work also provides a code-snippet showing how to implement such class loading mechanism. Android DexClassLoader is available in Android API from level 3, which means that all modern Android devices have this capability. Furthermore, Android has added support for loading classes directly from memory since API level 26, which means that this is only available in the newest versions of Android. Such support allows developers to load code without the necessity of storing it into the file system.

4 Empirical Evaluation

The goal of the empirical evaluation is to confirm the results presented in the original SEAS study [2]. In such study, SEAS was compared against traditional schedulers for distributed execution of atomic jobs, namely Round Robin and Random Scheduler, with regard to the amount of finished jobs given an energy budget. The original evaluation was performed using simulation and notebooks profiles, and SEAS outperformed both schedulers in some scenarios executing up to 20% more jobs. In this work, evaluation has been performed using smartphones and tablets instead of notebooks. Hence, results are not expected to be equals to the ones obtained in [2], but they are expected to confirm that SEAS outperforms Round Robin and Random schedulers. Unlike the original SEAS study where evaluation was performed only by simulation, experiments reported in this work have been run in a real environment.

As previously stated, we carried the empirical evaluation using Android devices. In particular, we performed the experiments using 3 smartphones and a tablet connected to a WiFi network. The proxy was also connected to the same WiFi network, i.e., all the communication was performed over a Local Area Network reducing issues such as latency, but the proposed implementation could also work over a more complex network topology, such as the Internet. Table 2 summarizes the specification of the devices used in the experiments. Such table contains CPU specification, RAM, BogoMips as reported in `/proc/cpuinfo`, and battery capacity. Notice that the reported BogoMips correspond to a single core because the current implementation does not run jobs in parallel. Furthermore, in some cases cores are not symmetrical, so only the largest reported BogoMips is used because jobs are CPU intensive. 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.

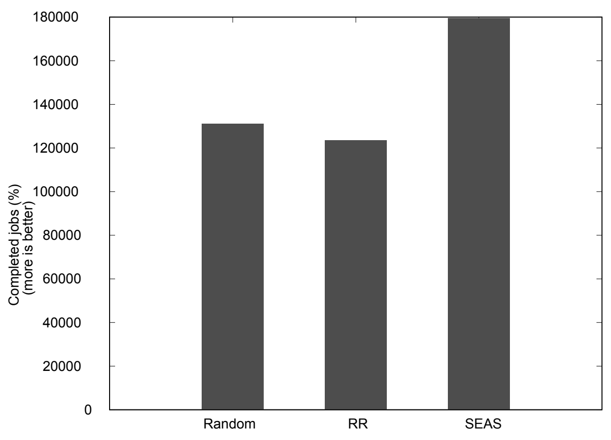


Figure 5: Finished jobs

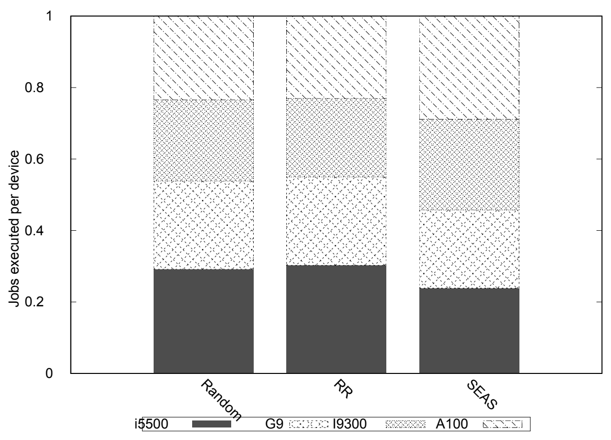


Figure 6: Relative executed jobs per device

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 5 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 were higher than the ones originally reported in [2]. However, there are several differences in this experiment that might account for these discrepancies. Firstly, in this work, smartphones and tablets were used instead of notebooks and netbooks. Secondly, the original SEAS simulator did not take into account energy consumption due to executing a task. As the mobile device operating system can dynamically scale CPU frequency according to the workload, mobile devices consume much energy when they are executing CPU intensive tasks. This was introduced in the simulation presented in [12], in which the simulator uses several profiles to estimate energy consumption. In that work, four different profiles were used for modeling each mobile device, each profile represented a CPU load, namely 0%, 30%, 75%, and 100% CPU usage. Thirdly, battery recovery effect has not been considered in the simulation. This effect is observed when available energy is less than the difference between battery charge and battery consumed. When a battery discharges 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 [9]. 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 6 depicts how the different schedulers distribute the jobs between different nodes. As expected, Round Robin and Random scheduler uniformly distributed the jobs, while SEAS has a clear preference for some device over others. As a result, SEAS improves energy efficiency with regard to finished jobs.

5 Conclusion and Future Works

The platform empirical evaluation presented in this work confirms that SEAS outperformed Random and Round Robin. Such results were expected as the original SEAS study [2] reported similar results through simulation. The main conclusion of this work is that, although simulation might not be 100% accurate, its results are reliable enough for drawing general conclusions about mobile Grid schedulers performance. Additionally, this work evidences that implementing an energy aware scheduler for mobile Grids is feasible.

This work also brings new research topics for future works. Firstly, future studies should focus on evaluating the proposed approach using other scenarios, such as distributed environments comprising more mobile devices and different types of jobs. In particular, more time-consuming jobs should be considered to make the experiments more comparable to experiments presented in other works [2, 12, 6]. Other future studied should aim at assessing simulation accuracy by testing the same scenarios through simulation and using SEAS implementation. Such evaluation will not only provide further evidence about the validity of simulation as experimental methodology for distributed environments that integrates mobile devices, but also shed light on which aspect of the simulation model must be enhanced to obtain more accurate results. Despite a SEAS implementation is available, simulation is a great tool for experiment because it does not require having hundreds of devices for experimenting and gives the means for performing more repeatable experiments [36].

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. Currently, the application does not support disconnections, which are common in mobile devices. Hence, we are currently implementing a mechanism for supporting mobile devices connected to poor quality networks. This will allow to evaluate the platform in real-life scenarios using mobile devices belonging to volunteers and not just dedicated devices. This is a key future work as the evaluation presented in this work was only performed using a reduced number of mobile devices. 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 [37]. 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. 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. [Online]. Available: https://www.doi.org/10.1504/IJWGS.2011.038386

[2] ——, “Mobile grid seas: Simple energy-aware scheduler,” in *3rd High-Performance Computing Symposium. 39th JAIIO*, 2010.

[3] 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: https://www.doi.org/10.1016/j.ins.2014.04.018

[4] 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: https://www.doi.org/10.1002/cpe.3297

[5] 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: https://www.doi.org/10.1145/2656214

[6] 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: https://www.doi.org/10.1016/j.pmcj.2015.08.003

[7] 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. [Online]. Available: https://www.doi.org/10.1007/s10723-016-9387-6

[8] C. Li and L. Li, “Tradeoffs between energy consumption and qos in mobile grid,” *The Journal of Supercomputing*, vol. 55, pp. 367–399, 2011. [Online]. Available: https://www.doi.org/10.1007/s11227-009-0330-5

[9] 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. [Online]. Available: https://www.doi.org/10.1109/JSAC.2010.100926

[10] P. Ghosh, N. Roy, and S. K. Das, “Mobility-aware efficient job scheduling in mobile grids,” in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid ’07)*, May 2007, pp. 701–706. [Online]. Available: https://www.doi.org/10.1109/CCGRID.2007.73

[11] S. S. Vaithiya and S. M. Saira Bhanu, *Mobility and Battery Power Prediction Based Job Scheduling in Mobile Grid Environment.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 312–322. [Online]. Available: https://doi.org/10.1007/978-3-642-24037-9_31

[12] 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: https://www.doi.org/10.1007/s00607-012-0245-5

[13] 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: https://www.doi.org/10.1145/2517448

[14] 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. [Online]. Available: https://www.doi.org/10.1109/ICDCSW.2012.59

[15] Y. Sawada, Y. Arai, K. Ootsu, T. Yokota, and T. Ohkawa, “An android cluster system capable of dynamic node reconfiguration,” in *2015 Seventh International Conference on Ubiquitous and Future Networks*, July 2015, pp. 689–694. [Online]. Available: https://www.doi.org/10.1109/ICUFN.2015.7182632

[16] 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. [Online]. Available: https://www.doi.org/10.1109/TMC.2014.2362753

[17] 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: https://www.doi.org/10.1016/j.future.2012.05.023

[18] 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. [Online]. Available: https://www.doi.org/10.1109/MNET.2017.1500293NM

[19] 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. [Online]. Available: https://www.doi.org/10.1109/ISPA.2012.115

[20] 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: https://www.doi.org/10.1016/j.jnca.2013.07.007

[21] 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. [Online]. Available: https://www.doi.org/10.1109/TMC.2012.239

[22] R. Magán-Carrión, J. Camacho, P. García-Teodoro, E. F. Flushing, and G. A. D. Caro, “A dynamical relay node placement solution for manets,” *Computer Communications*, vol. 114, no. Supplement C, pp. 36 – 50, 2017. [Online]. Available: https://www.doi.org/10.1016/j.comcom.2017.10.012

[23] L. Eastwood, S. Migaldi, Q. Xie, and V. Gupta, “Mobility using ieee 802.21 in a heterogeneous ieee 802.16/802.11-based, imt-advanced (4g) network,” *IEEE Wireless Communications*, vol. 15, no. 2, pp. 26–34, April 2008. [Online]. Available: https://www.doi.org/10.1109/MWC.2008.4492975

[24] R. Ratasuk, A. Prasad, Z. Li, A. Ghosh, and M. A. Uusitalo, “Recent advancements in m2m communications in 4g networks and evolution towards 5g,” in *2015 18th International Conference on Intelligence in Next Generation Networks*, Feb 2015, pp. 52–57. [Online]. Available: https://www.doi.org/10.1109/ICIN.2015.7073806

[25] 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: https://www.doi.org/10.1145/2348543.2348583

[26] 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. [Online]. Available: https://www.doi.org/10.1007/s11227-015-1425-9

[27] X. Wei, J. Fan, Z. Lu, and K. Ding, “Application scheduling in mobile cloud computing with load balancing,” *Journal of Applied Mathematics*, vol. 2013, 2013. [Online]. Available: https://www.doi.org/10.1155/2013/409539

[28] T. Shi, M. Yang, Y. Jiang, X. Li, and Q. Lei, “An adaptive probabilistic scheduler for offloading time-constrained tasks in local mobile clouds,” in *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on.* IEEE, 2014, pp. 243–248. [Online]. Available: https://www.doi.org/10.1109/ICUFN.2014.6876790

[29] 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: https://www.doi.org/10.1016/S0378-7753(01)00840-0

[30] 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

[31] R. V. Van Nieuwpoort, G. Wrzesiñska, 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: https://www.doi.org/10.1145/1709093.1709096

[32] 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: https://www.doi.org/10.1002/cpe.1392

[33] J. Cao, K. Arya, R. Garg, S. Matott, D. K. Panda, H. Subramoni, J. Vienne, and G. Cooperman, “System-level scalable checkpoint-restart for petascale computing,” in *22nd IEEE Int. Conf. on Parallel and Distributed Systems (ICPADS’16).* IEEE Press, 2016, pp. 932–941, also, technical report available as: arXiv preprint arXiv:1607.07995. [Online]. Available: https://www.doi.org/10.1109/ICPADS.2016.0125

[34] Y. Sawada, Y. Arai, K. Ootsu, T. Yokota, and T. Ohkawa, “Performance of android cluster system allowing dynamic node reconfiguration,” *Wireless Personal Communications*, vol. 93, no. 4, pp. 1067–1087, Apr 2017. [Online]. Available: https://doi.org/10.1007/s11277-017-3978-9

[35] 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: https://www.doi.org/10.1016/S0167-739X(01)00058-9

[36] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. [Online]. Available: https://www.doi.org/10.1002/spe.995

[37] J. M. Rodriguez, A. Zunino, A. Tommasel, and C. Mateos, *Encyclopedia of Information Science and Technology, Fourth Edition.* IGI Global, 2018, ch. Recurrent Neural Networks for Predicting Mobile Device State, pp. 6658–6670. [Online]. Available: https://www.doi.org/10.4018/978-1-5225-2255-3.ch577