Battery-aware centralized schedulers for CPU-bound jobs in mobile Grids

Matías Hirsch[a,*], Juan Manuel Rodriguez[a], Alejandro Zunino[a], Cristian Mateos[a]

[a] *ISISTAN-CONICET Research Institute. Campus Universitario, Tandil (B7001BBO). Tel.: +54 (249) 4439682 ext. 35. Fax.: +54 (249) 4439681*

Abstract

Given the relentless growing number of mobile devices, researchers have pointed out that distributed computing environments, such as clusters or even computational Grids, could increase the available resources by scavenging devices' capabilities. However, this negatively impacts on their batteries. We study centralized job schedulers that aim at exploiting clusters of mobile resources with rather stable devices in terms of connection time. These schedulers outperform a previous centralized scheduler, finishing up to 30% more jobs with the same amount of energy. We also show that the schedulers perform competitively compared to schedulers based on job stealing, which are more difficult to implement and consume more network resources and hence energy.

Keywords: Mobile Devices, Mobile Grid, Job Scheduling, SEAS, Job stealing

1. Introduction

Mobile devices have evolved from simple PDAs, with restricted memory and computational capabilities, to small computers able to handle complex computations. Current mobile devices can store several gigabytes of information, include fast multi-core CPUs and a few gigabytes of RAM, and have GPUs capable of rendering complex 3D graphics. In response, some studies [41, 22, 48] have proposed to use mobile devices for processing and visualizing scientific and medical data, while doing the associated heavy computations on fixed servers. This model is known as *offloading* [27]. Other studies [44, 23] have taken a step further and have proposed to perform all the heavy processing in the mobile devices.

Nowadays, mobile devices represent one of the commonest kind of computational devices in the world. This year the number of mobile devices exceeded the world population (7.3 billion)[1]. According to the Android official Web Site, more than one million Android devices are activated each day.

As a result of their capabilities and number, there is a great deal of joint computational power in mobile devices, mostly underused. Therefore, many researchers [12, 46, 16, 47, 30, 2, 45] have pointed out that unused mobile device capabilities can be scavenged for performing heavy computations [44]. To this end, several studies [47, 6, 2] have proposed developing distributed computing environments using mobile devices. This would not only allow mobile device resources to be scavenged, but also make them able to perform computations that a single mobile device could not handle otherwise [47, 25, 31]. In this sense, people carrying mobile devices everywhere with computing capabilities, such as smartphones, has motivated researchers [35] to exploit computation opportunities of local networking structures arising from the social interaction of groups of people.

In this context, two promising research lines in Mobile Computing is integrating mobile devices to traditional cluster/Grid Computing platforms or creating new platforms tailored for mobile devices [35, 47]. Although this presents similar challenges to the ones found in traditional distributed computing platforms, mobile devices pose new issues. Firstly, traditional platforms assume that computing nodes are connected to wired networks, which are fairly reliable, stable, and fast. In contrast, mobile devices are connected to wireless networks, which are less reliable/stable and slower. Another major difference with traditional cluster/Grid nodes is that

*Corresponding author.
Email address: matias.hirsch@isistan.unicen.edu.ar (Matías Hirsch)

[1] http://tinyurl.com/mokcut3

mobile devices rely on batteries. Therefore, a heavy computation executing on a mobile device might drain its battery if the energy consumption is not well managed. Sadly, this is not a trivial problem because accurately estimating the energy required to execute any computation is very challenging [9] when not impossible. Furthermore, in the general case, estimating how long a program would take to run implies solving the Halting problem [59]. Another problem is that mobile devices are not dedicated to the computing environment meaning that users might add unexpected load to the nodes [45].

Network communication itself represents another source of energy waste, which has motivated several works, particularly for wired networks [37, 38, 60]. In wireless networks, this problem is exacerbated by the combination of unreliable/unstable communication channels and the limited energy supply of mobile devices. Despite being convenient for mobility, wireless communication requires large amounts of energy, and different communication patterns have different energy consumption patterns [42]. This motivated researchers to study energy-efficient low level [15, 14, 50] and high level [3, 54] wireless communication protocols.

Although these issues might hinder incorporating mobile devices to computational environments, particularly Grids, there are significant advances in energy-efficient wireless protocols for connecting nodes to servers [33, 1, 56]. Besides, energy-aware programming techniques have arisen, such as battery-aware code refactoring techniques [43]. Still, a main problem is energy optimization at a higher level, since better environment-wide energy usage means more mobile devices up and ready to execute computations. Since different nodes have different computational/energy characteristics, bad scheduling decisions at the platform level are likely to produce energy waste despite intra-node optimizations. Previous works [16, 46, 30, 47, 45] have shown that traditional (energy-agnostic) schedulers are inefficient in this kind of distributed environments in regard to the rate of computation performed per unit of energy. These works have also showed that to increase energy efficiency levels, the schedulers should consider the inherent energy-related properties of the nodes.

In response, there are energy-efficient schedulers targeting data-intensive applications and mobility concerns that exploit wireless links properties [51] or use resource sharing models driven by market rules [17, 16]. Other efforts [30, 29, 13] do not target any specific kind of application and propose utility-based frameworks to coordinate mobile devices resources. These studies encourage mobile device owners to sell their unused computational capabilities in a variable price market. Therefore, the aim of these approaches is at optimizing an utility function that considers different variables, such as energy consumption, finished computation, and economical gain. Lastly, efforts such as [44, 46, 45] aim at CPU-intensive applications, exploiting available CPU cycles considering energy limitations.

In this paper we explore several criteria built upon energy-related and computing capability-related properties of nodes to exploit the aggregated computational resources of mobile devices. The performance of local (cluster-level) schedulers based on the proposed criteria is measured in terms of delivered throughput per energy unit. We assume that mobile devices are intended to solve atomic and independent computations (*jobs*). Furthermore, we target CPU-intensive jobs, which potentially benefits many applications [4, 57, 5, 20, 36].

This work differs from previous efforts, and particularly from [46, 45], in the following aspects:

- We introduce three battery-aware criteria for CPU-intensive applications, presented into a local centralized scheduling approach. These criteria aim at maximizing the amount of finished jobs using the available energy within a cluster of mobile devices.
- An enhanced version of the previously published SEAS criteria [46]. This new version features a more accurate method to obtain estimations of available energy. In case of battery-powered devices, knowing their available energy is necessary for exploiting resources offered by the device.
- A performance comparison of the proposed battery-aware criteria against the job stealing techniques presented in [45]. The comparison, which includes clusters with newer/more powerful hardware (Acer A100 tablet), shows that performance obtained by the centralized schedulers based on the proposed criteria is very competitive. It is worth noting that, even when job stealing schedulers consume more energy than centralized schedulers -due to overhead of stealing messages-, only the energy used to actually perform computations was measured.
- A preliminary study regarding the combination of the above energy-aware criteria with job stealing techniques. In [45], job stealing proved to increase the performance of a specific centralized scheduler based

on SEAS. Motivated by this, here we also evaluate whether the use of job stealing can also boost the performance of the centralized schedulers based on the new battery-aware criteria.

The rest of this paper is organized as follows. First, in Section 2, we discuss the most relevant related efforts. In Section 3, we describe the environment where the proposed centralized schedulers would operate and the proposed energy-aware criteria for job-node assignation. Section 4 describes the simulation tool used to evaluate our proposal, the definition of experimental scenarios and the evaluations of schedulers performance. Particularly, Section 4.2 shows the performance of centralized schedulers based on the new energy-aware criteria, while Section 4.3 evaluates their performance when equipped with job stealing techniques. Finally, Section 5 and Section 6 present the conclusions and future research opportunities, respectively.

2. Background and related work

The role of mobile devices in distributed computing has evolved from simple monitors that allow users to check the status of a cluster or Grid [18] to nodes that contribute to the infrastructure with computational resources [16, 30, 47, 2, 45, 31]. The main cause of this evolution is that mobile devices capabilities have grown exponentially in the last decade [40, 6] resulting in mobile devices able to perform complex computations [44].

Despite their computational capabilities, mobile devices rely on batteries that might become depleted as a result of performing computations [43]. Since different devices have different battery depletion rates, job scheduling in Grids including mobile devices -from now on mobile Grids- has a direct impact on the global energy efficiency level [16, 46, 30, 47]. This has motivated researchers to study different approaches for job scheduling in mobile Grids. The problem of scheduling jobs to mobile devices is in principle bound to the distributed environment topology because the topology imposes restrictions on what information a scheduler can know about the other nodes. Basically, the two alternatives for implementing distributed environments arranging mobile devices are: ad-hoc networks and infrastructure-based networks.

An ad-hoc network shares similarities with P2P networks since the former are also self-organized, operated under decentralized control and have the capability of providing connectivity in highly dynamic environments. Furthermore, ad-hoc networks are usually wireless multi-hop networks where nodes operate as both, end hosts and routers, forwarding packets wirelessly towards other mobile nodes that might not be within the direct transmission range of the sender [10]. These networks present more complex collision and interference situations than single-hop networks. In addition, changes in communication paths caused by mobility of nodes are very common and turn routing protocols into a complex and energy-consuming task. However, ad-hoc networks have been proposed as communication networks in scenarios that deal with a lack of pre-existing networking infrastructures, e.g., in emergency operations after natural disasters, in rescue missions, in military missions as distributed command-and-control system [49, 10, 55, 32] or as an extension of hotspots.

The other alternative to aggregate mobile devices into distributed environments is via infrastructure-based networks, which are characterized by the existence of a fixed backbone. A proxy-based setting is an example of such networking alternative and consists in connecting the mobile devices wirelessly to a special fixed node, called *proxy*, that presents a group of mobile devices as a single virtual resource to the entire Grid [47, 28]. Unlike ad-hoc networks, infrastructure-based networks favor application scenarios which assume more stable mobile environments in terms of devices' owners movement.

Figure 1 depicts a proxy-based mobile Grid infrastructure composed by Fixed Virtual Resources (FVR) and Mobile Virtual Resources (MVR) integrated by fixed computers and mobile devices, respectively. A Virtual Resource is seen as a single node by the global scheduler and differs from a traditional cluster mainly because it could be composed by rather heterogeneous hardware. Notice that an administrative domain could offer more than one Virtual Resource. Offering local resources behind a proxy to a Grid infrastructure is a strategy commonly adopted by traditional Grid platforms like Ibis-Satin [58], JCluster [61] and GridGain[2]. As a consequence, these platforms rely on *two-level schedulers*, in which meta-tasks are first scheduled to proxies by a global scheduler, and then these tasks are locally scheduled among nodes behind a proxy by a local

[2] http://www.gridgain.com

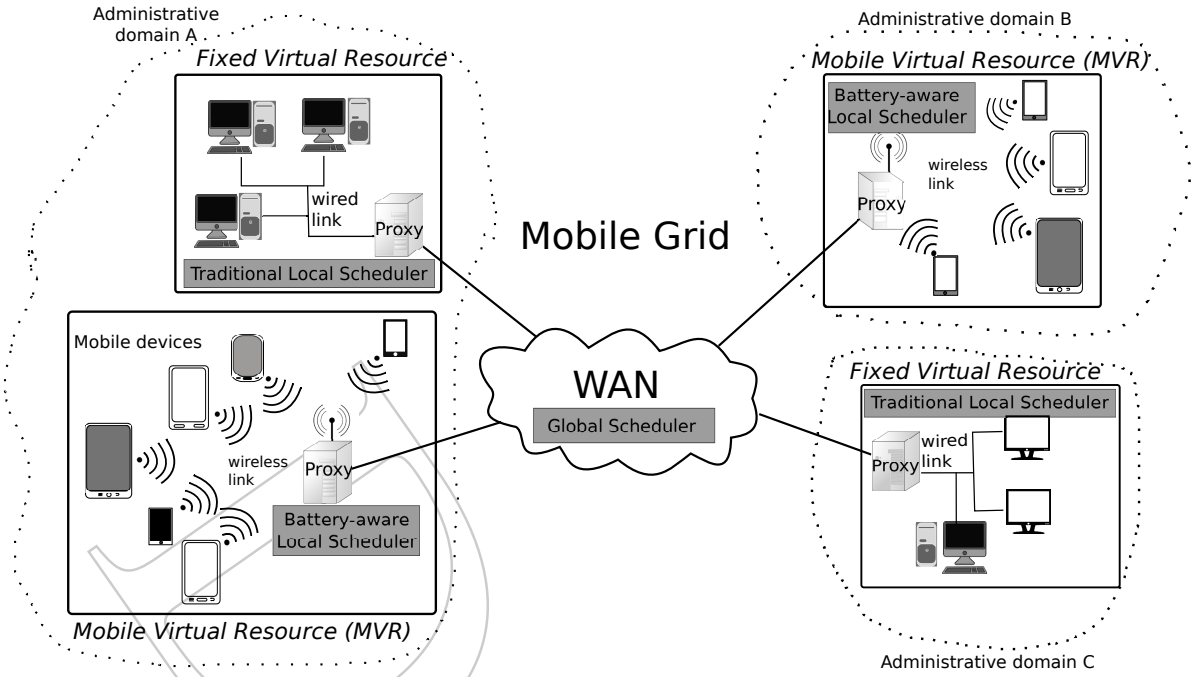


Figure 1: Proxy-based mobile Grid environment

–i.e., cluster-level– scheduler. Due to its simplicity, proxy-based infrastructures have been the starting point for defining different local scheduling policies in mobile Grids [46, 16, 30, 29, 45, 31].

In this sense, in [30, 29, 13], the authors discuss optimal approaches using Lagrange multipliers for scheduling jobs to mobile nodes for execution. These schedulers aim at maximizing an utility function of executing jobs using Lagrange multipliers and are designed for environments where not all jobs have the same importance or gain. This means that nodes with high-rated capabilities and strongly dedicated might put a higher price for executing a job than low-rated or less dedicated nodes. This might be the case in pay-per-use Grids [19]. The schedulers proposed in [30, 29, 13] are hard to implement in real-life Grids [47] because the utility function used requires knowing information that usually is not available, or it cannot be estimated easily. For instance, the energy consumption rate function of a node while executing a job. Firstly, this information depends not only on the consumption inherent to the node hardware, but also on how a job is coded because even small differences in code cause huge differences in energy consumption [43]. Furthermore, precise models to estimate battery consumption are based on complex differential equations [21], and solving them is a complex computational job itself. Another information that these schedulers need to know is the time required to execute a job in a node. Essentially, for predicting this in the general case, it would be necessary to solve the halting problem [59].

Moreover, the schedulers proposed in [17, 16] simulate a market where mobile devices buy and sell computational capabilities. Similarly to the schedulers presented in [30, 29, 13], these schedulers are designed to obtain an optimal utility according to an utility function. This is done by simulating a negotiation process between the proxy that submit jobs and the mobile devices. The schedulers are oriented towards minimizing the used network bandwidth when transfering job information and results, which is done by means of a framework based on LZ78 compression. Since optimizations are performed on the usage of network resources, CPU-intensive applications would hardly benefit from the framework unless they also need to transfer a considerable amount of data.

SEAS (Simple Energy-Aware Schedule) [46] is a local scheduler that aims at minimizing the energy consumed per job executed. SEAS was designed to be easily implemented in real-life environments, using only information available in common battery-based devices. When jobs are assigned to a proxy, it ranks all subordinated mobile devices according to which one can be allocated more jobs. For ranking a mobile device m,

SEAS uses the following estimation function: $resources\ per\ job_m = \frac{estimated\ remaining\ uptime_m * benchmark_m}{number\ jobs_m + 1}$, where $estimated\ remaining\ uptime_m$ is the estimated remaining uptime for the mobile device with the remaining battery power, $benchmark_m$ is the value obtained using some benchmark that represents the FLOPS (Floating-point Operations Per Second) the device is able to perform, and $number\ jobs_m$ represents the number of jobs allocated to that particular device. In scientific computing, the FLOPS can be estimated by means of the well-known Linpack or SciMark 2.0 benchmarks, and [44] shows how these benchmarks can be used to evaluate the computing capabilities of mobile devices.

As discussed above, estimating the remaining uptime is not trivial [21]. However, the original SEAS paper [46] proposes a rather simple technique that was fairly accurate when using notebooks and netbooks. 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 battery discharge notification, Android battery intents and the ACPI battery APIs. Assuming that the discharge rate is linear, for two events $i-1$ and i, the discharge rate can be calculated as $dr = \frac{c_i - c_{i-1}}{t_i - t_{i-1}}$ where, c_i and c_{i-1} are the battery charge levels reported by the events i and $i-1$, respectively. In addition, t_i and t_{i-1} are the times when these events occurred. Therefore, the estimated remaining uptime is $ut = \frac{c_i}{dr}$. Since the discharge rate is not linear [52], the estimation heavily varies from event to event. Thus, the SEAS battery estimator uses a modified version of the estimator that returns an average estimated total uptime minus the current uptime instead of the previously defined remaining time. This average is calculated using all previously estimated total uptime. The estimated total uptime is defined as the current uptime plus the estimated remaining uptime as defined above. Furthermore, the results of [13] places SEAS as the best algorithm in terms of energy consumption ratio. According to that study, SEAS overpasses the second best algorithm in a range of 8.3% and 9.5% in average.

SEAS was extended in [45] by adding the possibility of rebalancing the load among devices through job stealing techniques. The use of job stealing techniques implies that an underloaded node tries to take jobs from overloaded nodes in an attempt to balance the workload across the nodes. Even when SEAS has acceptable performance, it does not always make good scheduling decisions because the information used to schedule jobs still cannot be estimated precisely. By adding job stealing and studying several stealing policies, the negative impact of using uncertain battery information for scheduling can be minimized.

Lastly, ERRA (Energy efficient and Robust Resource Allocation) [51] aims at minimizing the energy consumption in mobile ad-hoc Grids. ERRA is a two-step scheduling scheme providing an energy-aware job allocation focused on node mobility. The first step consists in selecting nodes within an area with the highest probability to stay connected for a longer period. Using a Markov model, the authors maintain a history of movement patterns, which are then used to predict the next probable location of the nodes. The second step, uses a modified kNN (k-Nearest Neighbor) search algorithm to assign weights to the nodes based on the energy and transmission time costs of their links. Finally, the job allocation process uses the probability value of the nodes and the weights of their links to allocate a group of nodes to a job set. Different strategies of job allocations are proposed depending on the data transfer requirements and job interdependency type. All in all, the energy-aware resource allocation technique proposed in [51] is driven by the analysis of cost and availability of links between nodes rather than by the analisys of nodes computing capabilities. That makes the proposal more appropriate for scheduling data-intensive jobs rather than CPU-intensive jobs.

2.1. Classification of related scheduling works

This section presents a classification of related works –including this one– based on classical Grid scheduling categories plus other additional categories. The classical categories, extracted from [26], differentiate scheduling works according to the objective to optimize –single or multi objective–, environment type -static or dynamic-, processing mode -immediate mode or batch mode- and job interrelation -dependent or independent-. We propose additional categories to improve the differentiation of scheduling works for mobile clusters.

On one hand, we found that all related works of Section 2, including our proposal, fall into the same classical categories when they are classified based on their objective to optimize, environment type and processing mode, i.e., all works propose a multi-objective scheduler for a dynamic environment with immediate processing mode. Similarly, since all schedulers tackle at least two objectives, they are multiobjective. One of these objectives is energy, a concern that it is always present because nodes availability in mobile Grids depends on the energy of mobile devices batteries. This fact, in conjunction with connectivity concerns and unpredictable devices

load caused by devices' owner usage, among other facts, is the main reason for categorizing the environment of mobile Grid schedulers as dynamic. Moreover, the immediate mode characteristic is because in all the discussed schedulers, scheduling decisions are triggered with every job submitted to the system, i.e., jobs are not buffered but scheduled to nodes immediately.

Table 1 summarizes the scheduling works. The category of the first column classifies works by the type of *job interrelation* that schedulers are able to handle. While most of the analyzed schedulers handle independent jobs, i.e., bag-of-tasks applications, the scheduler proposed in [51] focuses on interdependent jobs, where relations among jobs are typically presented as directed acyclic graphs (DAG). Nevertheless, bag-of-tasks applications are in practice popular and commonplace in several application domains [39].

The *topology* column refers to the underlying networking structure used to connect and transfer data among the participant mobile devices. The pros and cons of the presented topologies, i.e., Ad-hoc network-based and proxy-based, make them suitable for different situations. In consequence, the selection of the most appropriate topology varies according to the characteristics of the resource allocation scenario. Most of the cited works, including this one, assume a proxy-based topology. On the other hand, [51], the most recent one, has abundant experimentation assuming an Ad-hoc network-based topology.

The *scheduling logic* column refers to how the scheduling process is managed and it has relation with the role assigned to each node of the distributed infrastructure, i.e., administrative and/or resource provider. According to this category, works can be classified into centralized and decentralized. The former suggests the existence of intermediate nodes –known as proxies or mediator nodes– that concentrate information of resource providers within a certain scope (in our proposal a mobile cluster). These intermediate or pure administrative nodes do not contribute to the overall Grid infrastructure with their resources but they coordinate or represent actual resource providers. In addition, worker nodes under the supervision of a proxy node do not cooperate between each other to perform scheduling decisions. For these reasons, the scheduling logic is considered centralized. This is the case of the works by Li & Li [30, 29, 13], where a market based model for resource allocation takes place at the mediator component, called mobile Grid proxy. Gosh & Das' [17, 16] proposals are also centralized because of the existence of a central component called Job Allocator (JA). Similarly, Rodriguez et al. [46] also advocate to centralized scheduling since the proxy makes all scheduling decisions based on the SEAS nodes ranking. In Sayed Shah's proposal [51], nodes of the mobile Grid could play dual roles, i.e., service broker or service provider, and scheduling decisions are divided into two levels for alleviating the processing burden of the scheduling criteria they propose. However, since both decision levels are performed by centralized nodes, the whole scheduling logic is classified as centralized.

On the contrary, decentralized scheduling logic suggests that all nodes of a mobile cluster are capable of both scheduling jobs and executing them. The effectiveness of the whole scheduling process at the cluster level arises from the aggregation of scheduling decisions each node perform within a small node neighborhood. The process could be summarized as follows: when a job is scheduled to one of the nodes, this latter may re-schedule the job to a neighbor if the neighbor is more appropriate for handling the job execution. The re-scheduling process continues until a node can not find a more appropriate neighbor. This type of scheduling logic is decentralized because all nodes play a dual-role, i.e, as job schedulers and as resource providers. This is the case of the other discussed work by Rodriguez et. al. [45], because the use of job stealing techniques implies that all nodes run logic to offload jobs from overloaded nodes, and these, in turn, contain logic to let other nodes steal queued jobs.

The *scheduling goal* column differentiates works by expliciting the objective to be optimized. Notice that this category differs from the classical one presented above -objective to optimize- in that the classical takes the arity of objectives into account rather than the objectives themselves. Moreover, we identified a correspondence between the *scheduling goal* and the *targeted application type* of the analyzed works. For instance, there are works targeting only data-intensive applications –[17, 16, 51]– that pursue the goal of minimizing the job response time, while works targeting CPU-intensive applications –[46, 45] and this work– aim at maximizing the throughput of the mobile cluster. Moreover, works whose targeted application type is not limited–[30, 29, 13]–, have as scheduling goal maximizing an utility function that integrates several objectives, e.g. payments per resource utilization, time and energy budgets, among others.

The column *required knowledge from jobs* differentiates works based on the difficulty to obtain the job features (e.g., execution time, energy consumption, number and size of packets to be transferred, application

Work	Job interrelation [26]	Topology	Scheduling logic	Scheduling goal	Targeted application type	Required knowledge from jobs
Li & Li ([30, 29, 13])	Independent	Proxy-based	Centralized	Maximize the utility of the mobile Grid	Not limited	High
Ghosh & Das ([17, 16])	Independent	Proxy-based	Centralized	Maximize revenue and minimize response time	Data-intensive	High
Sayed Shah ([51])	Dependent	Ad-hoc network-based	Centralized	Minimize response time	Data-intensive	Low
Rodriguez et. al ([46], **this work**)	Independent	Proxy-based	Centralized	Maximize throughput	CPU-intensive	None
Rodriguez et. al ([45])	Independent	Proxy-based	Decentralized	Maximize throughput	CPU-intensive	None

Table 1: Classification of scheduling works for mobile Grids

type, etc) required to feed the scheduling criteria. A value of *high* is assigned when the scheduling logic requires at least one job feature which is hard to estimate or even impossible to determine without executing the job, e.g., the job execution time. Moreover, a value of *low* is assigned when job features are circumscribed to user-provided high level job classifications that gives the scheduler a hint on the predominant type of resource on which job execution relies on, e.g, indicating whether a job is data-intensive or CPU-intensive. Such high level classifications could be provided with minimum effort by a domain expert. Finally, the *none* value characterizes works whose scheduling logic does not require to know any job feature.

From the analysis in this Section, we conclude that the present work is compatible with our previous efforts in terms of the topology assumed. Besides, the most distinctive feature of our proposal compared to works by other authors is that no job-related input is needed by the schedulers to operate. This feature endows our efforts with a special practical value.

3. Battery-aware centralized schedulers: Underpinnings and scheduling criteria

As anticipated at the introduction of this work, typical mobile devices usage patterns open a number of possibilities to exploit their underused capabilities. Moreover, it is increasingly common to see people carrying mobile devices wherever they go, and thus the association of crowds with high computing power sources is becoming commonplace. Crowds are present at innumerable daily life situations, but in this paper, we consider a particular subset of such situations in which mobility and residence time of people are relaxed. Examples of such situations occur in a library or a campus, where students and professors carrying their mobile devices share common places at predefined times. At offices or laboratories, employees spend almost a third part of the day. Also, recreational events are included in the set of such situations. For instance, football/tennis matches or musical concerts bring many spectators together. In all these situations, people usually occupy bounded areas and present motionless behavioral patterns for most of the time they stay at those places. These characteristics allow us to assume that the existence of a fixed networking infrastructure, with wireless access points or special fixed nodes like proxies, is viable if not the most usual form of connection for the participating mobile devices.

By taking advantage of those groups of nearby mobile devices forming different instances of Mobile Virtual Resources, this paper proposes schedulers for local, i.e., intra-MVR scheduling of CPU-bound jobs, leaving global, i.e., inter-MVR scheduling as future work. Within this context, the incorporation of energy concerns in the schedulers is crucial for taking more advantage of the available computing capabilities of nodes because CPU cycles, in mobile devices, are conditioned by batteries power. This has been the design guideline of SEAS [46], which offers performance advantages, in terms of jobs finished per energy unit, over traditional scheduling algorithms like random and round-robin. In a follow-up work [45], SEAS was combined with job stealing techniques. The results of such combination showed that an initial distribution of jobs made with SEAS can be improved by moving jobs from overloaded nodes to underloaded nodes. This load re-balancing based

on job stealing techniques proved to be a good complement to SEAS, since higher percentages of finished jobs were obtained. However, increasing the throughput through job stealing is not a cost-free re-balancing strategy because this decentralized technique requires more communication between nodes compared to centralized scheduling techniques.

In this paper, we aim at achieving at minimum competitive performance against SEAS with the re-balancing technique presented in [45], but maintaining the simplicity inherent to centralized scheduling. To this purpose, we study new battery-aware criteria designed from combinations of benchmarks, manufacturer data and real time information of real mobile devices. In our proposal, a battery-aware criterion is used by a scheduler for ranking nodes and determine which device is better than other for executing a CPU-intensive job.

As stated before, according to the characteristics of the targeted scenarios, the existence of a proxy component is typical. In such topology, proxies are components who play an administrative role solely. They do not offer computing resources by themselves, but through the mobile devices connected to it. A proxy acts as an intermediate between the mobile devices and the rest of the Grid. In this sense, a mobile device that joins a proxy passes through a registration phase to let the proxy know about the new capabilities to be managed [24]. During the registration phase, the proxy can obtain a feature profile associated to the device. In Grid infrastructures the presence of heterogeneous hardware is a rule, so this feature profile helps the proxy to identify and categorize, in terms of computing capability and battery capacity, the device that intends to join the Grid. Our approach explores a feature profile containing a set of specific attributes of a device including benchmarks results and manufacturer data. The former group is composed by the results obtained after running a benchmarking software, e.g., Linpack for Android[3] or SciMark 2.0[4], while the latter involves data about the battery specifications declared by the device manufacturer. Same model devices are supposed to have similar feature profiles. Therefore, instead of generating a feature profile for each device upon proxy registration, all the devices of the same models might be represented under the same feature profile.

Table 2 shows the feature profiles (as columns) for the devices used in the simulations performed to evaluate the analyzed schedulers, which are reported in Section 4.1. The first row of each feature profile shows the device computing capability expressed in Mega FLOPS (Million Float-point Operations per Second). These values represent the average of twenty runs obtained with Linpack for Android[5]. Furthermore, another unit of computing capability is presented by including the type and quantity of a set of different well-known benchmarks that the device could execute before the battery was depleted. This last information is summarized in the third and fourth rows of the table by indicating the total number of benchmarks executed during profiling, and the total elapsed time –expressed in minutes– the profiling took to be completed, respectively [44]. The feature profile also includes information about the battery capacity as informed by the manufacturer. Finally, the last row represents the rate of energy consumption derived from the benchmarks and the battery capacity information.

In addition to the feature profile, the criteria proposed take into account dynamic information. This information is part of the dynamic view of the mobile Grid and includes data reported periodically by a device, e.g., the remaining battery, and statistics maintained by the proxy, e.g., number of jobs assigned to a device and average job execution time in a device. As shown in [46], the combination of static and dynamic information as part of the criteria for scheduling decisions in the proxy-based approach is effective [13]. However, SEAS does not consider data about the relationship between energy consumption and jobs execution, or average of job execution time as part of the scheduling criteria. This paper aims at measuring the impact of including such information in the scheduling process.

The rest of the section is organized as follows. Section 3.1 describes a new battery estimation model for effectively handling tablets and smartphones that improves SEAS, which was designed for notebooks. The new model is adopted by the scheduling criteria proposed in Section 3.2. This last section presents three scheduling criteria that combine static feature profiles and dynamic information. These criteria are the Enhanced SEAS, in its improved version for smartphones and tablets (Section 3.2.1), the Job Energy aware Criterion (JEC)

[3] https://play.google.com/store/apps/details?id=com.greenecomputing.linpack&hl=en

[4] http://math.nist.gov/scimark2/about.html

[5] https://play.google.com/store/apps/details?id=rs.pedjaapps.Linpack

		Samsumg I5500	ViewSonic ViewPad 10s	Acer Iconia Tab A100
Mega FLOPS		7.60	17.07	61.66
Benchmark	Gaussian random generator	10	42	30
	Prime checker	10	42	30
	Fast Fourier Transform	11	43	30
	Sieve of Eratosthenes	10	43	30
	Towers of Hanoi	10	42	30
# Total Executed Benchmarks		51	212	150
Battery Capacity (μAh)		1,200	3,300	1,530
Job Energy Consumption Rate		23.53	15.57	10.20

Table 2: Feature profiles of mobile devices

(Section 3.2.2) and the Future Work aware Criterion (FWC) (Section 3.2.3).

3.1. The new battery estimation model

This paper introduces a change in regards to how the Original SEAS handles battery depletion. Section 2 explains that SEAS estimation works well when handling the discharging rate of mobile devices, such as netbooks and notebooks. However, for other types of mobile devices equipped with batteries that last longer, particularly tablets and smartphones, this estimation method has an important weakness. Mobile device profiling carried out in the context of this work revealed that it is necessary a considerable amount of time (several minutes or even hours) until the estimated uptime obtained with the Original SEAS battery model [46] becomes accurate.

Since, in an unplugged device, the SOC (State Of Charge)(%) should decrease as the time passes by, the battery model should report a higher SOC at time i than at time $i+1$. However, the SEAS estimation model needs to reach a *converging time* until the model starts to predict the uptime values correctly. Then, values reported before the converging time should be considered as spurious. When the model is applied to estimate the uptime of netbooks or notebooks the converging time is in the order of a few seconds, a minute at the most, but for tablets or smartphones the convergence could take up to several minutes or even hours. For this reason, using this model for scheduling on mobile Grids composed by tablets and smartphones might result in chaotic estimation patterns making SEAS ineffective. Therefore, the estimation model used in the present work has been redefined so as to use the remaining percentage or SOC reported by the operating system via battery events.

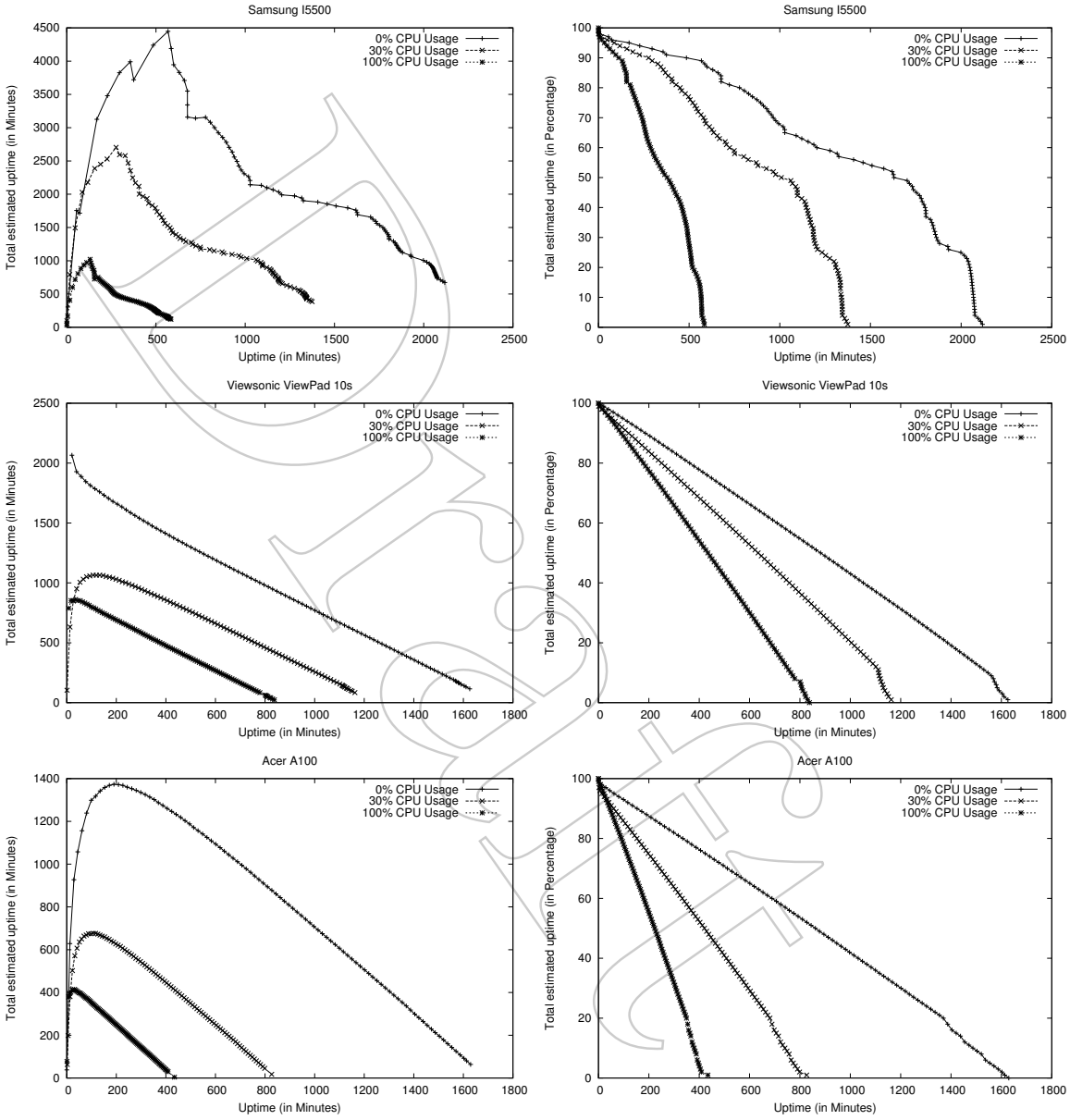
The SOC values, as reported by the OS, represent not only a more accurate and normalized form of estimating the remaining uptime, but it is also independent of the real capacity of the battery, making it applicable to different mobile devices. Figure 2 shows the uptime curves for the Samsumg I5500, the ViewPad 10S and the Acer A100 for different constant CPU usage levels. The figures on the left describe the estimated uptime values using the Original SEAS estimation model, whereas the figures on the right show the values resulting from the SOC values reported by the OS. Notice that with Original SEAS estimation model the maximum estimated uptime is not reported until a certain timestamp, meaning that the model needs all the information occurred at that timestamp to start reporting more accurate values. This phenomenon is not present when the SOC values are used to estimate the remaining uptime of a battery-powered device.

3.2. Proposed criteria for centralized schedulers

This section describes the new studied criteria to schedule jobs in MVRs. Broadly, these criteria allow to rank the devices included in a MVR based on static information provided by their feature profiles and on dynamic information summarized by the proxy or obtained from the managed devices. The main idea is that this rank could be used by a scheduler to select the node where a new job should execute once it arrives to the proxy. From now on, we will use "scheduler" and "criterion" as synonyms.

We propose three criteria that based on the information previously mentioned to exploit the computing capabilities of an MVR in an energy-aware fashion. Section 3.2.1 presents a criterion –called Enhanced SEAS–

Figure 2: Original SEAS uptime model (left) and the SOC based uptime model (right) for various devices



with the SOC component is based on theSOC values. Then, Section 3.2.2 describes the JEC, which bases its nodes ranking on a combination of a static metric of our own called Job Energy Consumption Rate and dynamic information. Finally, Section 3.2.3 describes the FWC, which is a criterion purely based on dynamic information. In addition to the number of assigned jobs and percentage of remaining battery, the FWC uses historic job execution data of devices.

3.2.1. The Enhanced SEAS

The node rank proposed by the Enhanced SEAS criterion is built upon the combination of three components, including the estimated remaining uptime of the node, the computation capabilities of the node measured in Mega FLOPS (Mega FLOPS row in Table 2) and the jobs the node has in its jobs queue. The remaining uptime of a node is obtained from SOC values, i.e, battery events, reported by the OS, which is more appropriate for the type of mobile devices considered in this work. Related to this last component, as mentioned in Section 3.1, due to the lack of accuracy in the battery estimation model of the Original SEAS when targeting hand-held devices, the Enhanced SEAS considers the remaining battery instead of the estimated uptime:

$$nodeRank = \frac{SOC * flops}{allocatedJobs + 1}$$

where *SOC* represents the estimated percentage of remaining battery charge, $flops$ is equivalent to the *benchmarkm* component of the Original SEAS formula and $allocatedJobs$ refers to the old *number jobsm* component.

3.2.2. The Job Energy aware Criterion (JEC)

The node rank in this criterion takes into account the relationship between the energy used and the number of jobs finished by a device. The values of these variables are extracted from the device feature profile. Energy used refers to the battery capacity in milliamperes as reported by the device manufacturer (Battery Capacity row from Table 2) while finished jobs refers to the number of benchmarks from [44] that a device is able to finish before the depletion of its battery occurs (# Total Benchmarks row from Table 2). The quotient of energy used by finished jobs gives the rate of energy consumption per job, i.e., $jobEnergyConsumptionRate = \frac{BatteryCapacity}{\#TotalBenchmarks}$.

Based on this, the ranking formula of this criterion is presented below:

$$nodeRank = \frac{SOC}{jobEnergyConsumptionRate} \times \frac{1}{allocatedJobs + 1}$$

This criterion combines dynamic information (*SOC* and $allocatedJobs$), with static information represented by the $jobEnergyConsumptionRate$ value. For simplicity, if two devices A and B have equal *SOC* and $allocatedJobs$ values, but device B has lesser $jobEnergyConsumptionRate$ than device A, then the criterion assigns a higher *nodeRank* value to device B understanding that this device will better take advantage of its energy to execute the job.

3.2.3. The Future Work aware Criterion (FWC)

This criterion, unlike JEC, bases its decisions on dynamic data. It considers that the future computational power of a node could be estimated by analyzing the computational power the node presented in the past. In other words, FWC assumes that the throughput achieved by a node in the past could be maintained in the future as well. This criterion is materialized via the formula presented below. Like the two previous criteria presented, *SOC* is obtained through the battery events periodically reported by a device, and $allocatedJobs$ is the number of jobs the proxy has scheduled to each node of the MVR. Moreover, the $averageTimeToCompleteJobs$ value is calculated by the proxy and represents the average time a node takes to complete each scheduled job. All in all, the ranking formula is as follows:

$$nodeRank = \frac{SOC}{averageTimeToCompleteJobs} \times \frac{1}{allocatedJobs + 1}$$

The left quotient of the formula penalizes higher average job execution times, while considers remaining battery. Since multiple job allocations could take place until the values of the left quotient are updated –because it is information partially derived from the events reported by the nodes after a 1% battery level drop–, the right quotient is included in the formula, which avoids overloading a particular node. When rank values are determined using this criterion, again, supposing two devices under equal conditions of remaining battery and assigned jobs, the one with lesser *averageTimeToCompleteJobs* value would be assigned a higher *nodeRank* value and consequently the probability of being selected for processing a new job will be higher as well.

A singular characteristic of this criterion is the cold start effect. While schedulers based on Enhanced SEAS and JEC can, in principle, start to operate effectively from the first allocations they do, FWC based schedulers usually need some time to achieve effectiveness because the criterion needs historic data to calculate the *averageTimeToCompleteJobs* value. In the meantime, if no jobs have finished in a node when its nodeRank is calculated, and there is a job currently executing in the node, then the *averageTimeToCompleteJobs* is approximated to twice the time the job has spent executing in the node so far. Otherwise, if no active jobs are executing in the node, then "1" is returned as *averageTimeToCompleteJobs* to avoid invalidating the left quotient of the formula by incorrectly associating the node a high rank.

4. Evaluation of the proposed scheduling criteria

To evaluate our scheduling criteria, we have simulated different instances of Mobile Virtual Resources. In this context, simulation in distributed computing is an accepted, well-known practice [7, 8, 45] because it reduces evaluation times and provides repeatable experiments. This means that different scheduling techniques can be quickly and fairly evaluated in the same environment. In Section 4.1, we present a description of the models used to simulate Mobile Virtual Resources and the different evaluation scenarios. Then, the simulated performance results of the proposed criteria as centralized scheduling scheme are reported in Section 4.2. Furthermore, in Section 4.3, the performance of our criteria is also evaluated in the context of the scheduling schemes explored in [45], this is, against and as part of a decentralized scheduling scheme.

4.1. Simulation conditions

We employed a software (*simulator*) based on a discrete event based simulation model [45]. The model uses events to model everything that might occur in a Mobile Virtual Resource (e.g., job arrivals, CPU usage changes, battery notifications, or job terminations). Therefore, to model nodes at different CPU usages, we first sampled information from real mobile devices, i.e., the battery depletion events under several pre-defined constant CPU usages. As a result, we obtained two different sets of events for each mobile device: battery events, i.e., the exact time a battery level drop was registered, and events aimed at accurately simulating CPU usage. To do this, we have caught CPU events at predefined times when sampling the battery consumption at different target CPU usages. This was done to deal with built-in software, e.g., the Linux kernel or the Android platform itself, that can prevent the CPU from keeping at a constant usage level.

Table 3 outlines the results in regard to the target CPU usage. Firstly, CPU usage is above 0, even when the Target CPU Usage is 0%. This is since Android bundled applications can periodically perform some operations, such as fetching e-mails or looking for software updates. On the other hand, the Android platform has a very active role on application life-cycle, which also require CPU time. Despite these facts, the standard deviation is acceptable in all cases, indicating that the dispersion of the measurements is small.

The simulator also required a set of jobs with their associated arrival time and floating-point operations. To simulate the execution of a job in a mobile device, the simulator uses two *usage profiles* of the same mobile device: the base usage profile –i.e., the default CPU consumption by Android itself and the mobile device owner– and the 100% CPU usage profile. The base usage profile models the situation of a mobile device when it is neither running any Grid job or user applications (i.e., 0%), or when the user is interacting with the device (i.e., 30%, which is a reasonable CPU usage under normal device usage). The 100% CPU usage profile is loaded when the mobile device is running a job. However, it does not mean the 100% of the CPU is dedicated to perform the job since the potential user-injected usage (base usage profile) is also considered as a fraction of this 100% CPU usage. We assume that each job is optimized and correctly coded to use as much CPU cores as available (running a job produces a CPU usage of 100%).

Device	Target CPU usage	Total Sampling Time (hh:mm:ss)	# of Samples (events)	Average CPU usage	Standard deviation
Samsung I5500	0%	35:23:08.021	7,030	3.83%	1.01%
	30%	15:37:12.229	3,466	30.06%	1.44%
	100%	09:45:25.627	2,065	99.98%	0.04%
ViewSonic ViewPad 10s	0%	26:55:05.166	5,469	10.24%	2.25%
	30%	19:28:40.890	4,151	30.10%	2.01%
	100%	13:50:03.965	2,949	99.99%	0.03%
Acer A100	0%	27:08:50.288	4,877	2.02%	1.49%
	30%	13:52:21.776	2,922	30.23%	1.50%
	100%	7:16:31.294	1,556	99.97%	0.06%

Table 3: CPU usage sampling results

When a simulated MVR starts up, all nodes load and synchronize their configured base usage profiles against a central clock maintained by the simulator. This synchronization is also performed every time a node swaps its base usage profile by its 100% usage profile and vice versa, which happens when a job starts and finishes, respectively (Figure 3). All in all, when a job arrives, the simulator schedules it to a selected node through the configured scheduling criterion. Then, the node proceeds with the execution of the job, which includes three steps:

1. Calculating the time T the job will spend executing in the nodebased on the size of the job (given in Mega FLOP – Million Float-point Operations) and the available Mega FLOPS of the node excluding the non-available Mega FLOPS given by its base usage profile.
2. Switching the associated base usage profile to the 100% usage profile –which is maintained during T–, and synchronize against the central clock.
3. After reaching T, raising a job finished event, and switching back to the base usage profile while synchronizing against the clock again.

Moreover, to design the evaluation scenarios, we considered the variables described below and summarized in Table 4:

- Mobile Virtual Resource (MVR) instance: the type and quantity of devices participating in the simulation.
- CPU user %: the base usage profile, or the average percentage of the CPU usage dedicated to perform user/OS activities only.
- Job size: the computational requirement of a job measured in Mega Float point Operations (MFLOP). Unlike the unit used to measure the computing capability of a device (Mega Float point Operations per Second), MFLOP is hardware independent.
- Job arriving rate: the amount of work per second arriving to the proxy, which is expressed in Million Float Operations To be Executed per Second (FLOTES).

All MVR instances contain 100 devices and combine 2 and 3 device models. The notation to indicate an MVR instance is $devicesGroup_1$-$devicesGroup_2$[-$devicesGroup_3$], where *devicesGroup* is *numberOfDevices* + *DeviceModel*. For example, "30A100-70I5500" refers to an instance composed by 30 Acer Iconia Tab A100 devices plus 70 Samsung I5500 devices.

With regard to CPU_User_% variable, it takes one of two values: 0% (fully dedicated device), and 30% (the device is able to give jobs up to a 70% of its CPU capability). To ensure a controlled experiment, all devices integrating a MVR instance run under the same base usage profile.

The two job sizes are *long jobs* and *short jobs*, varying between 19,393.65 and 59,180.95 MFLOP and between 3,232.27 and 9,696.82 MFLOP, respectively. Both job sets were generated following a continuous uniform distribution in those ranges to introduce high heterogeneity in the quantity of jobs with different sizes.

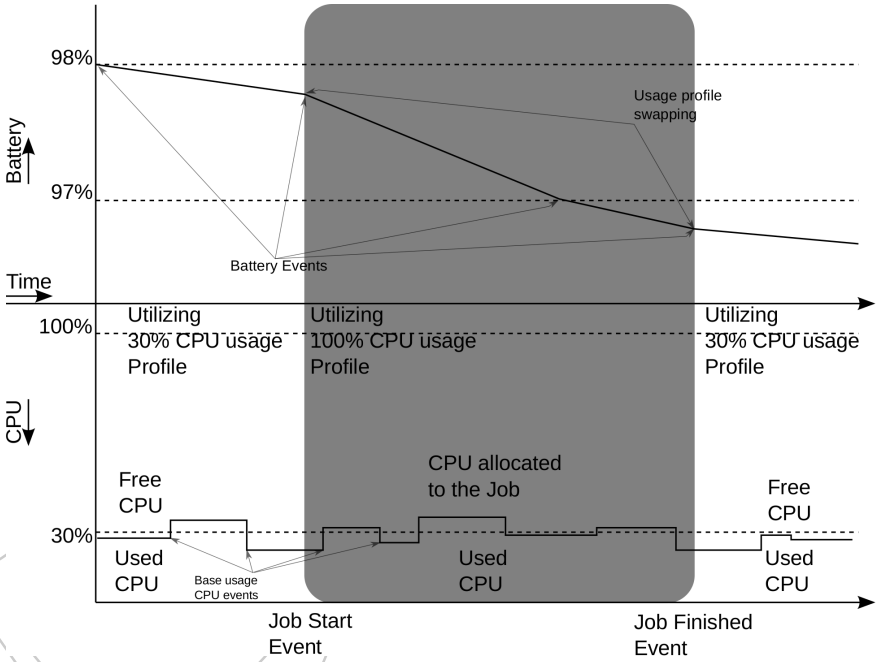


Figure 3: Usage profile swapping when simulating the execution of jobs in a node

Variable	Values			
MVR instance	30A100-70I5500	30ViewPad-70I5500	50ViewPad-50I5500	30A100-30ViewPad-40I5500
CPU User %	0	30		
Job size	short ([3,232.27 - 9,696.82] MFLOP)		long ([19,393.65 - 59,180.95] MFLOP)	
Job arriving rate (FLOTES)	$2.7x10^{11}$	$1.3x10^{12}$	$4x10^{12}$	

Table 4: Variables and values of the simulated scenarios

Moreover, the total amount of long jobs (2,306) and short jobs (13,822) in combination with the job sizes were chosen according to the highest computing capability among all MVR instances considered, so as to avoid reaching a hundred percent of finished jobs in all scenarios. The total amount of MFLOP represented by long and short jobs is approximately the same, which facilitates comparing the performance of different schedulers according to the same total amount of computation.

In respect to the Job arriving rates, we set up a time fraction in which the whole set of short jobs or long jobs arrives to the proxy. The time fractions were 5 minutes, 70 seconds and 20 seconds, resulting in $2.7x10^{11}$, $1.3x10^{12}$ and $4x10^{12}$ FLOTES respectively.

Finally, the scenarios simulated arise from the combinations between all values for all variables, i.e., a total of 48 scenarios exercised for each of the schedulers in Section 3.2. These scenarios share several characteristics with those presented in [45]. Specifically, they share a subset of the devices (Samsung I5500 and ViewSonic ViewPad 10s), some MVR instances, the values of CPU user percentage and the idea of handling long and short jobs. This allow us to compare the schedulers proposed in this paper with the ones introduced in [45].

4.2. Battery-aware criteria as part of a centralized scheduling

Before describing the results of the experiments, it is worth noting that we measure the scheduling performance as the amount of finished jobs per energy unit. In consequence, the selected metric to compare the schedulers was the percentage of finished jobs, so that the more finished jobs a scheduler achieves, the more efficient the energy usage is.

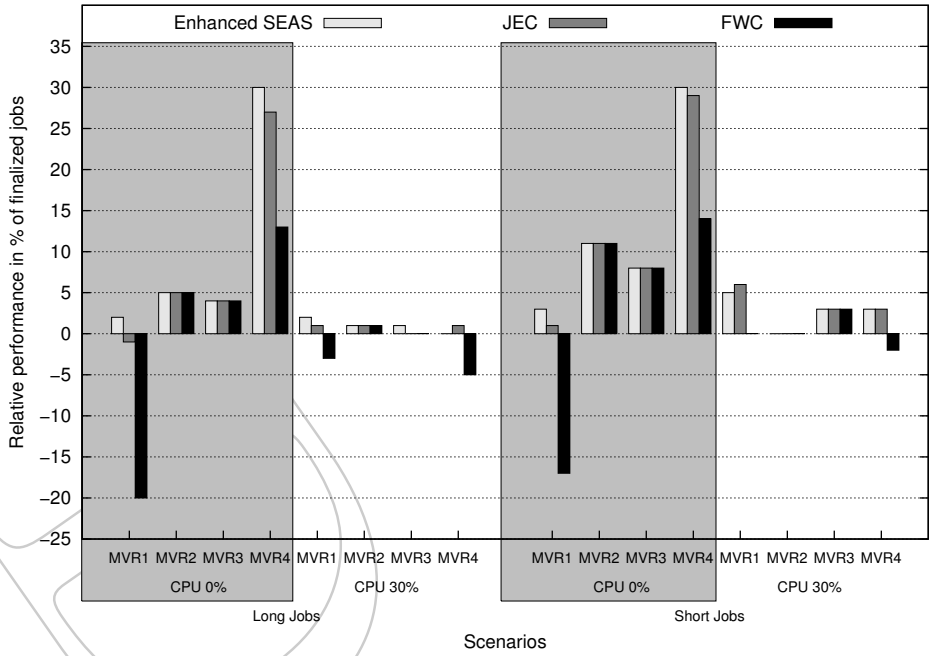


Figure 4: Relative performance of the proposed schedulers against the Original SEAS scheduler for job arriving rate = $4x10^{12}$FLOTES

The simulation results of the proposed criteria exploited from centralized schedulers are presented using two kinds of histograms. On one hand, the histograms of Figures 4, 5 and 8 show the relative performance of centralized schedulers based on the battery-aware criteria proposed, contrasted with the original version of the SEAS scheduler [46]. On the other hand, the histograms of Figures 6, 7 and 9 show the absolute performance values achieved by all the tested scheduling criteria, including the Original SEAS, for all the scenarios. This last type of histogram is used to show the causes of variations in the relative values.

All histograms show the simulation results for all scenarios associated to a particular job arriving rate. For clarity, different MVR instances are denoted as MVR1, MVR2, MVR3 and MVR4 indicating 30A100-70I5500, 30ViewPad-70I5500, 50ViewPad-50I5500 and 30A100-30ViewPad-40I5500, respectively. Furthermore, the left half of each histogram contains the values related to *long job* scenarios while the right half the values for *short job* scenarios. Additionally, scenarios for the base CPU usage of 0% are shown with a gray background, while the ones associated to the base CPU usage around 30% are presented with a white background. From now on, we will refer as "baseline" to the Original SEAS performance.

Regarding the collected results, Figure 4 shows the relative performance of the scheduling criteria compared to the Original SEAS performance when the job arriving rate is $4x10^{12}$ FLOTES. For all scenarios, the Enhanced SEAS outperforms the Original SEAS. Particularly, the improvement is more evident for scenarios with 0% CPU usage than for those with 30% CPU base usage profile. The JEC scheduler is the second best criterion. Its relative performance values are quite similar to those observed for the Enhanced SEAS except for the scenario that combines Long Job with 0% CPU usage and MVR1, where its achieved percentage of finished jobs is 1.4% bellow the baseline. With respect to the FWC scheduler, on one hand, it performs similar to the Enhanced SEAS and the JEC scheduler for scenarios where the Acer A100 (the most powerful device) is not part of the MVR instance. On the other hand, in MVR instances composed by the mentioned device (MVR1 and MVR4), the performance of FWC is not as good as the Enhanced SEAS or the JEC scheduler performance and, in some cases, it is lower than that of the baseline. This behavior may be caused by a chain of multiple factors. First, considering the formula on which the FWC criterion bases its scheduling decisions, the lack of historic information, i.e., the time a node spends completing jobs (cold-start), makes the FWC scheduler ranks all nodes with similar values. In consequence, the different processing power capabilities of nodes are ignored

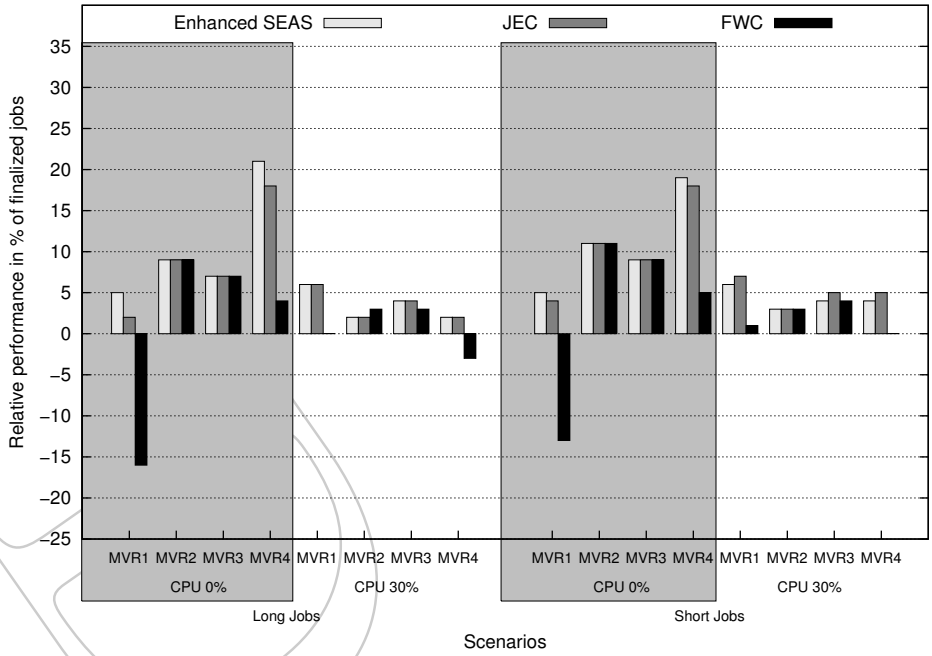


Figure 5: Relative performance of the proposed schedulers against the Original SEAS scheduler for job arriving rate = $1.3x10^{12}$FLOTES

causing that powerful and weak nodes are allocated with very similar workloads. Thus, given a predefined MVR instance, this consequence aggravates with higher job arriving rates and higher job sizes.

Figure 5 shows the relative performance of scheduling criteria when jobs arrive to the proxy at a slower rate, i.e., $1.3x10^{12}$FLOTES. Again, it can be observed that the relative performance of the Enhanced SEAS outperforms the baseline for all scenarios. Furthermore, the JEC scheduler also performs better than the baseline in all scenarios, obtaining, in most of the cases, similar values to the Enhanced SEAS. Moreover, the behavior of the FWC scheduler is similar to the one depicted in Figure 4. This is, FWC achieves competitive performance values for those scenarios not including the Acer A100 device as part of the MVR instance. For the scenarios including the Acer A100 (MVR1 and MVR4) the performance was below the average. However, in comparison with the fastest arrival rate ($4x10^{12}$ FLOTES), the performance values with this slower arrival rate represent smaller losses (or bigger gains depending on the scenario). This reinforces our previous statement about cold start of the FWC scheduler.

In general, the comparison of the relative performance values of Figure 5 and Figure 4 shows that the improvements of the proposed schedulers increase in 40 out of 48 cases. Moreover, 5 out of the remaining 8 cases experience decrements in the relative performance, while for the rest (3 cases), variations were negligible. The cause of these variations are explained by comparing the absolute performance values of Figure 6 and Figure 7. Notice that improvements are mainly caused by decrements of the baseline performance values. Similarly, the cases with decreased performance values –5 out of 6 cases of the scenario with 0% base usage profile and MVR4 instance– are mostly caused by variations of the Original SEAS values rather than of the other schedulers. A steady performance regardless the scenarios variations is desirable because this means that the criteria are slightly affected by variations in the execution conditions, and thus their applicability is higher.

Finally, Figure 8 shows the relative performance of the schedulers when jobs arrive at $2.7x10^{11}$ FLOTES. Despite obtaining lesser gain margins than those depicted in Figure 5 and Figure 4, the Enhanced SEAS and JEC outperformed the baseline in 31 out of 32 cases. Moreover, the FWC scheduler maintains a competitive relative performance for all scenarios except for most of the scenarios that include MVR1 and MVR4 instances. However, after analyzing the scenarios including MVR1 and MVR4 ordered from faster to slower job arriving rates presented until now, we observed that the relative performance of the FWC scheduler slightly improves as

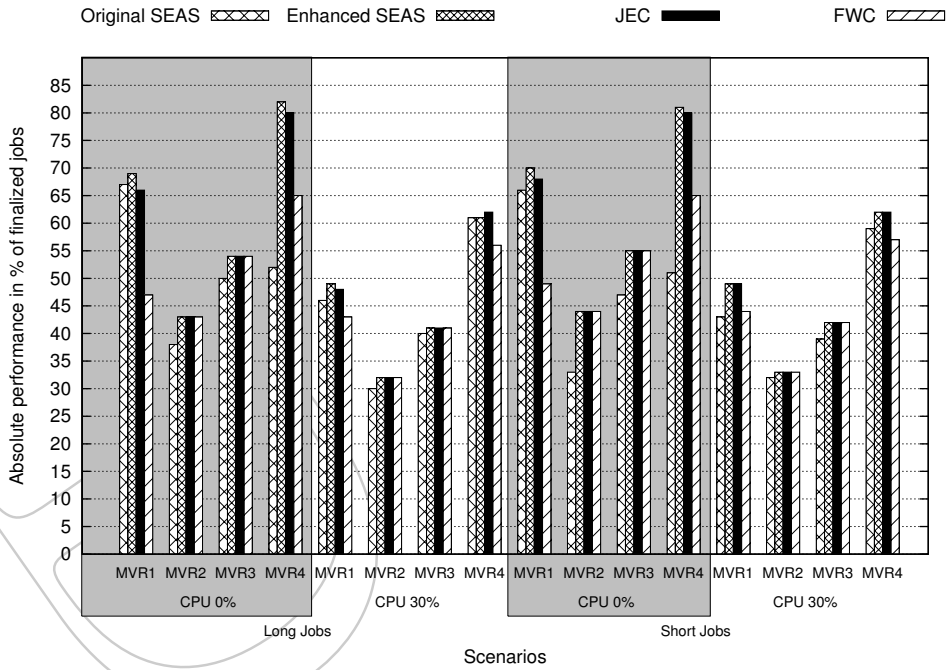


Figure 6: Absolute performance of all schedulers for job arriving rate = $4x10^{12}$ FLOTES

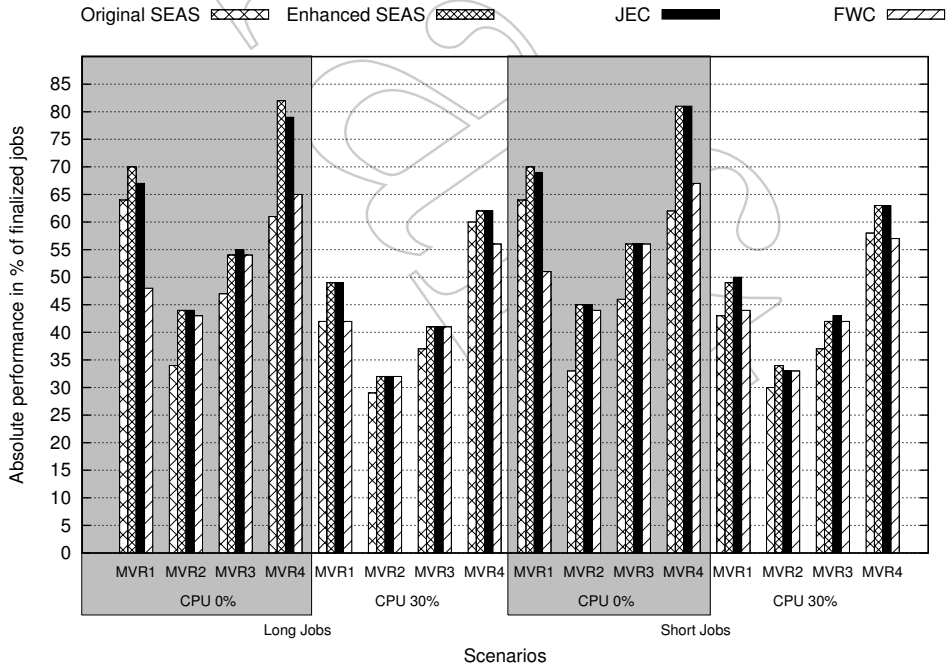


Figure 7: Absolute performance of all schedulers for job arriving rate = $1.3x10^{12}$ FLOTES

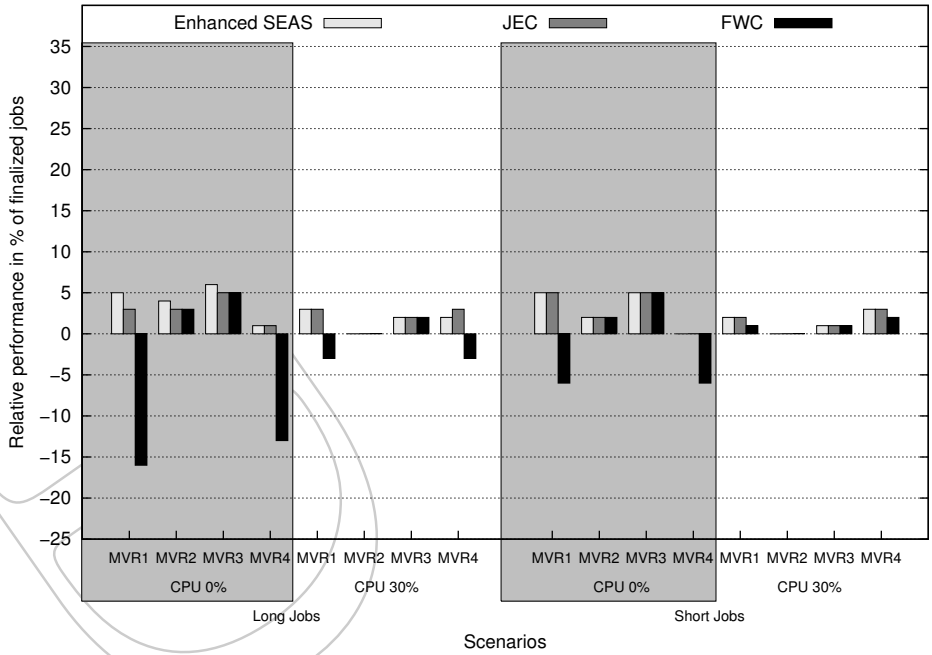


Figure 8: Relative performance of the proposed schedulers against the Original SEAS scheduler for job arriving rate = $2.7x10^{11}$FLOTES

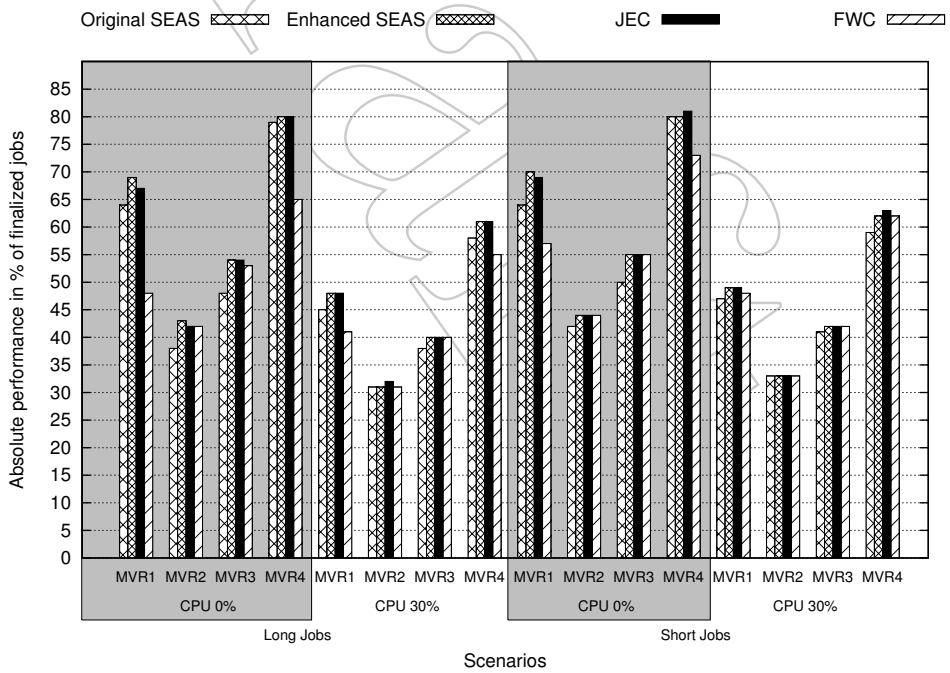


Figure 9: Absolute performance of all schedulers for job arriving rate = $2.7x10^{11}$ FLOTES

the rate becomes slower in 5 out of 8 cases. This evidences how the cold start affects the performance of the scheduler and how its performance varies according to the factors mentioned in the explanation of Figure 4.

The comparison of scenarios with job arriving rates of $2.7x10^{11}$ and $1.3x10^{12}$FLOTES show that with the slower rate the relative performance of all schedulers decreases. By analyzing the absolute performance values we see that, again, variations are mostly influenced by bigger changes registered by the performance values of the Original SEAS than those registered by its competitors. Unlike the last comparison of absolute values, the changes of the Original SEAS are mainly increments on the percentage of finished jobs. However, these increments were not enough to surpass the values achieved by its competitors for the majority of the scenarios.

All in all, the rate at which jobs arrive introduced slight variations on the percentage of finished jobs that the schedulers could complete. Although a deeper analysis could be done by having more values of the job arriving rate variable, we could say that the Original SEAS is more sensible to the rate at which jobs arrive to the Mobile Virtual Resource than the new proposed schedulers, and this sensitivity behaves in an unpredictable form. In addition, its overall performance was always surpassed by the other schedulers. More explicitly, in 97.91% of the experimental scenarios the Enhanced SEAS and the JEC scheduler performed better than the Original SEAS, while the FWC scheduler performed better in 68.75% of the scenarios. Table 5 discriminates the scenarios where each scheduler outperforms the baseline. Furthermore, when analyzing the performance values we found that, in 31 out of the 48 scenarios, the performance gain of the Enhanced SEAS is in between 3% and 30.6%. In 30 out of the 48 scenarios, the JEC scheduler performance gain is in between 3% and 29.6%. With respect to the FWC scheduler, its performance gain is in between 3% and 14.8% in 20 out of the 48 scenarios.

4.3. Evaluation of battery-aware criteria versus/as part of decentralized scheduling

In the previous section, we evaluated the proposed battery-aware criteria and the Original SEAS, all of them, in the context of a centralized scheduling scheme. This section presents an evaluation against a decentralized scheduling scheme. This evaluation is divided into two parts. Section 4.3.1 compares the best battery-aware criteria –i.e., Enhanced SEAS and JEC– in the context of a centralized scheduling scheme, against the Original SEAS as part of a decentralized scheduling scheme, i.e., combined with job stealing techniques [45]. As mentioned at the end of Section 2, the work in [45] proposed and successfully evaluated the combination of the Original SEAS with different job stealing techniques to mitigate sub-optimal scheduling decisions made by the former and produced as a consequence of uncertain battery information.

In Section 4.3.2, we appeal to the same decentralized scheduling scheme as a way of evaluating how much the scheduling decisions made by the battery-aware schedulers can be improved. In other words, the results described show the potential synergy between the proposed battery-aware criteria and job stealing techniques.

4.3.1. Battery-aware criteria in a centralized scheduling scheme versus the Original SEAS combined with job stealing

Before showing the resulting performance assessment of the best battery-aware criteria used in the context of a centralized scheduling scheme versus the schedulers in [45], we present the most relevant concepts of job stealing and the configurations extracted from [45] that are used for the comparison. Job stealing implies that underloaded nodes try to take jobs from overloaded nodes in an attempt to balance the workload across the nodes of the execution environment. When a node (the stealer) attempts to take jobs from another node (the victim), a victim selection strategy is applied. This selection is made upon a node ranking of all nodes of the mobile environment which is built following some criterion. Besides, once the stealer selects the victim, an offloading policy is applied to determine the number of jobs that will be stolen. In summary, a job stealing configuration comprises a criterion to rank the nodes in the mobile environment, a strategy to select *victim* nodes, and an offloading policy.

The options explored in [45] for victim selection were the Best Ranking Aware Stealing (BRAS), the Worst Ranking Aware Stealing (WRAS) and the random strategies. The difference between BRAS and WRAS is that the former aims at selecting the first node of the ranking, i.e., the best ranked node, while the second strategy selects the last node of the ranking, i.e., the worst ranked node. For example, if the criterion ranks the nodes in ascending order according to the remaining battery level, then BRAS/WRAS will select the node having the most/least remaining battery as victim. With a random strategy, as the name indicates, the node selection is performed randomly. Furthermore, the options explored referring to the offloading policy were Fixed and

Table 5: Centralized schedulers performance outline per scenario

Job Arriving Rate	Job Size	CPU user %	MVR instance	Original SEAS	Enhanced SEAS		JEC		FWC	
				Best	Best	Better	Best	Better	Best	Better
$4x10^{12}$ FLOTES	long	0	MVR1	✗	✓	✓	✗	✗	✗	✗
			MVR2	✗	✓	✓	✗	✓	✗	✓
			MVR3	✗	✓	✓	✗	✓	✗	✓
			MVR4	✗	✓	✓	✗	✓	✗	✓
		30	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✓	✓	✓	✓	✗	✓
			MVR3	✗	✓	✓	✗	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
	short	0	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✗	✓	✗	✓	✓	✓
			MVR3	✗	✗	✓	✗	✓	✓	✓
			MVR4	✗	✓	✓	✗	✓	✗	✓
		30	MVR1	✗	✗	✓	✓	✓	✗	✓
			MVR2	✗	✗	✓	✗	✓	✓	✓
			MVR3	✗	✗	✓	✗	✓	✓	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
$1.3x10_{12}$ FLOTES	long	0	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✓	✓	✗	✓	✗	✓
			MVR3	✗	✗	✓	✓	✓	✗	✓
			MVR4	✗	✓	✓	✗	✓	✗	✓
		30	MVR1	✗	✓	✓	✗	✓	✗	✓
			MVR2	✗	✗	✓	✗	✓	✓	✓
			MVR3	✗	✓	✓	✗	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
	short	0	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✓	✓	✗	✓	✗	✓
			MVR3	✗	✗	✓	✓	✓	✗	✓
			MVR4	✗	✓	✓	✗	✓	✗	✓
		30	MVR1	✗	✗	✓	✓	✓	✗	✓
			MVR2	✗	✓	✓	✗	✓	✗	✓
			MVR3	✗	✗	✓	✓	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
$2.7x10^{11}$ FLOTES	long	0%	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✓	✓	✗	✓	✗	✓
			MVR3	✗	✓	✓	✗	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
		30%	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✗	✓	✓	✓	✗	✓
			MVR3	✗	✓	✓	✗	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
	short	0%	MVR1	✗	✓	✓	✗	✓	✗	✗
			MVR2	✗	✗	✓	✗	✓	✓	✓
			MVR3	✗	✓	✓	✗	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✗
		30%	MVR1	✗	✗	✓	✓	✓	✗	✓
			MVR2	✗	✗	✗	✓	✓	✗	✓
			MVR3	✗	✗	✓	✓	✓	✗	✓
			MVR4	✗	✗	✓	✓	✓	✗	✓
# of total scenarios (best scheduler)				0	25		18		6	
# of total scenarios (better performance than the Original SEAS)						47		47		33

Scenario			Centralized		Decentralized			Centralized gain
Job size	CPU user %	MVR instance	Enhanced SEAS	JEC	JS_BF_SEAS	JS_WE_SEAS	JS_WF_SEAS	
long	0	MVR1	69.90	66.31	70.21	67.52	67.95	-0.30
		MVR2	43.54	43.32	43.50	39.72	38.46	0.04
		MVR3	54.64	54.60	54.60	51.17	51.26	0.04
		MVR4	82.05	80.01	85.91	52.25	51.17	-3.86
	30	MVR1	49.00	48.53	48.83	46.18	46.49	0.17
		MVR2	32.48	32.48	32.22	31.53	30.92	0.26
		MVR3	41.20	41.07	41.63	40.59	40.29	-0.43
		MVR4	61.49	62.58	62.27	61.27	60.71	0.30
short	0	MVR1	70.27	68.71	68.56	66.72	66.88	1.71
		MVR2	44.72	44.80	44.73	33.97	33.58	0.07
		MVR3	55.73	55.72	55.83	47.89	47.92	-0.10
		MVR4	81.62	80.58	86.78	52.52	51.48	-5.17
	30	MVR1	49.54	49.80	48.57	43.64	43.79	1.22
		MVR2	33.77	33.72	33.62	32.90	32.77	0.15
		MVR3	42.54	42.53	42.66	39.97	39.18	-0.12
		MVR4	62.81	62.96	63.03	59.51	59.72	-0.07

Table 6: Performance comparison of centralized scheduling criteria vs. Job Stealing with job arriving rate of $4x10^{12}$ FLOTES

Exponential. The steal function of the Fixed offloading policy is constant, i.e. -when possible- the same number of jobs is offloaded from the queue of the victim node every time a steal request takes place, while the steal function of the Exponential offloading policy attempts to steal 2^n jobs at each n_{th} steal request.

Now, we present the performance of battery-aware centralized scheduling against the Original SEAS with job stealing techniques. In order to narrow the analysis, we concentrated on the comparison of the best two centralized battery-aware criteria, i.e., Enhanced SEAS and JEC, and the best three job stealing configurations evaluated in [45]. The job stealing configurations are BRAS with Fixed policy (JS_BF), WRAS with Fixed policy (JS_WF) and WRAS with Exponential policy (JS_WE). Then, for instance, JS_BF_SEAS refers to the job stealing configuration of BRAS with Fixed policy and the Original SEAS criterion as node ranker.

Table 6 outlines the percentage of finished jobs for all scenarios exercised in Section 4.2 when jobs arrive to the proxy at $4x10^{12}$ FLOTES. The last column of this table shows the performance difference of the best centralized scheduler with respect to the best job stealing configuration for a particular scenario. Notice that at least one of the centralized schedulers is better than all job stealing configurations in 5 out of 8 scenarios with long jobs and 4 out of 8 scenarios with short jobs. In the remaining seven scenarios, the situation is the other way around, i.e., a job stealing configuration outperforms the percentage of finished jobs of all centralized schedulers. By analyzing the differences for long jobs scenarios we found that, in general, the difference moves in the range of [0.04 - 0.43] percentage of finished jobs while for short job scenarios the range is [0.07 - 0.15]. With exception of 4 out of 16 scenarios, whose differences are in the range of 5.17 and 1.22 percent, the performance values of the centralized and decentralized scheduling schemes are very similar. With respect to those exceptional cases, we see that in scenarios combining CPU user of 0% and the MVR4 instance, the JS_BF_SEAS obtained a clear advantage over the centralized schedulers but this advantage does not hold for CPU user of 30%. Being competitive in scenarios with CPU user above 0% is preferable since these scenarios represent more real mobile devices usage contexts.

Now, we analyze the performance values from Table 7 for scenarios with job arrival rate of $1.3x10^{12}$ FLOTES. Gains in favor to centralized schedulers are presented in 5 out of 8 long job scenarios and in 6 out of 8 short job scenarios. The remaining scenarios favor job stealing. In general, the magnitude of differences have slightly increased for long job scenarios –placed in between 0.13 and 0.95 percent– while barely decreased for short job scenarios –placed in between 0.02 and 0.07 percent–. The odd scenarios, i.e., those with the highest magnitude of gains, are the same to the ones presented in Table 6.

Finally, Table 8 shows the performance values for scenarios with job arrival rate of $2.7x10^{11}$ FLOTES. The

Scenario			Centralized		Decentralized			Centralized gain
Job size	CPU user %	MVR instance	Enhanced SEAS	JEC	JS_BF_SEAS	JS_WE_SEAS	JS_WF_SEAS	
long	0	MVR1	70.64	67.82	71.03	65.57	65.44	-0.39
		MVR2	44.19	44.02	43.58	35.99	34.91	0.61
		MVR3	54.77	55.12	54.16	47.70	47.40	0.95
		MVR4	82.26	79.92	86.95	62.84	62.14	-4.68
	30	MVR1	49.39	49.35	49.26	43.32	42.97	0.13
		MVR2	32.52	32.39	32.35	30.44	30.23	0.17
		MVR3	41.63	41.54	41.07	37.94	37.34	0.56
		MVR4	62.27	62.45	62.62	59.37	59.97	-0.17
short	0	MVR1	70.62	69.30	68.82	64.79	65.02	1.80
		MVR2	45.20	45.18	45.17	34.44	34.03	0.03
		MVR3	56.14	56.22	56.24	47.51	46.88	-0.02
		MVR4	81.36	81.17	87.35	63.54	63.27	-5.99
	30	MVR1	49.97	50.09	49.04	43.08	42.99	1.05
		MVR2	34.05	33.97	33.99	30.86	30.81	0.07
		MVR3	42.91	43.01	42.95	38.63	38.34	0.06
		MVR4	63.25	63.41	63.38	58.76	58.82	0.03

Table 7: Performance comparison of centralized scheduling criteria vs. Job Stealing with job arriving rate of $1.3x10^{12}$ FLOTES

number of long job scenarios where centralized schedulers obtained an advantage versus job stealing is 2 out of 8 while for short job scenarios it is 6 out of 8. Again, discarding the odd cases, in general, the differences are not so high so as to declare a clear winner. For long jobs the magnitudes are placed in between 0.04 and 0.30 while for short jobs are in between 0.02 and 0.11 percent.

To sum up the centralized schedulers outperformed job stealing in 28 out of 48 scenarios. From the remaining 20 scenarios, 19 of them were in favor to job stealing and there is one scenario where both scheduling schemes performed equal. Despite the number of scenarios in favor to one or another scheduling scheme, the magnitudes of gains are mostly around the 0.5 percent, which makes the election of a best scheme for a general case difficult.

4.3.2. Evaluation of battery-aware criteria combined with job stealing

Motivated by the results obtained in [45], where job stealing techniques improved sub-obtimal scheduling decisions made by the Original SEAS, we present a third set of experiments that was designed to measure the extent of improvement that the mentioned techniques might introduce over the proposed battery-aware criteria. These experiments include the Enhanced SEAS criterion, the JEC criterion and the BF job stealing configuration, i.e., the one including the BRAS selection strategy with Fixed offloading policy, which was the most competitive job stealing configuration from the evaluation of section 4.3. The Original SEAS with BF job stealing configuration is also included in this performance analysis and thus a comparison of all BF job stealing performance values could be done. Since the Enhanced SEAS criterion and the JEC criterion present competitive performance, we will refer to them as "battery-aware criteria". As explained at the beginning of Section 4.3, a job stealing configuration includes a criterion to rank the nodes of a mobile environment. The idea of combining a battery-aware scheduler with job stealing is to use the former as the nodes rank criterion. This is the reason why this third set of experiments also represents a measurement of the potential synergy between both scheduling schemes.

After comparing simulation results of battery-aware criteria with and without job stealing, we found that the performance improvement was not as significant as the one registered for the Original SEAS. More specifically, the improvement introduced by the BF job stealing combined with the Enhanced SEAS with respect to the centralized scheme is between -0.65 and 4.36 percent of finished jobs, while the corresponding JEC values are placed between -0.53 and 5.24. In contrast with the Original SEAS values, which are placed in between 0.21 and 35.8 percent of finished jobs, those are small improvement ranges. Cases with values near 0, i.e., registering

Scenario			Centralized		Decentralized			Centralized gain
Job size	CPU user %	MVR instance	Enhanced SEAS	JEC	JS_BF_SEAS	JS_WE_SEAS	JS_WF_SEAS	
long	0	MVR1	69.69	67.95	69.86	64.44	65.00	-0.17
		MVR2	43.37	42.80	43.54	39.72	39.20	-0.17
		MVR3	54.51	54.16	54.25	49.26	49.00	0.26
		MVR4	80.57	80.66	85.60	79.71	79.23	-4.94
	30	MVR1	48.53	48.40	48.83	45.06	45.58	-0.30
		MVR2	31.79	32.05	32.09	31.35	31.44	-0.04
		MVR3	40.89	40.63	40.89	38.68	39.16	0.00
		MVR4	61.49	61.88	61.71	59.11	58.37	0.17
short	0	MVR1	70.24	69.81	68.96	64.37	64.25	1.29
		MVR2	44.81	44.75	44.75	42.87	42.91	0.06
		MVR3	55.95	55.86	55.89	50.68	50.69	0.07
		MVR4	80.89	81.42	86.87	81.12	81.17	-5.45
	30	MVR1	49.64	49.71	47.74	47.24	47.01	1.98
		MVR2	33.68	33.86	33.74	33.69	33.75	0.11
		MVR3	42.66	42.70	42.63	41.74	41.48	0.07
		MVR4	62.99	63.15	63.17	59.99	60.01	-0.02

Table 8: Performance comparison of centralized scheduling criteria vs. Job Stealing with job arriving rate of $2.7x10^{11}$ FLOTES

differences under 1% are considered as tie cases. In other words, considering BF job stealing configuration as a performance booster, i.e., as a correction mechanism for sub-optimal job allocations, the improvement ranges of battery-aware criteria indicates that their allocation decisions are better than the Original SEAS allocation. However, the absolute performance of job stealing combined with battery-aware criteria for all scenarios, are very close to the values of the job stealing combined with the Original SEAS (Section 4.3.1).

In order to decide which of the combinations is the most convenient, we complement the performance analysis by measuring steals count. Steals produce network activity that could lead to waste considerable energy, thus achieving competitive performance with minimum steals requests is desirable. Table 9 shows, for each scenario, the percentage of job transfers that steals represent. Job transfers comprise job transfers initiated by the proxy and job transfers initiated by the nodes, i.e., steals. Each cell contains a percentage range considering the steals of the three job arrival rates. For instance, the value 11.51 means that 11.51% of the job transfers generated in the scenario were caused by steals. The *JS_BF_SEAS* column in Table 9 shows the range of values for the job stealing-BRAS-Fixed Original SEAS combination, while job stealing combinations with battery-aware schedulers are represented by *JS_BF_SEASEn* and *JS_BF_JEC*.

By comparing the JS_BF_SEAS column against the job stealing combinations with the battery-aware schedulers, we see that the steal ranges of the former are considerably greater than the latter. Even more, there are several scenarios where job stealing combined with battery-aware criteria did not generate any steal, meaning that there were no corrections to the mappings performed by the battery-aware criteria.

The job stealing scheme inherently involves costs referring to the energy consumed by message exchanges for making steal requests, eventually moving jobs from one node to another and updating internal structures of nodes. Thus, achieving low percentages of steals is desirable because it contributes to using less energy in scheduling duties and in consequence more energy is available to perform jobs or user-related activities.

5. Conclusions

The results of the set of tests performed in this work allow us to make several conclusions. Firstly, the experimental results from the Enhanced SEAS based centralized scheduler showed that the usage of the criterion presents advantages in performance and predictable behavior when compared with the Original SEAS. More explicitly, the performance of the Enhanced SEAS was better than that of the Original SEAS criterion in 97.91%

Scenario			Percentage of jobs transfers represented by steals		
Job size	CPU user %	MVR instance	JS_BF_SEAS	JS_BF_SEASEn	JS_BF_JEC
long	0	MVR1	[24.39 - 27.80]	[2.25 - 13.92]	[7.13 - 13.67]
		MVR2	[23.29 - 28.96]	0.00	0.00
		MVR3	[19.40 - 26.70]	0.00	0.00
		MVR4	[37.51 - 47.39]	[11.51 - 17.85]	[17.64 - 23.97]
	30	MVR1	[22.77 - 33.45]	[0.00 - 2.58]	0.00
		MVR2	[16.27 - 24.32]	0.00	0.00
		MVR3	[16.30 - 25.13]	0.00	0.00
		MVR4	[17.02 - 26.42]	[3.43 - 4.67]	0.00
short	0	MVR1	[4.15 - 6.92]	[0.21 - 0.35]	[2.70 - 6.52]
		MVR2	[5.02 - 13.03]	0.00	0.00
		MVR3	[7.00 - 11.18]	0.00	0.00
		MVR4	[7.93 - 28.43]	[2.09 - 2.57]	[7.40 - 9.61]
	30	MVR1	[2.59 - 9.02]	0.00	0.00
		MVR2	[0.00 - 6.11]	0.00	0.00
		MVR3	[3.42 - 6.68]	0.00	0.00
		MVR4	[4.95 - 6.62]	[0.37 - 1.47]	0.00

Table 9: Job transfers percentages represented by steals for battery-aware and Original SEAS combined with BRAS Fixed job stealing

of the exercised tests, and these performance values were not as fluctuating as the values obtained by the Original SEAS. Besides, the Enhanced SEAS based scheduler outperformed the rest of centralized battery-aware schedulers in the majority of scenarios, and, for this reason, it qualifies as the best battery-aware criterion.

With regard to the JEC based centralized scheduler, its performance is very competitive w.r.t. the Enhanced SEAS based scheduler, being slightly better in 37.5% of the scenarios and slightly worse or equal for the rest of the scenarios. These results position the JEC based scheduler as the second best battery-aware centralized scheduler. The JEC criterion exploits an alternative way of rating the computing capability of mobile devices, different than FLOPS, which synthesizes completed benchmarks tests with energy consumed into a single value called *job energy consumption rate*. In practice, the calculation of that rate involves collecting benchmarking information from a device, starting with a full charged battery until its depletion. That procedure could be regarded as the main weakness of the criterion, but it can be seen as a good stronghold because periodic calculation contemplates the operation time reduction of battery powered devices caused by the known deterioration problem in Li-ion batteries [53, 11]. More specifically, when rating the computing capability of a mobile device, e.g., through FLOPS, a value independent of the battery lifetime period is obtained. However, rating a mobile device based on its job energy consumption rate offers an up to date value of its real computing capability because potential reductions in its operation time could be reflected through a periodic recalculation of the rate value. It is worth mentioning that for the calculation of the job energy consumption rate of a mobile device, we run as many benchmarking tests as its battery could execute, i.e., from full charge to total depletion. However, an equivalent job energy consumption rate could be calculated by counting the number of benchmarking tests executed within a smaller time window, increasing practicality at the expense of losing accuracy. Since the performance of the JEC based centralized scheduler is barely inferior to the Enhanced SEAS based centralized scheduler, the qualitative feature described previously leads to a trade-off.

Regarding the FWC scheduler, the cold-start effect proved to be rather harmful in highly heterogeneous MVR instances such as those used in the experiments. However, after studying the performance evolution of such scenarios, it was observed that its performance slowly increases as the job arriving rate becomes slower. In any case, given the considerable less overall performance in comparison with the Enhanced SEAS and JEC based schedulers, and its major disadvantage given by the cold-start effect, the FWC scheduler obtained the third position in the rank of the proposed battery-aware centralized schedulers.

Moreover, the performance of the proposed centralized schedulers was compared against different configurations of a decentralized scheduling scheme. The results reveal that the Enhanced SEAS and the JEC criteria

together with a centralized scheduling scheme are very competitive with respect to the best job stealing configurations explored in [45]. In most cases, the absolute values in the percentage of finished jobs differences between both scheduling schemes are in average 0.19 with standard deviation of 0.20. There is a minority of twelve cases with a slight accentuated percentage of finished jobs, i.e., with performance differences between 1% and 6% of finished jobs. Six out of 12 cases were in favor to the centralized schedulers, whereas the remaining cases favored the decentralized scheduling schemes. However, it is worth noting that the only three slight accentuated cases corresponding to scenarios with CPU user of 30% were in favor to centralized schedulers. These cases represent more realistic situations of mobile devices than those representing a dedicated usage, i.e., with a CPU user of 0%.

In a third set of experiments, given the improvements that job stealing techniques introduced to the Original SEAS centralized scheduler [45], we explored the performance achieved from the combination of the best proposed battery-aware criteria with the best job stealing configuration from Section 4.3.1. The results showed that a maximum performance threshold seems to be achieved because values are very similar to those obtained by the job stealing combined with the Original SEAS. However, by analyzing the amount of steals generated by either job stealing combinations, it could be concluded that those including the proposed battery-aware criteria are more efficient since, with considerable less steals than the decentralized scheduler based on the Original SEAS criterion, similar performance values are achieved.

Given the equated performance of battery-aware criteria in the context of centralized and decentralized scheduling, the most convenient option could be selected by considering qualitative features. Specifically, by using job stealing the scheduling logic is spread out in all the nodes of the mobile environment. This involves logic to perform stealing requests, to discover and maintain the connections between nodes, among other activities. This means extra usage of processor and network resources than when adopting a centralized scheme. In centralized scheduling, the scheduling logic is not only simpler, because of the lack of stealing logic, but also could be concentrated in a single fixed node of the mobile cluster, i.e., the proxy which does not rely on battery power to run.

6. Future works

We are extending this work in several directions. Despite the job stealing technique has not introduced a notable performance boost to the scheduling decisions made by the proposed battery-aware centralized schedulers, we plan to evaluate the synergy in the context of other experimental conditions. Some examples are designing scenarios with different and mixed mobile usage patterns, introducing more heterogeneous jobs requirements and including networking activity as a battery consumption factor. With respect to the latter, currently, the simulator [45] 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 centralized job scheduling in Mobile Virtual Resources by considering jobs with rather high CPU processing times but very little bandwidth requirements, which make our results significant, and to fairly compare our work against job scheduling based on stealing. Nevertheless, an accurate network modeling will allow us to study data intensive applications such as sequence alignment or ray tracing [34], that pose new challenges to mobile cluster scheduling systems that must be addressed so as to achieve good energy efficiency.

Another possible extension to this work could be studying new criteria to contemplate not only the maximization of the system throughput but at the same time the minimization of other objectives, such as the response time. Given the benefits of including energy rates derived from benchmarking, this could be done by combining the number of jobs executed with the time employed to execute these jobs. In this way, schedulers focused on improving the response time along with the throughput of the system could be developed. Throughput measurement quantifies the amount of computation a system finishes within a time window, while response time indicates the total accumulated time that a set of jobs spend in a system. This metric, also known as flowtime, is the sum of the finish time minus the start time of each job. While the goal of throughput-oriented schedulers is to maximize finished jobs, flowtime-oriented schedulers aim at minimizing job finish time. Complementary to developing flowtime-oriented schedulers, the simulator used in the experiments will be configured to log timestamps of every state each job passes through, thereby disaggregated information is available to perform a detailed analysis of the data generated in each experiment.

Another possible future work is to adapt the proposed local or intra-MVR scheduling criteria for their use in global scheduling criteria, i.e., at inter-MVR level. While local scheduling criteria are based on single devices capabilities for taking scheduling decisions, global scheduler criteria, instead, would employ MVR capabilities for making the corresponding meta-scheduling decisions. The hypothesis is that considering battery-aware scheduling decisions in global schedulers could help to better exploit Grid environments composed by multiple MVRs. However, this does not mean that the schedulers proposed in this paper can not be readily applicable to real mobile Grids, since as a starting point meta-schedulers combining traditional Grid schedulers at the global level and of our centralized mobile schedulers at the local level could be used.

We also plan to expand and automate the software we use to perform profiles capturing of real mobile devices. Currently, this software collects battery depletion from CPU usage, and is launched and configured manually for each mobile device. The main idea is to profile the energy consumption of other activities such as data transferring and to further automate its configuration and launching. Then, by encouraging colleagues to run that automated software on their mobile devices, we will be able to create and maintain an up to date bank of profiles of different mobile devices not only for advancing on this and other research lines but also to share it with the research community.

To further evaluate our proposal in a real-life environment, we plan to design an evaluation that involves a middleware prototype for running CPU-intensive applications into real clusters of Android mobile devices. For the communication between proxies and mobile devices we plan to employ the push notification mechanism where requests for a given transaction are initiated by a server instead of by the receiver as in the pull mechanism. An alternative for materializing push notifications is through Google Cloud Messaging (GCM) [6]. We will also take into account common issues related to middleware design such as security and fault-tolerance (e.g., use proxy replicas).

Acknowledgments

We thank the reviewers for their comments to improve the paper. We acknowledge the financial support by ANPCyT through grant PICT-2012-0045. The first author acknowledges his Ph.D. fellowship granted by the CONICET.

[1] A. Amokrane, R. Langar, R. Boutaba, G. Pujolle, Energy efficient management framework for multihop tdma-based wireless networks, Computer Networks 62 (0) (2014) 29 – 42.

[2] J. Aron, Harness unused smartphone power for a computing boost, New Scientist 215 (2880) (2012) 18.

[3] M. Arroqui, C. Mateos, C. Machado, A. Zunino, RESTful web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones, Computers and Electronics in Agriculture 87 (0) (2012) 14 – 18.

[4] A. Barisone, F. Bellotti, R. Berta, A. De Gloria, Jsbricks: a suite of microbenchmarks for the evaluation of java as a scientific execution environment, Future Generation Computer Systems 18 (2001) 293–306.

[5] R. Baron, O. Lioubashevski, E. Katz, T. Niazov, I. Willner, Elementary arithmetic operations by enzymes: A model for metabolic pathway based computing, Angewandte Chemie International Edition 45 (2006) 1572–1576.

[6] F. Busching, S. Schildt, L. Wolf, Droidcluster: Towards smartphone cluster computing – the streets are paved with potential computer clusters, in: Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on, 2012, pp. 114–117.

[7] R. Buyya, M. Murshed, 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) (2002) 1175–1220.

[6] https://developers.google.com/cloud-messaging/

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[9] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, P. Cunha, Energy consumption and execution time estimation of embedded system applications, Microprocessors and Microsystems 35 (4) (2011) 426 – 440.

[10] M. C. Castro, A. J. Kassler, C.-F. Chiasserini, C. Casetti, I. Korpeoglu, Peer-to-peer overlay in mobile ad-hoc networks, in: Handbook of Peer-to-Peer networking, Springer, 2010, pp. 1045–1080.

[11] S. S. Choi, H. S. Lim, Factors that affect cycle-life and possible degradation mechanisms of a li-ion cell based on licoo2, Journal of Power Sources 111 (1) (2002) 130 – 136.

[12] D. C. Chu, M. Humphrey, Mobile ogsi.net: Grid computing on mobile devices, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 182–191.

[13] L. Chunlin, L. Layuan, Exploiting composition of mobile devices for maximizing user qos under energy constraints in mobile grid, Information Sciences 279 (0) (2014) 654 – 670.

[14] D. Costa, L. Guedes, F. Vasques, P. Portugal, Effect of frame size on energy consumption in wireless image sensor networks, in: Imaging Systems and Techniques (IST), 2012 IEEE International Conference on, 2012, pp. 239–244.

[15] S. Deng, H. Balakrishnan, Traffic-aware techniques to reduce 3g/lte wireless energy consumption, in: Proceedings of the 8th international conference on Emerging networking experiments and technologies, CoNEXT '12, ACM, New York, NY, USA, 2012, pp. 181–192.

[16] P. Ghosh, S. K. Das, Mobility-aware cost-efficient job scheduling for single-class grid jobs in a generic mobile grid architecture, Future Generation Computer Systems 26 (8) (2010) 1356 – 1367.

[17] P. Ghosh, N. Roy, S. Das, Mobility-aware efficient job scheduling in mobile grids, in: Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, 2007, pp. 701–706.

[18] F. J. González-Castaño, J. Vales-Alonso, M. Livny, E. Costa-Montenegro, L. Anido-Rifón, Condor grid computing from mobile handheld devices, SIGMOBILE Mobile Computing and Communications Review 7 (1) (2003) 117–126.

[19] J. Gray, Distributed computing economics, Queue 6 (3) (2008) 63–68.

[20] D. Hermelin, D. Rawitz, R. Rizzi, S. Vialette, The minimum substring cover problem, Information and Computation/information and Control - IANDC 206 (2008) 1303–1312.

[21] Y. Hu, S. Yurkovich, Battery cell state-of-charge estimation using linear parameter varying system techniques, Journal of Power Sources 198 (0) (2012) 338–350.

[22] D. Huynh, D. Knezevic, J. Peterson, A. Patera, High-fidelity real-time simulation on deployed platforms, Computers & Fluids 43 (1) (2011) 74 – 81.

[23] O. Karan, C. Bayraktar, H. Gümüşkaya, B. Karlık, Diagnosing diabetes using neural networks on small mobile devices, Expert Systems with Applications 39 (1) (2012) 54 – 60.

[24] A. Khalaj, H. Lutfiyya, M. Perry, The proxy-based mobile grid, in: Y. Cai, T. Magedanz, M. Li, J. Xia, C. Giannelli (eds.), Mobile Wireless Middleware, Operating Systems, and Applications, vol. 48 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, 2010, pp. 59–69.

[25] M. Kim, S. Park, Trust management on user behavioral patterns for a mobile cloud computing, Cluster Computing 16 (4) (2013) 725–731.

[26] J. Kołodziej, S. U. Khan, L. Wang, M. Kisiel-Dorohinicki, S. Madani, E. Niewiadomska-Szynkiewicz, A. Zomaya, C.-Z. Xu, Security, energy, and performance-aware resource allocation mechanisms for computational grids, Future Generation Computer Systems 31 (2014) 77–92.

[27] K. Kumar, Y.-H. Lu, Cloud Computing for mobile users: Can offloading computation save energy?, Computer 43 (2010) 51–56.

[28] D. Lee, H. Lee, D. Park, Y.-S. Jeong, Proxy based seamless connection management method in mobile cloud computing, Cluster Computing 16 (4) (2013) 733–744.

[29] C. Li, L. Li, A multi-agent-based model for service-oriented interaction in a mobile grid computing environment, Pervasive and Mobile Computing 7 (2) (2011) 270 – 284.

[30] C. Li, L. Li, Tradeoffs between energy consumption and qos in mobile grid, The Journal of Supercomputing 55 (2011) 367–399.

[31] W. Li, J. Wu, Q. Zhang, K. Hu, J. Li, Trust-driven and qos demand clustering analysis based cloud workflow scheduling strategies, Cluster Computing (2014) 1–18.

[32] D. Macone, G. Oddi, A. Pietrabissa, Mq-routing: Mobility-, gps- and energy-aware routing protocol in {MANETs} for disaster relief scenarios, Ad Hoc Networks 11 (3) (2013) 861 – 878.

[33] R. Mahapatra, A. D. Domenico, R. Gupta, E. C. Strinati, Green framework for future heterogeneous wireless networks, Computer Networks 57 (6) (2013) 1518 – 1528.

[34] C. Mateos, A. Zunino, M. Hirsch, M. Fernández, M. Campo, 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) (2011) 172–186.

[35] D. G. Murray, E. Yoneki, J. Crowcroft, S. Hand, The case for crowd computing, in: Proceedings of the Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds, MobiHeld '10, ACM, New York, NY, USA, 2010, pp. 39–44.

[36] A. Nicolaos, K. Vasileios, A. George, M. Harris, K. Angeliki, G. Costas, A data locality methodology for matrix-matrix multiplication algorithm, Journal of Supercomputing 59 (2012) 830–851.

[37] E. Niewiadomska-Szynkiewicz, A. Sikora, P. Arabas, M. Kamola, M. Mincer, J. Kołodziej, Dynamic power management in energy-aware computer networks and data intensive computing systems, Future Generation Computer Systems 37 (2014) 284–296.

[38] E. Niewiadomska-Szynkiewicz, A. Sikora, P. Arabas, J. Kołodziej, Control system for reducing energy consumption in backbone computer network, Concurrency and Computation: Practice and Experience 25 (12) (2013) 1738–1754.

[39] E. Pacini, C. Mateos, C. G. Garino, Distributed job scheduling based on swarm intelligence: A survey, Computers & Electrical Engineering 40 (1) (2014) 252–269.

[40] J. A. Paradiso, T. Starner, Energy scavenging for mobile and wireless electronics, IEEE Pervasive Computing 4 (1) (2005) 18–27.

[41] S. Park, W. Kim, I. Ihm, Mobile collaborative medical display system, Computer Methods and Programs in Biomedicine 89 (3) (2008) 248 – 260.

[42] A. Rice, S. Hay, Measuring Mobile Phone Energy Consumption for 802.11 Wireless Networking, Pervasive and Mobile Computing 6 (6) (2010) 593–606.

[43] A. V. Rodriguez, C. Mateos, A. Zunino, Mobile devices-aware refactorings for scientific computational kernels, in: 13th Argentine Symposium on Technology, AST2012. 41th JAIIO, 2012.

[44] J. Rodriguez, C. Mateos, A. Zunino, Are smartphones really useful for scientific computing?, Lecture Notes In Computer Science 7547 (2012) 38–47.

[45] J. Rodriguez, C. Mateos, A. Zunino, Energy-efficient job stealing for cpu-intensive processing in mobile devices, Computing (2012) 1–31.

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

[47] J. M. Rodriguez, A. Zunino, M. Campo, Introducing mobile devices into grid systems: a survey, International Journal of Web and Grid Services 7 (1) (2011) 1–40.

[48] K. Ryabinin, S. Chuprina, Adaptive scientific visualization system for desktop computers and mobile devices, Procedia Computer Science 18 (0) (2013) 722 – 731.

[49] T. Samad, J. S. Bay, D. Godbole, Network-centric systems for military operations in urban terrain: the role of uavs, Proceedings of the IEEE 95 (1) (2007) 92–107.

[50] P. Serrano, A. de la Oliva, P. Patras, V. Mancuso, A. Banchs, Greening wireless communications: Status and future directions, Computer Communications 35 (14) (2012) 1651 – 1661.

[51] S. C. Shah, Energy efficient and robust allocation of interdependent tasks on mobile ad hoc computational grid, Concurrency and Computation: Practice and Experience.

[52] W. X. Shen, C. C. Chan, E. W. C. Lo, K. T. Chau, Estimation of battery available capacity under variable discharge currents, Journal of Power Sources 103 (2) (2002) 180 – 187.

[53] K. Takeno, M. Ichimura, K. Takano, J. Yamaki, Influence of cycle capacity deterioration and storage capacity deterioration on li-ion batteries used in mobile phones, Journal of Power Sources 142 (1-2) (2005) 298–305.

[54] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, J. P. Singh, Who killed my battery?: analyzing mobile browser energy consumption, in: Proceedings of the 21st international conference on World Wide Web, WWW '12, ACM, New York, NY, USA, 2012, pp. 41–50.

[55] R. Torres, L. Mengual, O. Marban, S. Eibe, E. Menasalvas, B. Maza, A management ad hoc networks model for rescue and emergency scenarios, Expert Systems with Applications 39 (10) (2012) 9554 – 9563.

[56] M. F. Tuysuz, An energy-efficient qos-based network selection scheme over heterogeneous wlan - 3g networks, Computer Networks 75, Part A (0) (2014) 113 – 133.

[57] P. M. Vaidya, An algorithm for linear programming which requires $o(((m+n)n^2+(m+n)^{1.5}n)l)$ arithmetic operations, Mathematical Programming 47 (1990) 175–201.

[58] R. van Nieuwpoort, G. Wrzesinska, C. J. H. Jacobs, H. E. Bal, Satin: A high-level and efficient grid programming model, ACM Trans. Program. Lang. Syst. 32 (3).

[59] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, The worst-case execution-time problem – overview of methods and survey of tools, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) 36:1–36:53.

[60] M. Xu, Y. Shang, D. Li, X. Wang, Greening data center networks with throughput-guaranteed power-aware routing, Computer Networks 57 (15) (2013) 2880–2899.

[61] B.-Y. Zhang, Z. Mo, G. Yang, W. Zheng, Dynamic load balancing efficiently in a large-scale cluster, International Journal of High Performance Computing and Networking 6 (2) (2009) 100–105.