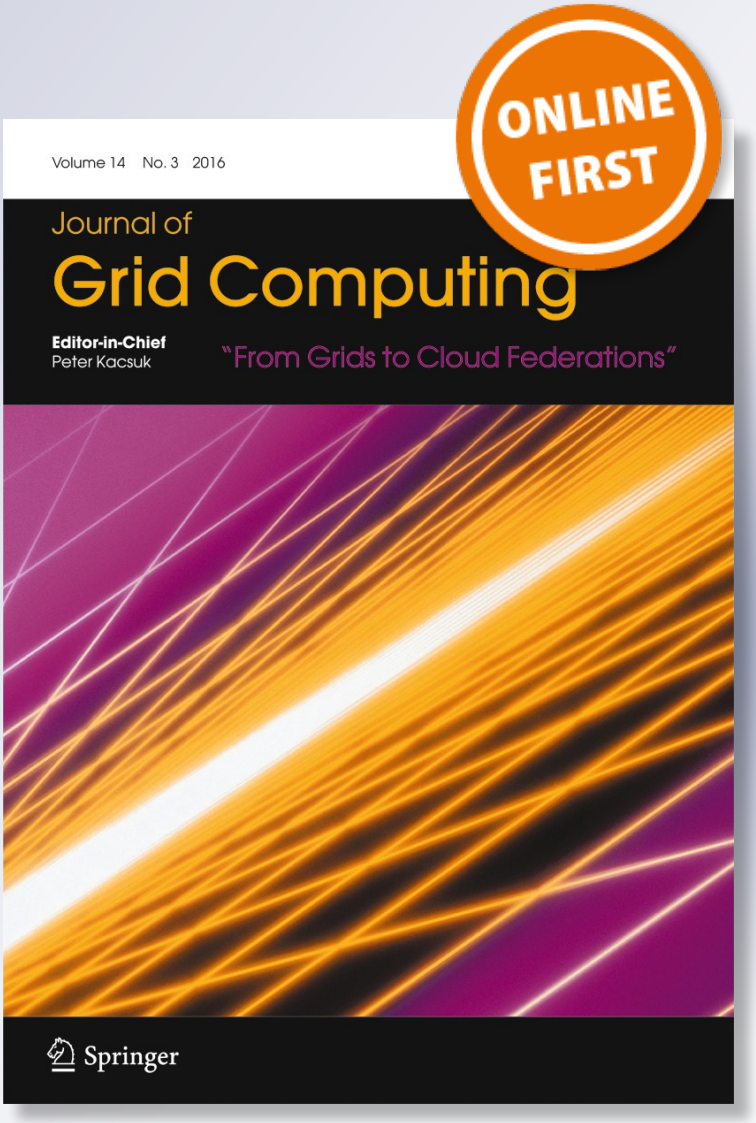
A Two-Phase Energy-Aware Scheduling Approach for CPU-Intensive Jobs in Mobile Grids

Matías Hirsch, Juan Manuel Rodríguez, Cristian Mateos & Alejandro Zunino

Journal of Grid Computing
From Grids to Cloud Federations

ISSN 1570-7873

J Grid Computing
DOI 10.1007/s10723-016-9387-6



Springer

J Grid Computing
DOI 10.1007/s10723-016-9387-6

A Two-Phase Energy-Aware Scheduling Approach for CPU-Intensive Jobs in Mobile Grids

Matías Hirsch · Juan Manuel Rodríguez · Cristian Mateos · Alejandro Zunino

Received: 26 June 2015 / Accepted: 1 November 2016

Abstract The profusion of mobile devices over the world and their evolved computational capabilities promote their inclusion as resource providers in traditional Grid environments. However, their efficient exploitation requires adapting current schedulers to operate with computing capabilities limited by energy supply and mobile devices that cannot be assumed to be dedicated, among other concerns. We propose a two-phase scheduling approach for running CPU-intensive jobs on mobile devices that combines novel energy-aware criteria with job stealing techniques. The approach was evaluated through an event-based simulator that uses battery consumption profiles extracted from real mobile devices. CPU usage derived from non-Grid processes was also modelled. For evaluating the first phase we compared the number of finalized jobs by all energy-aware criteria, while for the second phase we analyzed the performance boost introduced by job stealing. While the best first phase criteria finalized up to 90 % of submitted jobs, job stealing increased this percentage by up to 9 %.

Keywords Mobile grid · Mobile devices · CPU intensive application · Job scheduling · Job stealing

M. Hirsch (✉) · J. M. Rodríguez · C. Mateos · A. Zunino
ISISTAN-CONICET. UNICEN University,
Campus Universitario, Tandil, B7001BBO,
Buenos Aires, Argentina
e-mail: matias.hirsch@isistan.unicen.edu.ar

1 Introduction

Undoubtedly, mobile devices have evolved to small computers able to handle complex computations. Taking this fact into account, some studies [1, 2] have proposed to use mobile devices for processing and visualizing scientific and medical data, while delegating some of the necessary heavy computations to a fixed server. This technique is known as *offloading* [3, 4]. Moreover, other studies [5–7] have taken a step further and have proposed to perform all the heavy processing in the mobile devices.

Mobile devices represent one of the commonest kind of computational device in the world. The number of mobile devices over the world is steadily increasing and exceeded the world population during 2014 (7.3 billion). According to the Android 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 [7–12] have pointed out that unused mobile device capabilities can be scavenged for performing heavy computations, such as the ones commonly found in scientific computing [5]. To this end, several authors [10] have proposed developing distributed computing environments using mobile devices. This allow mobile device resources to be scavenged, and also make them able to perform computations that a single mobile device could not handle

Published online: 12 November 2016

otherwise by sharing their workload and operate cooperatively [10, 13].

In this context, one promising research line in Mobile Computing is integrating mobile devices to traditional cluster/Grid Computing platforms or creating new cluster/Grid Computing platforms tailored for mobile devices [10]. Although these goals present similar challenges to the ones found in traditional distributed computing platforms, mobile devices also introduce new challenges. 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 that are comparatively less reliable/stable and slower, and thus communication protocols need to be redesigned. Another major difference with traditional cluster/Grid nodes is the energy supply because 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. This is not a trivial problem mainly because *accurately* estimating the energy required to execute any computation is challenging [14]. Furthermore, other needed information cannot be estimated, like how long a program would take to run, which in the general case implies solving the Halting problem [15]. Another problem is that mobile devices are not dedicated to the computing environment, which means that mobile device owners might add unexpected or unpredictable load to the nodes from time to time [7].

Besides, a new challenge arises from the combination of unreliable/unstable wireless communication and the limited energy supply of mobile devices: using the network connection in an energy-efficient fashion. This has motivated researchers to study this problem in both low level [16] and high level [17, 18] network protocols, properly addressing at the same time errors due to the nature of wireless links (e.g. disconnections and data retransmissions).

Although these issues might hinder incorporating mobile devices to computational environments, particularly massively distributed ones, there have been significant advances in energy-efficient wireless protocols for connecting nodes to servers in the last years [19]. Still, a main problem is energy optimization at a global level, since better environment-wide energy usage means more mobile devices up and ready to execute computations. Since different nodes have different computational/energy properties, bad scheduling decisions at the inter-node level are likely to produce energy waste, despite of intra-node optimizations. Previous works [7–11] 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 also showed that to increase global energy efficiency levels, the schedulers should consider the inherent energy-related properties of the nodes.

In this work, we explore and compare several criteria build upon energy and computational related properties, materialized as schedulers that exploit the aggregated computational capabilities of a cluster of mobile devices in an energy-aware fashion. We assume that, relying on their batteries, mobile devices are intended to solve atomic and independent computations, which we call *jobs*. Furthermore, these jobs are CPU-intensive, which are present in many applications [20–22]. Such criteria do not rely on strong assumptions such as knowing jobs execution time beforehand. Moreover, the goal is to increase the number of Finalized Jobs per Energy unit (FJ/E).

The proposal, as is, could be employed for scavenging the aggregated capabilities of mobile devices from many real-world scenarios. Specifically, those that involve mobile devices availability over a fixed time period, in areas where local access points are feasible, and devices mobility is reduced in terms of coming in/out of that area. For example, hundred of students in a library, school or campus, hundred of employees in their daily routines at a building, thousand of spectators in a stadium enjoying a match of their favorite sport, etc. In situations like those, the harnessing of unused capabilities represented by aggregated computational resources of people mobile devices is possible. The contributions of this work are summarized below:

– A two-phase hybrid scheduling approach for CPU-intensive jobs in mobile Grids that exploits the advantages of energy-aware criteria, plus centralized and decentralized scheduling schemes. The first phase exploits the wide-view of resources with a centralized ranking-based scheduling, while the second phase exploits the self-organization

advantages of decentralized scheduling by allowing provider nodes to play an active role in balancing the load through Job Stealing techniques. Despite the advantages of these techniques were separately evaluated in our previous works [7, 8], now, their synergy is explored with new energy-aware criteria and improved simulation conditions.
- Three novel energy-aware criteria, easy to be included into real mobile Grid schedulers, for ranking nodes according to their computing capabilities and energy properties. Different instantiations of the two-phase scheduling approach based on these criteria are studied.
- A performance assessment of the hybrid scheduling approach considering energy consumption caused by devices network activity. Previous works [7, 8] disregard such aspect in the performance evaluation. Decentralized techniques like Job Stealing incurs in extra networking activity due to the necessity of nodes coordination including stealing messages, which increase the energetic cost. By taking into account the energy consumption due to network activity, a fair assessment of the benefits of centralized and decentralized schemes is performed. The energy consumption was obtained by profiling real mobile devices with a special power measurement equipment.
- Modelling the lack of resources ownership that reflects the CPU usage caused by device owners. The feature was initially included in [7] as part of the schedulers performance evaluation and simulated through a base profile with a fixed CPU usage value. In this work, the base profile was generated using a probabilistic model [23] derived from data collected of real users activity monitored during several months.

The rest of this paper is organized as follows. First, in Section 2, we describe and discuss the most relevant efforts related to scheduling jobs in mobile Grids. In Section 3, we present the novel two-phase scheduling approach from a general perspective, while Section 3.1 and Section 3.2 describe instantiations of the first and second phase of the approach with the proposed energy-aware criteria. Then, Section 4 outlines the evaluation of the approach. Its content is divided into three subsections: the experimental method followed (Section 4.1), the experimental scenarios (Section 4.2), and the scheduling performance results (Section 4.3). Moreover, Section 5 presents the conclusions of this work and future research opportunities.

2 Related Work

Recently, the role of mobile devices in distributed computing has evolved from simple monitors that check the status of a high performance environment to nodes that actually contribute to the infrastructure with computational resources [7, 9–13]. One reason of this evolution is that mobile devices capabilities have grown exponentially in the last decade [24] resulting in mobile devices able to perform complex computations [5]. However, mobile devices rely on batteries, which might become depleted as a result of performing such computations [25]. Furthermore, different devices have different energy-consumption/computation-performed rate making resource scheduling a non-trivial task [8–11].

Several researchers [7–9, 11, 26–28] have advanced towards tackling this problem, particularly studying approaches for job scheduling in mobile Grids. Since there is no standard on how to implement a Grid infrastructure that integrates mobile devices, different approaches makes different assumptions, consider different type of applications, and scheduling goals.

Table 1 presents qualitative features of the surveyed scheduling approaches that considers mobile devices as resource providers (note that it does not show details about finer contributions of same authors works). The first column presents the approaches that are further described in this section. The *topology* column classifies works according to how they assume the mobile Grid is structured, namely ad-hoc networks ("Ad-hoc") or based on a proxy ("Proxy"). Ad-hoc networks are decentralized and self-organized environments. In such networks, nodes acts as both host and routers allowing distant nodes to interact [34]. Since nodes can move, changes in communication paths are frequent, making routing a complex and energy-consuming task. However, ad-hoc networks have been proposed for lacking infrastructure scenarios, e.g. emergency operations after natural disasters or military missions as distributed command-and-control systems [34–36]. Moreover, these networks

Table 1 Classification of scheduling works for mobile Grids

Work	Topology	Scheduling logic	Scheduling goal	Application type	Required knowledge
Li & Li ([11, 26, 27])	Proxy	Centralized	Maximize the utility of the mobile Grid	Not limited	High
Wei et al. ([29])	Proxy	Centralized	Maximize profit and system uptime	Not limited	High
Ghosh & Das ([9])	Proxy	Centralized	Maximize revenue and minimize response time	Data-intensive	High
Sayed Shah ([28, 30])	Ad-hoc	Hybrid	Minimize energy consumption and response time	Data-intensive	Low
Loke et al. ([31])	Ad-hoc	Decentralized	Maximize speedup	CPU-intensive	None
Li et al. ([32])	Ad-hoc	Decentralized	Minimize response time, waiting time, offloading time	Not limited	High
Shi et al. ([33])	Ad-hoc	Decentralized	Satisfy task deadline with minimum energy consumption	Not limited	High
Rodriguez et al. ([7, 8], this work)	Proxy	Centralized, Decentralized and Hybrid, respectively	Minimize energy consumption and maximize # of finalized jobs per energy unit	CPU-intensive	None

are very resilient since do not have a single point of failure. The alternative –proxy-based mobile Grid– is simpler to develop, but not as resilient because it aggregates mobile devices into distributed environments via infrastructure-based networks, which are conceptually characterized by the existence of a fixed backbone. Figure 1 depicts a proxy-based mobile Grid environment 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 entire Grid environment and differs from a traditional cluster mainly because it could be composed by *heterogeneous* hardware. Offering local resources behind a proxy to a Grid environment is a strategy commonly adopted by traditional Grid platforms such as Ibis-Satin [37] and GridGain (www.gridgain.com).

The *scheduling logic* column refers to where the resource allocation component resides. According to this dimension, works can be classified into centralized, decentralized and hybrid. Centralized implies that there is a dedicated node or a pre-defined group of nodes exclusively in charge of scheduling decisions, while other nodes simply comply what is requested to them. On the contrary, with decentralized scheduling, nodes which actually contribute with resources to the Grid also have an active role in the scheduling decisions. Lastly, an hybrid scheduling logic means that allocation decisions are divided and collaboratively made by several but not always the same nodes, i.e. the scheduling role is alternatively played by the nodes of the distributed environment. This is the case of ERRA [28, 30], where an allocation decision is divided into two levels. A central node makes level 1 decisions and level 2 decisions are made by a node selected at level 1.

The *scheduling goal* dimension groups scheduling works by their objective function. The categories of that dimension relate with another dimension of the table, which is the *application type*. For instance, works that pursue the goal of minimizing the job response time, generally target data-intensive applications ([9, 28, 32]), while works that aim at maximizing throughput or speedup, focus on CPU-intensive applications ([7, 8, 31] and this work). Moreover, there are works whose scheduling goal is to maximize profit or an utility function that includes several variables – e.g. payments per resource utilization, time and energy

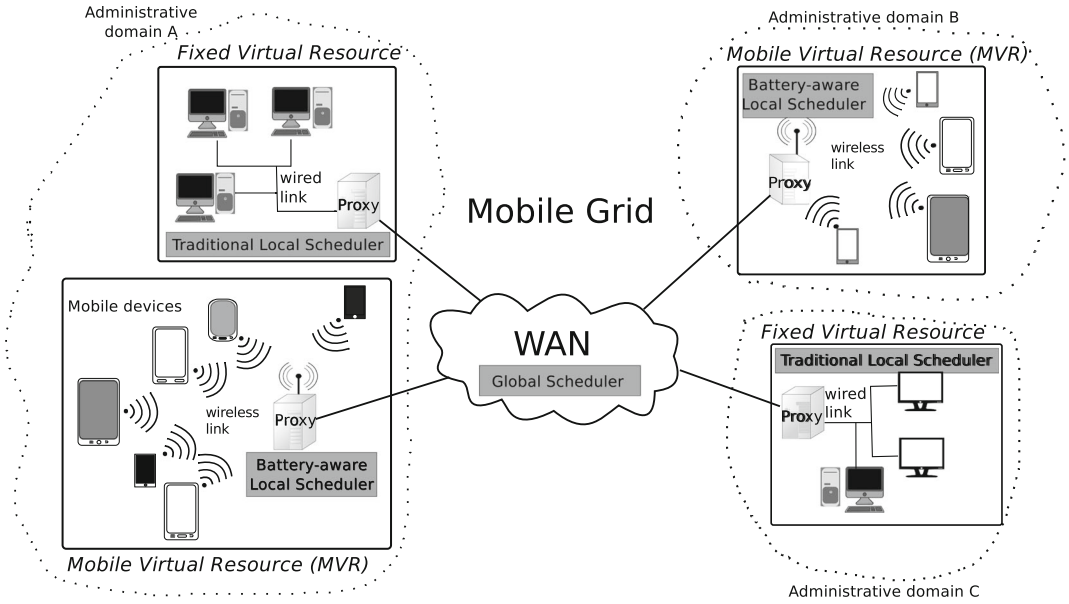


Fig. 1 Proxy-based Grid environment. In this paper, we study battery-aware local schedulers

budgets– and do not target any specific application type ([11, 26, 27, 29]).

Lastly, the column *required knowledge* refers to the extent to which the job requirements need to be specified to serve as input to the scheduler, e.g. execution time, energy consumption, input/output data, etc. *High* means that the scheduler needs at least a requirement which is hard or even impossible to determine without executing the job, e.g. the job execution time. Moreover, *low* means that requirements are superficially specified to give the scheduler a hint on the predominant type of resource on which the completion of jobs relies on, for example, data-intensive or CPU-intensive. Such high-level classifications could be provided with minimum effort by an expert. Finally, the *none* value is for the cases when the scheduler does not require to know any job-related information. Below we discuss in detail the related works in the table.

In [11, 26], the authors present near optimal approaches for assigning jobs to mobile nodes. These two schedulers aim at maximizing the utility of executing jobs using Lagrange multipliers, and were designed for environments where not all jobs have the same importance or delivered 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 of pay-per-use Grids [38].

The scheduler proposed in [9] simulates a market where mobile devices buy and sell computational capabilities. Similarly to [11, 26], these schedulers aim at obtaining a near optimal profit according to an utility function. This is done by simulating a negotiation process between the proxy and the mobile devices. The schedulers are oriented towards minimizing the used network bandwidth to transfer job information and results by means of a framework based on LZ78 compression. Since the framework focus is on optimizing network usage, the proposal seems to be appropriate for data-intensive applications and not for CPU-intensive applications unless the latter involve transferring large amounts of data.

ERRA [28], a two-step scheduling scheme that extends a previous work from the same authors [30], aims at minimizing energy consumption during data transferring in mobile ad-hoc Grids. The first step selects nodes within an area with the highest probability of staying connected for a longer period. That probability is inferred through the movements patterns history of nodes, which are fed to a Markov model to predict the next probable location. 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 devices and the weighted links to

allocate a group of nodes to a job set. The authors propose different job allocation strategies depending on the data transfer requirements and job interdependency type. However, since the resource allocation scheme is focused on exploiting nodes communication capabilities rather than their computing capabilities, the scheme seems more appropriate for data-intensive jobs than CPU-intensive jobs.

In [31], the authors evaluate the feasibility of applying a work stealing algorithm in different multi-layered Bluetooth ad-hoc network configurations for solving CPU-intensive tasks. The configurations include a single layer fan-out topology and a multi-layered or linear topology. In the first configuration, the delegator node schedules jobs to worker nodes which are only at one-hop distance from it. In the second configuration, job scheduling uses intermediary nodes to reach workers which are further than one-hop distance from the delegator. Intermediary nodes play the role of workers and delegators at the same time. The performance assessment comprises experiments that measure the working time, transmission time and speedup with varying job chunk sizes and the overall task size.

In [32] the authors propose an extension to traditional online (i.e. Minimum Execution Time and Minimum Completion Time) and batch (i.e. Min-Min, MaxMin and Sufferage) scheduling heuristics with communication concerns, to enable their utilization for offloading applications in mobile ad-hoc Clouds. Precisely, they propose to include communication time as part of the task completion time as such heuristics traditionally consider. To obtain that time authors propose to use the data to offload plus a linear equation that given the hops in a path between nodes determines the link bandwidth. In addition to the communication-aware version of traditional heuristics, the authors propose a new heuristic called Min-Hop that only considers communication time when offloading tasks.

Moreover, HACAS [29] is an instantiation of the Ant Colony algorithm for scheduling applications in a local mobile Cloud. HACAS pursues the goals of maximizing the profit of the system while balancing the load of provider nodes whose resource capacity is limited. Applications to be scheduled give certain profits to the system when completed, and the resources they need are specified generically. Given a set of applications to execute, authors formulate the scheduling problem as a binary knapsack problem where applications that best fit the aforementioned goals are selected for execution.

In [33], the authors propose a decentralized two-phase scheduler for Clouds with mobile devices as resource providers. In the discovery phase participant nodes periodically exchange information of their computing capacity, power and queue length. In the scheduling phase, a source node assigns tasks to nearby collaborators. Tasks are assumed to be submitted with their required amount of computation, size of data to be transferred and time constraints. Moreover, time of computation and communication on every potential processing node is known and used to generate a set of candidate nodes that best meet task time constraints. Each node of such set is assigned with a probability based on the relative energy consumption of tasks.

SEAS [8], designed to be easily implemented in real-life mobile Grid environments, aims at minimizing the energy consumed per executed job. Jobs are scheduled according to a node rank-based heuristic. For ranking a mobile device m, SEAS uses the following function:

$$resources\,per\,job_m = \frac{estimated\,remaining\,uptime_m \times benchmark_m}{number\ jobs_m + 1} \quad (1)$$

where $estimated\,remaining\,uptime_m$ is the estimated remaining uptime for the mobile device having remaining battery power, $benchmark_m$ is a value obtained with a benchmark that measures the FLOPS (Floating-point Operations Per Second) of the device, and $number\ jobs_m$ are the amount of queued jobs of the device.

Estimating the remaining uptime is not trivial [39]. However, SEAS [8] proposes a simple technique that was fairly accurate when using notebooks and netbooks. The estimation algorithm uses SOC (State of Charge) events by event-based battery APIs such as iOS[1], Android [40] and ACPI.[2] By assuming a lineal discharge rate between two events $i-1$ and i, the discharge rate dr can be calculated as:

$$dr = \frac{c_i - c_{i-1}}{t_i - t_{i-1}} \quad (2)$$

[1] http://tinyurl.com/ndl72fk.

[2] http://acpi.info/DOWNLOADS/ACPIspec50.pdf.

where c_i and c_{i-1} are SOCs reported by the events i and $i-1$, respectively, t_i and t_{i-1} are the times of these events. Therefore, the estimated remaining uptime is:

$$estimatedRemainingUptime = \frac{c_i}{dr} \quad (3)$$

Since the discharge rate is actually not lineal [41] the estimation heavily varies from event to event. Thus, SEAS uses the average of all previous estimations for minimizing this variation. A recent work [27], which is an extension of [11, 26], places SEAS as the best algorithm in terms of energy consumption ratio, outperforming the second best algorithm by 9.5 % for scenarios where job size varies, and by 8.3 % for scenarios where the number of mobile Grid users varies.

In [7], the authors propose a scheduler that uses the node ranking criteria of SEAS but applies a decentralized scheduling logic, i.e. job stealing for balancing the workload among nodes. Even when the SEAS has acceptable performance, it does not always make good decisions because the information used to schedule jobs cannot be estimated precisely in smartphones and tablets. By adding job stealing, the negative impact of using uncertain battery information can be minimized.

Works presented in this section, particularly those of distinct authors, differ in several aspects. Because of their scheduling purpose, nodes singularities covered, targeted application type and problem statement inputs assumed, works are strong in different application contexts. While the purposes of some efforts [9, 11, 26, 27, 29] are more relevant to commercial Grids, others [7, 8, 28, 30–33] seems to be more appropriate for –although not exclusively circumscribed to– volunteer computing schemes. Besides, not all works give the same importance to nodes singularities. For instance, the limited resource capacity has been considered directly in [7–9, 11, 26, 27, 29, 33] as a constraint of the problem statement or parameter of the scheduling logic, or indirectly in [28, 30–33] with scheduling schemes that select resources based on their energy consumption-related properties rather on availability indicators. Other nodes singularities are the non-dedicated nature of nodes and mobility, which have been considered only in [7, 9, 33] and [9] respectively. Lastly, several schedulers [3, 9, 11, 26, 27, 29, 32, 33] are hard to implement in real-life Grids [10] because they assume to know job information that is usually unavailable, difficult or impractical to estimate in the general case [15]. For instance, number of operations and therefore time and energy consumption of jobs while executing in a node. This information depends not only on the hardware, but also on how a job is programmed because even small differences in coding practices causes huge differences in time and energy consumption [25].

Centralized scheduling is cheaper than decentralized scheduling in the sense that less quantity of messages need to be exchanged between nodes for coordination. Decentralized scheduling makes worker nodes to play an active role in scheduling decisions that, according to [7], improves performance since it dynamically balances the load among nodes and compensates the lack of accurate battery predictions. However, the cost of that improvement was not empirically evaluated yet. This work proposal is to combine into a hybrid approach the advantages of these two distinct scheduling logics, plus new energy-aware criteria whose synergy is fairly evaluated with an improved simulation technique that includes energy consumption due to nodes network activity.

3 Intra-MVR Energy-Aware Two-Phase Scheduling: Formulation and Proposed Approach

Our hybrid two-phase scheduling approach focus on avoiding and correcting *sub-exploited* states at the MVR level. These states suggest underused periods of CPU cycles caused by the lack of accurate predictors that relates computing capability with energy properties, dynamic changes of resource availability, and a priori unknown requirement of jobs. The latter cause is an assumption of our problem statement, then the proposed hybrid two-phase approach is an aid to mitigate the first and second causes. Figure 2 illustrates a dynamic view of the approach. As jobs are submitted to the scheduler queue, they are distributed to nodes by following a mechanism that meets an objective function, or in this work, maximize system throughput. We call this initial job scheduling the first phase. Then, as sub-exploited states are detected, jobs are re-distributed by a second phase. The result is a job scheduling approach that maximizes nodes exploitation even when their computing capabilities vary due to unexpected CPU usage introduced by mobile devices owners.

The hybrid characteristic is from the fact that the first phase and second phase strategies combine

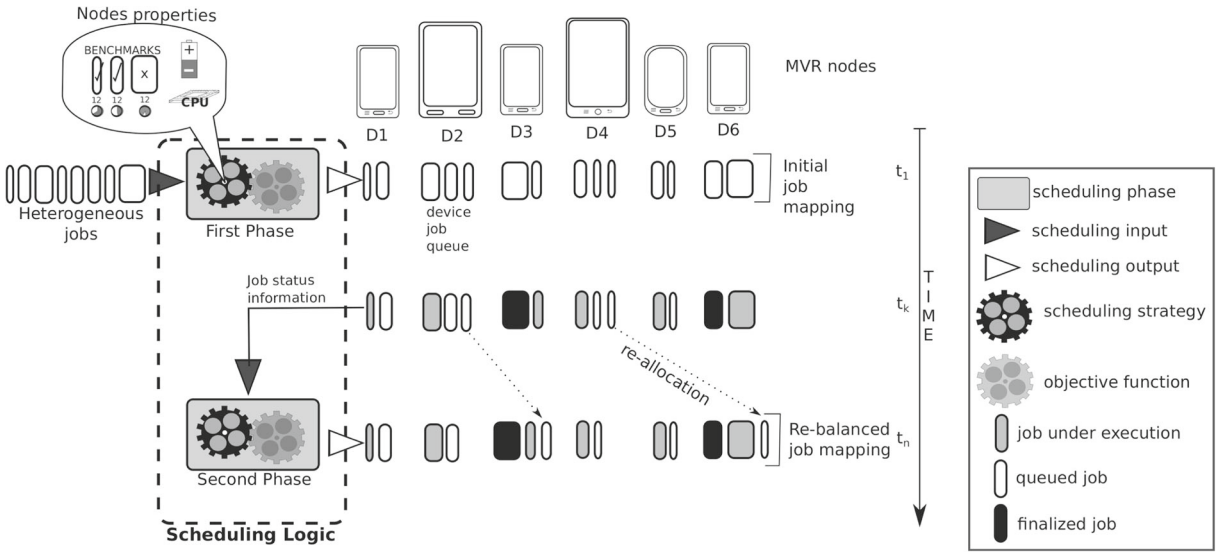


Fig. 2 Overview of the two-phase scheduling approach

a centralized ranking-based scheduler, described in Section 3.1, and a decentralized scheduler through the Job Stealing techniques detailed in Section 3.2. Both phases, in turn, use novel criteria to rank nodes according to their computing capabilities and energy-related properties. These criteria will be referred from now on as energy-aware criteria and include not only an enhanced version of SEAS criterion [8] (from now on E-SEAS) but also other currently unexplored computing/energy related properties derived from benchmarks and historic information provided by nodes.

Figure 3 illustrates the main concepts associated to our approach and their relationships. The combination of these concepts leads to different materialization of the two-phase scheduling approach proposed, and thus the figure also qualitatively outlines the pros and cons of each materialization. That figure complements the dynamic view outlined in Fig. 2.

3.1 First-Phase Scheduling: Centralized Scheduling Based on Energy-Aware Criteria

As mentioned in Section 3, the initial job scheduling, i.e. the first phase, is performed via a ranking-based scheduling that uses novel energy-aware criteria to rank (sort) mobile nodes. With respect to computational complexity, in this phase, the proxy receives let us say n jobs. For each job, the proxy sorts the available mobile nodes (let us say m) into a list based on one of such criteria, and picks the first node in the

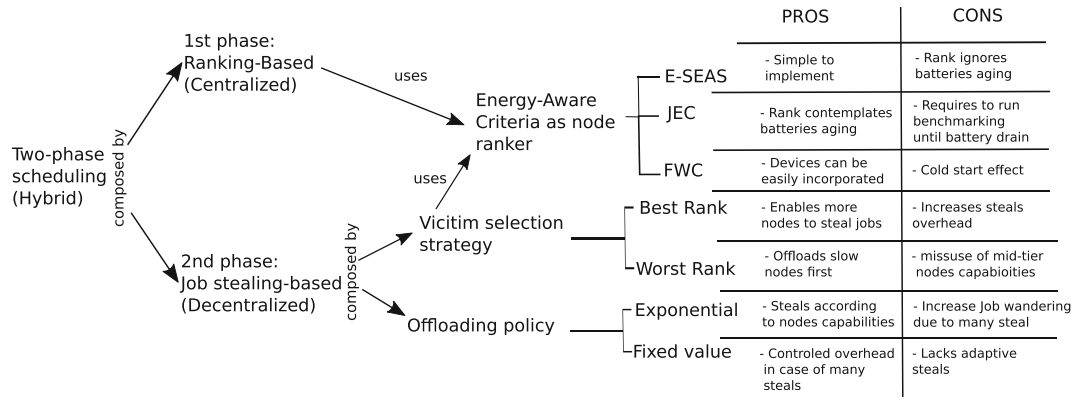


Fig. 3 Two-Phase Hybrid scheduling concepts and techniques

list to execute the job. Re-ordering must be done upon processing each job since assigning a job to a node in turn affects the ranking of that node, and besides the proxy periodically receives status update events that might affect the ranking of any node in the MVR. The complexity of the first phase of the algorithm is therefore $O(n * m * log(m))$. The convergence time of the first phase for n jobs is given by the time it takes to rank (sort) nodes times n.

Particularly, adopting a proxy-based topology for managing MVRs favors the task of making decisions at the MVR level because proxies act as intermediaries between nodes, i.e. mobile devices they coordinate and the rest of the mobile Grid environment. In this context, mobile devices willing to offer resources to the mobile Grid join a proxy and pass through a registration process to let the proxy know about the resources to be managed [42]. That registration process is an opportunity for the proxy to identify and categorize all devices, e.g. in terms of computing capability. For this purpose, we propose to associate each mobile device a *feature profile*. This feature profile contains a set of device properties, including benchmarks results and manufacturer data that are used by the proposed energy-aware criteria. 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 comprises information about the standard 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 examples of feature profiles (as columns) for different device models. In fact, these device models were used in the simulations performed to evaluate the two-phase scheduling approach proposed (see Section 4.2). The first row of each feature profile shows the computing capability of the device expressed in MFLOPS (Million Float-point Operations per Second). These values are the average of twenty runs obtained with Linpack for Android. Another way of measuring the computing capability is by aggregating the quantity of a set of different well-known benchmarks –second row– that the device was able to perform within a certain time period. That value is presented in the third row, while the time period is the time the battery last from full charge (100 %) to cut-off voltage (0 %) [5]. The feature profile also includes energy-related properties, i.e. battery capacity as informed by the manufacturer. The last row presents a novel way of rating the performance of a device by condensing the benchmark information with the battery capacity information into a single value called Job Energy Consumption Rate, which will be further developed in Section 3.1.2.

Since scheduling decisions based only on static properties read from the feature profile of a device intuitively might not be enough to represent the real computing capabilities of an MVR, it is necessary to take into account dynamic, i.e. runtime information as well. Dynamic information includes the level of available energy of the MVR, i.e. the devices SOC, and historic workload stats, i.e. number of jobs assigned to a device, average job execution time in a device, among others. The combination of static and dynamic information as part of the scheduling decisions in the proxy-based approach has been already proved to be effective [8]. However, that study does not consider the relationship between energy consumption and jobs execution, or average job execution times.

Below three alternatives for energy-aware criteria combining feature profiles and dynamic data are presented. These criteria are the E-SEAS, an improved version of SEAS criterion for smartphones and tablets (Section 3.1.1), the Job Energy aware Criterion (JEC) (Section 3.1.2) and the Future Work aware Criterion (FWC) (Section 3.1.3).

3.1.1 The E-SEAS

When using SEAS in smartphones and tablets equipped with batteries that last longer than notebooks and netbooks, sampling carried out in the context of this work revealed that it is necessary a considerable time (several minutes or even hours) until the battery estimations obtained with the SEAS battery model [8] become effective. Therefore, we introduce a change in regards to how the SEAS handles battery depletion.

Within a battery discharge cycle the SOC (State Of Charge indicated in %) is expected to decrease as the time passes by, meaning that the battery model at

[3] https://play.google.com/store/apps/details?id=rs.pedjaapps.Linpack&hl=en.

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

Table 2 Feature profiles examples. GRG (Gaussian Random Generator); PC (Prime Checker); FFT (Fast Fournier Transform); SE (Sieve of Eratosthenes); TH (Towers of Hanoi)

Device	MFLOPS	Benchmark					Executed Benchmarks	Battery Capacity (μAh)	Job Energy Consumption Rate
		GRG	PC	FFT	SE	TH			
Samsumg I5500	7.60	10	10	11	10	10	51	1,200	23.53
ViewSonic ViewPad 10s	17.07	42	42	43	43	42	212	3,300	15.57
Acer A100	61.66	30	30	30	30	30	150	1,530	10.20

time i should report a higher state of charge than at time $i+1$. However, the SEAS estimation model needs to reach a *converging time* until the model starts to predict the estimated remaining uptime values (Eq. 3) correctly. Then, values reported before the converging time can be considered as spurious. When this model was applied to estimate the remaining battery capacity of netbooks or notebooks, the converging time was 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 on mobile Grids composed by tablets and smartphones might result in chaotic scheduling patterns making the SEAS less effective. Therefore, the battery estimation model used in this work uses the SOC reported by the operating system (OS) via battery events.

The SOC-based estimation model, also referred from now on as percentage-based model, not only provides a more accurate and normalized battery remaining value but also it is independent of the real capacity of the battery. Figure 4 shows the uptime curves for the Samsumg I5500, the ViewSonic ViewPad 10s and the Acer A100 for different constant CPU usage levels, using the estimation model based on the percentage reported by the devices and the SEAS estimation model. It worth noting that the SEAS estimation model starts to report values below the maximum capacity of the battery. Then, after some time, this maximum is reached and the values reported become more accurate. This phenomenon is not present in SOC-based battery depletion models.

The battery estimation change described above motivated the creation of the E-SEAS criterion. Conceptually, its node rank is built upon the combination of the same three factors of the SEAS criterion, including the estimated remaining capacity of the node, the node computation capability measured in Mega FLOPS (Mega Flops row from Table 2) and the jobs in the node queue. But, unlike the SEAS, the remaining uptime information of a node is not estimated with the Eq. 3 but is directly inferred from the SOC reported by the OS, which is more appropriate for the kind of mobile devices considered in this work. The components of the formula for node ranking of E-SEAS are essentially almost the same as the ones used by the SEAS:

$$nodeRank = \frac{SOC * flops}{assignedJobs + 1} \tag{4}$$

where SOC represents the percentage of remaining battery charge, $flops$ is equivalent to the $benchmark_m$ component of the SEAS formula and $assignedJobs$ refers to the old $number\ jobs_m$ component. In other words, the main difference between SEAS and E-SEAS formulas is the replacement of the factor computed via Eq. 3 ($estimatedRemainingUptime$) by the SOC calculated by the Android OS.

3.1.2 The Job Energy Aware Criterion (JEC)

A node rank based on this criterion takes into account the relationship between the energy used and total of benchmarks a device can execute. These values are extracted from the device feature profile. Energy used refers to the battery capacity in milliampere-hour as reported by the device manufacturer (Battery Capacity row in Table 2) while total of benchmarks refers to the quantity of established benchmarking tests that a device is able to finalized before the depletion of its battery occurs [5] (# Total Executed Benchmarks row in Table 2). The quotient of energy used by total of benchmarks gives the rate of

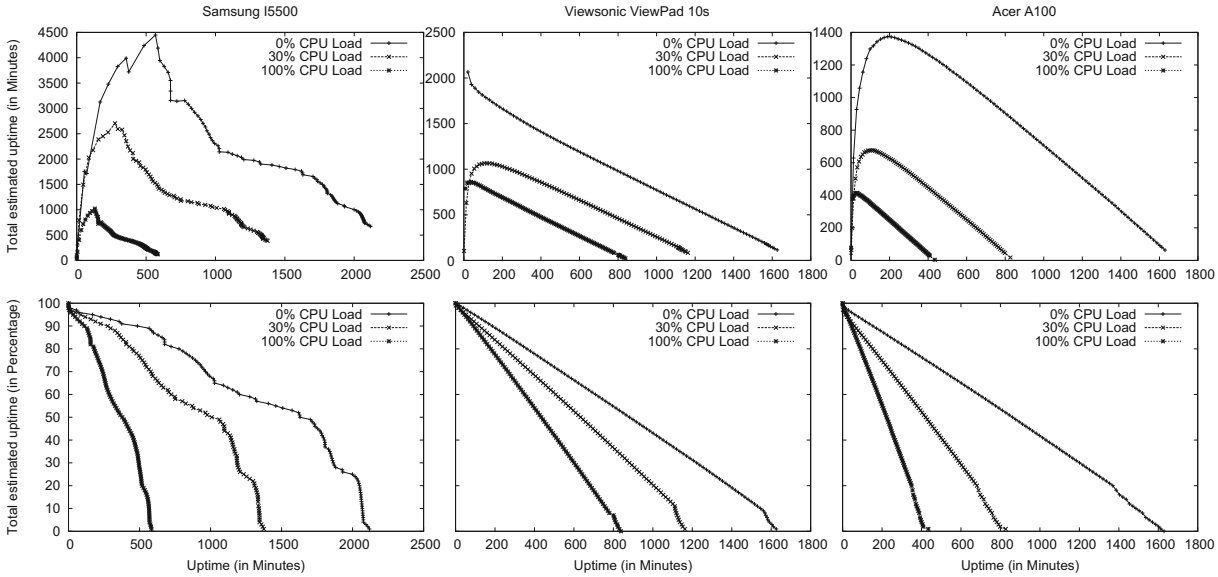


Fig. 4 Comparison between the SEAS battery estimation *(up)* and the SOC-based battery estimation *(bottom)* for the devices used in the experiments of this paper

energy consumption per benchmark, or from now on, $jobEnergyConsumptionRate$ (Eq. 5):

$$jobEnergyConsumptionRate = \frac{BatteryCapacity}{\#TotalBenchmarks} \tag{5}$$

Thus, the ranking formula defined by this criterion is presented in Eq. 6:

$$nodeRank = \frac{SOC}{jobEnergyConsumptionRate} * \frac{1}{assignedJobs + 1} \tag{6}$$

In summary, JEC combines runtime information *(SOC* and *assignedJobs*), and static properties associated to devices represented by the value of $jobEnergyConsumptionRate$. For example, in case two devices A and B have equal SOC and $assignedJobs$ values but device B has lesser $jobEnergyConsumptionRate$ than device A, then the criterion ranks device B higher than device A with the assumption that device B might take better advantage of its energy to execute jobs.

3.1.3 The Future Work Aware Criterion (FWC)

This criterion, unlike JEC, is purely based on runtime information. It considers that the future available computational resources of a device could be estimated by analyzing the computational resources the device presented in the past. In other words, FWC assumes that the computing performance delivered by a device in the past could be maintained in the future as well. Like for E-SEAS and JEC criteria, SOC is obtained through the battery events periodically reported by a device OS, and $assignedJobs$ is the number of jobs in the device queue. Moreover, the $averageTimeToCompleteJob$ value is calculated by the scheduler and represents the average time a device takes to complete each of its scheduled jobs. All in all, the ranking formula is defined in (Eq. 7):

$$nodeRank = \frac{SOC}{averageTimeToCompleteJob} * \frac{1}{assignedJobs + 1} \tag{7}$$

The left quotient of the formula penalizes high average job execution times while considers remaining battery percentage through SOC values while the right quotient avoids leaving some nodes less loaded than others. When comparing two devices with equal SOC and assigned jobs, the one with less $averageTimeToCompleteJob$ value would be assigned with a higher $nodeRank$ value. Consequently, the device with the highest rank value has the highest priority of being assigned a job.

A distinctive characteristic of this criterion is the cold start effect. While schedulers based on JEC and E-SEAS operate effectively from the first scheduled job, FWC based schedulers need some time to achieve effectiveness because FWC uses runtime historic information to calculate the $averageTimeToCompleteJob$ value. At the time of determining the $nodeRank$ of a device, if no jobs have been finalized yet, but there is a job currently executing in that device, then the $averageTimeToCompleteJob$ is computed as twice the time the job has spent executing so far. If no jobs have been finalized yet, nor a job is executing in the device, a value of "one" is returned as $averageTimeToCompleteJob$ to keep the left quotient as valid.

3.2 Second-Phase Scheduling: Distributed Re-Balancing Through Job Stealing and Energy-Aware Criteria

Even when our criteria consider runtime information to reflect the actual state of the available computing capabilities of devices, the unknown computational requirements of jobs, and the lack of update/accurate information -e.g. about future user CPU loads-, still affect the scheduling decisions of the first phase making them perfectible as the time passes. A second phase –re-balancing phase– is important to adapt scheduling decisions made in the past to periodically changing resource availability of the environment.

The Job Stealing technique has been effectively applied as a re-balancing mechanism in traditional distributed computing [37] and also in mobile Grids [7]. Basically, the Job Stealing technique aims at minimizing unused nodes in distributed environments, preventing jobs in a node from waiting to be executed when there are idle resources elsewhere. This behavior in conjunction with a detailed study of the dynamics of a mobile Grid environment inspired the formulation of the scheduling second phase, instantiated with Job Stealing.

In our approach, the use of Job Stealing involves configuring a selection strategy[5] to select the *victim* node and an offloading policy, i.e. the mobile device from which jobs will be stolen and the quantity of jobs to steal respectively. Steals are performed from an idle mobile devices or *stealer*. One victim selection strategy and one offloading policy represent a particular configuration of the Job Stealing technique. Given the number of possible configurations, we narrowed this set by selecting the best configurations in terms of FJ/E explored in [7]. In this context, one difference compared to [7] is the criteria used to rank the nodes: while [7] uses the SEAS, this work used the criteria proposed in Section 3.1. The considered selection strategies are:

Best Ranking Aware Stealing (BRAS): This strategy selects the mobile device with the highest ranking according to the criterion used. This strategy aims at offloading the least overloaded mobile devices. As a result, victims are more likely to become idle and, in turn, they would be themselves able to further offload other mobile devices. Basically, this strategy is expected to generate an offloading chain reaction.

Worst Ranking Aware Stealing (WRAS): Instead of selecting the best ranked mobile device, it selects the worst ranked one. In this case, the goal is to globally balance the load because idle mobile devices take jobs from the most loaded devices.

Moreover, the next two *policies* describe how a stealer determines the number of jobs it will try to steal upon each attempt. In practice, stealing several jobs at once might reduce the networking overhead because it requires establishing only one connection. Networking requires energy and reducing the need for that might extend battery life. For this issue, the two policies analyzed are:

Fixed Number: A stealer always steals the same (statically configured) number of jobs. For n-core mobile devices, each steal attempt might actually try to steal n jobs (the number of jobs the device is able to physically execute in parallel).

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

[5] This is inherent to the functioning of Job Stealing and should not be confused with the scheduling strategy in Section 3.

Another aspect that differentiates this Job Stealing instantiation with respect to that of [7] is the consideration of the network energy cost inherent to a decentralized scheduling logic. By introducing Job Stealing as part of the scheduling logic, mobile devices tend to be more active in terms of data transferring because when each one finalizes all its queued jobs looks for other mobile devices to steal their queued jobs. By considering that, such decentralized scheme implies a communication overhead, given by the nodes behavior that consist in sending stealing messages and eventually move jobs from a victim node's queue to the stealer's queue, the potential gains offered by this technique [10] is better evaluated in this work by considering the energy costs associated.

With respect to computational complexity of this second scheduling phase, upon the first job steal attempts are sent, there will be m_{busy} nodes with pending work to do which are busy executing one job, and m_{idle} nodes which are idle and hence will try to steal work from busy nodes, with $m = m_{busy} + m_{idle}$ being m the available mobile nodes in the MVR. Then, for each idle node m_{idle_i}, the proxy sorts the list of busy nodes according to a ranking criterion and picks the first or last element m_{busy_j} from this list (depending whether the BRAS or the WRAS strategy is used), thus m_{idle_i} can steal work from m_{busy_j}. Then, the complexity of this phase is $O(m_{idle} * m_{busy} * log(m_{busy}))$. The convergence time depends on the number of times the phase is initiated in a mobile node: considering that it is initiated when a node executes all its queued jobs, i.e. when the node becomes idle, the convergence time of that phase for that node is when the victim selection concludes. Eventually, this will cease when all idle nodes start the execution of a job that has been stolen from the queue of a busy node, or when all job queues in nodes are empty, after which no more steals are possible. Note that jobs whose execution is initiated in a node cannot be stolen by another node, which ensures convergence.

4 Evaluation

To evaluate the two-phase scheduling approach along with the energy-aware criteria, we have simulated different MVRs. Broadly, simulation in distributed computing as evaluation technique is an accepted and well-known practice [7, 43] because it reduces evaluation times and provides easily repeatable experiments. In Section 4.1, we present a description of the techniques, models and algorithms used to simulate MVRs and their nodes. The evaluation scenarios are presented in Section 4.2, while Section 4.3 discusses the results.

4.1 Simulation Technique

The simulator we used is based on the discrete event simulation model [7]. A discrete event simulation uses events to represent everything that might modify the state of the model, in particular, events affecting an MVR. In this sense, job arrivals, CPU usage changes, battery notifications, and job terminations are different kinds of events in our simulations. To simulate the dynamics of an MVR, we feed the simulator with events that reflect nodes battery changes and jobs arrivals. To generate the former, and specifically node battery changes for different CPU usages, we first sample battery level drops from real mobile devices under predefined target CPU usages. Notice that, although battery discharging might be seen as a continuous variable, mobile devices report state of charge (SOC) values as a discrete variable. For instance, Android uses discrete SOC and notify interested applications by means of Intents [40]. Then, we convert those samples into events for the simulator. From each mobile device, from each target CPU usage, two sets of events are obtained. One set contains the battery events, i.e. time and SOC registered during sampling. The other set of events aims at accurately simulating CPU usage and contains time and CPU usage values.

The software used for sampling mobile devices at specific target CPU usages allow us to achieved more real execution scenarios than when job execution is simulated by considering the available CPU as a fixed constant value, i.e. without CPU fluctuations. The software works as follows. To force the mobile device to keep a given target CPU usage, we have designed a component that generates CPU usage by performing floating-point operations in a dedicated thread, and a monitoring component that periodically adjusts a delay time between the operations so as to support lower target CPU usage. External software, e.g. the Linux kernel or the Android platform itself slightly impact on the target CPU usage and this becomes evident particularly when targeting 0 % CPU usage.

Algorithm 1 shows the thread logic of the CPU generator that sleeps for a while and then executes a set of floating-point operations. This logic repeats until the thread is externally killed. The sleeping time is adjusted by another thread that executes the Algorithm 2. The last thread uses the difference of the current CPU usage and the target CPU usage to modify the sleeping time so that the CPU usage generated is closer to the target CPU usage. The current CPU usage is calculated through the average of the last 30 measurements stored in the `/proc/stats` file (recall that Android runs a modified Linux kernel).

Algorithm 1 CPU usage generator thread

```
1.  procedure CPUUsageGenerator
2.      while true do
3.          if SLEEP > 0 then
4.              WAIT(SLEEP)
5.          end if
6.          count ← 0
7.          while count < CYCLES do
8.              performFloatingPointOps
9.              count ← count + 1
10.         end while
11.     end while
12. end procedure
```

Algorithm 2 CPU usage adjuster

```
1.  procedure CPUUsageAdjuster(TargetCPUUsage, Threshold)
2.      while true do
3.          cpu Usage ← getCPUUsage
4.          diff ← cpu Usage/TargetCPUUsage
5.          if −Threshold < 1− diff < Threshold
            then
6.              NotifyStable
7.              Log(cpu Usage)
8.          else
9.              sleep ← CPUUsage.getSleep()
10.             if sleep = 0 then
11.                 sleep ← 1
12.             end if
13.             CPUUsage.setSleep(sleep * diff)
14.         end if
15.     end while
16. end procedure
```

Table 3 outlines the results obtained with the CPU usage generator software for some target CPU usages (0 % and 100 %). Despite the usage introduced by Android bundled applications and the active role of the platform on the application life cycle, which also require CPU, the resulted Average CPU usage is very close to the target CPU usage. In addition, the standard deviation is acceptable in all cases, indicating that the dispersion of the measurements is small.

The simulator requires a final set of events that represent jobs arrivals with their associated arrival time and computational requirements in terms of mega floating-point operations (MFLOP). To simulate the job execution in a mobile device, the simulator uses two *usage profiles* of the same mobile device. By usage profile we do not refer to a profile of user actions but to the SOC changes that a device battery experiments for a specific CPU usage. One of those profiles is the *base usage profile* that represents the battery discharge rate of a device that is not executing a job of the distributed platform. Then, CPU usage that causes that battery discharge derives from user applications and/or simply Android and its default bundled applications. More details about the base usage profiles used in experiments are explained in Section 4.2. The other profile required by the simulator is the *full usage profile,* which represents the battery discharge rate of a device that is executing a job of the distributed platform. We assume that each job is optimized and correctly coded to use as much cores as available, or in other words a job execution always rises the CPU usage to 100 %. This does not mean that the 100 % of the computing cycles are dedicated to perform the job because the usage introduced by the base usage profile is also considered in that 100 % of CPU usage.

Usage profiles contain events for the simulator. When a simulated MVR starts up, all nodes start reading the first CPU and battery events of their base usage profile and continue reading the next events from that profile as the simulation time passes. Time is maintained by a central clock. Every time a node swaps its base usage profile by its full usage profile and vice versa, a synchronization against this central clock is performed. On one hand, the synchronization is to avoid start reading a profile always from the beginning instead of from the current simulation time. On the other hand, profile swapping is done to faithfully reproduce the energy consumption rates of the different CPU usages, e.g. energy is much faster drained

Table 3 CPU usage generator

Device	Target CPU usage of 0 %				Target CPU usage of 100 %			
	Sampling Time (hh:mm:ss)	Samples	CPU AVG	CPU STDEV	Sampling Time (hh:mm:ss)	Samples	CPU AVG	CPU STDEV
Samsung I5500	35:23:47	7,034	3.83 %	1.08 %	09:45:28	2,066	99.98 %	0.04 %
ViewSonic ViewPad 10s	27:15:39	5,476	10.23 %	2.26 %	13:57:44	2,977	99.99 %	0.03 %
Acer A100	27:08:26	4,877	2.02 %	1.49 %	07:16:44	1,556	99.97 %	0.06 %

when the CPU usage is 100 % than when it is 0 %. As Fig. 5 shows, the profile swapping between a base usage profile (z % of CPU usage) and a full usage profile (100 % of CPU usage) happens when a job starts and finalizes in a node, respectively. All in all, when a job is scheduled within an MVR and the hosting node starts to execute it, the node performs three steps:

1. Calculating the time the job will spend executing in the node. This time is calculated from the MFLOP of the job and the MFLOPS of the node excluding the non-available MFLOPS consumed by its base usage profile.
2. Swapping the base usage profile and the full usage profile mediated by a synchronization operation to continue reading the events of the last profile so that the new time stamp matches the current simulation time. The full usage profile is kept up to the time calculated in step 1.
3. Performing the inverse of step 2 to model the job completion. This swapping takes place when the executing time from step 1 is reached. The node raises a job finalized event and switches back to its base usage profile. Again, a new synchronization operation is performed. The process is repeated for all the jobs queued in a node and for all the nodes of the MVR while there are jobs to be processed and nodes able to process them.

Moreover, when a job is assigned to a node, particularly, an energy consumption caused by node networking activity occurs. That energy consumption is a feature of the current simulator software that was not included in the version used in [7]. We included such information in all events of a simulation that involves transferring data. For instance, when a node reports its SOC to the scheduler, the scheduler assigns a job to a node, a node sends the result of a finalized job, etc. We obtained the energy consumption information with a power monitor hardware and a set of experiments that measure the cost of sending data through a WiFi connection using the TCP protocol. Our measurements with a Samsumg I5500 throw a value of 0.005576 Joules/Kb sent with excellent signal strength, which

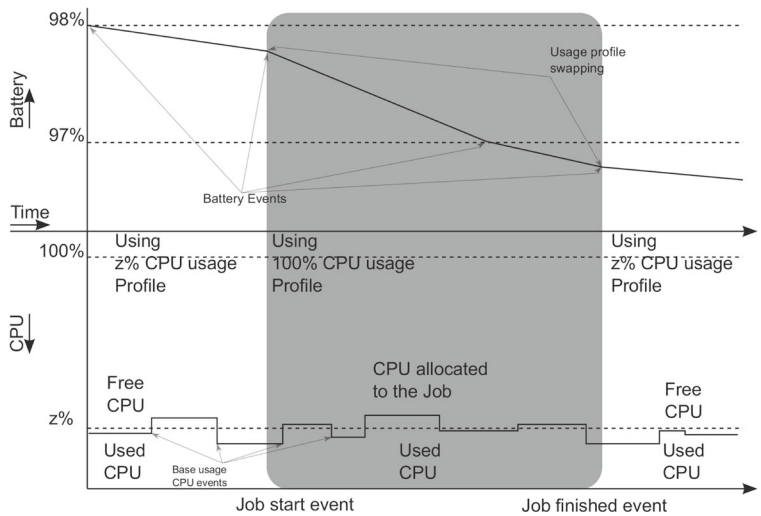


Fig. 5 Usage profile swapping when simulating the execution of jobs in a node

is very close to the value of 0.005 Joules/Kb reported in [44] with a Google G1 phone and the Nexus One. Through this network-aware extension to the original simulator [7], we are able to fairly measure the energy consumption overhead derived from communication and the administration protocols of the different scheduling approaches, e.g. with and without re-balancing.

4.2 Simulated Scenarios

The design of experimental scenarios aims at comparing the performance of the two-phase scheduling approach using the proposed energy-aware criteria by considering several variables including different MVR instances, i.e. arrangements of mobile devices, jobs sizes and base usage profiles. The rest of this section provides detailed information about these variables and values considered.

An MVR instance defines an arrangement of mobile devices that contribute with CPU cycles for the execution of Grid jobs. We define different MVR instances with the same number of mobile devices (100) and similar aggregated computing capacity (ACC) but energetically different. In theory, the ACC of a MVR could be expressed as the sum of all devices individual computing capacity (ICC), which in turn could be defined as $MFLOPS * batteryDuration$. The MFLOPS indicate the number of Mega Float point Operations a device is able to perform per second, while $batteryDuration$ is the maximum time, from full-charged battery, the device is able to run at 100 % of CPU usage. Same devices model are expected to be characterized by the same $MFLOPS$ and $batteryDuration$. The devices models used in our simulated MVR instances are Acer Iconia Tab A100, ViewSonic ViewPad 10s and Samsung I5500 whose ICC values are $1.6E^6$, $0.85E^6$ and $0.266E^6$ respectively. Then, MVR instances ACC equation is $ACC = 1.6E^6 * X + 0.85E^6 * Y + 0.266E^6 * Z \pm \varepsilon$, where X, Y and Z are the number of A100, ViewPad 10s and I5500 nodes respectively.

When creating MVR instances targeting the same ACC value but with the constraints that total number of nodes, i.e. $X + Y + Z$, should be the same and X, Y and Z should be positive integers, an error ϵ appears. That error, for the MVR instances created, is ±188,000 MFLOP, which represent a difference of approximately 0.24 % in the computing capacity. Table 4 outlines the values of X, Y and Z for the three MVR instances created along with their ACC values and total number of Joules.

Another variable of the simulated scenarios is Job size, which is expressed in Million Float point Operations (MFLOP). Long jobs size varies between 19,393.65 and 59,180.95 MFLOP while short jobs size is between 3,232.27 and 9,696.82 MFLOP. The long and short jobs sets were generated following a continuous uniform distribution, so that the various job sizes are equally probable. The base of this decision is to assure more heterogeneity in the quantity of jobs with different sizes than when using, for instance, a Normal distribution. The long jobs and short jobs sets sizes are 1,910 and 11,608, respectively. These values relate to the ACC of MVR instances. The idea was to exceed the MVR ACC so that the hundred percent of finalized jobs is avoided, which would hinder the comparison of the schedulers performance. Table 4 shows the ranges of expected number of finalized jobs according to the MVRs ACC values, which were calculated by dividing the MVR ACC by the job size average.

With the aim of adding more realism to the simulated scenarios, jobs were generated with input and output sizes because transferring data impacts on the available energy, and in turn, the computing capacity of mobile devices. Input size represents the data structures and arguments that need to be transferred to the node in charge of executing a job, while output size represents the job result size transferred to the proxy. The input and output data sizes were fixed values, i.e. they do not represent a variable of simulated scenarios, and were set to 1,024 bytes and 4 bytes respectively for all jobs. Those numbers are reasonable for CPU-bound jobs, which is the type of targeted applications of the proposed schedulers.

Another variable of the simulated scenarios is the base usage profile. This is the CPU usage caused by software other than Grid jobs, and comprises the usage derived from the device OS functioning and the device owner applications. We defined two values for this variable: *dedicated* and *non-dedicated*. A dedicated base usage profile means that the mobile device CPU usage is 0 % almost all the time, i.e. only the OS activity introduces slight variations of the CPU usage. On the contrary, a non-dedicated base usage profile means that the mobile device CPU usage fluctuates between 0 %, 30 % and 75 %, i.e. simulating

Table 4 Variables and values of the simulated scenarios

Variable	Values	Description
MVR instance	MVR1	Nodes: 10 A100, 64 ViewPad 10s, 26 I5500. ACC: 77.320 TFLOP. Energy: 6,445,008 J
	MVR2	Nodes: 20 A100, 41 ViewPad 10s, 39 I5500. ACC: 77.220 TFLOP. Energy: 5,035,752 J
	MVR3	Nodes: 30 A100, 18 ViewPad 10s, 52 I5500. ACC: 77.132 TFLOP; Energy: 3,626,496 J
Jobs size	Short	1,910 jobs in a range of [3,232.27 - 9,696.82] MFLOP
	Long	11,608 jobs in a range of [19,393.65 - 59,180.95] MFLOP
Base profile	Dedicated	no user activities (use 0 % of CPU)
	Non-dedicated	owner CPU usage varies between 0 %, 30 % and 75 % based on the probabilistic usage model of [23]

owner interaction. The fluctuation is not arbitrary but responds to the usage model presented in [23] where the authors studied the mobile device usage of 255 users during 7-28 weeks. The model derived comprises probabilistic distributions that best fitted the observed data and characterizes usage sessions, i.e. session length and time interval between sessions. That model was used to generate non-dedicated base usage profiles.

The simulated scenarios are twelve and arise from the combinations of all values for all variables presented in the second column of Table 4. The next subsection presents the results of all simulated scenarios for all of the schedulers combinations described.

4.3 Simulation Results

Schedulers performance is measured in number of finalized jobs. The more the jobs a scheduler finalizes, the more energy-efficient it is. The terms “device” and “node” will be used interchangeably.

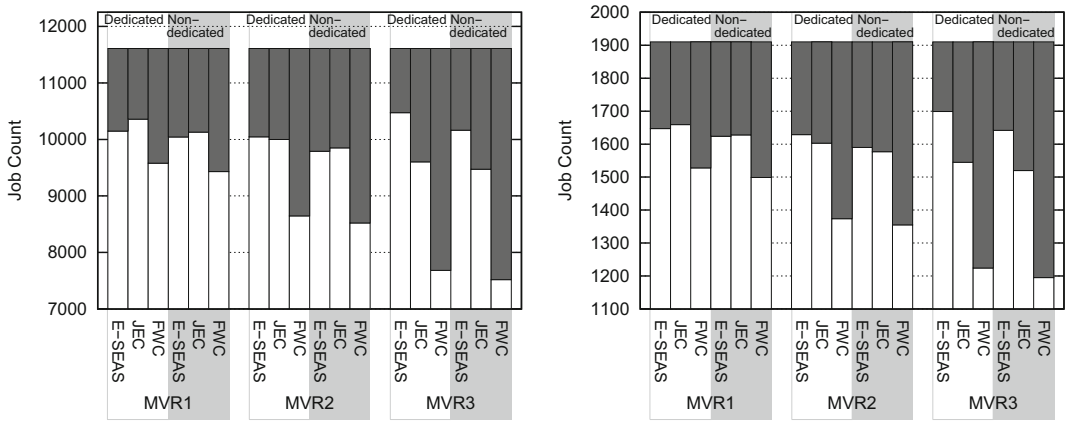
The results are organized as follows: Section 4.3.1 presents the performance analysis of the first phase for all energy-aware criteria and an analysis of job allocations by device type. Section 4.3.2 presents the results of the second phase, which are contrasted with those of the first phase. Finally, Section 4.3.3 presents a discussion of the best energy-aware criteria via an efficiency analysis.

4.3.1 First Phase: Evaluation of Ranking-Based Energy-Aware Criteria

The first phase results for short job scenarios shown in Fig. 6a describe similar performance patterns to those of long jobs scenarios shown in Fig. 6b. Furthermore, results of non-dedicated scenarios decrease in a range of 1-3 % with respect to dedicated scenarios and show similar performance patterns. For these reasons, Figs. 6a and b are enclosed into a single analysis. The best performance alternates between JEC-based and E-SEAS-based schedulers while FWC-based scheduler is always relegated to the third place. However, a cross-MVR analysis reveals that the number of finalized jobs by JEC-based and FWC-based schedulers notably degrades as more A100 devices integrate the MVR instance. On the contrary, the performance of E-SEAS-based scheduler slightly increases with the presence of more A100 devices, or at least, its performance is less affected.

We explore the schedulers assignments through the job-to-node-type allocations snapshot shown in Fig. 7, where only dedicated results are presented since the non-dedicated ones only differ in the existence of relatively more non-finalized jobs. In Fig. 7a it is observed that non-finalized jobs are mainly concentrated in A100 devices and a relative small quantity in I5500 devices. This shows that E-SEAS-based scheduler tends to overload the devices with high MFLOPS values. This fact helps to explain why, in the cross-MVR comparison with increasing number of A100 devices, the E-SEAS-based scheduler improves its performance. On the contrary, JEC-based and FWC-based schedulers allocations as shown in Fig. 7b and c respectively, tend to overload devices with low MFLOPS values, i.e. I5500 devices. Then, with the increasing number of A100 devices, i.e. with higher MFLOPS, the remaining computing cycles are underused by these last two schedulers.

The cause of these behaviors is associated to the node rank values obtained with each criterion. When analyzing rankings derived from E-SEAS we saw, e.g. that for an I5500 node to rank higher than an A100 node, the former should have more than 8 times



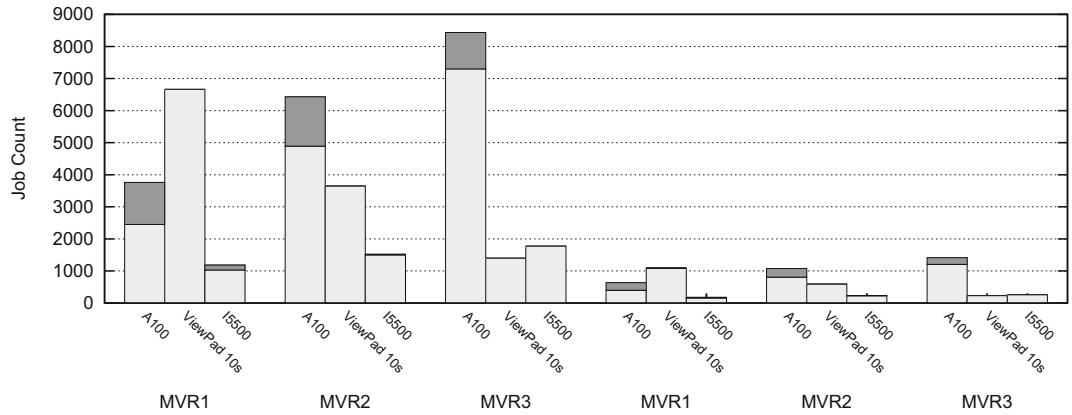
(a) Short job scenarios (b) Long job scenarios

Fig. 6 First phase energy-aware criteria performance (*white bars:* finalized jobs; *grey bars:* non finalized jobs)

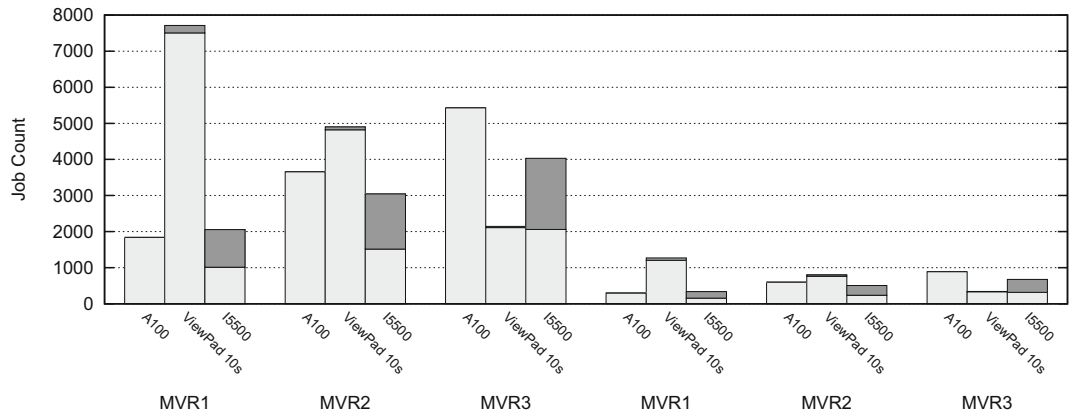
the remaining battery percentage of the A100. For example, by assuming that both nodes have not been assigned with any jobs yet, when the I5500 has approximately 40 % of remaining battery, the A100 needs only 5 % of its remaining battery to rank equal. That makes an E-SEAS-based scheduler prone to saturate with jobs the nodes with high MFLOPS first, and then those nodes with less MFLOPS passing through the intermediate ones. The opposite situation occurs for node rankings made with JEC. Again, assuming the case in which the weakest and the strongest nodes are compared and none have been assigned any job yet, for an I5500 node to rank better than an A100 node, the former needs approximately more than 2.3 times the remaining battery percentage of the latter. This is a condition easier to meet for an I5500 node than when it is ranked with the E-SEAS. Thus, a JEC-based scheduler tend to assign jobs first to the weak nodes and then to the strong ones. However, this allocation logic seems to create bigger underused periods of CPU cycles –i.e. sub-exploited states– than E-SEAS as ICC disparity among of nodes increases. The behavior of FWC is quite similar to JEC. The biggest difference is that the multiplier factor of remaining battery percentage, which weak nodes need to overcome to rank better than strongest nodes, varies as the time passes. For instance, for short jobs, this value starts with 1, because there is no information about the time a job last until the strongest nodes finalize their first allocated job. After that occurs, the multiplier factor changes to 2 because weakest nodes are still executing the first assigned job and the FWC ranking formula considers them to be in the middle of the execution time. All in all, the multiplier factor increases in multiple of 2 until 8 because the weakest node is 8 times weaker than the strongest one. This happens because the formula of FWC is based on pure dynamic components while E-SEAS and JEC formulas use a static component, i.e. MFLOPS and $JobEnergyConsumptionRate$ respectively.

4.3.2 Second Phase: Evaluation of Job Stealing with Energy-Aware Criteria

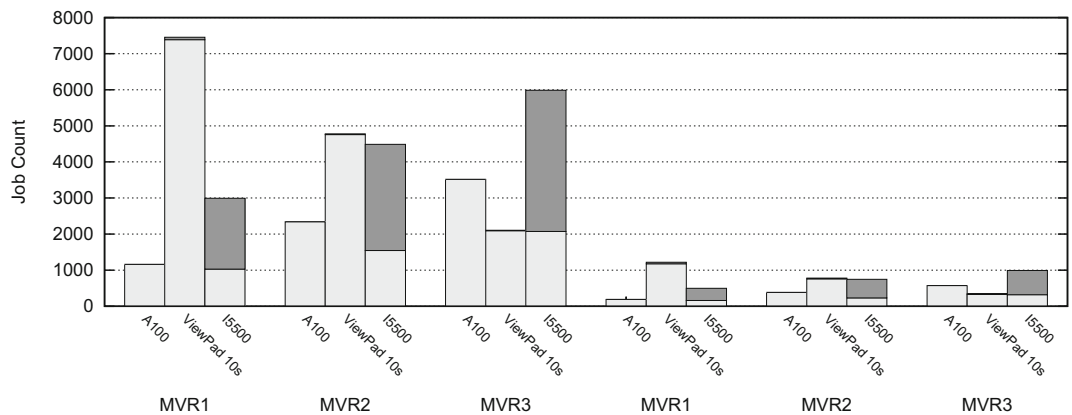
Below we report the results of the scheduling second phase and compare them with those achieved in the first phase. A separate analysis is presented for each energy-aware criterion. First phase results are indicated with the name of the criterion –e.g. E-SEAS, JEC, FWC–. Second phase results are indicated with a notation that includes JS (referred to Job Stealing) followed by a letter that indicates the victim selection strategy, i.e. W for Worst Randing-Aware Strategy and B for Best Ranking-Aware Strategy, and lastly another letter to indicate the offloading policy, i.e. E for Exponential policy and F for Fixed policy. When referring to the second phase, Job Stealing configuration or re-balancing phase results should be read the same. Figure 8a shows the two-phase scheduling performance using E-SEAS for short job scenarios. Notice that re-balancing through any of the Job Stealing configurations using E-SEAS as node ranking criterion does not improve the performance of the first phase also performed using E-SEAS. The cause is analyzed



(a) E-SEAS short job and long job scenarios



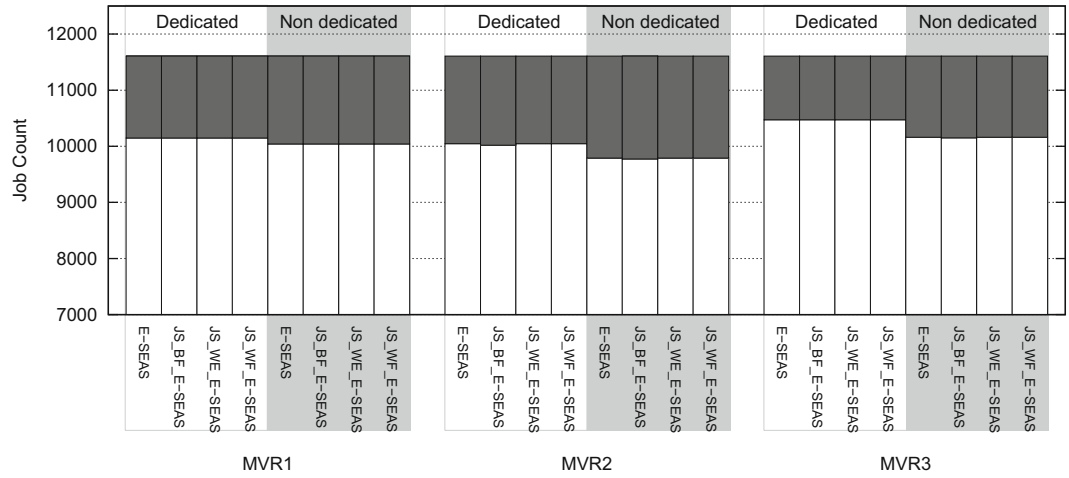
(b) JEC short job and long job scenarios



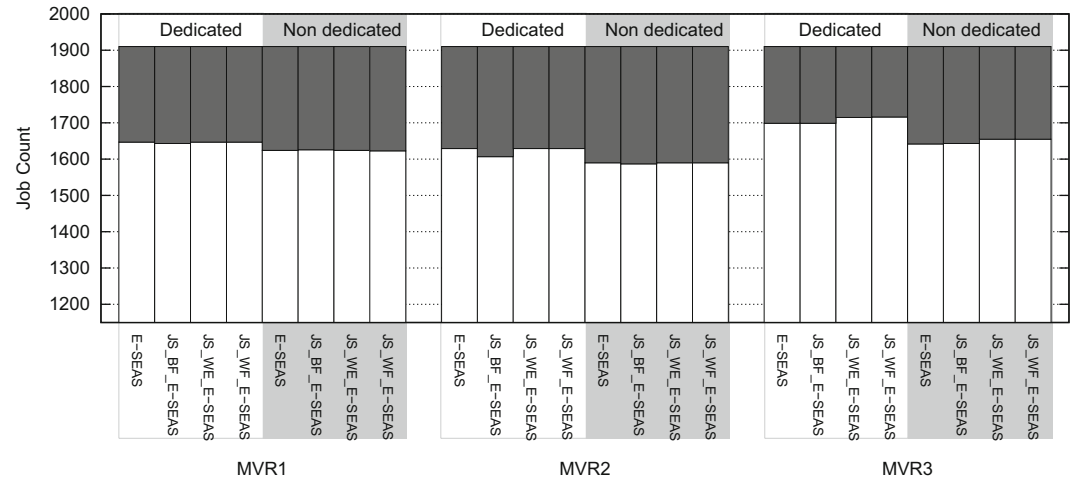
(c) FWC short job and long job scenarios

Fig. 7 First phase allocation snapshot (Bars on the *left half* represent short job scenarios; bars on the *right half* represent long job scenarios. *Light grey bars:* finalized jobs; *Dark grey bars:* non finalized jobs)

later, but in short we can anticipate that this relates to the nodes characteristics that E-SEAS overloads first. The results of long job scenarios, depicted in Fig. 8b, are somewhat different. Re-balancing slightly



(a) Short job scenarios



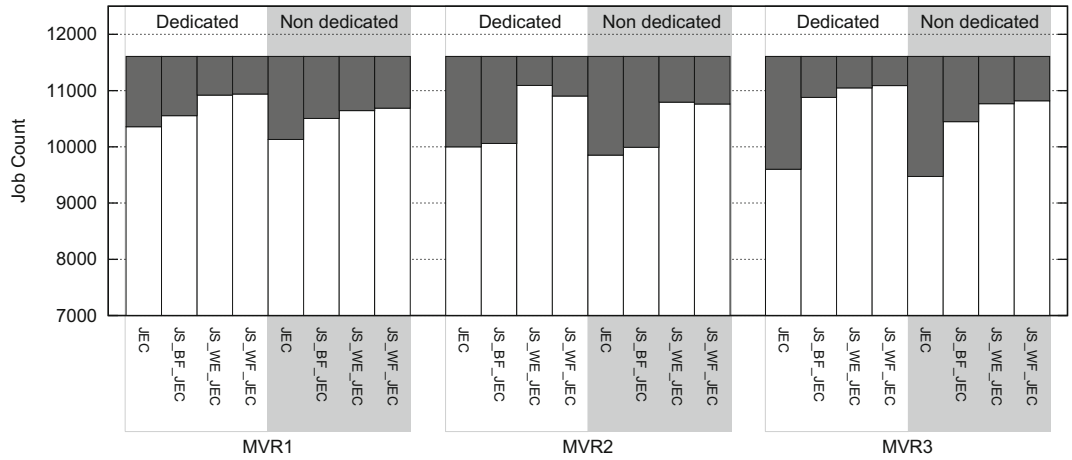
(b) Long job scenarios

Fig. 8 E-SEAS performance: *First* and *second* phases (*white bars:* finalized jobs; *grey bars:* non finalized jobs)

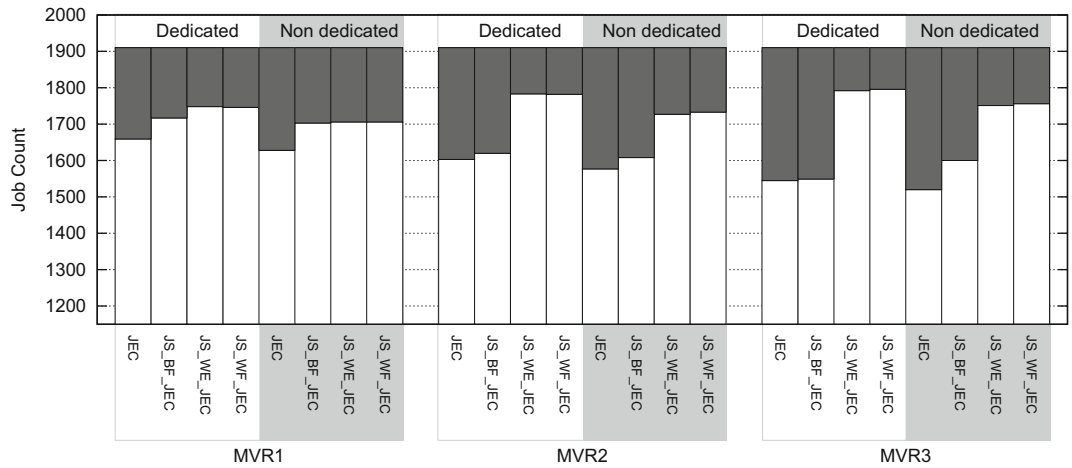
improved the performance achieved by the first phase, particularly for the MVR 3. In this case, JS_WE_E-SEAS and JS_WF_E-SEAS were the second phase configurations that slightly boost the performance of the first phase.

Now we analyze the results depicted in Fig. 9 of the two-phase scheduling performance using JEC. For all short jobs scenarios shown in Fig. 9a, all JS configurations using JEC significantly improve the first phase performance using JEC. The most efficient configurations are JS_WF_JEC and JS_WE_JEC. Furthermore, unlike JEC first phase results, the performance of these configurations does not decrease with the increment of powerful devices, meaning that re-balancing using JEC suppress that undesired effect, i.e. renders the scheduling performance less influenced by such type of devices, at least with the MVR instances studied. Regarding long job scenarios results, Fig. 9b reveals that the re-balancing has a similar effect to that observed in short job scenarios.

Figure 10 depicts the behavior of two-phase scheduling using FWC. For short jobs scenarios shown in Fig. 10, all Job Stealing configurations improve the first phase using FWC. JS_WF_FWC and JS_WE_FWC obtained the highest performance boost while JS_BF_FWC the lowest one. Once again, long



(a) Short job scenarios



(b) Long job scenarios

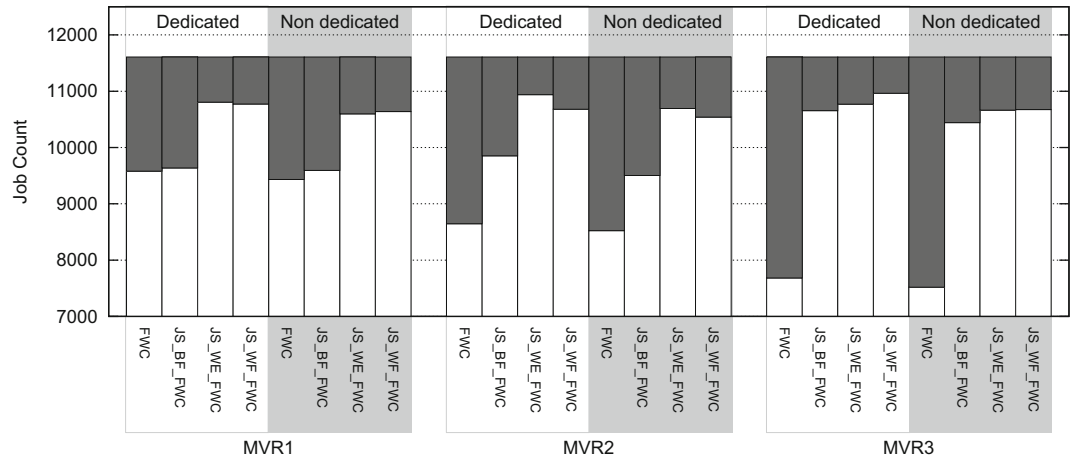
Fig. 9 JEC performance: *First* and *second* phases (*white bars:* finalized jobs; *grey bars:* non finalized jobs)

jobs scenarios results of Fig. 10a replicate performance gains of short job scenarios.

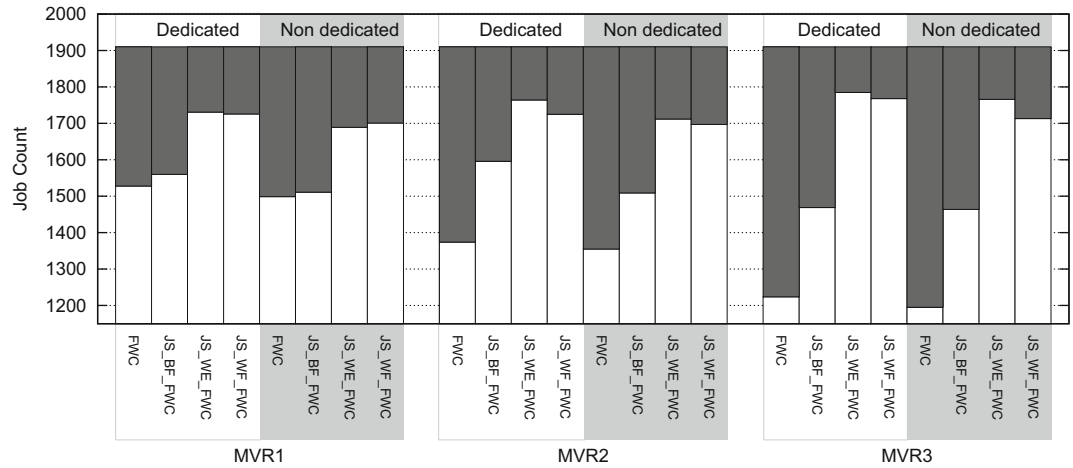
The negligible performance boost of E-SEAS relates to the most overloaded nodes type and the time those nodes stay connected to the proxy. If overloaded nodes leave the MVR before least loaded or balanced nodes, then steals never occur or its frequency is low. That effect is observed in the first phase scheduling using E-SEAS, where the overloaded nodes are the A100 nodes and leave the MVR first. It is worth mentioning that it is assumed that queued (not started) jobs of nodes that leave the MVR are not re-scheduled and are considered as non-finalized jobs. We leave that function to a fault tolerance mechanism that is out of the scope of this work.

4.3.3 Discussion

Now, we describe a performance cross-comparison of the best two-phase schedulers instantiations based on energy-aware criteria along with energy implications. For short job scenarios, irrespective of the MVR instance, E-SEAS-based schedulers are always worse than the rest of the schedulers. From 11,608 jobs, they finalized around 617 (5.3 %) and 1,044 (8.9 %), 646 (6 %) and 1,003 (9.3 %) less jobs than the best scheduler in dedicated and non-dedicated scenarios respectively. JEC-based schedulers present the best average performance for all scenarios. FWC-based schedulers average performance is barely inferior to that of JEC-based schedulers, i.e they finalized



(a) Short job scenarios



(b) Long job scenarios

Fig. 10 FWC performance: *First* and *second* phases (*white bars:* finalized jobs; *grey bars:* non finalized jobs)

between 126 (1 %) and 151 (1.3 %), 91 (0.8 %) and 156 (1.4 %) less jobs in dedicated scenarios and non-dedicated scenarios. Similarly, for long job scenarios the relative positions of schedulers is maintained with respect to those identified for short jobs, i.e. JEC-based performs the best followed by FWC-based and then by E-SEAS-based schedulers. In numbers, from 1,910 long jobs, in dedicated scenarios, E-SEAS-based schedulers finalized between 80 (4.2 %) and 184 (9.6 %) less jobs than JEC-based schedulers while the difference of finalized jobs between FWC-based and JEC-based schedulers is between 11 (0.57 %) and 19 (0.99 %). In non -dedicated scenarios, E-SEAS-based schedulers finalized between 83 (4.8 %) and 143 (8.2 %) less jobs than JEC-based schedulers, while FWC-based schedulers finalized between 17 (0.99 %) and 53 (3 %) less jobs than JEC-based schedulers.

The highest throughput two-phase schedulers are JS_WE_JEC and JS_WF_JEC, and now are evaluated with the steal revenue metric (sixth column of Table 5), defined as the extra finalized jobs over steals produced: $\frac{(extra\,Finalized\,Jobs)}{\#of\,Steals}$. The extra finalized jobs is the difference between the finalized jobs in the second phase and those in the best first phase. Steals, though necessary to increase the number of finalized jobs, produce overhead due to the extra network activity from exchanging steal messages and jobs data

Table 5 Best first phase and best second phase schedulers comparison based on their steals revenue values

Jobs size	MVR instance	Base profile	Best 1st phase	Best JEC-based 2nd phase		# JEC-based JS steals		Steals revenue	
				JS_WE	JS_WF	JS_WE	JS_WF	JS_WE	JS_WF
Short jobs	MVR1	Dedicated	10,358 (JEC)	10,919	10,939	4,674	619	0.120	0.938
		Non-dedicated	10,130 (JEC)	10,644	10,688	3,810	596	0.135	0.981
	MVR2	Dedicated	10,047 (E-SEAS)	11,091	10,903	9,824	1,220	0.106	0.701
		Non-dedicated	9,851 (JEC)	10,793	10,761	6,189	1,015	0.152	0.896
	MVR3	Dedicated	10,472 (E-SEAS)	11,049	11,089	16,138	1,914	0.062	0.322
		Non-dedicated	10,162 (E-SEAS)	10,768	10,818	10,735	1,548	0.056	0.423
Long jobs	MVR1	Dedicated	1,659 (JEC)	1,748	1,746	221	104	0.402	0.836
		Non-dedicated	1,628 (JEC)	1,706	1,706	211	92	0.369	0.848
	MVR2	Dedicated	1,629 (E-SEAS)	1,783	1,782	511	210	0.301	0.728
		Non-dedicated	1,590 (E-SEAS)	1,727	1,733	378	171	0.362	0.836
	MVR3	Dedicated	1,699 (E-SEAS)	1,792	1,796	687	319	0.135	0.304
		Non-dedicated	1,642 (E-SEAS)	1,751	1,756	590	271	0.184	0.421

input transfers. Besides, since a node sends many steal requests until it actually initiates the execution of any stolen job, there are unused CPU cycles, i.e. a sub-exploited state is generated. So, the less steals a job stealing configuration produces per extra finalized job, the more efficient it is considered. Table 5 shows in the last column the steal revenue value along with the data used in the calculation. The values of the fourth column and fifth columns (with its sub-columns) are number of finalized jobs. Fifth sub-columns show only values for JS_WE_JEC and JS_WF_JEC because these are the schedulers with the highest throughput.[6]

The higher the steal revenue, the more efficient the Job Stealing configuration is. JS_WF_JEC configuration outperforms JS_WE_JEC in 7 out of 12 scenarios. There is one scenario where both configurations have the same throughput and JS_WE_JEC wins in the remaining 4 scenarios. However, the steal revenue of JS_WF_JEC is better than JS_WE_JEC for all scenarios. This is because the offloading policy of JS_WF produces much less steals than that of JS_WE. The quantity of steals produced could serve to guide the selection of the best balanced scheduler for the case of communication channels with lower/higher latencies and energy costs. Table 6 outlines the performance implications of the best first phase and the best second phase schedulers from the perspective of the energetic constraints imposed by the MVR instances. The FJ/E increases as total Joules of MVR instances decreases (see Table 4). Then, MVR1 achieves less FJ/E than MVR2, and the latter, in turn, achieves less FJ/E than MVR3. These values align with the fact that the $ACC/TotalEnergy$ relation of MVR1 is bigger than that of MVR2, and the latter bigger than that of MVR3. With regard to improvements in energy utilization of the second phase over the first phase, it is over 4.57 % in all scenarios. Those improvements seem to be affected by the MVR *homogeneity*, which was quantified as the standard deviation of the sum of FLOPS that represent each group of mobile devices over the MVR ACC. The less that value is, the more heterogeneous the MVR is. Then, MVR2 improvement is the highest, followed by those of MVR3 and MVR1. Apart from the improvement differences, attributable to MVR homogeneities, it is observed from Table 5 fifth column that schedulers performance between MVR instances does not vary in more than 1.5 % and 2.6 % of finalized short and long jobs respectively. This suggests that schedulers adapt to MVRs with similar computing capacity but different $ACC/TotalEnergy$ relation without compromising the final throughput.

[6]There is one scenario where the second most efficient scheduler was JS_WE_FWC. However, it was not included because its performance metric value is very inferior to the JS_WF_JEC scheduler, which is the next better positioned scheduler after JS_WE_FWC.

Table 6 FJ/E in e^{-4} and energy exploitation improvement in % of 2nd phase with regard to 1st phase

MVR: Homogeneity	Dedicated				Non-Dedicated			
	Short		Long		Short		Long	
	1st Phase	2nd Phase	1st Phase	2nd Phase	1st Phase	2nd Phase	1st Phase	2nd Phase
MVR1: 32.60	16.07135	16.97283	2.57409	2.71218	15.71759	16.58338	2.52599	2.64701
Improvement	5.31 %		5.09 %		5.22 %		4.57 %	
MVR2: 17.33	19.95134	22.02452	3.23487	3.54068	19.56212	21.43275	3.15742	3.44139
Improvement	9.41 %		8.64 %		8.73 %		8.25 %	
MVR3: 25.04	28.87636	30.57773	4.68496	4.95244	28.02154	29.80287	4.52779	4.84214
Improvement	5.56 %		5.40 %		6.06 %		6.49 %	

5 Conclusions and Future Works

We have proposed an hybrid two-phase approach for scheduling CPU-bound jobs in MVRs considering varying job requirements, device combinations and available CPU cycles. The goal was to assess the synergy of a centralized first phase with a decentralized second phase. The first phase was instantiated with a ranking-based scheduling while the second phase with Job Stealing techniques. Both phases, in turn, use novel energy-aware criteria as node rankers.

The first phase best throughput was achieved by the E-SEAS-based and the JEC-based schedulers. However, the E-SEAS-based scheduler throughput did not improve when re-balancing was applied. On the contrary, JEC-based and FWC-based first phase schedulers highly improved their throughput with the re-balancing phase. Nonetheless, despite FWC-based schedulers were very competitive in the second phase, their performance in the first phase was not. This outcome is particularly undesirable if the proxy has a downtime and the second phase cannot be initiated because the resulting allocation would be under the throughput of JEC-based or E-SEAS-based schedulers. For this reason, JEC-based schedulers are considered the best average schedulers because they are competitive in the first phase and the best in the second phase of the scheduling. Besides, JEC criterion contemplates the reduction of a node computing capacity caused by the aging problem of Li-ion batteries [45] because, unlike MFLOPS, the $JobEnergyConsumptionRate$ value associated to a device varies if it is obtained before and after its battery suffers the aging effect.

From the second phase evaluation, it is observed that the WRAS strategy for selecting a victim obtained always better performance than the BRAS strategy, and w.r.t the policy used to determine the job steals quantity, the Fixed policy registered very superior steals revenue values than the Exponential policy. The latter generates much more useless steals that should be avoided, e.g. as the size of job input increases, because they incur in higher energy misuse.

A general conclusion of this two-phase scheduling approach is that a re-balancing phase performed with the Job Stealing technique helps to increase the FJ/E achieved with a single phase scheduling, even when considering the energy footprint due to network activity of nodes. However, the node ranking criteria used for the first phase allocation heavily influenced the performance boost that could be obtained with the re-balancing phase. More explicitly, the most effective instantiation of the re-balancing phase occurred when the first allocation phase left an active distributed pool of jobs. By active pool of jobs we mean jobs queued in nodes whose time to leave from the MVR is as far in time as possible. This increases the possibility to produce steals that in the best case will take advantage of underused periods of computing cycles to finalize more jobs.

We are extending this work in several directions. First, given the momentum gained by Swarm Intelligence algorithms and their application to scheduling in distributed computing [46], we plan an instantiation of the second phase with such algorithms and compare the performance achieved with the Job Stealing techniques. Furthermore, instantiations of the approach other than independent CPU-bound jobs could be

provided to tackle other type of jobs, e.g. inter-dependent CPU-bound jobs (workflows) and network-bound jobs. We also plan to add mobility criteria to the scheduler, so that MVRs do not assign jobs to mobile devices that will not be reachable by the time the job execution is completed. Furthermore, this can also be a stealing strategy.

We will study new scheduling criteria targeting other performance metrics. Given the benefits of including energy rates derived from benchmarking, new criteria could be proposed by combining quantity of jobs executed with the time employed to execute these jobs. In this way, schedulers focused on improving the flowtime could be developed. Another metric, which is interesting to be considered is fairness, or the balanced distribution of work load among all mobile devices of an MVR. Given that the utility of mobile devices is governed by the availability of energy provided by their battery, a fair scheduler is one that uses, from all mobile devices, an equal proportion of their available energy in executing jobs of the MVR.

Another future work is to analyze the proposed local scheduling criteria for MVRs in the context of global scheduling criteria. More specifically, criteria applied by local schedulers that arrange job mappings based on the capabilities of devices within an MVR could be replicated into global schedulers criteria, which operate at the Grid level. 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 (e.g. round robin) and our proposed mobile schedulers at the local, MVR level could be used. The detection of over-exploited states is also a topic in the future works agenda since it would allow a scheduler that operates, e.g. at intra-MVR level, to offer, to a meta-scheduler that operates at a higher level, e.g. inter-MVR level, a job execution service that does not overestimate the computing resources and achieve at the same time the highest rate of finalized jobs.

Acknowledgments We acknowledge the financial support by ANPCyT through grants PICT-2012-0045 and PICT-2013-0464.

References

1. Huynh, D., Knezevic, D., Peterson, J., Patera, A.: High-fidelity real-time simulation on deployed platforms. Comput. Fluids **43**(1), 74–81 (2011)
2. Ryabinin, K., Chuprina, S.: Adaptive scientific visualization system for desktop computers and mobile devices. Procedia Computer Science **18**(0), 722–731 (2013)
3. Shiraz, M., Gani, A., Shamim, A., Khan, S., Ahmad, R.: Energy efficient computational offloading framework for mobile cloud computing. Journal of Grid Computing **13**(1), 1–18 (2015)
4. Khan, A.u.R., Othman, M., Khan, A., Abid, S., Madani, S.: Mobibyte: An application development model for mobile cloud computing. Journal of Grid Computing, 1–24 (2015)
5. Rodriguez, J.M., Mateos, C., Zunino, A.: Are smartphones really useful for scientific computing? Lect. Notes Comput. Sci **7547**, 38–47 (2012)
6. Karan, O., Bayraktar, C., Gümüşkaya, H., Karlik, B.: Diagnosing diabetes using neural networks on small mobile devices. Expert Syst. Appl. **39**(1), 54–60 (2012)
7. Rodriguez, J.M., Mateos, C., Zunino, A.: Energy-efficient job stealing for cpu-intensive processing in mobile devices. Computing **96**(2), 87–117 (2014)
8. Rodriguez, J.M., Zunino, A., Campo, M.: Mobile Grid Seas: Simple Energy-Aware Scheduler. In: 3Rd High-Performance Computing Symposium (2010). 39Th JAIIO
9. Ghosh, P., Das, S.K.: Mobility-aware cost-efficient job scheduling for single-class grid jobs in a generic mobile grid architecture. Futur. Gener. Comput. Syst. **26**(8), 1356–1367 (2010)
10. Rodriguez, J.M., Zunino, A., Campo, M.: Introducing mobile devices into grid systems: a survey. International Journal of Web and Grid Services **7**(1), 1–40 (2011)
11. Li, C., Li, L.: Tradeoffs between energy consumption and qos in mobile grid. J. Supercomput. **55**, 367–399 (2011)
12. Aron, J.: Harness unused smartphone power for a computing boost. New Scientist, 215 (2880)
13. Li, W., Wu, J., Zhang, Q., Hu, K., Li, J.: Trust-driven and qos demand clustering analysis based cloud workflow scheduling strategies. Clust. Comput., 1–18 (2014)
14. Callou, G., Maciel, P., Tavares, E., Andrade, E., Nogueira, B., Araujo, C., Cunha, P.: Energy consumption and execution time estimation of embedded system applications. Microprocess. Microsyst. **35**(4), 426–440 (2011)
15. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem – overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3), 36:1–36:53 (2008)
16. Serrano, P., de la Oliva, A., Patras, P., Mancuso, V., Banchs, A.: Greening wireless communications: Status and future directions. Comput. Commun. **35**(14), 1651–1661 (2012)
17. Arroqui, M., Mateos, C., Machado, C., Zunino, A.: RESTful web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones. Comput. Electron. Agric. **87**(0), 14–18 (2012)

18. Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., Singh, J.P. In: Proceedings of the 21St International Conference on World Wide Web, WWW'12. Who Killed My Battery?: Analyzing Mobile Browser Energy Consumption, pp. 41–50. ACM, New York (2012)
19. Mahapatra, R., Domenico, A.D., Gupta, R., Strinati, E.C.: Green framework for future heterogeneous wireless networks. Comput. Netw. **57**(6), 1518–1528 (2013)
20. Nicolaos, A., Vasileios, K., George, A., Harris, M., Angeliki, K., Costas, G.: A data locality methodology for matrix-matrix multiplication algorithm. J. Supercomput. **59**, 830–851 (2012)
21. Hermelin, D., Rawitz, D., Rizzi, R., Vialette, S.: The minimum substring cover problem. Information and Computation/information and Control - IANDC **206**, 1303–1312 (2008)
22. Baron, R., Lioubashevski, O., Katz, E., Niazov, T., Willner, I.: Elementary arithmetic operations by enzymes: a model for metabolic pathway based computing. Angew. Chem. Int. Ed. **45**, 1572–1576 (2006)
23. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in smartphone usage. In: Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, pp. 179–194 (2010)
24. Busching, F., Schildt, S., Wolf, L.: Droidcluster: Towards smartphone cluster computing – the streets are paved with potential computer clusters. In: 2012 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 114–117 (2012)
25. Rodriguez, A.V., Mateos, C., Zunino, A.: Mobile Devices-Aware Refactorings for Scientific Computational Kernels. In: 13Th Argentine Symposium on Technology, AST 2012 (2012). 41Th JAIIO
26. Li, C., Li, L.: A multi-agent-based model for service-oriented interaction in a mobile grid computing environment. Pervasive and Mobile Computing **7**(2), 270–284 (2011)
27. Chunlin, L., Layuan, L.: Exploiting composition of mobile devices for maximizing user qos under energy constraints in mobile grid. Inf. Sci. **279**(0), 654–670 (2014)
28. Shah, S.C.: Energy efficient and robust allocation of interdependent tasks on mobile ad hoc computational grid, Concurrency and Computation: Practice and Experience
29. Wei, X., Fan, J., Lu, Z., Ding, K.: Application scheduling in mobile cloud computing with load balancing. J. Appl. Math (2013)
30. Shah, S., Park, M.S.: An energy-efficient resource allocation scheme for mobile ad hoc computational grids. Journal of Grid Computing **9**(3), 303–323 (2011)
31. Loke, S.W., Napier, K., Alali, A., Fernando, N., Rahayu, W.: Mobile computations with surrounding devices: Proximity sensing and multilayered work stealing. ACM Trans. Embed. Comput. Syst. **14**(2), 22:1–22:25 (2015)
32. Li, B., Pei, Y., Wu, H., Shen, B.: Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds. J. Supercomput., 1–28 (2015)
33. Shi, T., Yang, M., Jiang, Y., Li, X., Lei, Q.: An Adaptive Probabilistic Scheduler for Offloading Time-Constrained Tasks in Local Mobile Clouds. In: Ubiquitous and Future Networks (ICUFN), vol. 2014 Sixth International Conf on, pp. 243–248. IEEE (2014)
34. Castro, M.C., Kassler, A.J., Chiasserini, C.-F., Casetti, C., Korpeoglu, I.: Peer-to-peer overlay in mobile ad-hoc networks, pp. 1045–1080. Springer (2010)
35. Macone, D., Oddi, G., Pietrabissa, A.: Mq-routing: Mobility-, gps- and energy-aware routing protocol in MANETs for disaster relief scenarios. Ad Hoc Networks **11**(3), 861–878 (2013)
36. Torres, R., Mengual, L., Marban, O., Eibe, S., Menasalvas, E., Maza, B.: A management ad hoc networks model for rescue and emergency scenarios. Expert Syst. Appl. **39**(10), 9554–9563 (2012)
37. van Nieuwpoort, R., Wrzesinska, G., Jacobs, C.J.H., Bal, H.E.: Satin: A high-level and efficient grid programming model. ACM Trans. Program. Lang. Syst. **32** (3)
38. Xu, H., Yang, B.: An incentive-based heuristic job scheduling algorithm for utility grids. Futur. Gener. Comput. Syst. **49**(0), 1–7 (2015)
39. Hu, Y., Yurkovich, S.: Battery cell state-of-charge estimation using linear parameter varying system techniques. J. Power. Sources **198**(0), 338–350 (2012)
40. Mednieks, Z., Dornin, L., Meike, G.B., Nakamura, M.: Programming Android, 2nd Edn. Java Programming for the New Generation of Mobile Devices, O'Reilly Media (2012)
41. Shen, W.X., Chan, C.C., Lo, E.W.C., Chau, K.T.: Estimation of battery available capacity under variable discharge currents. J. Power Sources **103**(2), 180–187 (2002)
42. Khalaj, A., Lutfiyya, H., Perry, M.: The Proxy-Based Mobile Grid. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (eds.) Mobile Wireless Middleware, Operating Systems, and Applications, Vol. 48 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 59–69. Springer, Berlin (2010)
43. Calheiros, R.N., Ranjan, R., Beloglazov, A., de Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011)
44. Rice, A., Hay, S.: Measuring Mobile Phone Energy Consumption for 802.11 Wireless Networking. Pervasive and Mobile Computing **6**(6), 593–606 (2010)
45. Takeno, K., Ichimura, M., Takano, K., Yamaki, J.: Influence of cycle capacity deterioration and storage capacity deterioration on li-ion batteries used in mobile phones. J. Power. Sources **142**(1-2), 298–305 (2005)
46. Pacini, E., Mateos, C., García Garino, C.: Distributed job scheduling based on swarm intelligence: A survey. Comput. Electr. Eng. **40**(1), 252–269 (2014). 40th-year commemorative issue