Mobile Grid SEAS: Simple Energy-Aware Scheduler

Juan Manuel Rodriguez, Alejandro Zunino, and Marcelo Campo

ISISTAN - UNICEN. Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina
Tel.: +54 (2293) 43-9682 ext. 35. Fax.: +54 (2293) 43-9683
Also CONICET (Consejo Nacional de Investigaciones Científicas y Técnicas)

Abstract. In the last few years, several millions of high-capable mobile devices were sold around the globe. Therefore, integrating mobile devices to Grid systems may increase significantly their resources. However, mobile devices have more limited resources than personal computers or servers. In particular, minimizing energy consumption is very important because mobile devices rely on battery as energy supply. We propose a novel job scheduler that aims to use the energy in an efficient way. To implement it, we developed a simple but effective battery estimation model. In some experiments our energy aware scheduler outperformed traditional Grid schedulers to effectively address energy constrains in mobile Grids.

1 Introduction

In the last few years, mobile devices have evolved from being merely data-access devices to being capable of processing and storing significant amounts of data. Additionally, each new generation of mobile devices has better connectivity technologies, more storage memory and faster processors than previous generations [12]. In addition, mobile devices are more common now than ever [23]. For instance, 64.1 millions of smartphones were sold in 2006 while the number of sold smartphones in 2008 was 139.3 millions. However, as a result of their small displays and difficult input methods, mobile devices are more suitable for short and unilateral interactions, such as reading e-mail during a bus trip or sending a short text message (SMS), than for long and complex interactions [19]. In consequence, mobile devices remain unused most part of the time.

On the other hand, the concept of Grid computing has been gaining relevance since mid-90s when was introduced by Ian Foster [6]. Basically, Grid computing aims at combining the computer resources, which are usually unused by their owners, to obtain a performance similar to a supercomputer. Grid computing is an alternative for organizations that need intensive computing capabilities, but do not have the budget to buy and maintain a supercomputer.

Considering that mobile device capabilities are underused most part of the time and Grid systems always require more capabilities, we think that combining mobile device with Grid systems may increase its capabilities by taking advantage of underused mobile devices. In particular, integrating mobile devices into the Grid promises advantages not only for the Grid, but also for mobile device users because a mobile device can use the Grid to execute tasks it cannot perform otherwise [8]. Furthermore, mobile devices can provide several sensors, such as cameras, microphones, GPS among others, to analyze their environment [13]. Finally, people collecting data in the field, such as biologists or census takers, can employ mobile devices to upload collected data to the Grid in real-time [2].

Despite its advantages, integrating mobile devices into the Grid presents a wide variety of challenges that are not present when using desktop or server computers, which we will call fixed devices. Firstly, mobile devices wireless connections are not as reliable as fixed devices wired connections. Basically, a wireless connection can be interrupted for more reasons than a wired connection. For instance, an interference might result from a wall between the connected devices or because a mobile device moves away of the wireless connection range [24]. Secondly, compared with fixes devices, mobile devices computational resources are very limited [1]. Finally, mobile devices power source rely on battery technology. As a result, a mobile device may fail because its battery is empty [17,12]; in contrast, fixed devices, which are connected to the power grid, rarely fail because of its energy source.

The combination of those problems is hindering the adoption of mobile devices as a part of the Grid. In fact, the integration of mobile devices as Grid resources introduces more drawbacks than advantages. Although there are several works [1,2,7,8,10,12] that deal with some of these issues, there are still many open problems.

Nowadays, researchers divided Grid system into several subtypes in accord with the main purpose of the Grid [1]. The first type, called *Computational Grids*, consists of Grids designed for high performance computing. Here, each Grid node shares its computational capabilities to solve a complex problem. Usually, a *Computational Grid* is fed with jobs that are task indivisible by the Grid. Therefore, they must be assigned to a particular computer in the Grid. For instance, PrimeGrid[1] is indented to find large prime numbers by distributing the computation across volunteer computers. On the other hand, there are Grids whose purpose is to share large amounts of data; these Grids are known as *Data Grids*. Finally, *Utility Grids* are design not to handle large amounts of data or computation, but for coordinating the information across a highly distributed system, such as a disaster management application.

Although mobile devices have not the same computational power as other devices, integrating mobile devices into *Computational Grids* might be useful because the ever increasing capabilities and number of mobile devices, such as smartphones, PDAs, and notebooks, in the market [4,9,23]. Currently, there are in the market smartphones with 1GHz processors and several hundred of RAM, such as the Nexus One[2] or the well-known iPhone 4. As a result of their number and current capabilities, mobile devices might be a significant source of computational power for a *Computational Grids*.

This paper focuses on how jobs are scheduled in job-based *Computational Grids* when mobile devices are present. We propose a novel technique for scheduling jobs when nodes that depend on battery are a part of the Grid. This scheduling technique aims to minimize the energy consumption of the mobile devices while maximize the throughput of the Grid. To do this, this technique considers not only the performance

[1] PrimeGrid: `http://www.primegrid.com/`

[2] Nexus One technical specifications: `http://www.google.com/phone/static/en_US-nexusone_tech_specs.html`

of the mobile node in terms of computational speed, but also what the energy cost of using a particular node is.

Having introduced the focus of this paper, the rest of it is organized as follows: Section 2 presents a brief description of Grid concepts and the problems related with using mobile devices within a Grid system. Section 3 summarize other proposed scheduling techniques that consider mobile devices. Then, in Section 4, we introduce our algorithm to schedule jobs in Grids that integrates mobile devices. Experiments on efficient battery estimation in real time and simulations to evaluate our algorithms against traditional Grid scheduling are presented in Section 5. Finally, Section 6 presents the conclusions of this work as well as future works in the area.

2 Background

Grid computing is a concept that appeared in the mid-90s to describe a new way of implementing distributed applications. Ian Foster defined a computational Grid as: "a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service" [6]. The first characteristic of a Grid is that its resources are in different places and interconnected with a network, usually the Internet. Furthermore, coordinating those resources implies that there is not a hierarchical relationship. Therefore, central control should be avoided when possible because each central control policy must be agreed by all Grid participants. Since there should not be a central control for Grid resources, each of them might be running different operating systems. As a consequence, open-protocols are required to ensure the interoperability of those systems. Finally, a Grid system must provide non trivial qualities of service to fulfill the Grid user expectation. For instance, a Grid user should not be worried about Grid resource faults because the Grid should provide fault-recovery systems.

Foster [6] also proposed a four-layer architecture to develop Grid systems. Each layer represents a different abstraction of a Grid. Lower layers treat a Grid system as a set of components interconnected by a network. While, higher layers shows a Grid system as a single entity. The layers are:

– Fabric: this layer presents all the resources that a Grid system can use. The function of this layer is to present in a uniform way resources of the same type. For instance, a storage system should have the same operations independently if it is a data-base, a file-system in a machine or a file-system distributed across a local network.
– Resource and connectivity protocols: this layer has the different protocols that Grid elements in the fabric use to interact with each other.
– Collective services: collective services are operations that show a Grid system as a single entity. In this layer, there are services like “run in the Grid” or “storage in the Grid”. These services are a transparent way to interact with a Grid system because a user does not need to know which particular elements of the Grid is actually performing the operation.
– User applications: This is the layer where the Grid user deploys their applications to be executed. These applications can access to the Grid resources through the services described above.

The other aspect of this work is related with mobile devices. These devices have the capacity of being with their users at any time because they are designed to be carried anywhere. As a result of being designed for mobility, mobile devices must be small and carry their own energy source. Those constraints result in devices with limited resources, in comparison with standard desktop computers, because their small size and limited operation caused by their energy source [16].

Battery is a major concern when using mobile devices because their energy density has not been increased as the same rate as mobile device capabilities [17]. It has been pointed out that despite the rapid advances in battery technology, radical changes in battery technology are needed to meet the current demands of energy [20]. Additionally, battery consumption is very difficult to be estimated [3] because the battery charge does not have a linear behavior [3].

3 Related work

Several authors [1,7,8,10,14] have proposed different methods to integrate mobile devices into traditional Grid systems. Those proposals go from simply visualizing a Grid system and modifying some parameters from a mobile device [8], to a real integration where mobile devices share their resource as other Grid members do [14].

In [18], a scheduling algorithm for mobile Grid is described. Although this algorithm does not directly take into account energy consumption, it is designed considering that mobile devices tend to frequently disconnect from the Grid. This scheduler considers that both connection and disconnection are Poisson process. Basically, the scheduler tries to obtain the node which has more probability of being connected when it finishes executing the job. This algorithm needs to know the execution time of a job in a node to calculate the aforementioned probabilities. Therefore, the scheduler determines which node is the best candidate to execute a job.

In [22], the authors analyze and compare several scheduling algorithms for applications based on communicating sub-tasks. Basically, an application is composed by several tasks, and some of these tasks use as input the result of other tasks. Therefore, an application can be modeled as an acyclic graph of tasks, where dependencies between tasks are the graph edges. All the schedulers aim to minimize the percentage of total energy consumed by executing a task. This work considers that several parameters of the tasks and the devices, such as how much energy a device consumes per execution time unit, or how many execution time units are needed to complete a task, are known. In addition to this limitation, the scheduler makes a static assignation. This means that this scheduler does not consider the possibility of a device failure.

A single-class job scheduler is proposed in [7]. Single-class job means that all the jobs in the Grid have the same CPU, memory and network requirements, limiting the Grid to solve only one problem. This work assumes that the assignation of a job to a mobile device is made by an entity called Job Allocator. Basically, the Job Allocator and a node play an incomplete information game. This game is a negotiation where the Job Allocator offers a pay to get the job done and the mobile device wants that pay. Both the mobile device and the Job Allocator are trying to reach a bargain because each one needs something that the other has. As time passes, the different players will learn how other players negotiate. For instance, a Job Allocator might learn that a particular node accepts certain type of offer in the 70% of the cases. This knowledge allows the players to behave in a more intelligent way when they negotiate. In the end, the behavior of all Grid nodes tends to generate a self-regulated market because if a mobile device wants to charge too much for executing a job, no Job Allocator will work with that mobile device. On the other hand, if a Job Allocator offer is low, no mobile device will accept that offer.

In [14], an energy consumption aware scheduling algorithm is described. This algorithm considers that mobile devices charge for executing jobs. This algorithm aims to not only maximize the efficiency of energy use, but also minimize the cost of assigning jobs. Several functions are defined to express the cost of assigning a job to a node and the energy consumption of a job in a node. Having these functions defined, the problem is reduced to a maximization and minimization problem. The authors proposed to solve the problem by applying the Lagrange multipliers method. Therefore, this method assures that the resulting resource assignation will be optimum. However, this scheduler requires knowing exactly the value of several variables, such as energy capacity of the resources, computational capacity of the resources, the time required to finish a job in a particular resource, among others, before assigning the resources. These requirements make this scheduler difficult to be implemented.

4 SEAS: a simple energy-aware scheduling algorithm

The aforementioned algorithms have the disadvantage that they need to know too much information that is usually unavailable or difficult to guess, such as battery estimation or job execution time [1,3]. To deal with this complexity we propose a simple algorithm that does not try optimize resource allocation, instead it tries to estimate a good resource allocation based on the current state of the mobile devices. We call this scheduler Simple Energy-Aware Scheduler (SEAS) because it was designed thinking in simplicity and fast resource allocation.

The first problem to design an energy-aware scheduler is to have a realistic and practical battery estimation. However the most realistic battery models consist of many complex differential equations that requires knowing several physical parameters, which are only known by battery makers. In addition, these equations might require too much computational time to be solved [3]. As a consequence, we decided to formulate a simpler model that can make not so accurate estimates, but they can be done in real-time without knowing all technical parameters. The simplest algorithm consists in measuring the current battery charge (*bc*) and the current time (*ct*), and wait for a change in the battery charge. When a change happens, the algorithm measures the new battery charge (*nbc*) and the current time (*nct*) and uses this information to calculate the current discharge rate (dr) as follows:

$$dr = \frac{bc - nbc}{nct - ct}$$

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

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

Unfortunately, the discharge rate changes over the time [21]. Therefore, this process must be repeated every time the battery charge change to keep the estimation updated. We found out that estimations made with this algorithm have significant variations in subsequents updates. Thus, we modified this algorithm to return an average remaining time instead of returning the previously defined remaining time. This average is calculated using the estimated uptime, which is defined as the current uptime plus the estimated remaining time as defined above. Therefore, the new estimated remaining time is the average estimated uptime minus the current uptime. Algorithm 1 describes how the remaining time is calculated.

Algorithm 1 Battery time estimation

```
 1: procedure BATTERYESTIMATIONTHREAD($battery, clock$)
 2:     $startTime \leftarrow clock.getTime$
 3:     $oldTime \leftarrow clock.getTime$
 4:     $oldCharge \leftarrow battery.getCharge$
 5:     $previusEstimations \leftarrow$ new Vector                          ▷ Empty Array
 6:     while true do                                                     ▷ Never ends
 7:         WAITFORBATTERYCHARGEUPDATE
 8:         $newTime \leftarrow clock.getTime$
 9:         $newCharge \leftarrow battery.getCharge$
10:         $dischargeRate \leftarrow (newTime - oldTime)/(oldCharge - newCharge)$
11:         $estimatedUptimeTime \leftarrow newTime - startTime + newCharge * dischargeRate$
12:         ADD($estimatedUptimeTime, previusEstimations$)
13:         $newEstimatedUptimeTime \leftarrow$ AVERAGE(previusEstimations) $- (newTime - startTime)$
14:         UPDATEESTIMATEDUPTIMETIME($newEstimatedUptimeTime$)
15:         $oldTime \leftarrow newTime$
16:         $oldCharge \leftarrow newTime$
17:     end while
18: end procedure
```

With regard to our scheduling algorithm, it was conceived to map an indivisible computational task, called job, to a particular mobile device. Although not necessarily true, we assume that those jobs have similar requirements, being similar to a single-class job Grid [7]. In addition, we assume that all mobile devices have the capability of measuring their remaining battery capacity.

To design the SEAS, we considered that mobile devices are connected to the Grid through proxies. As Figure 1 shows, a proxy is a server whose purpose is to hide mobile devices from the rest of the Grid [1,7,14]. Therefore, mobile devices and their proxy are seen by the Grid as a single resource. As a result, the SEAS is independent form the rest of the Grid scheduling policies.

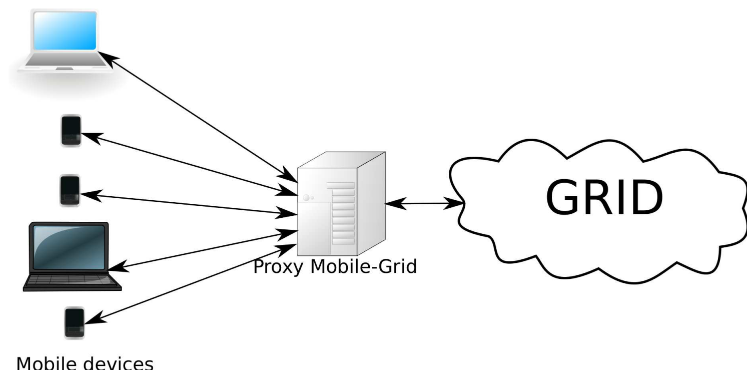


Fig. 1: Mobile devices connected using a proxy

The algorithm consists in two parts: measuring mobile devices capabilities and assigning a resource to a job. Every mobile device is responsible for measuring not only its performance, but also its estimated remaining up-time based on the remaining battery. Firstly, measuring the performance of a system is not a simple task and there are many ways to do it [15]. We used a simple benchmarking approach that aims to estimate how many floating-point operations per second (flops) can be performed by a system. In addition, we considered that modern microprocessors have a better performance when running multi-thread application [5]. Therefore, we run several times our benchmark, increasing the number of threads until the performance does not increase any further to determine the optimal number of threads.

For an efficient resource assignation, it is necessary that each mobile device informs its estimated battery time to the proxy. The proxy has a registry of all mobile devices connected with battery, benchmark and assigned job information. When a new job arrives to the proxy, the proxy uses the stored mobile device data to calculate how many resources per job are, and based on that value the proxy creates a ranked list of the mobile devices to assign the job to the best ranked one. The assigned resource per job in the i^{th} mobile device is calculated as follows:

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

It is worth noting that a node with a high benchmark value and a low estimated time value might have fewer resources than other mobile device with a lower benchmark value, but a higher estimated time. This is because, although the first mobile device is expected to finish jobs faster than the second one, it has the capability of executing more jobs. Basically, the mobile device that has more resources per job is considered to be the best option to solve the new job because it has more available resources. When a job is assigned to a node, it is responsible for scheduling the job. The current SEAS version uses a First Come First Served (FCFS) scheduling policy.

5 Experiments and Simulations

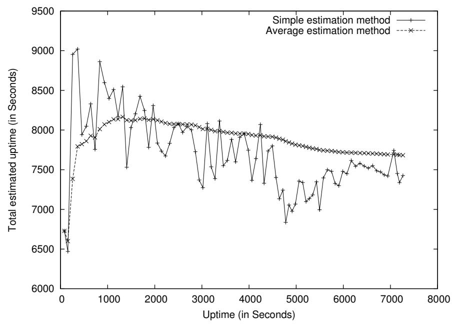
We evaluated this work from two different points of view: the effectiveness of the battery estimation algorithm, and the effectiveness of the scheduler. To evaluate the battery estimation algorithm, we profiled the battery usage from different notebooks while they were being used by real users. The data of the profile was obtained using the Advanced Configuration & Power Interface (ACPI)[3]. Table 1 describes the characteristic of the profiled notebooks and their batteries.

Notebook	Technical description	Battery
Notebook-1	Intel Core 2 Duo T5300 (1.73GHz) 2 GB Ram DDR2 Hard Drive: 160Gb SATA DVD +/- RW	6 Cells 4400 mAh
Notebook-2	Intel Core 2 Duo T7100 (1.80GHz) 4 Gb Ram DDR2 Hard Drive: 250GB SATA DVD +/- RW	6 Cells 4800 mAh
Notebook-3	Intel Core 2 Duo T2400 (1.83 GHz) 4 Gb Ram DDR2 Hard Drive: 100 GB SATA DVD +/- RW	6 Cells 4800 mAh
Notebook-4	Intel Core 2 Duo P7350 (2.00GHz) 4 Gb Ram DDR3 Hard Drive: 360 GB SATA DVD +/- RW	8 Cells 4000 mAh
Netbook-1	Intel ATOM processor N270 (1.60 GHz) 1GB Ram DDR2 Hard Drive: 160 GB	6 Cells 5900 mAh

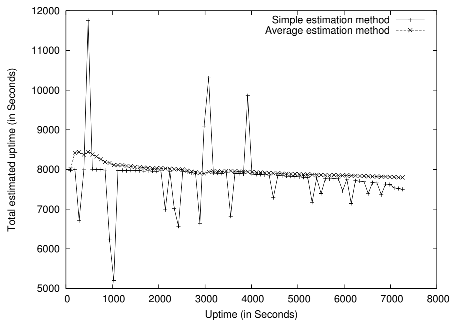
Table 1: Profiled notebooks

Based on these profiles, the two values of estimated time were calculated: the simple one, which is based on current energy consumption; and the one based on an average of estimated uptime. Figure 2 depicts how the estimated uptime changed when the notebooks were used. There are two lines in each figure, one for each of the proposed methods. Since it is desirable that estimated uptime does not change during one execution, the flatter the line is the better the estimation. In all the cases, the measures taken using the simple estimation method had an standard deviation greater than 5% of the average value, while the measures taken using the average estimation method had a standard deviation minor than 3%. Comparing these methods with other proposed methods was not possible because the other methods require knowing technical

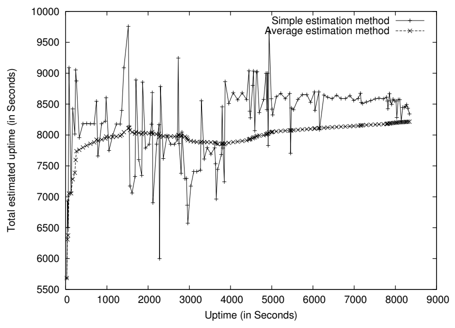
[3] ACPI: `http://www.acpi.info/`



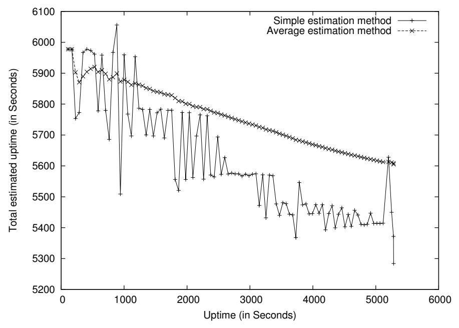
(a) Notebook-1



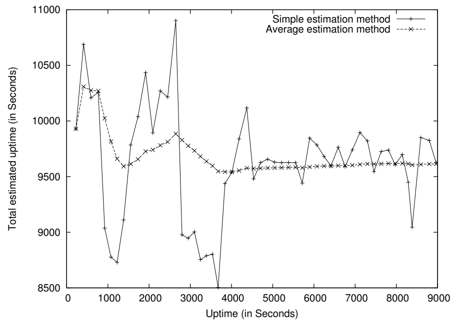
(b) Notebook-2



(c) Notebook-3



(d) Notebook-4



(e) Netbook-1

Fig. 2: Estimated uptime

information about the batteries that cannot be obtained by using ACPI. In addition, the other methods also need to exactly know how many energy will be consumed during a job execution. However, the energy consumption may vary as a result of the hardware use, such as disk, DVD or CPU, turning difficult to make a good prediction. Besides, since mobile devices run not only Grid applications, but also other user applications, hardware utilization varies independently of the Grid job that is running on the mobile device.

For evaluating the SEAS paradigm we implemented it as a load balancing service provider for GridGain[4], which is a well-known extensible Grid middleware. Additionally, GridGain comes with different load balancers that were compared with our scheduler. In order to evaluate the SEAS algorithm, several simulations based on the profiles taken from the notebooks where executed. First, a benchmark was executed on all the machines to determine the speed of each machine. The fastest notebook was Notebook-4, however, Notebook-2 was able to execute more operations because it can operate for extended periods of time when working on battery. It is worth noting that we consider that nodes disconnect from the Grid only because of battery depletion.

Second, several possible battery profiles were generated to simulate each machine. In the simulation, each node has the capability of running up to 4 jobs at the same time without performance penalties. In regard to jobs, the execution time represent the time that takes to execute the job on an Grid average node. Besides, the number and execution time of the jobs in each simulation were adjusted to require all Grid capacity. This means that ideally the 100% of the jobs can be executed before the last node runs out of battery. However, since each job computation cannot be divided, it is possible that not all the jobs can be completed. Besides, even though we assumed similar jobs, we simulated several cases where this assumption is not true. In particular, in some simulation the largest job execution time could be three times the shortest job execution time.

The different simulations are described in Table 2. The first 5 simulations consider that the Grid execute jobs that require long time, taking into account the battery duration, to be executed. Sim-1 stands for a small Grid composed by 7 nodes. The second simulation (Sim-2) is a Grid that consists of 400 nodes of different capabilities. Sim-3, Sim-4 and Sim-5 are different cases of Grids integrated by very dissimilar devices. Notebook-2 is a fast device, but with limited battery. On the other hand, Netbook-1 is a slow device that can execute more operations than the Notebook-2 because it has a better battery. Finally, the three last simulations are Grids similar to the three previous Grids, but executing jobs that require less processor time (8 minutes in average).

Figure 3 shows the result of the simulation. From these results it can be seen that SEAS performed better than the other two algorithms in all the cases. Moreover, all algorithms performed better with the smaller jobs, probably because the schedulers can distribute the calculations in a more fair way. In addition, the random scheduler tends to behave similarly to the Round-Robin scheduler when the number of jobs is higher and its execution time is shorter. This result probably stems form the fact that the random scheduler assigns a job to a node with a uniform probability. Therefore, when the number of jobs is considerable bigger than the number of nodes, the number of jobs

[4] GridGain: `http://GridGain.com/`

Simulation	Nodes in the Grid	Jobs to be executed
Sim-1	3 Notebook-1 1 Notebook-2 1 Notebook-3 1 Notebook-4 1 Netbook-1	110 jobs of 24 minutes +/-24%
Sim-2	100 Notebook-2 100 Notebook-3 100 Notebook-4 100 Netbook-1	5999 jobs of 24 minutes +/-24%
Sim-3	100 Notebook-2 100 Netbook-1	3560 jobs of 24 minutes +/-24%
Sim-4	150 Notebook-2 50 Netbook-1	3856 jobs of 24 minutes +/-24%
Sim-5	50 Notebook-2 150 Netbook-1	3490 jobs of 24 minutes +/-24%
Sim-6	100 Notebook-2 100 Netbook-1	11033 jobs of 8 minutes +/-50%
Sim-7	150 Notebook-2 50 Netbook-1	11569 jobs of 8 minutes +/-50%
Sim-8	50 Notebook-2 150 Netbook-1	10497 jobs of 8 minutes +/-50%

Table 2: Configuration of the simulations

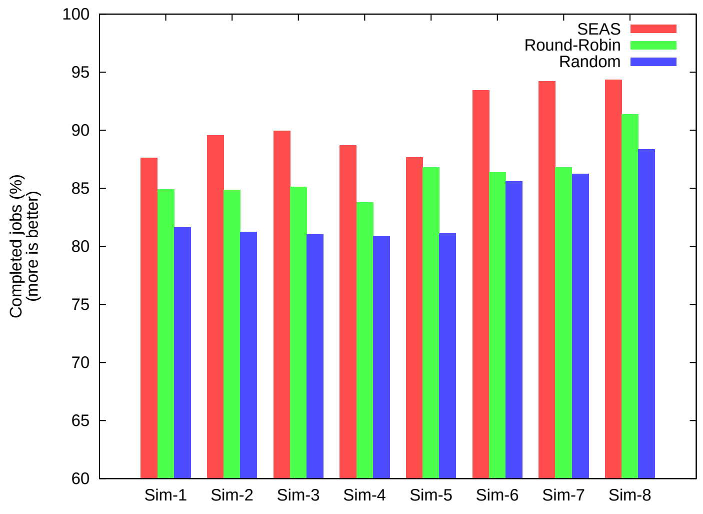


Fig. 3: Simulation results

assigned to a node converge to the number of jobs divided by the number of nodes, which is the Round-Robin scheduler expected behavior. Another important factor is scheduler speed. Our scheduler was executed on an AMD Phemon X3 and needed on average 234.06 ms to schedule 11569 jobs in 200 nodes, while the Round-Robin scheduler needed 53.9 ms and the random scheduler needed 22.99 ms.

6 Conclusions

As several authors have stated [1,7,10,8,14], integrating mobile devices into Grid systems will significantly increase Grid systems capacities. Grid schedulers that take into consideration energy are required to properly integrate mobile devices. This work presents a novel energy-aware Grid scheduler that has proved to perform better than traditional Grid schedulers in simulations.

Furthermore, the simulations also put in evidence that all the evaluated scheduling algorithms are more effective when jobs are small. This is probably because the scheduler can perform a finer assignation when jobs are shorter. For instance, if there is a job that requires 400 minutes to be executed in a particular device and there are two of those devices with enough battery to execute for 300 minutes, the job cannot be finished because it cannot be split. However, if there are two jobs that need 200 minutes of execution, each job can be assigned to a different device for execution.

From this work it is evident that there are a range of complex issues that if properly solved would enable the development of a more effective scheduler for Grid systems that use mobile devices. One of the problems is to estimate the capacity of a mobile device. This problem is related not only with estimating the remaining battery time, which already is a complex problem [3,20,21], but also estimating the amount of work that a mobile device can do in that time. Another problem is to estimate how complex a job is, because better estimations allow the scheduler to calculate how many resources a particular work will consume, allowing a better resource assignation [11].

Future work related with Grid scheduling will focus on creating better battery estimation models, different benchmarking techniques and methods for estimating job complexity. For the battery estimations, we will develop algorithms for estimating how mobile devices will be used based on their users' context [24,20,23]. Since energy consumption is directly affected by the mobile device usage, we think user profiles may help estimating battery life. With regard to benchmarking, we are studying several benchmarking techniques that have been proved in other domains [15], such as traditional servers. These benchmarks are essential to know how a mobile device will behave when a new job arrives. Finally, knowing a job complexity will help the scheduler to calculate how many resources should be assigned in a mobile device, instead of our current assumption that all jobs consume about the same amount of resources. Finally, we are adapting the approach to execute experiments using data obtained from Android-based smartphones.

Acknowledgments

We acknowledge the financial support provided by ANPCyT through grants PAE-PICT 2007-02311 and PAE-PICT 2007-02312.

References

1. Ashish Agarwal, Douglas O. Norman, and Amar Gupta. Wireless Grids: Approaches, architectures, and technical challenges. Working papers 4459-04, Massachusetts Institute of Technology (MIT), Sloan School of Management, 2004.
2. Yaw Anokwa, Carl Hartung, Waylon Brunette, Gaetano Borriello, and Adam Lerer. Open source data collection in the developing world. *Computer*, 42(10):97–99, 2009.
3. Vijayasekaran Boovaragavan, S. Harinipriya, and Venkat R. Subramanian. Towards real-time (milliseconds) parameter estimation of lithium-ion batteries using reformulated physics-based models. *Journal of Power Sources*, 183(1):361 – 365, 2008.
4. Ming-Chiao Chen, Jiann-Liang Chen, and Teng-Wen Chang. Android/osgi-based vehicular network management system. *Computer Communications*, In Press, Corrected Proof:–, 2010.
5. Theofanis Constantinou, Yiannakis Sazeides, Pierre Michaud, Damien Fetis, and Andre Seznec. Performance implications of single thread migration on a chip multi-core. *SIGARCH Comput. Archit. News*, 33(4):80–91, 2005.
6. Ian Foster and Carl Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure.: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2. a. edition, December 2003.
7. Preetam Ghosh and Sajal K. Das. Mobility-aware cost-efficient job scheduling for single-class grid jobs in a generic mobile grid architecture. *Future Generation Computer Systems*, In Press, Corrected Proof:–, 2009.
8. Francisco J. González-Casta no, Javier Vales-Alonso, Miron Livny, Enrique Costa-Montenegro, and Luis Anido-Rifón. Condor grid computing from mobile handheld devices. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(1):117–126, 2003.
9. Adrian Holzer and Jan Ondrus. Mobile application market: A developer's perspective. *Telematics and Informatics*, In Press, Corrected Proof:–, 2010.
10. Karin Anna Hummel and Harald Meyer. Self-organizing fair job scheduling among mobile devices. *Self-Adaptive and Self-Organizing Systems Workshops, IEEE International Conference on*, 0:230–235, 2008.
11. JoonMin Gil KwangSik Chung Taeweon Suh JongHyuk Lee, SungJin Song and HeonChang Yu. *Advances in Grid and Pervasive Computing*, chapter Balanced Scheduling Algorithm Considering Availability in Mobile Grid, pages 211–222. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009.
12. Konstantinos Katsaros and George C. Polyzos. Towards the realization of a mobile grid. In *CoNEXT '07: Proceedings of the 2007 ACM CoNEXT conference*, pages 1–2, New York, NY, USA, 2007. ACM.
13. Wolfgang Kiess and Martin Mauve. A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Networks*, 5(3):324 – 339, 2007.
14. Chunlin Li and Layuan Li. Energy constrained resource allocation optimization for mobile grids. *Journal of Parallel and Distributed Computing*, 70(3):245–258, 2010.
15. Yue Luo, J. Rubio, L.K. John, P. Seshadri, and A. Mericas. Benchmarking internet servers on superscalar machines. *Computer*, 36(2):34 – 40, feb 2003.

16. Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. Ibis for mobility: solving challenges of mobile computing using grid techniques. In *HotMobile '09: Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, New York, NY, USA, 2009. ACM.
17. Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–27, 2005.
18. Y.-B. Ko S.-M. Parkm and J.-H. Kim. *Service-Oriented Computing - ICSOC 2003*, chapter Disconnected Operation Service in Mobile Grid Computing, pages 499 – 513. Lecture Notes In Computer Science. SpringerLinks, 2003.
19. Suprateek Sarker and John D. Wells. Understanding mobile handheld device use and adoption. *Commun. ACM*, 46(12):35–40, 2003.
20. Bruno Scrosati and Jürgen Garche. Lithium batteries: Status, prospects and future. *Journal of Power Sources*, 195(9):2419 – 2430, 2010.
21. 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*, 103(2):180 – 187, 2002.
22. Sameer Shivle, H.J. Siegel, Anthony A. Maciejewski, Prasanna Sugavanam, Tarun Banka, Ralph Castain, Kiran Chindam, Steve Dussinger, Prakash Pichumani, Praveen Satyasekaran, William Saylor, David Sendek, J. Sousa, Jayashree Sridharan, and José Velazco. Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment. *Journal of Parallel and Distributed Computing*, 66(4):600 – 611, 2006. Algorithms for Wireless and Ad-Hoc Networks.
23. Joel West and Michael Mace. Browsing as the killer app: Explaining the rapid success of apple's iphone. *Telecommunications Policy*, 34(5-6):270 – 286, 2010.
24. Lee W. McKnight William Lehr. Wireless Internet access: 3G vs. WiFi? *Telecommunications Policy*, 27:351–370, 2003.