**Noname manuscript No.**
(X)

# Energy-efficient Job Stealing for CPU-intensive processing in Mobile Devices

**Juan Manuel Rodriguez · Cristian Mateos · Alejandro Zunino**

the date of receipt and acceptance should be inserted later

**Abstract** Mobile devices have evolved from simple electronic agendas and mobile phones to small computers with great computational capabilities. In addition, there are more than 2 billion mobile devices around the world. Taking these facts into account, mobile devices are a potential source of computational resources for clusters and computational Grids. In this work, we present an analysis of different schedulers based on job stealing for mobile computational Grids. These job stealing techniques have been designed to consider energy consumption and battery status. As a result of this work, we present empirical evidence showing that energy-aware job stealing is more efficient than traditional random stealing in this context. In particular, our results show that mobile Grids using energy-aware job stealing might finish up to 11% more jobs than when using random stealing, and up to 24% more jobs than when not using any job stealing technique. This means that using energy-aware job stealing increases the energy efficiency of mobile computational Grids because it increases the number of jobs that can be executed using the same amount of energy.

**Keywords** Mobile Grid · Mobile Devices · Job Stealing · CPU intensive application · Job Scheduling

## 1 Introduction

Unlike years ago, today's mobile devices have the capability of executing complex software that requires large amount of CPU/memory and might also need other special capabilities, such as 3D graphics rendering. In addition, current storage technologies make it possible to store several gigabytes in a small card, which means that mobile devices have the ability of storing large amount of data. In addition to mobile devices internal capabilities, they have the capability of using wide-range wireless networking technologies [25] meaning that mobile devices might be connected to Internet or to other devices most of the time. All

Juan Manuel Rodriguez · Cristian Mateos · Alejandro Zunino
ISISTAN Research Institute. UNICEN University.
Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina.
Tel.: +54 (249) 4439682 ext. 35. Fax.: +54 (249) 4439683
also Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Juan Manuel Rodriguez E-mail: juanmanuel.rodriguez@isistan.unicen.edu.ar

in all, mobile devices have evolved from simple agendas and PDAs to small computers with similar capabilities to few year-old computers [45]. Furthermore, there are more than 2 billion mobile device owners, and people in established markets usually own 2 or more mobile devices [42].

Taking into account these facts, several researchers [17,33,26,14,27,42,44,45,12,19,1,11] have studied how to scavenge mobile devices resources to solve complex computational problems, such as scientific problems [43]. Although mobile devices capabilities are limited when compared to server or desktop machines, the large amount of mobile devices might compensate their limitations [45]. In fact, these works [27,42,45] present mobile devices as part of distributed computer environments, such as traditional computational Grids or clusters. By "computational" we mean Grids whose main purpose is to offer CPU processing time to user applications, while "traditional" means distributed environments relying on fixed (and not mobile) computing resources, such as PCs or servers. As this paper concerns CPU-intensive applications, computational Grids will be referred simply to as "Grids".

Although there are plenty of works, such as [38,51,47,35,37,49], studying traditional Grid and clusters, mobile devices introduce new research issues to current distributed environments [45]. Some of the issues are intrinsic to mobile devices, while others emerge from combining mobile devices with traditional distributed computing. For instance, a typical issue in mobile devices is their different software platforms [2] with different APIs and the fragmentation within each unique platform [15]. This issue makes it difficult to run the same software in different mobile devices even when they are running the same operating system. Moreover, a problem of merging mobile devices with fixed distributed environments is security because different mobile devices are owned by different people to keep their personal data, such as pictures and contacts. Therefore, each mobile device owner wants to have their data as well as their communications with their services secure [46]. This usually does not happen in some traditional distributed environments, such as clusters or Clouds, where all of their components are owned by a single organization, so intra-net communication is more controlled.

A recurrent problem when dealing with mobile devices is energy consumption. This is because mobile devices energy source are batteries, and when a mobile device consumes all its battery, the mobile device cannot remain functional. An interesting fact is that in general mobile device capabilities have been exponentially growing, but battery capacity has barely improved [41,43]. This issue affects mobile devices when connected to distributed environments irrespective of the distributed environment purpose. For instance, some researchers have studied the viability of connecting mobile devices through wireless networks to peer-to-peer (P2P) network overlays, such as Kademlia, and the impact of these connections on battery consumption [22]. This work is oriented to content sharing, such as music or video. In addition, other researchers aim at minimizing battery consumption during data transfer [40] within a program, for example transferring a serialized object, which usually requires less data transfer than content itself.

Furthermore, there is a line of research that aims at reducing battery consumption by offloading complex computational tasks from mobile devices into distributed environments [42, 24]. In particular, [24] analyzes the possibility of offloading mobile devices processing into Cloud Computing environments, such as Amazon EC2[1]. The authors discuss different usage scenarios and how the associated privacy and security mechanisms might affect this approach efficiency.

[1] http://aws.amazon.com/ec2/

In contrast, several researchers [33, 14, 26, 12, 44, 30, 45, 8, 1] have proposed using mobile devices as resources to execute complex computational tasks. These researchers argue that mobile devices are a considerable source of computational resources and they are able to handle this kind of tasks [43]. However, these works have identified two main issues that are not present in distributed environments where only fixed servers and desktops machines are connected. The first problem is that wireless networks are not as fast, reliable and low-latency as wired networks. The second issue is again that mobile devices run on battery. Hence, when using mobile device resources, the distributed environment have to take into account this factor to avoid draining mobile devices batteries, while not hurting the distributed environment global performance.

This work is focused on the second issue, i.e., considering battery consumption when scavenging mobile devices resources. In particular, this work analyzes how to apply the well-known job stealing [3] technique to balance load in distributed environments where mobile devices are used to perform CPU-intensive computational tasks. Basically, a job is an atomic unit of work that is assigned to a particular node, which executes the job. In job stealing, an unloaded node tries to take jobs from loaded nodes in an attempt to balance the workload across the nodes. In this case, the goal is to determine whether job stealing also balances battery consumption and therefore to try to maximize the amount of jobs that can be executed by a set of mobile devices without requiring incrementing their battery capacity.

This work is based on the Simple Energy-Aware Scheduler (SEAS) [44], a job scheduler for CPU-intensive processing in mobile Grids [12], or Grids comprising mobile devices only. The SEAS was designed to assign jobs to mobile devices taking into account their computational capabilities, their battery current charge and their battery consumption rate. When a job is assigned to a mobile device, the job is enqueued until the mobile device can execute it. In addition, battery consumption rate varies according to several factors, such as workload and network usage, and mobile devices battery sensors are not very accurate. As a result of these facts, the scheduler choice might not be the best possible choice, but rescheduling is not supported. Hence, a bad choice made by SEAS cannot be corrected and this might have a negative impact on the global mobile Grid energy efficiency. In this context, job stealing might rebalance the distributed environment as job execution progresses, improving the energy efficiency of the mobile Grid as a whole.

In brief, the main contribution of this paper is the assessment of different job stealing algorithms in SEAS to reduce overall battery consumption in mobile Grids, which is a major issue in the area [28]. Therefore, the goal of our study is twofold: reduce energy consumption by improving job scheduling in mobile Grids, but also obtaining better throughput by using the energy potentially saved to execute a larger number of jobs. Interestingly, the analyzed job stealing algorithms, which were originally proposed for traditional Grids, can be implemented easily in current devices. This is a key feature because some of the existing scheduling algorithms specifically proposed for mobile Grids [27, 30] require to know too much about the mobile Grid environment making them difficult, or sometimes impossible, to be implemented in real systems [45]. According to our experiments, a mobile Grid using job stealing might solve up to 24% more jobs that the same mobile Grid using SEAS. This means that with the same amount of energy, the mobile Grid finished more jobs when using job stealing.

The rest of this paper is organized as follows. Section 2 surveys related works on mobile Grids as well as job stealing algorithms. Section 3 presents our approach for using job stealing in mobile Grids, which focuses on efficient energy usage. Then, Section 4 discusses the experiments performed to validate our approach. Finally, Section 5 concludes the paper and outlines future research lines.

## 2 Related works

Since mobile devices gained the ability of connecting to the Internet through wireless networks, researchers have been studying how to exploit them in distributed computing environments. Firstly, mobile devices were proposed as visualization and management devices of this kind of environments [13]. Although some researchers still put mobile devices only in this role [19], others are studying mobile devices capabilities for scientific computing [43, 32, 21]. These researches aim at not only determining whether mobile devices are capable of executing scientific computation tasks, but also providing guidelines of how to implement this kind of software efficiently.

Although [43] pointed out that mobile devices are considerable slower than desktop machines, which is expected, [43] showed that mobile devices can perform a substantial amount of work when on battery. This means that they deliver a good task percentage execution per energy unit ratio. This is important because in mobile Grids, mobile devices are supposed to belong to different mobile device owners, which might together contribute to perform a larger work that can be divided in smaller execution units.

The motivation of mobile device owners for contributing can be different. Some mobile device owners might be willing to freely contribute to some project. This scheme has proved to be successful in traditional Grid projects, such as the ones carried out by the World Community Grid[2] or SETI@home[3]. Besides, mobile device owners might share resources because they expect to also use other mobile device resources when they need them [29]. Even more, potential mobile Grid users might consider paying external mobile device users for using this latter's mobile devices [14]. Since encouraging mobile device owners to share is a main factor for mobile Grids to suceed, there are some works [12, 10, 31] discussing alternatives towards achieving this goal. However, this topic is out of this paper scope and constitutes in itself a fresh research line in the area.

Apart from sharing issues, for which as pointed out interesting advances have been achieved, developing mobile Grids is also challenging because of mobile devices' resource limitations and intermittent network connectivity [9]. At the same time, there is a clear motivation of supporting computational mobile Grids since they are promising mainly because tree facts:

- mobile devices computational capabilities are significant [43],
- these capabilities can be used by scavenging unused mobile devices capabilities [1], and
- the ubiquity of mobile devices [42].

However, the success of computational mobile Grid heavily depends on the ability of using mobile device resources without draining the batteries. Therefore, the selection of the resource to use is very important [18, 12, 30, 8]. This problem is still open mainly because the information needed for taking optimal scheduling decisions is not usually available in real mobile Grids. In this context, our goal is to improve the amount of jobs executed using mobile Grids. To do this, we have extended the SEAS algorithm [44] with job stealing techniques.

The next section presents background on mobile Grid schedulers, and in particular, describes the SEAS, which is the scheduler this work is based on. Then, Section 2.2 discusses related works on job stealing, i.e., the techniques used to increase the energy efficiency and throughput of SEAS.

[2] World Community Grid `http://www.worldcommunitygrid.org/`

[3] SETI@home `http://setiathome.berkeley.edu/`

### 2.1 Mobile Grid schedulers

Mobile Grids present new challenges when compared with traditional Grids because mobile devices are too different from servers and desktop computers. One of these issues is how to schedule jobs for maximizing the number of executed jobs with the same battery capacity. Intuitively, existing schedulers aim at finishing as many jobs as possible before the mobile devices batteries become depleted [26,27,28,12,44,45,30].

Examples of these kinds of schedulers are presented in [26,27,30] where different optimal job schedulers for Grids of mobile devices are proposed. These schedulers take into consideration different important variables, but all of them aim at optimizing job assignation to minimize energy consumption while maximizing utilities. Utility is defined by giving a value to each job, i.e., the value of executing a job "A" might be different from the value of executing a job "B". To know whether a job can be assigned, these schedulers need to know exactly how many work-units, e.g., operations, are necessary for completing each job, how much energy a work-unit consumes in each device, the time limits for each job and the amount of available energy in each device. Assuming all this information is known, the scheduler assigns the jobs using a non-linear optimization function that aims at maximizing the utility, while minimizing energy consumption. In particular, the optimization method is based on Lagrange multipliers. The main drawback of this approach is that these assumptions are very unlikely to hold in real-life deployments. In addition, these schedulers do not take into account owners' usage of the mobile devices, i.e., these latter are assumed to be dedicated computing nodes.

In [12], the authors propose another scheduling and pricing strategy. The authors assume that the connection between a Grid and the mobile devices is an edge router, called *Grid controller*. This mobile Grid model is frequently used, and the Grid controllers are generically known as Proxies [45]. The proposed scheduler operates as a market, where mobile devices offer their computational capabilities to the Grid. In this market, the Grid controller works as a broker, selecting a mobile device based on historical information of price and the probability of accepting a job. Then, the Grid controller makes an offer for having the job executed. The mobile device might accept the offer, reject it or negotiate for a better price. If the offer is accepted or rejected, the scheduling process finishes. But in case the negotiation continues, the Grid controller re-offers and the process continues until the job is accepted by the mobile device, or until the Grid controller or the mobile device finish the negotiation. To perform the negotiation, the Grid controller must know the work-units required by the job, the mobile device processing rate, the mobile device remaining uptime estimation and the CPU time price.

Although there are other proposed schedulers, in general, they all share the same assumptions [44,45]. As a result, they all inherit the same weaknesses. Firstly, battery estimation is one of the most important problems. Although there are models for estimating this [4,16], they require to know several variables about the battery, such as electrolyte concentration, electrolyte potential, or solid-phase potential, usually not available in real mobile devices. In addition, these models are based on complex mathematical models that might represent complex jobs themselves for mobile devices, making them unsuitable to be used as part of mobile Grid schedulers because scheduling decisions also consume battery. The other problem is knowing how long it would take to execute a job in a mobile device. In general, this problem is impossible to solve because solving it would mean solving the halting problem [50]. However, [7] aims at estimating both time and battery consumption for executing a particular application. Yet, this approach uses other complex models that

make it unsuitable for scheduling because in some cases it may take longer to perform the model simulation than to actually run the applications, thus wasting energy.

Furthermore, the SEAS [44] is an algorithm that uses simple mathematical models to estimate how many energy units per job are assigned in a particular mobile device. Having this information, the SEAS tries to keep this proportion balanced across all mobile devices connected to the Grid. Since this work is heavily based on the SEAS, APPENDIX A present a comprehensive description of the scheduler.

Interestingly, the SEAS has proven to be more suitable for mobile Grids than traditional scheduling algorithms for Grids, such as Round Robin, or Random assignation. However, the SEAS might experience serious performance loss under certain common conditions. Firstly, if all jobs arrive at the initial time, the battery estimation might not be good enough because the estimation algorithm has not enough data yet to make an accurate estimation. Secondly, the SEAS does not take into account mobile devices workload from user applications. Finally, if jobs present too much variation in computational requirements, the required resources per job estimation might be biased. Since using a resource consumes energy, wrongly performing the required resources per job estimation means wrongly estimating the required energy for a job, which might in turn result in an energy waste.

In order to overcome these issues, we evaluated the outcome of equipping SEAS with traditional job stealing techniques. Job stealing is a well-known paradigm for job scheduling in parallel and distributed environments that has proven to be effective, algorithmically simple, and easy to implement. The next section introduces related works in the job stealing area.

## 2.2 Job stealing

Broadly speaking, regardless the parallel or distributed environment targeted, the problem of scheduling a set of jobs on several computational resources can be addressed by following two paradigms: *work sharing* and *work stealing*. In work sharing, whenever a resource or node has some jobs to execute, it attempts to move some of the jobs to underutilized resources. In opposition, in work stealing –from now on job stealing– idle resources periodically try to take jobs from overloaded resources.

The benefits of one approach over the other have been in the past subject of long debates [3]. In traditional Grid environments, job stealing is very popular, as evidenced by a number of job schedulers proposed to efficiently harness Grid resources based on deterministic resource victim selection (e.g., [38]) or those that rely on random decisions (e.g. [51, 47, 49]). Within the former group, the Javelin 3 [38] middleware arranges Grid resources as a unique tree, and includes a scheduling algorithm by which whenever a resource runs out of jobs to execute, selects a neighboring resource to request jobs from based on the tree structure. In other words, an idle resource first attempts to steal job from its children, if any, and if unsuccessful, from its parent. This ensures that all the jobs assigned to the subtree rooted at a resource are processed before that resource takes new jobs from its parent.

With respect to random job stealing algorithms, the JCluster platform [51] proposes Transitive Random Stealing, by which each resource remembers the originating resource (history information) from which a job was last received after a steal attempt and sends requests directly to that resource (the short-cut path). In addition, the stealer resource also forwards this history information to other resources which want to take a job from the stealer node (the transitive policy). On the other hand, the WSPE (Work Stealing Programming Environment) [47] Grid programming environment proposes Round Stealing, which enhances

the original Random Stealing algorithm. Instead of randomly selecting a neighbor resource to send a steal request, the resource sends individual asynchronous steal requests to each neighbor in rounds. As soon as the first reply (i.e., job to execute) arrives, the obtained job is pushed onto the resources local job queue. As the communication demands of the proposed algorithm are high, WSPE complements its scheduler with a peer-to-peer network overlay that arranges resources based on network latency.

Furthermore, Satin [49] is a Grid platform that includes the Cluster-aware Random task Stealing mechanism (CRS). With CRS, when a Grid resource becomes idle, it attempts to steal an unfinished task both from resources belonging to the same local cluster or *external* resources (i.e., from clusters reached via WAN links), however intra-cluster steals have higher priority than inter-cluster ones, minimizing expensive WAN communication. This job stealing mechanism is indirectly exploited by some Grid meta-schedulers such as JGRIM (Java GRidifying by Injecting Metaservices) [35] and BYG (BYtecode Gridifier) [37], which partially rely on Satin for handling job execution.

## 3 Energy-efficient job stealing

In traditional distributed computing environments, job stealing is used as a computational load balancing mechanism [49]. Basically, this technique aims at minimizing unused nodes in such environments. In addition, this prevents jobs from waiting to be executed when there are idle resources, which improves throughput. Since these are important concerns in mobile Grids as well, job stealing might be applied in this context. However, it is necessary to consider the potential energy efficiency levels offered by these techniques [45] to successfully apply job stealing in mobile Grids.

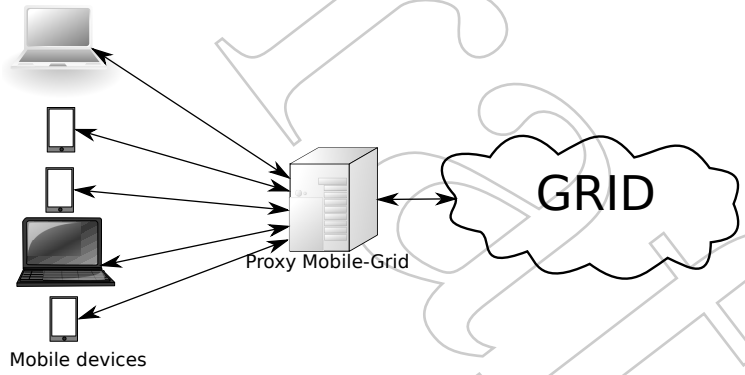


**Fig. 1** Proxy-based mobile Grid

Fig. 1 depicts the mobile Grid architecture on which this work is based. Basically, this infrastructure is centralized from the mobile devices point of view. The central component, called *proxy*, is the one that handles the interaction between the traditional Grid and the mobile devices as well as mobile devices interactions. In this infrastructure, nodes from the traditional part of the Grid see the mobile Grid as a single virtual resource [26, 44, 12] to which they can ask for executing jobs, but internally it is a distributed environment. From now on, this work is focused on the proxy and mobile devices. In this context, the former receives jobs to execute, and it assigns them to the mobile devices to perform the execution.

It is worth mentioning that, in its present form, the proxy based architecture has two main drawbacks [45]: scalability and reliability. Firstly, the number of mobile devices that can be connected to a proxy is limited by the computational capabilities of the proxy. Secondly, if a proxy fails, all the mobile devices connected to that proxy would become unavailable to the Grid. Yet, the scalability problem can be easily solved by adding new and independent proxies to the Grid. On the other hand, the reliability issue might be solved using redundancy or reconnecting the mobile devices to other proxies. Although these issues are out of this paper scope, notice that they have been tackled in different previous works [18,17,23,20]. Some of these advances could be then applied in the near future to ensure scalability and reliability to the architecture.

As mentioned above, this work is based on the SEAS algorithm. This means that when the proxy receives a job execution request, it uses the SEAS algorithm to assign the job to a mobile device. Notice that the SEAS assumes that the jobs are independent and atomic computational units, which means that a job never depends on other jobs' output and the job cannot be divided for executing in several mobile devices. In the original SEAS approach, there are only two possible outcomes for the job. One of them is that the job is executed in the mobile device and it returns the execution result to the proxy. In this case, if the mobile device has no more jobs assigned, it remains idle until the proxy assigns it a new job. The other one is that the mobile device runs out of battery before finishing the job execution, which means that the job execution is canceled. In addition, when a node runs out of battery, it does not only cancel the currently executing job, but also all the jobs enqueued in that node waiting to be executed. Notice that when a mobile device runs out of battery, several jobs might be canceled at once, namely the ones which are executing and the ones which are enqueued waiting to be executed in the device.

By introducing job stealing, mobile devices are more active during job assignation. According to this fact, the proxy works as described above receiving jobs and assigning them to mobile devices. However, mobile devices behave differently because when a mobile device finishes executing all its assigned jobs, it looks at other mobile devices to steal their jobs. Essentially, when a mobile device becomes idle, it selects another mobile device, which is called *victim*, and tries to offload it.

The first issue is how an idle mobile device, called *stealer*, selects a victim. We have analyzed three different *strategies* for selecting the victim:

Random Stealing (RS): This selection strategy consists on choosing a random mobile device as a victim. We selected this strategy because it is widely used in job stealing algorithms for traditional Grids [51,47,49]. This strategy usually performs well in many scenarios and its computational cost is very low.

Best Ranking Aware Stealing (BRAS): This selection strategy consists on picking up the mobile device that is best ranked according to the SEAS ranking criteria and naturally has enqueued or unfinished jobs. This strategy aims at offloading the least overloaded mobile devices.

Worst Ranking Aware Stealing (WRAS): This selection strategy is also based on the SEAS ranking strategy, but, instead of selecting the best ranked mobile device, it selects the worst ranked one. In this case, the goal is to globally balance the load because the mobile devices that have no load take the load from the most loaded ones.

Notice that the SEAS ranking formula uses three variables: a benchmark factor of each device, the estimated uptime of a device, and the number of jobs assigned to that device (please refer to APPENDIX A for details on the formula). Basically, the ranking consists in multiplying the benchmark by the estimated uptime to determine how many units of work

the mobile device can perform in a given time. Then, this number is divided by number of jobs plus one in order to determine how many units of work would be available, on average, to execute each job if other job is assigned to that mobile device. In this context, BRAS tries to offload nodes that are likely to finish their jobs, so they become idle and start stealing jobs quickly. As a result, victims are more likely to become idle, and, in turn, they would be themselves able to further offloading other mobile devices. Essentially, this strategy is expected to generate an offloading chain-reaction. In contrast, WRAS offloads the nodes that are overloaded to make more likely that this mobile device can finish all the remaining jobs assigned to them. As a result, we expect that if more nodes are more likely to finish all their jobs, more jobs would be finished by the time all mobile devices run out of battery.

In addition to selecting a victim, the stealer should also determine how many jobs it will steal. Stealing several jobs at once might reduce the networking overhead because it requires establishing only one connection. Since networking requires using a lot of energy [25,22] reducing the need for that might extend the battery life. For this issue, we have analyzed two *policies*:

Fixed Number: In this policy, a stealer always steals the same (statically determined) number of jobs. In this model, the stealer always behaves the same upon each steal attempt. In these experiments, we fixed this number to one because is the maximum number of jobs that a node can execute at once. As will be explained in the next Section, we employed single-core mobile devices in the experiments. For $n$-core mobile devices, each steal attempt might actually try to retrieve around $n$ jobs.

Exponential: This policy exponentially increases the number of stolen jobs based on how many times the node became idle. Basically, if a mobile device becomes idle for the $n^{th}$ time, it would steal $2^n$ jobs. For example, the first time, the stealer steals one job ($2^0$), the second time, the stealer steals two jobs ($2^1$), and so on.

One victim selection strategy and one offloading policy represent a particular combination that can be used in a mobile Grid. The next section presents an evaluation of the six possible combinations between strategies and policies in terms of throughput and energy usage when applied to mobile Grids.

## 4 Evaluation

In order to evaluate the different job stealing algorithms, we have performed several simulations. Simulation is a common method to evaluate different algorithms for distributed computing [5,6] because it reduces time and costs, making it possible to test approaches before a large-scale deploy is carried out. In addition, simulation makes experiments easily replicable.

For making the simulations as real as possible, we have profiled different mobile devices battery consumption under several CPU load conditions. Then, we used the profiles to extrapolate how mobile devices would behave when working as part of a mobile Grid. Therefore, Section 4.1 outlines the procedure followed to profile mobile device battery consumption, as it is a crucial aspect of our assessment. Then, Section 4.2 presents our simulation approach. Finally, Section 4.3 discusses the experimental results.

### 4.1 Mobile device profiling

For performing intra-mobile device profiling, we developed an application to take into account three variables: time, CPU load and battery charge. Although the profiler was implemented for Android, the profiling method can be easily adapted to any other mobile device platform. The proposed profiler measures two of them, namely time and battery charge, while it tries to keep a constant CPU load (Target CPU Load), which is given as a parameter. To measure time, the profiler uses the mobile device internal clock. For measuring battery, the profiler uses the event based system battery report, in this case via the Android Intent API. Essentially, the profiler saves the battery charge and the time upon each new battery event issued by the Android system.

To assure that the measures are consistent with a particular CPU load, the profiler must force the mobile device to keep a particular Target CPU Load. To do this, we have designed a subsystem that generates CPU load by performing floating-point operations in a dedicated thread. However, constantly executing floating-point operations consumes 100% of the CPU. Therefore, a monitoring component constantly adjusts a delay time between the operations.

Algorithm 1 shows the logic of the thread that consumes CPU. Basically, the thread that executes this algorithm sleeps during a period of time and then executes a set of floating-point operations, and it repeats this process until the thread is externally killed. The sleeping time is being constantly adjusted by the Algorithm 2, which runs in another thread. In contrast, the number of operations executed in each cycle is fixed in the constant CYCLES. This constant had to be defined because the granularity of the time in the method "wait" is milliseconds. Since current mobile devices processors are relatively fast, it was impossible to control the CPU load by only executing a floating-point operation and sleeping. In all the profiles we executed, the CYCLES constant was fixed in one million. Yet, this number could not work well with slower or faster processors.

Algorithm 2 shows the algorithm of the thread that adjusts the sleep time for generating the target CPU load. As mentioned, this algorithm runs in another thread and adjusts the CPU usage. Firstly, it measures the CPU use by calculating the average of 30 measures taken in 200 milliseconds. Each measure is calculated using the information reported by Android through the `/proc/stats` file, in the same way as the Linux command top[4] . Then, the thread calculates the rate between the current CPU use and the target CPU use. Using this rate, it modifies the sleep time in the CPU loader thread in the same proportion in an attempt to move the CPU usage closer to the target. Notice that when the target CPU usage is 0%, none of these threads are started.

To obtain the profiles, we ran the software in some mobile devices with fully charged batteries and plugged to the electrical power line. Then, when the CPU load was near to the Target CPU Load by a threshold of 5%, we unplugged the mobile devices and left the profiler running without human interaction until the batteries were depleted. The profiler logged all the recollected data in a file on each mobile devices SD card.

From a technical point of view, we had to take some precautions to minimize Android scheduler and power manager impact on the profiles. The first problem is that the Android scheduler is very aggressive and might terminate the profiler, which intensively uses the CPU, especially when the mobile device screen is locked. To prevent this situation from happen, both threads, the CPU loader and the CPU loader adjuster, run within a Service in

[4] Linux top command: `http://procps.sourceforge.net/`

**Algorithm 1** CPU loader thread

```
1: procedure CPULOADER
2:     while true do                                              ▷ Never ends
3:         if SLEEP > 0 then
4:             WAIT(SLEEP)                  ▷ Waits for SLEEP milliseconds without using CPU
5:         end if
6:         count ← 0
7:         while count < CYCLES do
8:             PERFORMFLOATINGPOINTOPS                            ▷ Uses CPU
9:             count ← count + 1
10:        end while
11:    end while
12: end procedure
```

**Algorithm 2** CPU loader adjuster

```
1: procedure CPULOADERADJUSTER(TargetCPULoad, Threshold)
2:     while true do                                              ▷ Never ends
3:         cpuLoad ← GETCPUUSAGE          ▷ Average 30 measurements separated by 200 ms
4:         diff ← cpuLoad/TargetCPULoad
5:         if −Threshold < 1 − diff < Threshold then
6:             NOTIFYSTABLE          ▷ Informs that the CPU load is near the target CPU load
7:             LOG(cpuLoad)
8:         else
9:             sleep ← CPULoader.GETSLEEP()  ▷ Gets the current value of the variable SLEEP in the
   CPU loader thread
10:            if sleep = 0 then
11:                sleep ← 1
12:            end if
13:            CPULoader.SETSLEEP(sleep ∗ diff)  ▷ Adjust the sleeping time  ▷ Adjust the value of
   SLEEP in the CPU loader thread
14:        end if
15:    end while
16: end procedure
```

foreground. Basically, a Service[5] is a special class of the Android framework that represents a component that performs long-running operations without providing any graphical interface. In particular, when a Service is in foreground, it tells the Android scheduler that this latter should avoid killing the Service unless extremely necessary[6].

The other main issue is that Android might reduce the CPU speed to preserve battery. However, we assume that the CPU will be fully used when processing computations within a mobile Grid, which must be took into account in our profiling application. Android provides a mechanism, called *power locks*, that allows applications to tell whether it needs to keep some part of the system to be active. In particular, the above service asks for a *partial wake lock*[7] that keeps the CPU active, but not the screen or the keyboard light on. We also keep a WiFi connection active and connected to the Internet so that bundled applications, such as e-mail, can stay active. Besides, a mobile device connected to a Grid is expected to be connected to a wireless network.

[5] Android Services: `http://developer.android.com/guide/topics/fundamentals/services.html`

[6] Android Processes and Threads: `http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html`

[7] Android Power Lock: `http://developer.android.com/reference/android/os/PowerManager.html`

| Device | Target CPU load | Total Profiling Time (hh:mm:ss) | # of Measurements | Average CPU load | Standard deviation |
|---|---|---|---|---|---|
| Samsung I5500 | 0%[8] | 35:23:47.082 | 7034 | 3.83% | 1.08% |
| | 30% | 22:57:43.507 | 4831 | 30.26% | 3.92% |
| | 75% | 11:15:20.006 | 2385 | 75.28% | 4.04% |
| | 100% | 9:45:28.792 | 2066 | 99.98% | 0.04% |
| ViewPad 10s | 0%[8] | 27:15:39.293 | 5476 | 10.23% | 2.26% |
| | 30% | 19:29:27.824 | 4154 | 30.11% | 2.01% |
| | 75% | 13:49:49.690 | 2951 | 76.92% | 1.76% |
| | 100% | 13:57:44.642 | 2977 | 99.99% | 0.03% |

**Table 1** Profiler CPU load

Table 1 outlines the results obtained by the profiler in regard to the injected CPU load. Firstly, it can be noticed that the profiled mobile devices have always the CPU slightly loaded when the Target CPU Load is 0%. This might result from the fact that bundled applications can periodically perform some operations, such as fetching e-mails or look for software updates. On the other hand, the Android platform has a very active role on application life-cycle, such as managing Intents, which also use CPU. Despite these facts, the other profiles were generated with a CPU load very approximated to the Target CPU Load. In addition, the standard deviation is very low, indicating that the dispersion of the measurements is small.

Finally, we executed Java versions of the well-known Linpack and SciMark 2.0 benchmarks on the Samsung I5500 and ViewPad 10s. These benchmarks were used to determine how fast these mobile devices are for performing scientific computational tasks [43]. As a result of these benchmarks, we have determined that the Samsung I5500 has 7.6 megaflops, while the ViewSonic ViewPad 10s has 35.49 megaflops. Although these are the Linpack results, the SciMark 2.0 results were very similar, being these results 7.24 and 35.06 for the Samsung I5500 and the ViewSonic ViewPad 10s respectively.

### 4.2 Mobile Grid simulation

As stated earlier, simulation is a widely accepted practice for evaluating distributed systems performance [5,6]. This is because it is difficult to deploy a distributed environment to evaluate different approaches. In addition, simulated environments allow researchers to fairly compare different approaches. Hence, we used a simulated environment to assess job stealing in mobile Grids, which is also an accepted practice in mobile Grids [26,27,30].

To perform the experiments, we then used an event based simulation software of our own. This means that everything that might occur in the mobile Grid is considered by the software as an event. For instance, a job arrival, battery notification, or job terminations are all different kind of events for the simulator. Hence, we had to convert all the profiled information to events for the simulator. In addition, we generated jobs with their associated arrival time and requirements in terms of floating-point operations each input job needs to finish.

In order to simulate a job execution in a mobile device, the simulator uses two profiles of the same mobile device: the base profile –i.e., the default CPU consumption by Android itself and the mobile device owner– and the 100% CPU load profile. The base profile represents the use of the mobile device when it is not running any job as a mobile Grid node. The 100% CPU load profile is used when simulating that the mobile device is running a job

[8] The CPU load registered is generated by the Android OS and bundle applications

assigned from the mobile Grid. As the reader can note, we assume that each job is optimized and correctly coded to use as much CPU as it is available.

The intra-device model used to switch between profiles when a job is started or finished, and calculating how long it would take to finish the job, is based on the following assumptions:

- A job uses the unused CPU of the base profile. For instance, if the user is currently using 31% of the CPU, the job will use the remaining 69%.
- The CPU used by a job might vary if a base profile CPU event occurs while the job is executing. This is because we assume that the base profile represents user CPU usage and the mobile Grid should not interfere with users' tasks. Therefore, if a user requires more CPU, the mobile Grid should allow it to happen.
- The CPU usage of a profile between events is constant. This means that the CPU load only changes on registered events.
- The battery consumption between two battery events is lineal. This is because mobile devices do not allow users to analyze what happens between battery events.To calculate how long it takes to perform a job in a particular mobile device, the simulator analyzes what occurs between events. Firstly, when the job arrives, it calculates using Equation 1 whether the job will finish before the next battery event or not:

$$job\,time = \frac{job\,operations}{device\;flops * (1 - current\,CPU\;use)} \tag{1}$$

If the job finished before the next battery event, the simulator adds the corresponding event into the events queue. Otherwise, the simulator calculates how many operations will be performed before the next CPU event and updates the number of operations in the job (see Equation 2), so the simulator can perform the previous analysis when the new CPU event arrives. This continues until either the job is finished or the mobile device depletes its battery, leaving the mobile Grid.

$$updated\,operations = job\,operations - device\;flops * (1 - current\,CPU\;use) * time\,to\,battery\,event \tag{2}$$

As suggested, the model used to simulate profile switching is based on the assumption that the remaining battery between two events can be calculated as a lineal function, and a mobile device uses 100% of its CPU when it is running a job. Basically, when a mobile device profile changes, the simulator calculates the actual battery charge and then calculates, using the new profile, how long it would take until the next battery event. Therefore, the simulator removes the actual battery event and adds a new one that will trigger later on.

Fig. 2 depicts how the simulator switches between the base profile, which is based on a 30% CPU target profile, and the 100% CPU target profile. As it can be seen in the upper part, battery events occur when the battery discharges by 1% or a job starts or finishes. This figure also shows that the gradient of the remaining battery might vary when a battery event unrelated to a job execution occurs (the third battery event from left to right). Finally, the figure, in its bottom part, also shows that the base profile is used to determine the CPU percentage a job will use.

### 4.3 Experimental results

To evaluate how the different job stealing techniques behave on mobile Grids, we defined 16 different execution scenarios. All these scenarios use a mobile Grid comprising 100 mo-

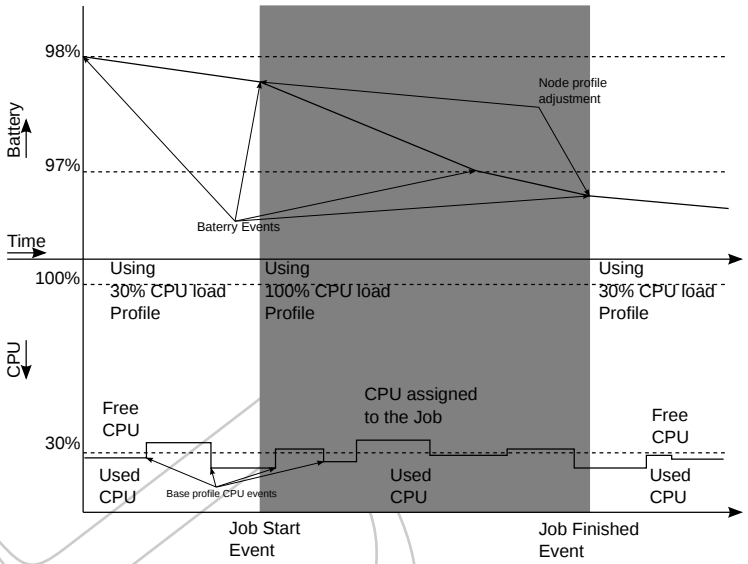


**Fig. 2** Profile switch model

bile devices, following some of the profiles mentioned in the previous section as well as different number of jobs. Firstly, we selected two types of mobile Grid deployments: one consisting of 70 Samsung I5500 and 30 ViewPad 10s, the other consisting of 50 Samsung I5500 and 50 ViewPad 10s. Secondly, we selected two base profiles for each type: the 0% CPU load and the 30% CPU load. As a result, we obtained four different mobile Grid configurations. Regarding to mobile Grid constitution, we considered two cases: one were low-end devices are more common that high-end ones, and other were both types are equally common. In addition, the two base profiles used represent one in which mobile devices are not used at all and other in which they are used for non CPU intensive tasks, such as reading mail or browsing the Web.

For each of these mobile Grid configurations, we generated two different job configurations, namely short and long jobs. Short jobs take to execute an average of 5 minutes with a standard deviation of 2.5 minutes in a normal distribution. In contrast, long jobs take to execute an average of 30 minutes with a standard deviation of 15 minutes in a normal distribution. In both cases, it is assumed that the mobile device load is 0%. Notice that the short/long job configurations are instead different for each mobile Grid configuration because the numbers of operations for obtaining these times are different not because of the CPU profiles, but the hardware. For instance, the short jobs for the 70 Samsung I5500 and 30 ViewPad 10s Grid have 4790.97 million operations on average, while the short jobs for the 50 Samsung I5500 and 50 ViewPad 10s Grid have 6464.55 million operations on average. The following equation is used to obtain the number of operations for each configuration.

$$job\,average\,operations = time * average\,flops \tag{3}$$

where, the average flops is defined as the average mobile device flops:

$$average\,flops = \frac{\sum mobile\,device\,flops}{number\,of\,mobile\,devices} \tag{4}$$

| Grid nodes | Base profile | Job Configuration |
|---|---|---|
| 70 Samsung I5500 - 30 ViewPad 10s<br>50 Samsung I5500 - 50 ViewPad 10s | 0% CPU load<br>30% CPU load | Short Id.<br>Long Id.<br>Short Sat.<br>Long Sat. |

**Table 2** Grid simulation configurations

Finally, for each mobile Grid-job length combination, we generated another two configurations based on how much work is assigned to the Grid. The first configuration, which we call *ideal* (id. for short), consists of assigning to the mobile Grid approximately the maximum number of jobs that the mobile Grid ideally could finish before depleting the overall mobile devices energy capacities. The second configuration, which we call *saturated* (sat. for short), consists of assigning to the mobile Grid approximately twice as much as the maximum number of jobs that the mobile Grid could handle according to the ideal configuration. Basically in this case, what changes is not the mobile Grid hardware, but the number of jobs. The following equation, in which *uptime* refers to the uptime in the 100% CPU use profile, was used to estimate the ideal number of jobs:

$$ideal\,jobs = \frac{ViewPad\,flops * uptime + I550\,flops * uptime}{job\,average\,operations} * (1 - base\,profile\,CPU) \quad (5)$$

In each of these 16 scenarios, which are the combination of the different settings described in the columns of Table 2, we tested the original SEAS [44] and the SEAS with job stealing using the different strategy-policy combinations. This means that there were seven schedulers: six with job stealing, which resulted from combining the three strategies (RS, BRAS and WRAS) with the two policies (Fixed, Exponential or Exp), and one without job stealing. All in all, 112 scenarios were evaluated during the simulations.

The simulations were designed to determine which approach executes more jobs using the same mobile Grid configuration. This is because executing more jobs with the same mobile Grid configuration means that the mobile Grid as a whole is using less energy per job [26,27,12,45], meaning that the mobile Grid would be more energy-efficient. In other words, given the same set of jobs and the same overall energy capacity, we evaluate which stealing technique better improves the original SEAS under the designed scenarios, if applicable. Another issue we studied is how many job transferences (i.e., moving a job from a device to another) each stealing strategy generates. This is important because in real life mobile Grids each stealing attempt might represent extra energy consumption because it might use network resources at both ends [25,22].

However, notice that networking is out of this paper evaluation. Hence, the simulations presented in this Section assume a perfect network that means that a) data transfer are instantaneous, b) there is no off-line time and c) using the network does not consume battery. Although these assumptions do not hold in real life mobile Grids, they are commonly accepted in the research area when analyzing and experimenting with mobile Grid resource managers [33,26,12,44,45], and schedulers in particular. In fact, a) and b) apply to all the tested job stealing techniques, and thus a fair experimental testbed is used.

With regard to c), and according to [42], sending a small packet (1KB or less) from a standard mobile device through a WiFi link requires around 0.02 Joules, while sending a 10KB packet or higher requires 0.15 Joules. In our context, a job stealing request is a

| Metric | Samsung I5500 | ViewPad 10s |
|---|---|---|
| Volts | 3.70 | 8.20 |
| MAh | 1,200 | 3,300 |
| Wh | 4.40 | 27.06 |
| W-s (Joules) | 15,984 | 97,416 |

**Table 3** Battery-related characteristics of the mobile devices used

small packet, whereas transferring a job from a mobile node to another would require one medium-sized packet as we are dealing with CPU-intensive (and not data-intensive) jobs. As will be discussed later, the worst execution in terms of average steal requests resulted in 22,825 steals. Considering the worstcase scenario in which each job steal request leads to a job transfer (i.e., all requests are successful) the total required energy in Joules is $22,825 * (0.02+0.15) = 3,880.25$. Then, considering the battery data shown in Table 3, this energy, when compared to the amount of energy available in the two 100-node Grid configurations (70 Samsung I5500/30 ViewPad 10s, and 50 Samsung I5500/50 ViewPad 10s) represents $\frac{3,880.25}{70*15,984+30*97,416} = 0.068\%$ and $\frac{3,880.25}{50*15,984+50*97,416} = 0.096\%$, respectively. Indeed, even when there is certainly some impact, these very small percentages confirm that ignoring network energy consumption in our simulations does not compromise the significance of the results. It is worth noting that, in the calculations, the energy necessary to maintain a WiFi connection active (around 0.024 Joules [42]) is not computed as mobile device profiling was performed with the WiFi connection active (see Section 4.1).

Turning to the simulations performed, we carried out 10 different runs for each scenario In each run, two variables were analyzed: the percentage of finished jobs and the percentage of stolen jobs. Table 4 presents the average percentage of finished jobs as well as the standard deviation obtained in the 10 simulations. In all the cases, the standard deviation is less than 1.5%. This indicates that the results were fairly stable in regard to finished jobs across the different simulation runs. An interesting fact is that in the saturated configurations, all the job stealing techniques tend to finish approximately the same amount of jobs, which is always higher than the amount finished by the SEAS.

The other variable analyzed is the number of stolen jobs over total jobs. Notice that this metric can be more than 100% if the stealing technique produced more steals than jobs are in the simulation, e.g., if each job is stolen 2 times on average, this metric value would be 200%. Table 5 shows the average and standard deviation of this metric in the different simulation scenarios. The standard deviation for this variable can be as high as 16.4%, but this happens when the average is high, e.g., more than 130%. Therefore, these cases do not affect the analysis because these values are far from optimal results. Yet, the average standard deviation is 2.08%, which indicates that the deviation is not that high. In fact, the Pearson correlation between the average of each scenario and the standard deviation is 0.875 ($\rho < 0.0001$). This means that in the best cases, i.e., when the number of steals is small, the deviation tend to be small too, so the best cases did not vary in the different runs. Having this into consideration, the following analyses were performed using the average values previously discussed.

The first evaluation aims at studying the percentage of the ideal number of jobs each scheduler can successfully execute. Therefore, the analysis is limited to the eight scenarios with the ideal number of jobs. Fig. 3 outlines the results of this study. Basically, the simulations revealed that in 64.28% of the cases more than 90% of the jobs were finished and in 44.64% of the cases more than 95% of the jobs were executed implying that all the presented techniques have a fair performance. Another interesting fact is that WRAS with

| Configuration | SEAS | RS Fixed | BRAS Fixed | WRAS Fixed | RS Exp | BRAS Exp | WRAS Exp |
|---|---|---|---|---|---|---|---|
| 30 ViewPad 70 I550 / 0% / Short Id | 74.5% +/- 0.206% | 96.8% +/- 0.173% | 97.9% +/- 0.065% | 98% +/- 0.078% | 87% +/- 0.758% | 93.9% +/- 1.477% | 97.8% +/- 0.118% |
| 30 ViewPad 70 I550 / 0% / Short Sat | 27.4% +/- 0.082% | 33.3% +/- 0.087% | 33.3% +/- 0.117% | 33.3% +/- 0.113% | 33.3% +/- 0.084% | 33.3% +/- 0.11% | 33.3% +/- 0.108% |
| 30 ViewPad 70 I550 / 0% / Long Id | 77% +/- 0.486% | 86.8% +/- 0.901% | 92.6% +/- 0.442% | 95% +/- 0.456% | 86.2% +/- 1.64% | 92.3% +/- 0.959% | 94.7% +/- 0.441% |
| 30 ViewPad 70 I550 / 0% / Long Sat | 28.1% +/- 0.143% | 34% +/- 0.314% | 34.1% +/- 0.243% | 34% +/- 0.295% | 34% +/- 0.27% | 34% +/- 0.21% | 34% +/- 0.267% |
| 50 ViewPad 50 I550 / 0% / Short Id | 86.7% +/- 0.175% | 97.9% +/- 0.084% | 98.5% +/- 0.062% | 98.6% +/- 0.122% | 88.8% +/- 1.375% | 95.3% +/- 0.482% | 98.8% +/- 0.045% |
| 50 ViewPad 50 I550 / 0% / Short Sat | 33% +/- 0.064% | 36.7% +/- 0.078% | 36.7% +/- 0.071% | 36.7% +/- 0.066% | 36.7% +/- 0.076% | 36.7% +/- 0.062% | 36.7% +/- 0.074% |
| 50 ViewPad 50 I550 / 0% / Long Id | 89.4% +/- 0.423% | 87.4% +/- 0.711% | 95.3% +/- 0.635% | 96.7% +/- 0.27% | 88.6% +/- 0.781% | 96.4% +/- 0.259% | 96.5% +/- 0.348% |
| 50 ViewPad 50 I550 / 0% / Long Sat | 32.9% +/- 0.174% | 37% +/- 0.228% | 37.1% +/- 0.199% | 37% +/- 0.175% | 37% +/- 0.192% | 37.1% +/- 0.213% | 37% +/- 0.206% |
| 30 ViewPad 70 I550 / 30% / Short Id | 79.2% +/- 0.337% | 95.9% +/- 0.206% | 97.8% +/- 0.091% | 97.8% +/- 0.051% | 85.7% +/- 0.983% | 97.4% +/- 0.26% | 97.8% +/- 0.043% |
| 30 ViewPad 70 I550 / 30% / Short Sat | 32% +/- 0.125% | 36.5% +/- 0.118% | 36.5% +/- 0.125% | 36.5% +/- 0.136% | 36.5% +/- 0.13% | 36.5% +/- 0.152% | 36.5% +/- 0.131% |
| 30 ViewPad 70 I550 / 30% / Long Id | 79.9% +/- 0.632% | 85.8% +/- 0.949% | 92.1% +/- 0.726% | 94.6% +/- 0.356% | 85.9% +/- 0.684% | 94.2% +/- 0.534% | 95.4% +/- 0.128% |
| 30 ViewPad 70 I550 / 30% / Long Sat | 32% +/- 0.2% | 36.5% +/- 0.24% | 36.5% +/- 0.296% | 36.4% +/- 0.31% | 36.4% +/- 0.276% | 36.4% +/- 0.24% | 36.4% +/- 0.247% |
| 50 ViewPad 50 I550 / 30% / Short Id | 89.2% +/- 0.349% | 97% +/- 0.182% | 98.4% +/- 0.069% | 98.6% +/- 0.037% | 88.8% +/- 1.119% | 97% +/- 0.493% | 97.3% +/- 0.189% |
| 50 ViewPad 50 I550 / 30% / Short Sat | 37.4% +/- 0.157% | 40.5% +/- 0.155% | 40.5% +/- 0.119% | 40.5% +/- 0.13% | 40.5% +/- 0.133% | 40.5% +/- 0.121% | 40.5% +/- 0.107% |
| 50 ViewPad 50 I550 / 30% / Long Id | 89.1% +/- 0.814% | 86.6% +/- 1.019% | 93.3% +/- 1.044% | 96.7% +/- 0.214% | 88.9% +/- 0.786% | 96.4% +/- 0.289% | 96.8% +/- 0.287% |
| 50 ViewPad 50 I550 / 30% / Long Sat | 36.8% +/- 0.318% | 40.6% +/- 0.29% | 40.5% +/- 0.352% | 40.5% +/- 0.354% | 40.5% +/- 0.321% | 40.5% +/- 0.406% | 40.5% +/- 0.402% |

**Table 4** Rate of finished jobs over total jobs (average $\pm$ standard deviation)

Fixed Number policy finished 95% of the jobs in 6 out of the 8 scenarios, while WRAS with Exponential policy finished 95% of the jobs in 7 out of the 8 scenarios.

Once we performed this simulation, we analyzed which of the job stealing techniques executed more jobs in each one of the 16 scenarios. Basically, WRAS with the Fixed Number policy executed more works in 5 out of the 16 scenarios, while WRAS strategy with the Exponential policy performed better in 4 out of the 16 simulations. Fig. 4 shows how the different techniques behaved in the different scenarios. Each scenario is normalized by the technique that finished more jobs, which is represented by the white color. This figure evidences that WRAS in its two variations performed better that any other approach in general, with a near best performance when it did not have the best performance.

The better performance of WRAS might be because when a mobile device becomes idle, it offloads work from the worst ranked mobile device. Since the mobile device that steals a job is idle, it is likely that its rank is much higher than the worst ranked device. Therefore, the stealer's rank would be slightly affected, while the victim's rank might be more affected. As a result of the stealing, rank dispersion will be reduced, while average rank might increase in the best case. A better rank implies that is less likely that the mobile devices have many jobs when their batteries become depleted. In contrast, BRAS did not

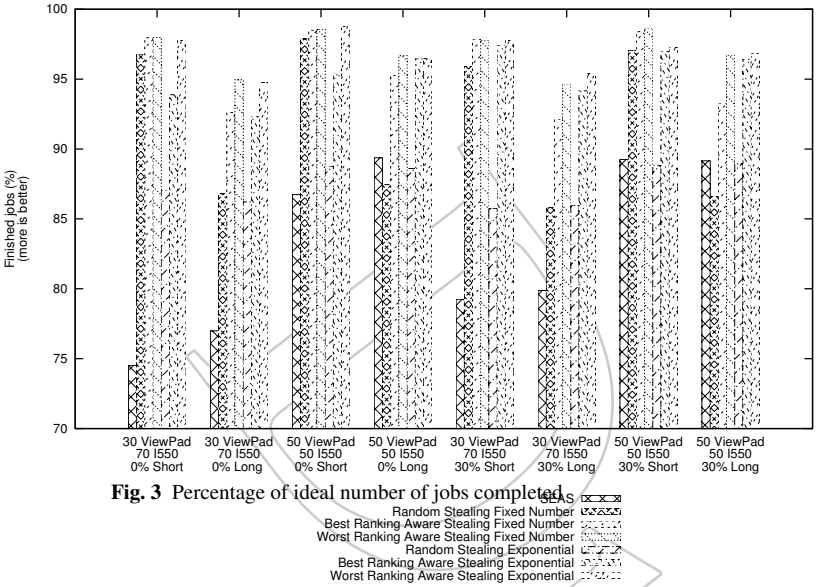


**Fig. 3** Percentage of ideal number of jobs completed

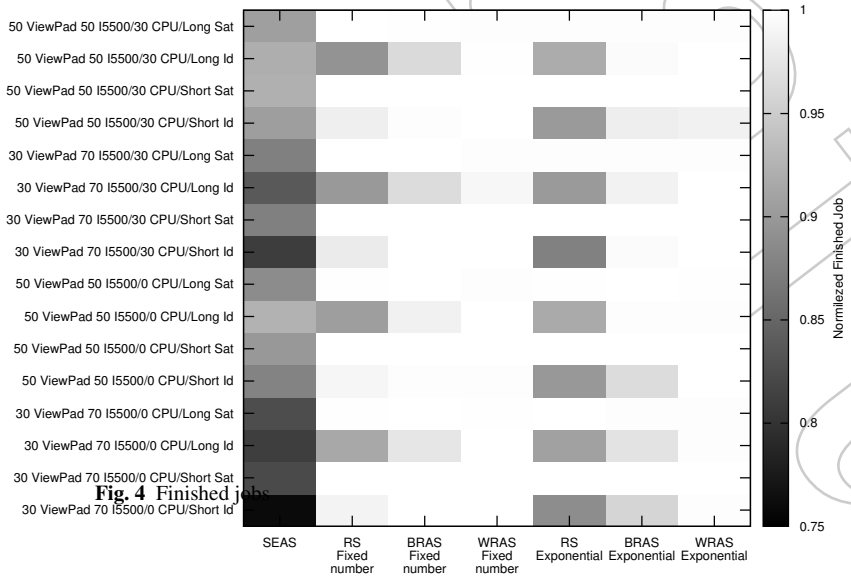


**Fig. 4** Finished jobs

| Configuration | RS Max | BRAS Max | WRAS Max | RS Exp | BRAS Exp | WRAS Exp |
|---|---|---|---|---|---|---|
| 30 ViewPad 70 I550 / 0% / Short Id | 68.1% +/- 2.2% | 31.9% +/- 0.8% | 29% +/- 0.2% | 173.9% +/- 14.5% | 59.4% +/- 6.5% | 32.5% +/- 0.1% |
| 30 ViewPad 70 I550 / 0% / Short Sat | 15.5% +/- 0.2% | 7.1% +/- 0% | 7.1% +/- 0% | 28.8% +/- 4.1% | 11% +/- 0.1% | 10.9% +/- 0.2% |
| 30 ViewPad 70 I550 / 0% / Long Id | 129% +/- 8.5% | 81.8% +/- 4.4% | 62.7% +/- 1.1% | 153.5% +/- 8.5% | 51.2% +/- 3.5% | 34.8% +/- 0.6% |
| 30 ViewPad 70 I550 / 0% / Long Sat | 30.5% +/- 1.8% | 10.7% +/- 0.2% | 10.7% +/- 0.2% | 30.5% +/- 2.3% | 11.4% +/- 0.3% | 11.5% +/- 0.5% |
| 50 ViewPad 50 I550 / 0% / Short Id | 35.9% +/- 1.5% | 17.9% +/- 0.4% | 15.4% +/- 0.1% | 72.7% +/- 5.7% | 33% +/- 2.6% | 16.4% +/- 0.2% |
| 50 ViewPad 50 I550 / 0% / Short Sat | 8.7% +/- 0.4% | 4.5% +/- 0.1% | 4.4% +/- 0.1% | 12.2% +/- 1.3% | 4.9% +/- 0.1% | 4.8% +/- 0.2% |
| 50 ViewPad 50 I550 / 0% / Long Id | 104.8% +/- 4.5% | 62.8% +/- 9.9% | 36.4% +/- 0.5% | 62.9% +/- 4.2% | 30.9% +/- 0.9% | 21.6% +/- 0.7% |
| 50 ViewPad 50 I550 / 0% / Long Sat | 22.8% +/- 0.8% | 15.9% +/- 0.1% | 15.9% +/- 0.1% | 15.7% +/- 1.3% | 7.5% +/- 0.4% | 7.3% +/- 0.4% |
| 30 ViewPad 70 I550 / 30% / Short Id | 63.1% +/- 2.4% | 30.3% +/- 1% | 24.6% +/- 0.3% | 157.1% +/- 16.4% | 30.1% +/- 1% | 24.2% +/- 0.2% |
| 30 ViewPad 70 I550 / 30% / Short Sat | 14.3% +/- 0.5% | 6% +/- 0.2% | 6% +/- 0.1% | 25% +/- 5.2% | 7.8% +/- 0% | 7.8% +/- 0% |
| 30 ViewPad 70 I550 / 30% / Long Id | 178.2% +/- 6.7% | 95.8% +/- 8% | 58.5% +/- 1.2% | 110.3% +/- 12.3% | 50.8% +/- 2.7% | 36.4% +/- 0.6% |
| 30 ViewPad 70 I550 / 30% / Long Sat | 36.6% +/- 2% | 15.7% +/- 0.4% | 15.6% +/- 0.4% | 24.5% +/- 2.1% | 10.5% +/- 0.1% | 10.4% +/- 0.1% |
| 50 ViewPad 50 I550 / 30% / Short Id | 36.6% +/- 1.9% | 21.8% +/- 0.6% | 14.4% +/- 0.1% | 61.4% +/- 6.3% | 24.1% +/- 2.4% | 14.6% +/- 0.1% |
| 50 ViewPad 50 I550 / 30% / Short Sat | 9.2% +/- 0.4% | 4% +/- 0.1% | 4% +/- 0.1% | 9.5% +/- 0.7% | 5.8% +/- 0.1% | 5.8% +/- 0.1% |
| 50 ViewPad 50 I550 / 30% / Long Id | 122.5% +/- 10.7% | 48.3% +/- 5.3% | 30.6% +/- 0.6% | 51.1% +/- 2.7% | 23.5% +/- 1.8% | 15.6% +/- 0.6% |
| 50 ViewPad 50 I550 / 30% / Long Sat | 29.3% +/- 1.9% | 22.3% +/- 0.3% | 20.5% +/- 0.3% | 14.6% +/- 1.5% | 8% +/- 0% | 8% +/- 0.1% |

**Table 5** Percentage of stolen jobs (average $\pm$ stDev)

work as well as WRAS did, but better than the original SEAS and RS though. This might happen because stealing chain-reaction from the best ranked node does not manifest as fast as expected. As a result of this, worst ranked nodes are never offloaded which leads to a lot of job execution failures.

Fig. 5 presents more evidence that the WRAS strategy was the job stealing technique that had the best performance on average. Basically, this figure depicts how many less jobs on average a scheduler was able to finish when compared with the technique that performed better. For instance, the value for SEAS is 13.36% meaning that on average, SEAS finished 89.36% of the jobs that the best scheduler for that scenario finished.

Another analysis consisted in determining the number of jobs transferred by the different job stealing techniques. This is important because this kind of action requires network usage. This, in practice, might have a negative impact on battery consumption [25, 22, 24, 45]. Therefore, we analyzed the number of steals per job in the simulation. Notice that we only consider node-to-node transfers, which are generated by the stealing algorithm, and we do not consider proxy-to-node transfers generated by jobs arrival. This decision allowed us to eliminate noise and accurately measure the performance of the stealing algorithms in respect to job steals. Fig. 6 illustrates the number of steals per job in each simulation. In

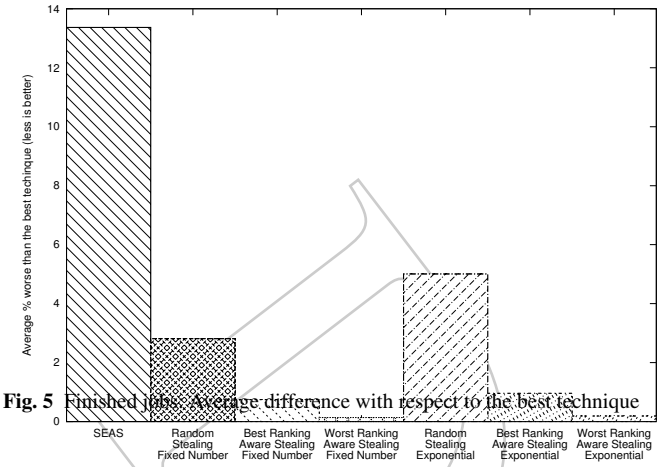


**Fig. 5** Finished jobs. Average difference with respect to the best technique

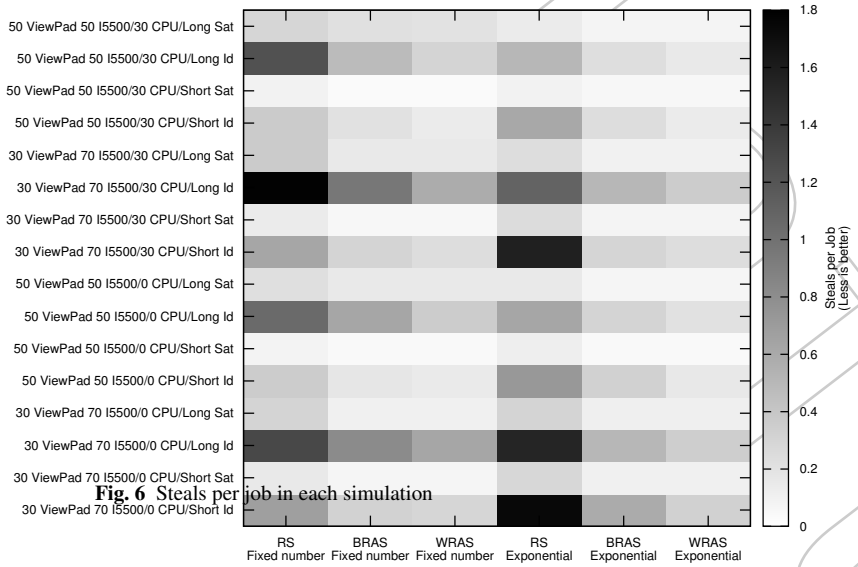


**Fig. 6** Steals per job in each simulation

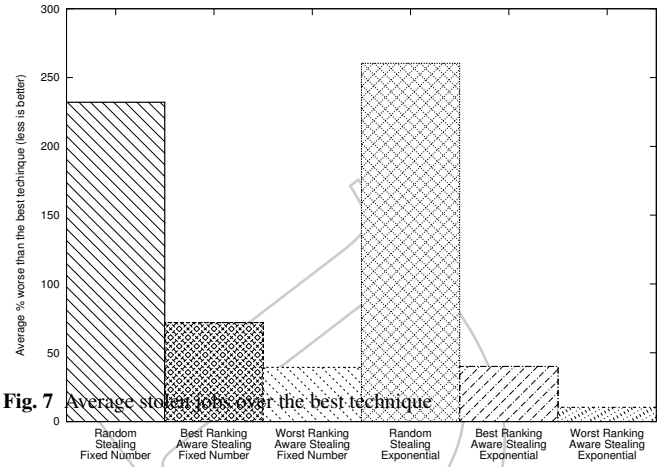


**Fig. 7** Average stolen jobs over the best technique

the worst case, the number of steals that were produced in the simulations where more than two times the number of jobs. In contrast, the number of steals per job in the best scenario was 0.039.

Again, WRAS in its two variants was the strategy that achieved the best results. WRAS with the Fixed Number policy performed a steal, on average, 0.222 times per job. In this case, the worst scenario was 0.627 steals per job, while the best scenario was 0.040 steals per job. In contrast, WRAS with the Exponential policy performed better having an average of 0.164 steals per job, while its worst and best scenarios had 0.364 steals per job and 0.048 steals per job, respectively. Finally, Fig. 7 depicts how many jobs, in percentage, over the best scheduler, each particular scheduler stole on average. For example, the value for WRAS with Exponential policy was 10.54% meaning that it performed 10.45% more steals than that the best scheduler in an average scenario. Notice that in some scenarios that percentage is zero because WRAS with Exponential policy was the best scheduler for that scenario.

In these simulations, the BRAS also performed worse than the WRAS. This might stem from the fact that BRAS was expected to generate a stealing chain-reaction. Although this chain-reaction was not fast enough to finish more jobs than WRAS, it is large enough to issue more job steals. In brief, the BRAS strategy generates more steals than WRAS, but not enough to outperform it with respect to finished jobs. Finally, RS finished fewer jobs requiring a large amount of stealings. RS is a computationally costless alternative to the other techniques because it performs well on traditional distributed environments. However, according to our simulations, energy constrained distributed systems are very sensitive to schedule decisions meaning that non-energy aware scheduling techniques are likely to result in low energy efficiency.

Regarding to the policy, these simulations show that the Exponential policy worked better than the Fixed Number policy for BRAS and WRAS, but not for RS. This might be because the Exponential policy tends to offload overloaded nodes very quickly, while increasing the load of underloaded ones. However, when this policy is used randomly, it is likely that the offloaded node is not the best or worst producing more steals. As a result, the offloaded node will produce another steal increasing the number of steals without really

| Configuration | SEAS | RS Fixed | | BRAS Fixed | | WRAS Fixed | | RS Exp | | BRAS Exp | | WRAS Exp | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F.J. | F.J. | S.J. | F.J. | S.J. | F.J. | S.J. | F.J. | S.J. | F.J. | S.J. | F.J. | S.J. |
| 30 ViewPad 70 I550 / 0% / Short Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 30 ViewPad 70 I550 / 0% / Short Sat | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| 30 ViewPad 70 I550 / 0% / Long Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| 30 ViewPad 70 I550 / 0% / Long Sat | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 50 ViewPad 50 I550 / 0% / Short Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| 50 ViewPad 50 I550 / 0% / Short Sat | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 50 ViewPad 50 I550 / 0% / Long Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| 50 ViewPad 50 I550 / 0% / Long Sat | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| 30 ViewPad 70 I550 / 30% / Short Id | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ |
| 30 ViewPad 70 I550 / 30% / Short Sat | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 30 ViewPad 70 I550 / 30% / Long Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| 30 ViewPad 70 I550 / 30% / Long Sat | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| 50 ViewPad 50 I550 / 30% / Short Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 50 ViewPad 50 I550 / 30% / Short Sat | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 50 ViewPad 50 I550 / 30% / Long Id | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| 50 ViewPad 50 I550 / 30% / Long Sat | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Total best performance | 0 | 1 | 0 | 5 | 1 | 5 | 7 | 1 | 0 | 0 | 1 | 4 | 7 |
| Total best for both metrics | N/A | 0 | | 0 | | 3 | | 0 | | 0 | | 2 | |

F.J.: Finished jobs
S.J.: Stolen jobs

**Table 6** Experimental scenarios: Summarized results

balancing the mobile Grid load. Briefly, the WRAS strategy with Exponential policy was the best combination in most of the studied scenarios and it performed fairly well in the other scenarios.

Table 6 shows which technique had the best performance in the simulations for each scenario. This table takes into consideration two metrics, namely finished jobs and stolen jobs. Firstly, it can be seen that the schedulers using job stealing techniques always outperformed the scheduler without job stealing, i.e., the SEAS. Secondly, RS in conjunction with either policies only finished more jobs than other combinations two times and the former always stole more jobs than the other combinations, which is undesirable. Regarding finished jobs, the best techniques were BRAS and WRAS with the Fixed number policy. On the other hand, regarding stolen jobs, WRAS in its two variants performed better. However, when both metrics are simultaneously taken into consideration, WRAS with Fixed number in particular,

and more importantly WRAS in general worked better (23 out of 32 scenario-metric combination). Besides, in contrast with what happens in traditional distributed environments [51, 47,49], RS performance was rather poor. Finally, regardless the particular scenario, the simulations using job stealing techniques always reported better results than the simulations using only the SEAS regardless the Grid conformation or the size and number of jobs to be solved.

## 5 Conclusions

Several authors [13,14,22,24,26,44,27,45] have recognized that mobile devices are the new frontier for distributed systems. These authors have proposed using mobile devices in different ways ranging from simple front-end applications for accessing extra-device computational resources [13,24,19] to fully integrated devices [33,12,30]. In this context, this paper presents an analysis of different job stealing strategies when applying them in mobile Grids.

This work results show that job stealing can increase the jobs that a mobile Grid can execute. This means that job stealing improves the energy efficiency of mobile Grids because they can execute more jobs with the same amount of energy. As a corollary, energy is managed more efficiently by using energy aware job stealing. In addition, this work analyzes several job stealing techniques to determine which of them are applicable for mobile Grids. Notice that the analyzed job stealing techniques can be implemented on mobile devices because they do not need information that is complex to be obtained [45], such as job execution time or the battery required for executing jobs [30].

Traditional distributed computing environments using RS for selecting a job stealing victim have proven to be a viable option to CPU-intensive computing with good results [51, 47,49]. However, RS has not had a good performance in mobile Grids according to the experimental results reported in this paper. Although RS executed more works than SEAS in general, RS in its two variants performed, on average, worst than WRAS in its two versions. In addition, RS stole up to 759% more works than the best technique for each scenario, stealing on average 240% more works than the best technique for each scenario. Since each steal requires using network and in turn networking requires energy [22,24], a technique that requires more steals might be less energy efficient that other techniques which require less steals.

In order to better assess how the job stealing process might impact on energy efficiency, we will extend this work to take into account the network usage and its impact on mobile devices batteries. Currently, our simulator assumes that transferring a job from one device to another does not consume energy, which is not the case in real devices. The reason of assuming this in this paper is to move forward towards evaluating the feasibility of job stealing in mobile Grids by considering jobs with rather high CPU processing times but very little bandwidth requirements, which make our results significant. Data intensive scientific applications such as sequence alignment or ray tracing [36], pose new challenges to Grid scheduling systems that must be addressed so as to achieve good energy efficiency. Complementary, apart from network energy consumption, we will also study more realistic networking scenarios taking into account network latency, speed and failure possibility [25]. In addition, one of the most important network-related aspects to cover and study in the future is scenarios with nodes leaving and joining the Grid, which is unpredictable. Certainly, autonomic computing refers to the capability of a distributed platform of self-managing and self-adapting to unpredictable environmental changes, and hence this paradigm is in prin-

ciple a good path towards building more elaborated schedulers capable of coping with this aspect. In fact, autonomic computing is often materialized via bio-inspired algorithms such as Ant Colony Optimization, which have proved to be useful when managing resources in conventional Grids [34, 39]. Then, bio-inspired techniques remain a possible solution to explore to tackle this problem. The support for experimenting with this new scenarios will be added to the current version of our simulation software.

In addition, we will develop a real mobile Grid system that implements both the architecture of Fig. 1 and the different studied job stealing techniques to evaluate them in a real environment. To this end, we are currently developing a job execution mobile Grid middleware for master-worker applications based on Android 2.3 and higher, which is a platform version used by more than half of all active devices[9]. Finally, we will analyze different payment schemes for encouraging mobile device owners to share their mobile device resources [14]. To do this, we are considering adding price negotiation into the job stealing and the SEAS [12].

## References

1. Aron, J.: Harness unused smartphone power for a computing boost. New Scientist **215**(2880), 18 (2012). DOI 10.1016/S0262-4079(12)62255-6. URL http://www.sciencedirect.com/science/article/pii/S0262407912622556
2. Blom, S., Book, M., Gruhn, V., Hrushchak, R., Köhler, A.: Write once, run anywhere - a survey of mobile runtime environments. Grid and Pervasive Computing, International Conference on **0**, 132–137 (2008). DOI 10.1109/GPC.WORKSHOPS.2008.19
3. Blumofe, R., Leiserson, C.: Scheduling multithreaded computations by work stealing. In: Foundations of Computer Science, Annual IEEE Symposium on, vol. 0, pp. 356–368. IEEE Computer Society, Los Alamitos, CA, USA (1994). DOI 10.1109/SFCS.1994.365680
4. Boovaragavan, V., Harinipriya, S., Subramanian, V.R.: Towards real-time (milliseconds) parameter estimation of lithium-ion batteries using reformulated physics-based models. Journal of Power Sources **183**(1), 361 – 365 (2008). DOI 10.1016/j.jpowsour.2008.04.077. URL http://www.sciencedirect.com/science/article/B6TH1-4SFS0MD-4/2/e4746e187e06b4aebb77c9a930f56b7f
5. Buyya, R., Murshed, M.: GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation: Practice and Experience **14**(13), 1175–1220 (2002)
6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011). DOI 10.1002/spe.995
7. Callou, G., Maciel, P., Tavares, E., Andrade, E., Nogueira, B., Araujo, C., Cunha, P.: Energy consumption and execution time estimation of embedded system applications. Microprocessors and Microsystems **35**(4), 426 – 440 (2011). DOI 10.1016/j.micpro.2010.08.006. URL http://www.sciencedirect.com/science/article/pii/S0141933110000529
8. Choi, S., Lee, J., Yu, H., Lee, H.: Replication and checkpoint schemes for task-fault tolerance in campus-wide mobile grid. In: T.h. Kim, H. Adeli, H.s. Cho, O. Gervasi, S.S. Yau, B.H. Kang, J.G. Villalba (eds.) Grid and Distributed Computing, *Communications in Computer and Information Science*, vol. 261, pp. 455–467. Springer Berlin Heidelberg (2011). URL http://dx.doi.org/10.1007/978-3-642-27180-9_56
9. Chu, D.C., Humphrey, M.: Mobile ogsi.net: Grid computing on mobile devices. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID '04, pp. 182–191. IEEE Computer Society, Washington, DC, USA (2004). DOI 10.1109/GRID.2004.44
10. Duan, L., Kubo, T., Sugiyama, K., Huang, J., Hasegawa, T., Walrand, J.: Incentive mechanisms for smartphone collaboration in data acquisition and distributed computing. In: INFOCOM, 2012 Proceedings IEEE, pp. 1701 –1709 (2012). DOI 10.1109/INFCOM.2012.6195541

[9] Platform Versions: http://developer.android.com/resources/dashboard/platform-versions.html

11. Fernando, N., Loke, S.W., Rahayu, W.: Mobile cloud computing: A survey. Future Generation Computer Systems **29**(1), 84 – 106 (2013). DOI 10.1016/j.future.2012.05.023. URL http://www.sciencedirect.com/science/article/pii/S0167739X12001318
12. Ghosh, P., Das, S.K.: Mobility-aware cost-efficient job scheduling for single-class grid jobs in a generic mobile grid architecture. Future Generation Computer Systems **26**(8), 1356 – 1367 (2010). DOI 10.1016/j.future.2009.05.003. URL http://www.sciencedirect.com/science/article/pii/S0167739X09000648
13. González-Castaño, F.J., Vales-Alonso, J., Livny, M., Costa-Montenegro, E., Anido-Rifón, L.: Condor grid computing from mobile handheld devices. SIGMOBILE Mobile Computing and Communications Review **7**(1), 117–126 (2003). DOI 10.1145/881978.882005
14. Gray, J.: Distributed computing economics. Queue **6**(3), 63–68 (2008). DOI 10.1145/1394127.1394131
15. Ham, H.K., Park, Y.B.: Mobile application compatibility test system design for android fragmentation. In: T.h. Kim, H. Adeli, H.k. Kim, H.j. Kang, K.J. Kim, A. Kiumi, B.H. Kang (eds.) Software Engineering, Business Continuity, and Education, *Communications in Computer and Information Science*, vol. 257, pp. 314–320. Springer Berlin Heidelberg (2011)
16. Hu, Y., Yurkovich, S.: Battery cell state-of-charge estimation using linear parameter varying system techniques. Journal of Power Sources **198**(0), 338 – 350 (2012). DOI 10.1016/j.jpowsour.2011.09.058. URL http://www.sciencedirect.com/science/article/pii/S0378775311018295
17. Huang, Y., Venkatasubramanian, N.: Supporting mobile multimedia applications in mapgrid. In: Proceedings of the 2007 international conference on Wireless communications and mobile computing, IWCMC '07, pp. 176–181. ACM, New York, NY, USA (2007). DOI 10.1145/1280940.1280978. URL http://doi.acm.org/10.1145/1280940.1280978
18. Huang, Y., Venkatasubramanian, N., Wang, Y.: MAPGrid: A New Architecture for Empowering Mobile Data Placement in Grid Environments. In: Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, pp. 725 –730 (2007). DOI 10.1109/CCGRID.2007.69
19. Huynh, D., Knezevic, D., Peterson, J., Patera, A.: High-fidelity real-time simulation on deployed platforms. Computers & Fluids **43**(1), 74 – 81 (2011). DOI 10.1016/j.compfluid.2010.07.007. URL http://www.sciencedirect.com/science/article/pii/S0045793010001829
20. Ibrohimovna, M., Groot, S.: Proxy-based fednets for sharing personal services in distributed environments. In: Wireless and Mobile Communications, 2008. ICWMC '08. The Fourth International Conference on, pp. 150 –157 (2008). DOI 10.1109/ICWMC.2008.25
21. Kaushik, A., Vidyarthi, D.P.: A cooperative cell model in computational mobile grid. International Journal of Business Data Communications and Networking **8**, 19–36 (2012). DOI 10.4018/jbdcn.2012010102
22. Kelenyi, I., Nurminen, J.: Energy aspects of peer cooperation measurements with a mobile dht system. In: Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on, pp. 164–168 (2008). DOI 10.1109/ICCW.2008.36
23. Khalaj, A., Lutfiyya, H., Perry, M.: The proxy-based mobile grid. In: Y. Cai, T. Magedanz, M. Li, J. Xia, C. Giannelli, O. Akan, P. Bellavista, J. Cao, F. Dressler, D. Ferrari, M. Gerla, H. Kobayashi, S. Palazzo, S. Sahni, X.S. Shen, M. Stan, J. Xiaohua, A. Zomaya, G. Coulson (eds.) Mobile Wireless Middleware, Operating Systems, and Applications, *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 48, pp. 59–69. Springer Berlin Heidelberg (2010). URL http://dx.doi.org/10.1007/978-3-642-17758-3_5
24. Kumar, K., Lu, Y.H.: Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? Computer **43**(4), 51 –56 (2010). DOI 10.1109/MC.2010.98
25. Lehr, W., McKnight, L.W.: Wireless Internet access: 3G vs. WiFi? Telecommunications Policy **27**, 351–370 (2003)
26. Li, C., Li, L.: Utility-based scheduling for grid computing under constraints of energy budget and deadline. Computer Standards & Interfaces **31**(6), 1131–1142 (2009). DOI 10.1016/j.csi.2008.12.004. URL http://www.sciencedirect.com/science/article/B6TYV-4V70RB2-4/2/65554f30c6e3068ba1697c540f160003
27. Li, C., Li, L.: Energy constrained resource allocation optimization for mobile grids. Journal of Parallel and Distributed Computing **70**(3), 245–258 (2010). DOI 10.1016/j.jpdc.2009.06.003. URL http://www.sciencedirect.com/science/article/B6WKJ-4WKTWWB-1/2/7f0834c24e7b7dd44adae8e22ce49ad5
28. Li, C., Li, L.: Energy efficient resource management in mobile Grid. Mobile Information Systems **6**, 193 –211 (2010). DOI 10.3233/MIS-2010-0099. URL http://iospress.metapress.com/content/3R214MM142481741
29. Li, C., Li, L.: A multi-agent-based model for service-oriented interaction in a mobile grid computing environment. Pervasive and Mobile Computing **7**(2), 270 – 284 (2011). DOI 10.1016/j.pmcj.2010.10.006. URL http://www.sciencedirect.com/science/article/pii/S1574119210001173

30. Li, C., Li, L.: Tradeoffs between energy consumption and qos inÂ mobile grid. The Journal of Supercomputing **55**, 367–399 (2011). URL 10.1007/s11227-009-0330-5
31. Li, Z., Shen, H.: Game-theoretic analysis of cooperation incentive strategies in mobile ad hoc networks. Mobile Computing, IEEE Transactions on **11**(8), 1287 –1303 (2012). DOI 10.1109/TMC.2011.151
32. Lin, C.M., Lin, J.H., Dow, C.R., Wen, C.M.: Benchmark dalvik and native code for android system. In: Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on, pp. 320 –323 (2011). DOI 10.1109/IBICA.2011.85
33. Litke, A., Skoutas, D., Tserpes, K., Varvarigou, T.: Efficient task replication and management for adaptive fault tolerance in mobile grid environments. Future Generation Computer Systems **23**(2), 163 – 178 (2007). DOI 10.1016/j.future.2006.04.014. URL http://www.sciencedirect.com/science/article/pii/S0167739X0600080X
34. Ludwig, S., Moallem, A.: Swarm intelligence approaches for grid load balancing. Journal of Grid Computing **9**(3), 279–301 (2011)
35. Mateos, C., Zunino, A., Campo, M.: On the evaluation of gridification effort and runtime aspects of JGRIM applications. Future Generation Computer Systems **26**(6), 797–819 (2010)
36. Mateos, C., Zunino, A., Hirsch, M., Fernández, M., Campo, M.: A software tool for semi-automatic gridification of resource-intensive java bytecodes and its application to ray tracing and sequence alignment. Advances in Engineering Software **42**(4), 172–186 (2011)
37. Mateos, C., Zunino, A., Trachsel, R., Campo, M.: A Novel Mechanism for Gridification of Compiled Java Applications. Computing and Informatics **30**(6), 1259–1285 (2011)
38. Neary, M.O., Cappello, P.: Advanced eager scheduling for java-based adaptive parallel computing. Concurrency and Computation: Practice and Experience **17**(7-8), 797–819 (2005). DOI 10.1002/cpe.v17:7/8
39. Pacini, E., Mateos, C., García Garino, C.: Schedulers based on ant colony optimization for parameter sweep experiments in distributed environments. In: S. Bhattacharyya, P. Dutta (eds.) Research on Computational Intelligence for Engineering, Science and Business. IGI Global (2012). In Press.
40. Palmer, N., Kemp, R., Kielmann, T., Bal, H.: 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, pp. 1–6. ACM, New York, NY, USA (2009). DOI 10.1145/1514411.1514426
41. Paradiso, J.A., Starner, T.: Energy scavenging for mobile and wireless electronics. IEEE Pervasive Computing **4**(1), 18–27 (2005). DOI 10.1109/MPRV.2005.9
42. Rice, A., Hay, S.: Measuring Mobile Phone Energy Consumption for 802.11 Wireless Networking. Pervasive and Mobile Computing **6**(6), 593–606 (2010). DOI 10.1016/j.pmcj.2010.07.005. URL http://www.sciencedirect.com/science/article/pii/S1574119210000593
43. Rodriguez, J., Mateos, C., Zunino, A.: Are smartphones really useful for scientific computing? Lecture Notes In Computer Science **7547**, 38–47 (2012)
44. Rodriguez, J.M., Zunino, A., Campo, M.: Mobile grid seas: Simple energy-aware scheduler. In: 3rd High-Performance Computing Symposium. 39th JAIIO (2010)
45. Rodriguez, J.M., Zunino, A., Campo, M.: Introducing mobile devices into grid systems: a survey. International Journal of Web and Grid Services **7**(1), 1–40 (2011)
46. Rosado, D.G., Fernández-Medina, E., López, J., Piattini, M.: Systematic design of secure mobile grid systems. Journal of Network and Computer Applications **34**(4), 1168 – 1183 (2011). DOI 10.1016/j.jnca.2011.01.001. URL http://www.sciencedirect.com/science/article/pii/S1084804511000026
47. Rosinha, R.B., Geyer, C.F.R., Vargas, P.K.: WSPE: a peer-to-peer grid programming environment. Concurrency and Computation: Practice and Experience **21**(13), 1709–1724 (2009). DOI 10.1002/cpe.v21:13
48. Shen, W.X., Chan, C.C., Lo, E.W.C., Chau, K.T.: Estimation of battery available capacity under variable discharge currents. Journal of Power Sources **103**(2), 180 – 187 (2002). DOI 10.1016/S0378-7753(01)00840-0. URL http://www.sciencedirect.com/science/article/B6TH1-44V3JXV-2/2/77bf800f9c9901c16d550f67a4a31e6b
49. Van Nieuwpoort, R., Wrzesińska, G., Jacobs, C., Bal, H.: Satin: A high-level and efficient Grid programming model. ACM Transactions on Programming Languages and Systems **32**(3), 9:1–9:39 (2010)
50. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem – overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3), 36:1–36:53 (2008). DOI 10.1145/1347375.1347389
51. Zhang, B.Y., Yang, G.W., Zheng, W.M.: Jcluster: an efficient Java parallel environment on a large-scale heterogeneous cluster. Concurrency and Computation: Practice and Experience **18**(12), 1541–1557 (2006). DOI 10.1002/cpe.v18:12

## APPENDIX A: Simple Energy Aware Scheduler (SEAS)

The SEAS [44] is a scheduling algorithm for mobile Grids that is designed to perform scheduling along with as few estimations, such as device estimated remaining battery, as possible. The SEAS is a centralized scheduler, i.e., all the mobile devices that are considered by the scheduler must be connected to a central server which is called *proxy*. Basically, the proxy receives a job execution request and assigns the job to a mobile device. In order to select a mobile device, it ranks them according to which might assign more resources per job. For a mobile device $m$, this value is calculated as follows:

$$resources\,per\,job_m = \frac{estimated\,uptime_m \times benchmark_m}{number\,jobs_m + 1} \tag{6}$$

where $estimated\,uptime_m$ is the estimated uptime for the mobile device with the remaining battery power, $benchmark_m$ is the value obtained using some benchmark that represents the MIPS (Million Instructions Per Second) the device is able to perform, which in scientific computing might be the Linpack or the SciMark 2.0 [43], and $number\,jobs_m$ represents the number of jobs assigned to that particular device. This function adds one to the number of jobs because it calculates which would be the node rank if a new job is added to it.

The only estimation the SEAS needs is each mobile device remaining uptime. The proposed estimation algorithm is based on the fact that battery APIs are event-based and the battery information is reported as a discrete variable. Examples of this are the iOS[10], Android[11] and ACPI[12] battery APIs. A basic way of estimating remaining uptime with this event system is assuming a lineal discharge rate. Therefore, it is possible to calculate the discharge rate when two consecutive battery events happen. For two events $i-1$ and $i$, the discharge rate $dr$ can be calculated as follows:

$$dr = \frac{c_i - c_{i-1}}{t_i - t_{i-1}} \tag{7}$$

where, $c_i$ and $c_{i-1}$are the battery charge reported by the events $i$ and $i-1$, respectively. $t_i$ and $t_{i-1}$are the times when these events occurred. Therefore, the remaining uptime $ut$ might be estimated as follows:

$$ut = \frac{c_i}{dr} \tag{8}$$

However, the discharge rate is actually not lineal [48] and hence the estimation heavily varies from event to event. Thus, the SEAS uses a modified version of the estimator that returns 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 3 describes how the remaining time is calculated. According to [44], this algorithm tends to work better over time and is suitable for the SEAS purpose.

[10] iOS battery discharge notification: `https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIDevice_Class/Reference/UIDevice.html#//apple_ref/c/data/UIDeviceBatteryStateDidChangeNotification`

[11] Android Battery Intent: `http://developer.android.com/reference/android/content/Intent.html#ACTION_BATTERY_CHANGED`

[12] Advanced Configuration and Power Interface Specification version 5: `http://acpi.info/DOWNLOADS/ACPIspec50.pdf`

**Algorithm 3** SEAS: Enhanced battery time estimation algorithm

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