CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* 2015; **00**:1–30 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

# Energy-Aware Task Allocation for Small Devices in Wireless Networks

Carmela Comito[1], Deborah Falcone[2], Domenico Talia[2], Paolo Trunfio[2]

[1]*ICAR-CNR, Rende (CS), Italy*
[2]*DIMES, University of Calabria, Rende (CS), Italy*

**SUMMARY**

**The continuous advances in wireless networking and mobile computing technologies have paved the way to the spreading of new classes of distributed applications running on networks of small devices such as smartphones and tablets. An issue that still prevents a wider implementation of distributed applications in wireless networks is the lack of task allocation strategies addressing both the energy constraints of small devices and the decentralized nature of wireless networks. In this paper we focus on this two-fold issue by proposing an energy-aware scheduling strategy for allocating computational tasks over a wireless network of small devices in a decentralized but effective way. The main design principle of our scheduling strategy is finding a task allocation that prolongs the total lifetime of the network and maximizes the number of alive devices by balancing the energy load among them. A simulation analysis has been performed to assess the performance of the proposed strategy in different network and application scenarios. The results show that by using the proposed energy-aware task allocation approach, the network lifetime is extended and the number of alive devices is significantly higher compared to alternative scheduling strategies, while meeting application-level performance constraints.**

KEY WORDS: Energy-Aware; Scheduling; Task Allocation; Mobile computing; Distributed Systems

## 1. Introduction

Recent developments on hardware and software technologies of wireless networks and mobile devices, such as smartphones and tablets, together with the general reduction of their costs, have significantly raised the interest on applications combining small devices and wireless

E-mail: ccomito@dimes.unical.it (C. Comito), dfalcone@dimes.unical.it (D. Falcone), talia@dimes.unical.it (D. Talia), trunfio@dimes.unical.it (P. Trunfio)

scenarios [14]. In particular, an increasing number of distributed applications is appearing, in which small mobile devices collaborate with each other in a peer-to-peer (or *mobile-to-mobile*) style, where the devices can act both as task requestors (clients) and task executors (servers) [34]

An issue that still prevents a wider implementation of distributed applications in mobile-to-mobile scenarios is the lack of task allocation strategies addressing both the energy constraints of battery-operated devices and the decentralized nature of wireless networks. In this paper we focus on this two-fold issue by proposing an Energy-Aware (EA) scheduling strategy for allocating computational tasks over a wireless network of small devices in a decentralized but effective way. Decentralization is achieved by clustering devices into local groups, named *clusters*. Such a cooperative architecture can be seen as a set of requestors, i.e., applications generating tasks to be executed, and a clustered set of resources, i.e., devices characterized by different levels of energy and processing capabilities, where tasks can be executed.

The proposed model may be applied to any network of mobile devices (e.g., smartphones) that share their energy to perform possibly critical tasks. Risk scenarios such as earthquakes, floods, fires, accidents, are situations in which mobile devices may cooperate to share their energy. The basic principle behind such cooperative scenarios is that, whenever an energy-limited device has a critical task to execute, this task can be assigned to another device that has enough resources to handle its execution. As a concrete use case, consider the case of a team of rescue workers in a given urban area hit by one of the above events. The mobile devices can be connected together to form an ad hoc network through which data generated by any node (e.g., smartphone's sensor readings) can be effectively spread to all other nodes in the network. Let us assume that, in order to monitor each other's position, every node periodically spreads its current location through the network, e.g., latitude/longitude data taken from the GPS of the device. In this way, every node locally collects data about the positions of the other devices over time. Using a trajectory pattern mining algorithm, a node may locally analyze this positioning data to discover the most frequent trajectories of the neighboring nodes, for example to find the paths that likely lead to a safe point at the moment. According with the proposed model, if the remaining energy of the device is low, the trajectory pattern mining task can be assigned to another device that has enough energy to handle it.

To make the most of all available resources, the EA strategy proposed in this paper tries to find a suitable distribution of tasks among clusters and individual devices. Specifically, the main design principle of our scheduling strategy is finding a task allocation that prolongs the total lifetime of a wireless network and maximizes the number of alive devices by balancing the energy load among them. To this end, the EA scheduler implements a two-phase heuristic-based algorithm. The algorithm first tries to assign a task locally to the cluster where the execution request has been generated, so as to maximize the cluster residual life. If the task cannot be assigned locally, the second phase of the algorithm is performed by assigning the task to the most suitable node all over the network of clusters, maximizing this way the overall network lifetime. We characterize the energy consumption of mobile devices defining an energy model in which the energy costs of both computation and communication are taken into account. Even though we consider a scenario involving mobile devices, we do not focus on mobility issues such as unstable links, asymmetric channels and changing topology. In particular, we assume that links are stable and topology does not change over time. This assumption holds

both in scenarios where the devices stay within the same area, and in scenarios where the devices move all together (i.e., group mobility). Considering more general mobility scenarios and related issues (e.g., unstable links) is out of the scope of this work.

There are several works in literature whose goal is minimizing the overall energy dissipation of the system [12, 13]. However, this goal does not capture the nature of cooperative networks of small devices. The reason is that minimizing the overall energy dissipation can lead to heavy use of energy-effective devices, regardless of their remaining energy. The consequent short lifetime of such devices will very likely compromise the system performance. This weakness is a major motivation of the proposed energy-balanced task allocation scheme.

An extensive simulation analysis has been performed to assess the performance of the proposed EA strategy in different network and application scenarios. The simulation results show that by using the proposed energy-aware task allocation approach, the network lifetime is extended and the number of alive devices is significantly higher compared to alternative scheduling strategies, while meeting application-level performance constraints. In details, our algorithm: (i) is effective in prolonging network lifetime by reducing the energy consumption; (ii) is able to complete a greater number of tasks in the same settings; (iii) in all the simulations, is able to keep alive all the devices thanks to its energy-balanced task allocation scheme.

The remainder of the paper is organized as follows. Section 2 describes the energy model. Section 3 presents the energy-aware task allocation scheme and related algorithms. Section 4 presents the performance results. Section 5 discusses related work. Finally, Section 6 concludes the paper.

## 2. Energy Model

Mobile nodes are battery powered, which makes energy a critical concern. Thus, the main aim in mobile networks is to conservatively consume the energy in order to increase the lifetime of the network.

Energy consumption of mobile devices depends on the computation and the communication loads. We define $\mathrm{E_i}$ as the energy consumption of device $\mathrm{d_i}$ in a time interval $\delta t$, which is the sum of energy consumption for communication, $\mathrm{ET_i}$, and computation, $\mathrm{EC_i}$, of all the tasks assigned to device $\mathrm{d_i}$ within time interval $\delta t$:

$$\mathrm{E_i} = \mathrm{EC_i} + \mathrm{ET_i} \tag{1}$$

To the scope of the validation of the proposed task allocation strategy, the energy consumption of the devices has to be established. This means that we have to be able to determine both the terms contributing to the energy depletion. Consequently, we need to establish some real values to use in the simulation such as the energy model matches reality.

In order to account the energy for computation we characterized the energy consumption behaviour of mobile devices when they perform computational tasks. More specifically, we experimentally measured the energy consumed for computation by the devices when executing a set of sample algorithms. The detailed experimental evaluation is provided in section 2.2.

We modeled the energy for communication by exploiting a well-known and widely-adopted energy characterization model [23, 24] that has been validated experimentally. Details are presented in the following section.

### 2.1. Communication Energy

To determine the energy consumed for communication by a node is necessary to distinguish the node state. The network interface of a mobile device can be in four states: (i) transmit mode; (ii) receive mode; (iii) idle mode (this is the default mode for ad-hoc network and in this state a node can transmit or receive); (iv) sleep mode (characterized by really low power consumption). In this state the interface can neither transmit nor receive until it is woken up and changes state. In mobile networks, nodes must always be ready to receive traffic from neighbors due to the absence of base station nodes. Thus, a network interface operating in ad-hoc mode can not be in a sleep mode but it has to continuously listen to the wireless channel consuming this way a constant idle energy power. Therefore, every node overhears every packet transmission occurring in its transmission range consuming this way energy uselessly. This idle energy consumption is referred to as *overhearing*. Due to overhearing, a new cost in the computation of per-packet energy consumption is introduced and it is the cost for discarding overheard packets. Therefore, to model the energy consumed for communication, the costs to send, receive and discard a packet must be included. Consequently, the energy consumed by a device $d_i$ for communication can be defined by the following equation:

$$ET_i = E_{send_i} + E_{receive_i} + E_{discard_i} \tag{2}$$

A packet may be sent through a broadcast or a point-to-point channel. With the former the packet is received by all hosts within the sender's transmission range; with the latter the packet is discarded by non-destination hosts.

According to [23, 24] we assume a commonly used energy model to evaluate the energy consumption behavior of the mobile network interface. Based on this model, the cost for a node to send or receive a message is modelled as a linear function. In this function there is a fixed cost associated with channel acquisition and an incremental cost proportional to the size of the message. The fixed channel access costs, denoted as $b_{send}$ and $b_{recv}$, and the incremental costs, $m_{send}$ and $m_{recv}$, are the same for broadcast and point-to-point.

The cost, $E_{send_{ij}}$, for a node $d_i$ to send a point-to-point packet to a node $d_j$ is described by the following equation:

$$E_{send_{ij}} = m_{send} * |MSG| + b_{send} \tag{3}$$

where $|MSG|$ is the size (number of bits) of the message exchanged among nodes $d_i$ and $d_j$.

The cost to receive a point-to-point packet is modelled through the following equation:

$$E_{rec_{ij}} = m_{recv} * |MSG| + b_{recv} \tag{4}$$

In case of broadcast transmission, the cost to send a packet is represented through Equation 5 while the cost to receive a packet is given by equation 6:

$$E_{send_{broad_i}} = m_{send} * |MSG| + b_{send} \tag{5}$$

$$E_{rec_{broad_i}} = m_{recv} * |MSG| + b_{recv} \tag{6}$$

Thus, the energy cost of node $d_i$ for sending (Equation 7) and receiving (Equation 8) packets depends on the used transmission mode:

$$E_{send_i} = \begin{cases} E_{send_{ij}} \text{ if point-to-point} \\ E_{send_{broad_i}} \text{ otherwise} \end{cases} \tag{7}$$

Table I. Communication energy cost coefficients of IEEE 802.11-based wireless network interface.

| | | $\frac{m \cdot w \cdot S}{byte}$ | $byte$ | $\frac{m \cdot w \cdot S}{byte}$ |
|---|---|---|---|---|
| Point-to-Point send | $E_{send_{ij}}$ | 1.9 | $\times size$ | 420 |
| Broadcast send | $E_{broad_i}$ | 1.9 | $\times size$ | 250 |
| Point-to-Point and Broadcast receive | $E_{recv_{ji}}$ | 0.42 | $\times size$ | 330 |

$$\mathrm{E_{receive_i}} = \begin{cases} \mathrm{E_{rec_{ij}}} \text{ if point-to-point} \\ \mathrm{E_{rec_{broad_i}}} \text{ otherwise} \end{cases} \tag{8}$$

Non-destination nodes within the transmission range of either the transmitting or receiving nodes overhear the traffic. The cost of discarding is comparable to the one of a broadcast receiving:

$$\mathrm{E_{discard_i}} = \mathrm{m_{discard}} * |\mathrm{MSG}| + \mathrm{b_{discard}} \tag{9}$$

[24] describes a series of experiments reporting detailed measurements of the per-packet energy consumption of a IEEE 802.11 wireless network interface operating in ad-hoc mode. In those experiments, energy consumption by a network interface to send, receive or discard broadcast and point-to-point messages was determined by direct measurements of the input voltage and current draw at the network device by inserting a small resistance in series with the device. Measurements were made using a 100 MHz digital oscilloscope and 15 MHz 1X probes. Coefficients for the equations defined by the linear formulation above have been determined by performing these measurements for packets of various sizes and applying linear regression.

As stated in [23], the IEEE 802.11 protocol does not define broadcast acknowledgement or retransmission, while, regarding point-to-point traffic, each element of the protocol in Equation 3 and Equation 4 takes into account retransmissions.

The experiments shown that the linear model proposed by Feeney et al. is appropriate. The correlation coefficient is over .99 in nearly every instance. Approximately 50-90 packet measurements, each with an uncertainty of about 7%, contributed to the calculation of each linear equation. The standard error of the calculated coefficients was generally less than 5%.

Those results, shown in Table I, provide a solid experimental basis for energy-aware design and evaluation of a network interface operating in the IEEE 802.11 environment. Accordingly, we adopted those results as energy measures to use in our simulation study.

### 2.2. Computation Energy

In this section we focus on the estimation of the energy consumption for computation.

As our study focuses on energy evaluation of data intensive tasks, the computation load concerns execution of data mining algorithms over smartphones. Specifically, we aim at identifying the energy consumption characteristics of some commonly used data mining tasks running on-board a smartphone: J48, for data classification; K-means, for cluster analysis, and Apriori, for association rules discovery.

We characterize the performance of those data mining algorithms running over a specific device and with respect to a given data set, in terms of the following metrics: energy consumption, memory usage, execution time and percentage of CPU used.

An Android application has been implemented to perform the experimental evaluation. The application has been designed with a twofold objective:

1. integrate the Weka [5] implementation of the reference data mining algorithms in an Android setting;
2. monitor mobile device resources like battery level, CPU occupancy and memory usage.

This last feature has been implemented by exploiting the development tools made available by Android. In particular, the application is able to detect information about resources usage through some files of the Linux kernel of Android. Those files belong to the *proc* directory that contains information concerning the current state of the Linux kernel, allowing this way applications and users to explore the system status. Within such a directory one can find information about hardware status (e.g. current battery level), as well as CPU occupancy and memory usage of each running process.

The energy cost computation is based on battery depletion by exploiting the assumption of proportionality between the percentage of battery level and the energy consumption. Accordingly, the energy cost has been computed as follows. The battery level percentage is gathered through the *proc* files and the corresponding battery level value in $mAh$ is calculated considering the maximum battery level of the device. This value is then converted into electric charge ($Coulomb$) and multiplied to the nominal voltage (equal to 3,7 $V$) to obtain the residual energy of the device, as expressed by the following equation:

$$E(t) = Q(t) * V \tag{10}$$

where $Q(t)$ is, thus, the battery residual charge (Coulomb) and $V$ the nominal voltage.

The application described above once installed on an Android smartphone allows for the (i) execution of data mining algorithms and the (ii) gathering of statistics related to the device resources collected during the execution of a given algorithm. In particular, the application through a simple user interface allows for the selection of the algorithm and the data set to be used. Moreover, through the interface it is possible to specify either a set of common performance parameters like data set size, or algorithm-specific performance parameters like the number of clusters for the K-means algorithm or the minimum support for Apriori.

Input data is an integral part of data mining applications. To support our experimental evaluation, two sample real-word data sets, from the UCI KDD archive [20], have been used as data sources: Covertype with J48; US Census 90 with K-means and Apriori. The first data set contains information about forest cover type for a large number of sites in the U.S. The second one contains data extracted from the U.S. Census Bureau Web site as part of the 1990 U.S. census.

Given a task $\mathrm{t_a}$ and size s of the dataset to be analyzed, the EMC and EEC values for a given device $\mathrm{d_i}$, $\mathrm{EMC_i(t_a, s)}$ and $\mathrm{EEC_i(t_a, s)}$, are obtained on the basis of memory and energy consumption measurements performed on that device.

Table II reports CPU occupancy, memory usage, energy consumption and execution time of the three data mining algorithms on datasets of increasing size, based on their execution

Table II. EMC, EEC, CPU occupancy and Execution time of three data mining algorithms on datasets of increasing size, based on their execution on a set of HTC Hero devices.

| Technique | Algorithm | Dataset | Dataset size (MB) | EMC (MB) | EEC (J) | CPU (%) | TIME (sec) |
|---|---|---|---|---|---|---|---|
| Classification | J48 | Covertype | 0.1 | 19.47 | 178.49 | 96.23 | 300 |
| | | | 0.2 | 20.15 | 396.94 | 98.21 | 540 |
| | | | 0.4 | 23.87 | 791.21 | 97.43 | 2040 |
| | | | 0.8 | 27.68 | 2592.60 | 97.36 | 8160 |
| Clustering | K-means | US Census | 0.1 | 16.73 | 33.97 | 98.03 | 55 |
| | | 90 | 0.2 | 17.95 | 107.89 | 97.65 | 150 |
| | | | 0.4 | 19.72 | 251.75 | 97.02 | 300 |
| | | | 0.8 | 23.08 | 305.69 | 97.97 | 600 |
| | | | 1.6 | 26.40 | 575.42 | 97.82 | 1320 |
| Association rules discovery | Apriori | US Census 90 | 0.1 | 15.86 | 2.82 | 96,92 | 6 |
| | | | 0.2 | 16.97 | 5.91 | 98,03 | 12 |
| | | | 0.4 | 18.06 | 12.85 | 98,24 | 26 |
| | | | 0.8 | 19.87 | 35.96 | 98,13 | 73 |
| | | | 1.6 | 23.32 | 179.82 | 96,87 | 300 |
| | | | 3.2 | 26.92 | 310.36 | 95,44 | 512 |

on a set of HTC Hero smartphones with Android OS. Each algorithm has been executed 10 times; the values reported in the table are the average of the CPU occupancy, execution time, memory usage and energy consumed at the different execution.

As expected, the table shows that the energy consumption grows with the dataset size for all the data mining algorithms. For example, for J48, the EEC value ranges from about 178 to 2,593 J, by increasing the dataset size from 0.1 to 0.8 MB. Similar trends can be observed also for the execution time and memory consumption. For instance, for J48, the EMC value passes from about 19 MB with the dataset of 0.1 MB, to about 28 MB with the dataset of 0.8 MB. Conversely, the CPU occupancy remains rather constant with the data set size for all the algorithms.

## 3. Energy-Aware Task Allocation Strategy

In this section we present the Energy-Aware (EA) task allocation strategy over a reference mobile-to-mobile network architecture.

### 3.1. Reference Architecture

We consider a multi-hop wireless ad-hoc network in which small *devices* (or *nodes*) are divided into local groups, named *clusters*. Generally, geographically adjacent devices are assigned to the same cluster. Under a cluster-based structure, nodes may be assigned different roles, such as *cluster-head* or *cluster member*. A cluster-head normally serves as the local coordinator for its cluster, performing intra-cluster transmission arrangement, data forwarding, and so on. A cluster member is a non-cluster-head node without any inter-cluster links [17, 18, 19, 21, 1].

In this work we assume, as a reference, the cluster-based architecture shown in Figure 1, which is meant to support mobile-to-mobile (M2M) collaborations between small devices. Nodes within a cluster interact through ad-hoc connections (e.g., wi-fi), that we refer to as

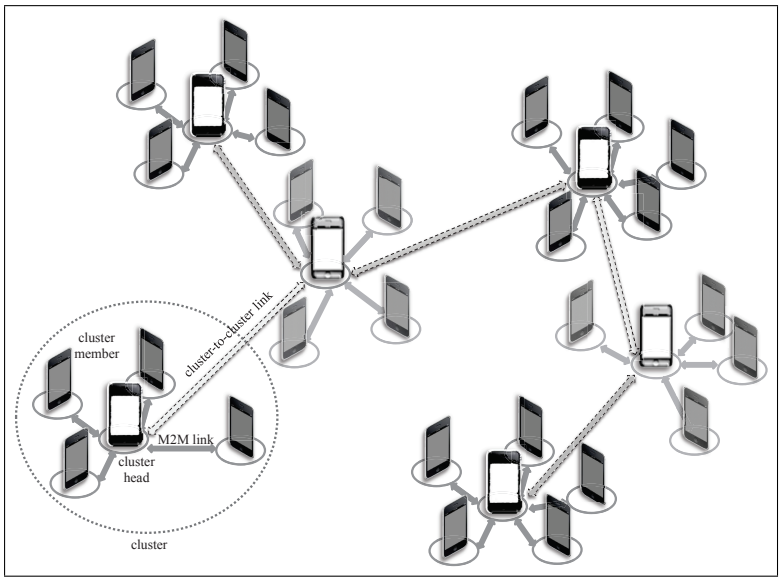



Figure 1. Reference architecture for mobile-to-mobile collaboration.

M2M links. Interactions between clusters (cluster-to-cluster links) take place through ad-hoc connections between the respective cluster-heads.

The architecture is based on a fully distributed cluster formation algorithm in which nodes take autonomous decisions. No global communication is needed to setup the clusters but only local decisions are taken autonomously by each node. This means that the architecture is self-organized into local groups: when devices meet each other, i.e., when they are within the same transmission range, they can form a local group. The self-organization nature of the clustering scheme distributes the responsibility among all the devices. In a previous work [15], we described the cluster formation algorithm. This algorithm follows a bottom-up strategy that finds successive clusters using previously established ones by maximizing the network residual life. Since the focus of this paper is not on the cluster formation algorithm, details can be found in [15].

All types of interactions in the architecture shown in Figure 1 take place either to submit a task, or to perform its distributed allocation, as detailed in the remainder of the section.

### 3.2. Scheduling Model

The scheduling problem is the process of mapping a given application onto a target architecture by: (1) selecting which task of the application shall be executed; (2) allocating that task to a resource; (3) computing start and execution times for the task; (4) repeating these steps until

all tasks are scheduled. It is common in the literature to use the terms task allocation and task scheduling interchangeably. However, scheduling is commonly used to describe all of the above mentioned steps as well as to describe the computation of start and execution times only. Task allocation is, therefore, a step of the more general scheduling problem; it can also be seen as a global scheduling or meta-scheduling that distributes the tasks among the devices. Once tasks have been allocated, the problem becomes one of defining a feasible local schedule that manages task execution for each node. In this paper we focus on the task allocation problem and we refer to task allocation or task scheduling interchangeably.

We introduce the following model to support the description of the scheduling strategy.

- $\mathcal{D}$= $\{d_1, d_2, ..., d_m\}$ is the set of devices in the network.
- $\mathcal{T}$= $\{t_1, t_2, ..., t_n\}$ the set of tasks to be executed, which are assumed to be independent each other.

A generic task $t_i$ is characterized by the following features:

- execution time of $t_i$ to process a data set of size s on a device $d_j$;
- energy consumption of $t_i$ to process a data set of size s on a device $d_j$;
- memory consumption of $t_i$ to process a data set of size s on a device $d_j$;

Before going into the details of the scheduling policy, some notations are introduced.

- $M_i(t)$: memory availability of device $d_i$ at time t.
- $RE_i(t)$: residual energy available at device $d_i$ at time t.
  $P_i(t)$: instantaneous power of device $d_i$ at time t.
- $EEC_i(t_j, s)$: estimated energy consumed for computation by device $d_i$ to run a task $t_j$ over a data set of size s.
- $EET_i(t_j, s)$: estimated energy consumed for communication by device $d_i$ to run a task $t_j$ over a data set of size s.
- $EMC_i(t_j, s)$: estimated memory consumption of device $d_i$ to run a task $t_j$ over a data set of size s.

Given this model, we define the concept of *residual life* $RL_i(t)$ of device $d_i$ at time t, as follows:

$$RL_i(t) = RE_i(t)/P_i(t). \tag{11}$$

Similarly, we define *network lifetime* at time t, $RL_{net}(t)$, as the total residual life of the devices in the network. Therefore, the goal of our scheduling strategy is finding a task allocation that maps each task $t_i \in \mathcal{T}$ to a device $d_j \in \mathcal{D}$ so as to maximize $RL_{net}(t)$.

### 3.3. Scheduling Strategy

The task allocation problem has been proven to be NP-Complete in its general form [22]. Even though optimal algorithms have been proposed for some restricted versions of the problem, heuristic-based algorithms have been proposed for the more general versions of the problem allowing to find good allocations in polynomial time [25].

Accordingly, we propose a two-phase heuristic-based decentralized algorithm. When an assignment decision has to be made for a task, the first phase, referred to as *local assignment*, tries to assign the task locally to the cluster where the execution request has been generated, so as to maximize the cluster residual life. If the local assignment failed because none of the devices within the cluster can execute the task, the cluster-head activates the second phase, referred to as *global assignment*, responsible for task arbitration among clusters. In this case, the task will be assigned to the most suitable device, all over the network of clusters, that maximizes the overall network lifetime.

We formalize the problem of task allocation as an optimization problem. As said before, the aim of the optimization is to maximally extend the network lifetime and the number of alive devices. To this aim, tasks are allocated to the devices such as to balance the load among them. We optimize the problem by iteratively trying to improve a candidate solution. A feasible allocation is optimal if the corresponding cluster residual life (in case of local assignment) or network lifetime (in case of global assignment) is maximized among all the feasible allocations.

A device $d_i$ can be candidate for the assignment of a task $t_a$, if it satisfies the following constraints:

1. $d_i$ has enough memory to perform $t_a$ over a data set of size s, i.e. $EMC_i(t_a, s) < RE_i(t)$.
2. $d_i$ has enough energy to perform $t_a$ over a data set of size s, i.e. $EEC_i(t_a, s) < RE_i(t)$.

During the local assignment phase, a cluster-head, or the set of neighboring cluster-heads in case of the global assignment, will choose the local node, among the ones satisfying the above constraints, that will prolong the life of the corresponding local group by using the following objective function:

$$RL_{LG_j}(t) = \text{Max} \sum_{i=1}^{N_{LG_j}} \alpha_i RL_i(t) \tag{12}$$

where $RL_{LG_j}$ denotes the residual life of local group $LG_j$, $N_{LG_j}$ is the number of nodes within the local group $LG_j$, $RL_i$ is the residual life of node $d_i$ in the group, and parameter $\alpha_i$ takes into account the importance of $d_i$ in the local group. The node associated with the maximum value of the objective function will be selected by the cluster-head as candidate node. Note that throughout the experimental evaluation the parameter $\alpha_i$ is set to 1 thus, all the nodes have the same role within the local group.

If the global assignment phase is activated, the final decision is taken by considering all the candidate nodes proposed by the neighboring clusters. The task will be assigned to the local group that maximizes the network lifetime:

$$RL_{net}(t) = \text{Max} \sum_{j=1}^{N} \alpha_j RL_{LG_j}(t) \tag{13}$$

where N is the number of clusters in the network.

### 3.4. Scheduling Algorithm

When a node, referred to as requesting node, wants to execute a task, it does two operations:

```
Method allocateTaskReq
Input: task t, data set size s

begin
    if (this.isClusterHead()) then
        allocateTask(t, s);
    else
        myClusterHead.allocateTaskReq(t, s);
    end
end
```

Figure 2. Task allocation request.

1. it requests its cluster-head to handle the task assignment process (see Figure 2);
2. it sets a timeout within which it wants to receive the task execution result. If the timeout expires and the task is not completed (for execution or communication problems), the requesting node resubmits the task. The timeout is set based on the estimated execution time of the submitted task.

Upon receiving a task allocation request, the cluster-head triggers, through the method shown in Figure 3, the activation of the task allocation strategy to find the optimal assignment for that task. It purses first a local optimum to reduce the transmission costs.

To this aim, the cluster-head verifies whether the residual life of its group is greater than a given threshold ($local_{thr}$) of its peak value. If the check is successful, it starts up the local assignment phase. If the check is negative or the local assignment failed because none of the nodes within the cluster can execute the task, the cluster-head activates the global assignment phase. Note that in case of negative check on $local_{thr}$ , the global phase is activated only if at least one neighboring group, whose residual life is greater than a given threshold ($global_{thr}$), is found. If this last condition is not satisfied, the local phase is activated anyhow. Details about those threshold values and how they have been established are given in section 4.1.

The scheduling starts with the local assignment phase where the cluster-head tries to assign the task to a node within the group that maximizes the cluster residual life (see Figure 4). In this way, the cluster-head determines a candidate node in the group meeting the requirements to execute the task in terms of energy and memory constraints. The local assignment phase can be executed both by the cluster-head of the requesting node and by the cluster-heads of neighboring clusters involved in a global assignment.

The global assignment phase, whose pseudo-code is shown in Figure 5, receives as input the set of neighboring clusters $\mathcal{C}$. A cluster-head identifies the set of neighboring clusters by sending a discovery message; then, all the cluster-heads within its transmission range reply with an acknowledgment message; thus, the neighboring set includes the clusters whose cluster-heads sent the acknowledgment. Among the neighboring clusters are considered as input only the ones having a residual life greater than the global threshold $global_{thr}$, so as to avoid overloading of the local groups. This threshold has been selected after some experiments as a value that

```
Method allocateTask
Input: task t, data set size s

begin
    localAssignmentFailed ← false;
    C ← determineNeighboringClusters();
    if (RL_LGi > local_thr or C=∅) then
        < d_candidate, ERL_LGi > ← localAssignment(t, s);
        if (< d_candidate, ERL_LGi >≠ ∅) then
            if (d_candidate = this) then
                startTask(t, s);
            else
                d_candidate.executeTask(t, s);
            end
        else
            localAssignmentFailed ← true;
        end
    end
    if ((RL_LGi ≤ local_thr or localAssigmentFailed) and C ≠ ∅) then
        globalAssignment(t, s, C);
    end
end
```

Figure 3. Task allocation by the cluster-head of local group $LG_i$.

ensures a good balance between communication and computation costs. For each local group in $\mathcal{C}$, the algorithm selects a candidate node that best performs the task. The complexity of this operation is linear in the number of clusters. Since this number is very low (e.g. 5 in a network of 100 devices), the complexity of this operation is limited. Then, for each candidate node, the algorithm estimates the network residual life as it would be if that task would have been assigned to that node. The network residual life is estimated using Equation 13, which in turn is calculated after having estimated the residual life of each local group using Equation 12. Finally, among all the candidate nodes, the task is assigned to the one that ensures the highest network lifetime, referred to as executor node.

## 4. Performance Evaluation

To assess the effectiveness of the proposed EA allocation strategy, we carried out a performance evaluation using a custom discrete-event simulator. The goal of the evaluation is to assess whether the EA strategy is able to extend the network lifetime and the number of alive devices compared to alternative scheduling strategies, while meeting application-level performance constraints.

```
Method localAssignment
Input: task t, data set size s
Output: candidate node d_candidate, estimated residual life
ERL_LG_i of local group LG_i if t is assigned to d_candidate

begin
    if (this = CH_i or (this ≠ CH_i and RL_LG_i > global_thr)) then
        RL_LG_curr ← 0;
        ERL_LG_i ← 0;
        d_candidate ← ∅;
        foreach node d_i ∈ LG_i do
            if (d_i.hasSkill(t, s)) then
                RL_i ← d_i.estimateNodeResidualLife(t, s);
                RL_LG_curr ←estimateGroupResidualLife(d_i,RL_i);
                if (RL_LG_curr > ERL_LG_i) then
                    ERL_LG_i ← RL_LG_curr;
                    d_candidate ← d_i;
                end
            end
        end
    end
    return < d_candidate, ERL_LG_i >;
end
```

Figure 4. Local assignment of task t within local group $LG_i$.

### 4.1. Evaluation Setup

A custom discrete-event simulator has been implemented to perform the evaluation. As a first step, the simulator builds a network of 100 mobile devices distributed over an area of 250,000 $m^2$, and let them grouping into clusters based on the algorithm described in [15]. As a result of the clustering procedure, 20 devices act as cluster-heads, with an average cluster size of 5. Then, an initial energy capacity ranging from 3,000 J to 11,000 J is assigned to each device, following a normal distribution.

After the initial setup, mobile devices start generating a set of tasks to be executed. To the purpose of this work we focused on data mining tasks, as collaborative data analysis represents a quite general example of application scenario. We experimentally measured computing and energy requirements of a set of data mining algorithms on data sets of varying size. Based on these measurements, in our simulation, each task is characterized by the amount of energy required for its execution, ranging from 30 to 3000 J, and the execution time, ranging from 45 seconds to 160 minutes. Both the amount of energy and the execution time follow a Zipf distribution with skewing factor equal to 0.8. The task arrival event is a Poisson process with a rate $\lambda$ ranging from 80 to 2400 tasks/hour. The transmission energy is based on the model presented in Section 2.

```
Method globalAssignment
Input: task t, data set size s, set of neighboring clusters C

begin
    D ← ∅;
    foreach local group LG_i ∈ C do
        < d_candidate, RL_LG_i > ← CH_i.localAssignment(t, s);
        D.add(< d_candidate, RL_LG_i >);
    end
    ERL_net ← 0;
    d_best ← ∅;
    foreach node d_i ∈ D do
        RL_net ← estimateNetworkResidualLife(d_i, RL_LG_i);
        if (RL_net > ERL_net) then
            ERL_net ← RL_net;
            d_best ← d_i;
        end
    end
    CH_best ← getClusterHead(d_best);
    if (d_best = this) then
        startTask(t, s);
    else
        if (CH_best = this) then
            d_best.executeTask(t, s);
        else
            CH_best.requireTaskExecution(d_best, t, s);
        end
    end
end
```

Figure 5. Global assignment of task t.

We compared the performance of the EA scheduler with that achieved by three alternative scheduling strategies implemented over the same clustered architecture introduced in Section 3.1:

- *Round Robin* (RR). RR assigns tasks to each device in equal portions and in circular order. Each cluster-head maintains a circular queue of local nodes and neighboring cluster-heads. When a task has to be allocated, the cluster-head assigns it to the current pointed node in the queue. The queue is scanned circularly, until at least one of the device is alive. RR does not make any check about energy requirements and availability, thus, a task is assigned to a device even if this does not have the required energy to execute it.
- *Smart Round Robin* (SRR). SRR is a modified version of RR that still maintains a circular queue composed of local nodes and cluster-heads of neighboring clusters. Differently from RR, SRR does take into account the residual energy of nodes to make

scheduling decisions. Thus, SRR assigns a task to a node only if this has enough energy to run it.

- *Energy Consolidation* (EC). EC follows a two-phase allocation approach similar to that of the EA scheduler. When an assignment decision has to be made for a task, first the local phase is activated by allocating the task to the most discharged node in the cluster, among the ones having enough energy to run it. If there are no local nodes that can run the task, the global phase is activated and the task is assigned to the most discharged node among the neighboring clusters, after having verified that the node has enough energy. The main difference between EA and EC is in the optimization function. EA maximizes the number of alive devices prolonging the network lifetime by balancing the energy load among the devices. Conversely, EC consolidates the devices by filling first the least loaded ones, thus avoiding energy fragmentation in the network.

For each of the scheduling strategies considered (EA, RR, SRR and EC) we implemented three different ready queue management policies: (i) *First In First Out* (FIFO), where processes are assigned the CPU in the order they request it; (ii) *Shortest Job First* (SJF), that assigns the CPU to the process that has the smallest next CPU burst; and (iii) *Completely Fair Scheduler* (CFS) that assigns the CPU to the process that has been waiting for the CPU for the largest amount of time.

CFS is the preemptive scheduling algorithms adopted by Linux 2.6.23 and used in Android mobile devices. CFS tries to give tasks a fair amount of the processor time. When one or more tasks are not given a fair amount of time, then time to execute is assigned to out-of-balance tasks. To determine the balance, CFS maintains the amount of time assigned to each task in the so called *virtual runtime*. The smaller is the virtual runtime of a task, the smaller is the amount of time a task has been permitted access to the processor, while the higher its need for the processor.

### 4.2. Performance Results

Since the goal of the EA scheduling policy is to maximize the number of alive devices, we evaluated how effective is the EA strategy in achieving these goals compared to the alternative scheduling strategies introduced above. As the comparison in terms of network lifetime is significant only if the compared schedulers execute exactly the same tasks, we focus in particular on the following metrics, which allow to highlight the performance even if the compared schedulers do not execute exactly the same tasks: (i) the *number of alive devices*, (ii) the *number of completed tasks*, (iii) the amount of useful work performed by the devices that we refer to as *workload*, and (iv) the ratio between completed tasks and assigned tasks that we refer to as *completion rate*. The workload metric has been introduced to take into account the fact that tasks are heterogeneous. In fact, because of task heterogeneity, using only the number of alive devices and the number of completed tasks as performance metrics is not enough to evaluate the actual computation carried out by the devices.

We organized the simulations in four categories. Each category evaluates the performance of the schedulers with respect to the following system parameters: (i) number of submitted tasks; (ii) computational load; (iii) task arrival rate; (iv) energy distribution. After the simulation

Table III. Number of alive devices (AD), Number of completed tasks (CT), Workload (W) and Completion time (CT) w.r.t. number of submitted tasks, using EA, RR, SRR and EC with FIFO, SJF and CFS policies.

| | | #tasks: 150 | | | #tasks: 300 | | | #tasks: 600 | | | #tasks: 1200 | | | #tasks: 2400 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | FIFO | SJF | CFS | FIFO | SJF | CFS | FIFO | SJF | CFS | FIFO | SJF | CFS | FIFO | SJF | CFS |
| | EA | 100 | 100 | 100 | 100 | 100 | 99.98 | 99.72 | 99.72 | 99.56 | 91.38 | 91.34 | 89.66 | 55.48 | 55.48 | 57.06 |
| AD | RR | 99.54 | 99.54 | 99.54 | 94.28 | 94.28 | 94.28 | 70.08 | 70.02 | 69.84 | 23.76 | 24.58 | 24.4 | 10.98 | 11.1 | 11.68 |
| | SRR | 99.6 | 99.6 | 99.6 | 96.78 | 96.8 | 96.76 | 82.64 | 82.48 | 82.54 | 39.66 | 39.78 | 38.96 | 27.34 | 26.46 | 27.48 |
| | EC | 99.44 | 99.82 | 97.96 | 95.24 | 97.24 | 89.6 | 75.3 | 80.14 | 64.6 | 38.8 | 42.18 | 26.8 | 16.68 | 15.74 | 6.56 |
| | EA | 148 | 148 | 148 | 299.96 | 299.96 | 299.96 | 590.24 | 590.16 | 590.26 | 979.32 | 979.04 | 973.68 | 1098.26 | 1098.42 | 1076.02 |
| CT | RR | 147.44 | 147.52 | 147.38 | 290.5 | 292 | 289.94 | 534.42 | 545.7 | 535.82 | 772.12 | 807.48 | 788 | 797.86 | 840.28 | 818.2 |
| | SRR | 147.5 | 147.58 | 147.46 | 294.14 | 295.66 | 293.7 | 552.08 | 564.36 | 549.98 | 780.6 | 814.76 | 779.88 | 788.88 | 823.3 | 789.18 |
| | EC | 145.9 | 145.9 | 145.98 | 289.56 | 289.56 | 289.78 | 559.96 | 559.78 | 562.5 | 826.42 | 871.14 | 820.3 | 878.42 | 957.02 | 855.06 |
| | EA | 118.21 | 118.21 | 118.21 | 245.45 | 245.45 | 245.45 | 472.06 | 471.81 | 472.07 | 642.76 | 642.51 | 641.24 | 656.18 | 656.08 | 652.82 |
| W | RR | 117.05 | 117.07 | 116.73 | 231.16 | 231.20 | 225.06 | 410.88 | 411.36 | 381.51 | 566.03 | 568.53 | 501.28 | 579.08 | 583.38 | 515.37 |
| (KJ) | SRR | 117.20 | 117.22 | 116.92 | 238.51 | 238.63 | 233 | 439.60 | 440.04 | 404.98 | 585.37 | 590.65 | 507.45 | 588 | 592.70 | 507.67 |
| | EC | 117.05 | 115.19 | 113.83 | 241.92 | 235.85 | 226.89 | 479.37 | 460.73 | 440.94 | 660.61 | 649.34 | 583 | 668.31 | 665.23 | 585.48 |
| | EA | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.99 | 0.99 | 0.99 | 0.98 | 0.98 | 0.98 |
| CR | RR | 1 | 1 | 1 | 0.97 | 0.97 | 0.97 | 0.89 | 0.91 | 0.90 | 0.79 | 0.83 | 0.81 | 0.78 | 0.82 | 0.80 |
| (%) | SRR | 1 | 1 | 1 | 0.98 | 0.99 | 0.98 | 0.93 | 0.95 | 0.92 | 0.85 | 0.89 | 0.85 | 0.84 | 0.88 | 0.85 |
| | EC | 0.99 | 0.99 | 0.99 | 0.97 | 0.97 | 0.97 | 0.94 | 0.94 | 0.94 | 0.91 | 0.90 | 0.92 | 0.84 | 0.83 | 0.92 |

results, we present experimental results to measure the computation and communication overhead of the proposed EA allocation strategy. Finally, we describe how the algorithm parameters used in the simulations have been selected.

### *4.2.1. Performance evaluation w.r.t. number of submitted tasks*

The aim of this first set of simulations is to compare the performance of the EA task allocation scheme versus EC, RR and SRR with respect to the number of submitted tasks. We consider an increasing number of submitted tasks, from 150 to 2400, that arrive following a Poisson distribution with frequency $\lambda = 80$ tasks per hour. All the allocation strategies are evaluated considering the three ready queue scheduling policies introduced earlier: FIFO, SJF and CFS.

Table III reports the comparison in terms of *alive devices* (AD), *completed tasks* (CT), *workload* (W) and *completion rate* (CR). The results show that the ready queue scheduling policies have a limited impact on the schedulers behavior.

In contrast to EA, RR and SRR strategies, EC behaves slightly different on the basis of the ready queue scheduling policy used. More precisely, when EC uses the SJF policy, it improves the number of completed tasks that however is far from the number achieved by EA. On the contrary, when EC uses the CFS strategy it gets worse in terms of alive devices, reaching the same bad results as RR and SRR. In terms of workload, EC presents more or less the same performance of EA, in case of the SJF and FIFO policies, whereas in case of the CFS policy EA performs significantly better than EC. Accordingly, with respect to completion rate, the scheduler EC reaches the 95% using the CFS policy, improving the score of the other policies.

To get a clearer view of the schedulers performance, we plotted some results. We consider only the FIFO policy because of the achievements similarity with the other policies. Figure 6(a)compares the number of alive devices of EA, with those of RR, SRR and EC. As expected, the number of alive devices decreases with the number of submitted tasks, while the number of completed tasks increases. Nonetheless, with EA the number of alive devices decreases significantly only after the submission of 1200 tasks. The number of alive devices with EA is greater than those of RR, SRR and EC, and the advantage of EA increases with the number of submitted tasks. Similarly, EA completes a higher number of tasks than those of RR, SRR and EC as can be noticed in Figures 6(b). This trend is even more evident for a number of submitted tasks greater than 600.

Figure 7(a) shows that for very low number of submitted tasks (around 330) all the strategies present almost the same behavior. By increasing the number of submitted tasks we can see that EA and EC outperform both RR and SRR carrying the same workload. With the same workload, the difference in the number of alive devices and executed tasks between EA and EC can be explained as follows. EC consolidates the devices, so the alive devices are more loaded compared to those of EA and are able to execute heavier tasks. As EA balances the energy load among the devices in the network, it remains with more devices alive that are less charged than those of EC.

As shown in the previous figures, EA outperforms the other strategies both in the number of alive devices and the number of completed tasks. This is mainly due to the fact that EA does not allocate all the submitted tasks but only those that can be executed by the network. In fact, exploring the completion rate in Figure 7(b), we can notice that EA completes around 100% of the allocated tasks, whereas RR and SRR around 75% and, finally, EC completes around 80% of the allocated tasks. The behavior of EC depends on the fact that this strategy consolidates the most powerful devices, discharging those with lower energy. Thus, during the assignment process it may happen that EC leaves the devices to which a task has been assigned with an energy load just enough to execute the just assigned task. For this reason, it might happen that some of these devices consume a small amount of energy for communication before the execution of the task, resulting in the inability of executing the assigned task.

As the devices are heterogeneous in terms of initial energy level, the energy load balancing effect of EA is particularly effective. Figure 8 shows how the remaining energy is distributed among all the devices in the network. In particular, it is shown how this distribution changes by increasing the number of submitted tasks, for EA, RR, SRR and EC. More precisely, it is specified the percentage of devices with (i) no remaining energy ($RE = 0$), (ii) low remaining energy ($0 < RE <= 1000$) (iii) medium remaining energy ($1000 < RE < 5000$) and (iv) high remaining energy ($RE >= 5000$). One can note the effectiveness of the EA algorithm in balancing the energy level among the devices: the percentage of devices with medium remaining energy is always rather high, regardless of the number of submitted tasks. In particular, up to 600 tasks, 85% of devices remains with high energy values and up to 1200 tasks it keeps alive 95% of devices. Conversely, with RR, SRR and EC schedulers, when the number of submitted tasks increases the percentage of devices without energy grows always more, reaching 90% with RR, 75% with SRR, and 83% with EC after the submission of 2400 tasks.

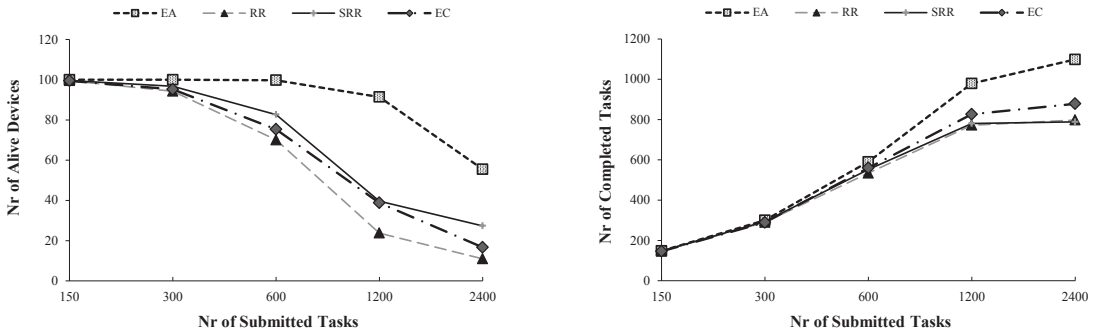



Figure 6. (a) Number of alive devices and (b) Number of completed tasks w.r.t. number of submitted tasks, using EA, RR, SRR and EC.

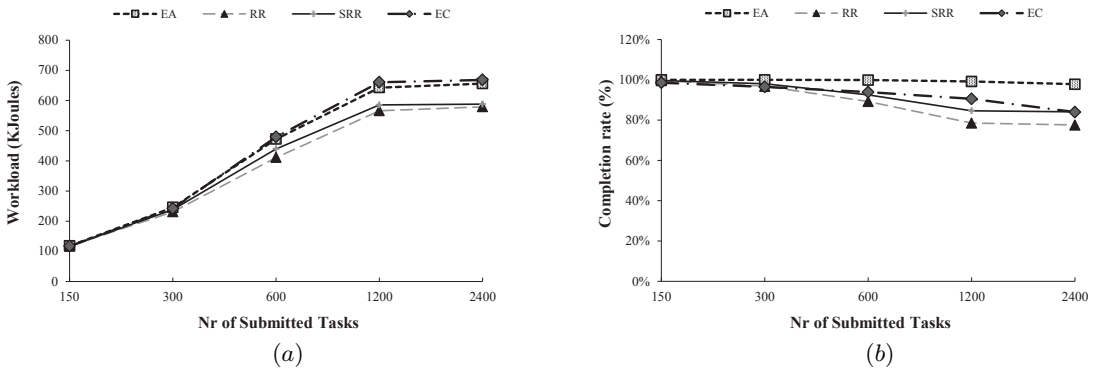



Figure 7. (a) Workload and (b) Completion rate w.r.t. number of submitted tasks, using EA, RR, SRR and EC.

In the remaining simulations we do not report on results of the CFS scheduling policy. This is because all the considered tasks have the same priority. Thus, we could not exploit the rationale of CFS that is essentially based on task priorities. Moreover we do not show other results concerning SRR, because we noted that its behaviour is almost the same as RR.

### *4.2.2. Performance evaluation w.r.t. computational load*

In the second set of simulations we compared the performance of the EA scheduler versus EC and RR considering tasks of increasing computational load, ranging from 250 to 4000 J. We set the number of submitted tasks to 2500, that arrive with a frequency $\lambda = 80$ tasks per hour. At each run, 2500 homogeneous tasks are submitted, all having the same computational load. We report only the results of the FIFO policy because in each benchmark all the task are the same.

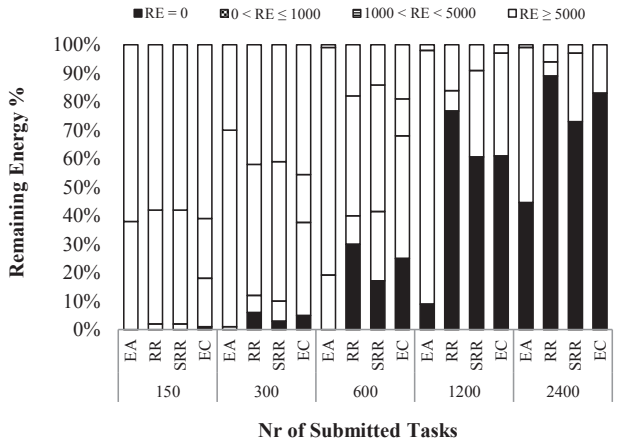



Figure 8. Distribution of remaining energy using EA, RR, SRR and EC w.r.t. number of submitted tasks.

By increasing the computational load EA, EC and RR complete a decreasing number of tasks. Figure 9(b) shows that all the related schedulers complete the same number of tasks, but since EA and EC allocate only the tasks that can be executed, they avoid switching off the devices. Conversely, RR turns off a number of devices that increases always more till reaching 100% when the computational load is greater than 500 J, as shown in Figure 9(a). Differently, for EA and EC the number of alive devices increases with the computational load. This is because a lower number of tasks has been completed, and, thus, the alive devices remain more charged.

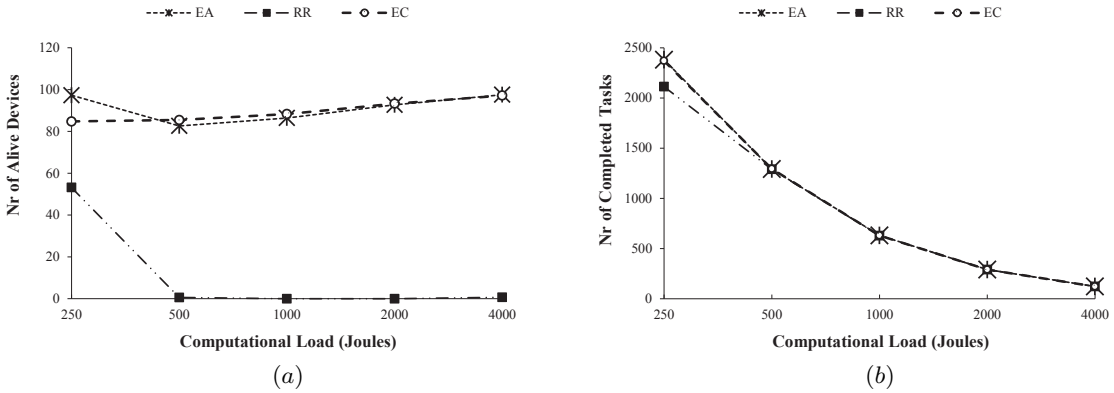



Figure 9. (a) Number of alive devices and (b) Number of completed tasks w.r.t. computational load, using EA, RR and EC.

#### *4.2.3. Performance evaluation w.r.t. task arrival rate*

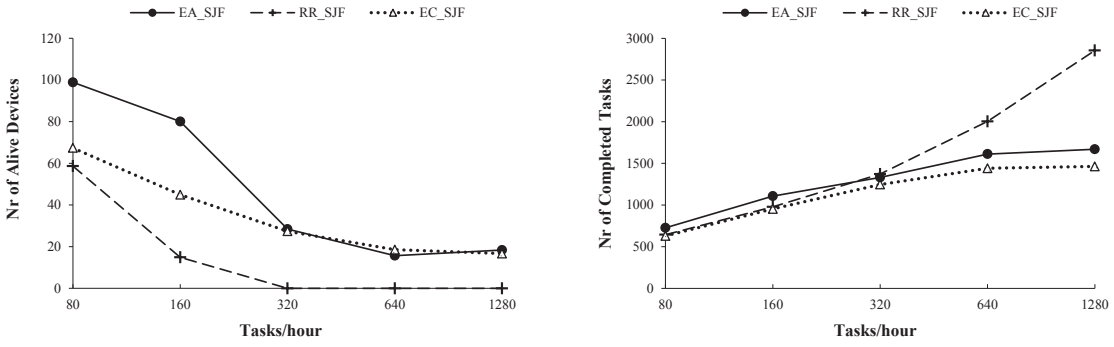
The aim of this third set of simulations is to compare the performance of EA versus RR and EC considering task arrival rates ranging from 80 to 1280 tasks per hour. In this case, the ready queue scheduling policies influence the performance of EA, RR and EC.

Before presenting the performance results, we briefly outline the expected behavior of the local scheduling policies when the submission frequency increases. By increasing the task arrival frequency, the devices consume more communication energy, so less energy remains for executing tasks for all the strategies. This is particularly evident using the FIFO policy, where the number of completed tasks decreases with the task arrival frequency, conversely to what happens with the SJF policy where the number of completed tasks increases. The reason of this can be explained as follows. By increasing the task arrival frequency, a larger number of lightweight tasks are submitted. This happens because we use a Zipf distribution of the energy required to execute the tasks, therefore the number of lightweight tasks is higher. In this condition, the SJF policy assigns the CPU to the most lightweight task moving it before all the preceding heavier tasks in the ready queue. This way, SJF increases the number of completed small tasks. RR is the scheduler that takes more advantage from the SJF policy because it allocates all the submitted tasks even if not all of them can be executed by the devices in the network. In contrast, EA and EC do not allocate a task if none of the available nodes satisfy the required energetic constraints, as will be shown in Figure 11(a).

Let's first discuss the case where all the strategies use the SJF policy. Figure 10(a) shows that the number of alive devices for EA decreases with the task arrival frequency up to 320 tasks per hour, while, for higher frequency values the number of alive devices decreases more slowly until remaining almost constant. This is because EA completes less tasks, as can be noted from the graph in Figure 10(b). For what concerns EC, for frequency values lower than 320 tasks per hour, it maintains alive a considerable lower number of devices compared to EA, as expected; for higher values the two strategies keep the same number of alive devices. Despite this, we can see in Figure 10(b) that the number of completed tasks for EC is always lower than that of EA. RR remains with the lowest number of alive devices and, in particular for frequency values greater than 320 all the devices are turned off.

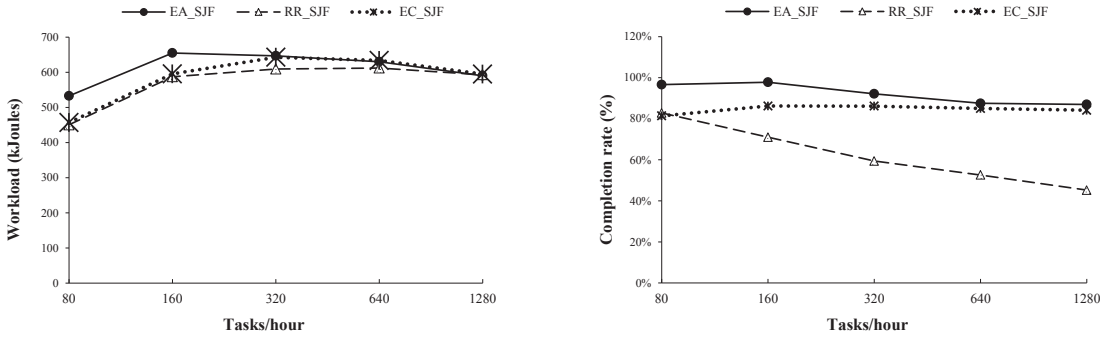
As said before, RR with SJF is the only strategy that takes advantage from the increasing frequency. As such, as can be noted from Figure 10(b), for frequency values higher than 320, the number of tasks completed by RR is considerably higher than those of both EA and EC. As stressed above, the advantage of RR in the number of completed tasks is due to the fact that RR allocates all the submitted tasks even if not all of them can be executed by the devices. Figure 11(a) shows that, in contrast, EA completes always almost all the allocated tasks down to 90% only for very high frequency values. EC completes a number of tasks ranging from 80 to 85%. In contrast, the number of tasks completed by RR decreases significantly with the frequency, reaching 40% with a frequency of 1280 tasks per hour. Thus, the advantage in terms of number of completed tasks of RR is due to the fact that it executes more lightweight tasks. This is confirmed by the amount of workload performed by the three reference strategies. In Figure 11(b) one can see that EA bears a higher workload. Only for very high frequency values, around 1000 tasks per hour, the workload of RR and EA strategies becomes the same, whereas, for frequency values greater than 320, EC behaves like EA. Furthermore, we highlight here

that RR obtains this high number of completed tasks sacrificing the devices that at the end are all turned off.




(a) (b)

Figure 10. (a) Number of alive devices and (b) Number of completed tasks w.r.t. task arrival frequency, using EA, RR and EC with SJF policy.

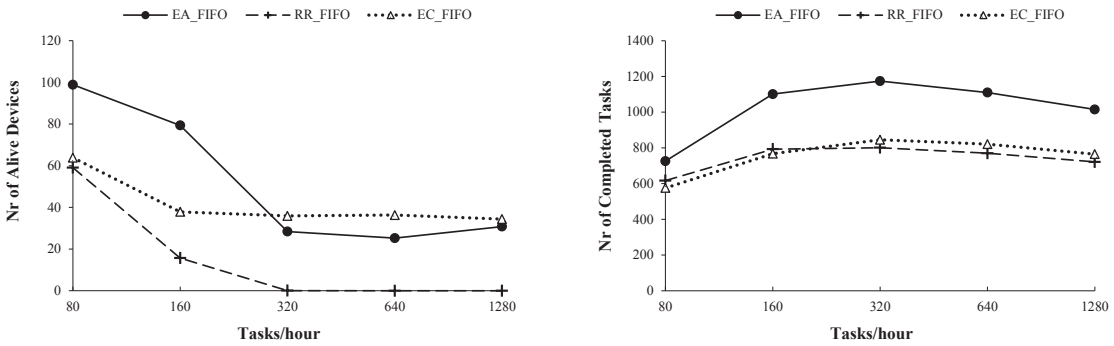



(a) (b)

Figure 11. (a) Workload and (b) Completion rate w.r.t. task arrival rate using EA, RR and EC with SJF policy.

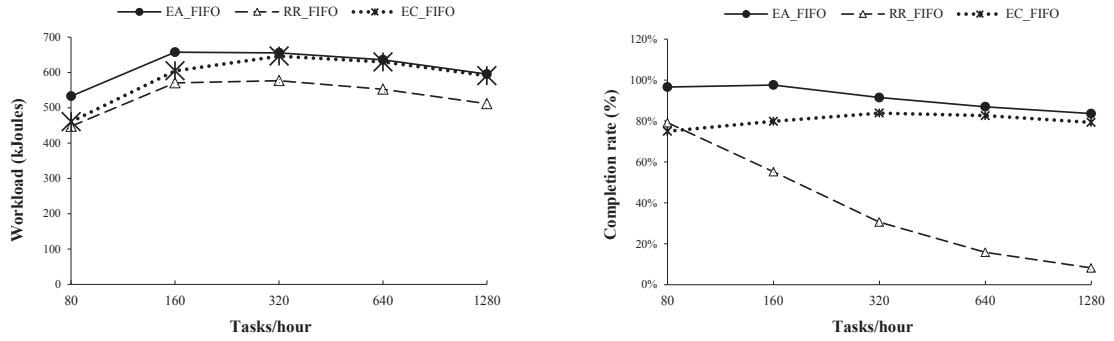
Figure 12(a) shows the number of alive devices of EA, RR and EC when the FIFO scheduling policy is used. As expected, the number of alive devices for all the strategies decreases with the tasks arrival frequency. Once again EA is able to keep the higher number of alive devices for frequency values lower than 320. For higher values, EC maintains a slightly larger number of alive devices. This is mainly due to the fact that, as one can see from Figure 12(b), EC, mostly for frequency values greater than 320, executes a number of tasks much lower than EA. RR once again for frequency values greater then 320 switches off all the devices. However, differently from what happens with the SJF policy, here RR completes a significantly lower number of tasks than EA as reported in Figure 13(a). This figure shows that: (i) EA completes always almost all the allocated tasks dropping to 90% only for very high frequency values; (ii)

EC completes a number of tasks ranging from 80 to 85%; (iii) the number of tasks completed by RR is always quite low until falling to 5% with a frequency of 1280 tasks per hour. Even more, the workload of RR is always substantially lower than EA, while the workload sustained by EC equals the one of EA for frequency values greater than 320 (see Figure 13(b)).




(a) (b)

Figure 12. (a) Number of alive devices and (b) Number of completed tasks w.r.t. task arrival rate, using EA, RR and EC with FIFO policy.




(a) (b)

Figure 13. (a) (a) Workload and (b) Completion rate w.r.t. task arrival rate using EA, RR and EC with FIFO policy.

Figures 14(a) and 14(b) show the waiting time and the completion time of the tasks for all the three allocation schemes. For the SJF case, both the waiting time and the completion time of EA and RR are very similar: RR is slightly better only for frequency values greater than 640. The EC times are always higher than those of EA and RR but for frequency values greater than 320 the difference gets smaller. For the FIFO case all the strategies take higher times: RR is faster than EA only for frequency values greater than 320. However, the difference is important only for very high frequency values. On the contrary, EC is always significantly slower than both EA and RR.

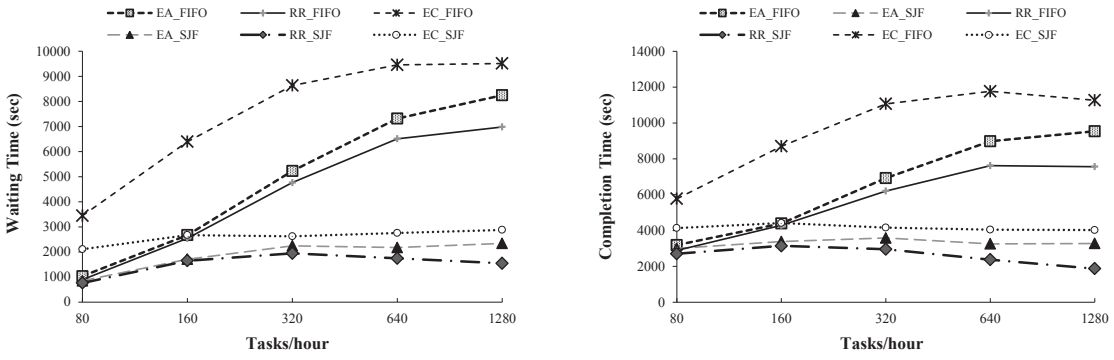



Figure 14. (a) Waiting Time and (b) Completion Time w.r.t. task arrival rate using EA, RR and EC with FIFO and SJF policies.

### *4.2.4. Performance evaluation w.r.t. energy distribution*

The aim of this fourth set of simulations is to evaluate the performance of EA against RR and EC by varying the distribution of the initial energy level among the devices in the network. In particular, we considered four energy configurations:

- CONF_1: the energy is assigned to the devices following a normal distribution with average at 40% of the peak value (this configuration has been used also for the simulations presented above).
- CONF_2: all the devices start with the same energy level, equal to 40% of the peak value.
- CONF_3: the energy is uniformly distributed in the range 15%-65% of the peak value.
- CONF_4: the energy is assigned using a bimodal distribution, with local maxima at 30% and 75% of the peak value.

For all the configurations, the total energy level available is constant and equal to about 700 kJoule. In each configuration, 2500 tasks are submitted with a frequency $\lambda = 80$ tasks per hour.

Figures 15(a), 15(b), and 16 show only the results for the FIFO policy because they are similar to those of SJF. EA and EC adapt their behavior to the different configurations always assigning the tasks respectively to the most energy powerful devices and to the less energy powerful ones. Differently, RR allocates the tasks without taking into account the energy availability of the devices. Thus, the performance of RR depends on the specific combination of devices and tasks: each time a task has to be allocated, the effectiveness of RR depends on the remaining energy of the device at the beginning of the scheduling queue and on the load of the task to be allocated. For example, if the most charged device is at the beginning of the scheduling queue when a high computational task has to be allocated, there is a chance of executing the task without turning off the device. This is particularly evident in Figure 15(a) for configurations CONF_1 and CONF_3 in which 90% of devices is turned off. Figure

15(a) shows that EA keeps alive almost the same number of devices in the different network configurations, and this number is always greater than the number of alive devices of RR. EC keeps alive a variable number of devices and this number is always lower than that of EA. This is because EC consolidates the devices by assigning the tasks to the less energy powerful devices, causing this way the turning off of the weakest devices. For this reason, EC remains with a lower number of alive devices that are more loaded compared to the ones of EA. Thanks to this, EC is able to execute heavier tasks, and, thus, this is the reason for the difference in the number of executed tasks with EA as confirmed by Figure 16, which shows that the workload carried out by both strategies is the same.

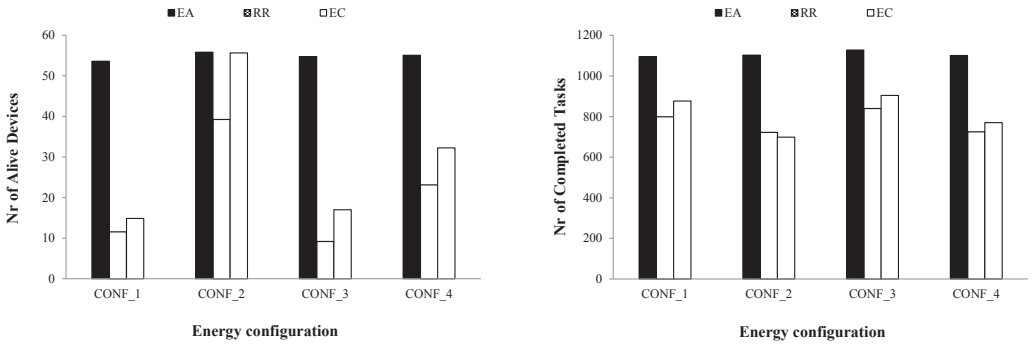



Figure 15. (a) Number of alive devices and (b) Number of completed tasks w.r.t. energy configurations, using EA, RR and EC.

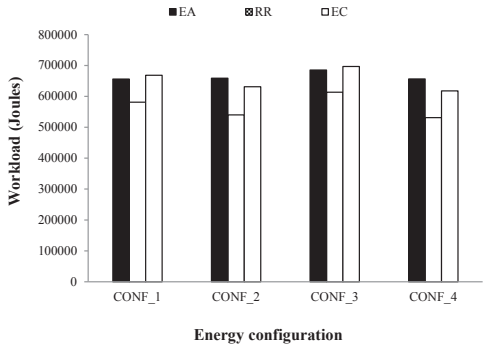



Figure 16. Workload w.r.t. energy configurations, using EA, RR and EC.

Table IV. Number of completed tasks (CT), Workload (W), Communication overhead (TO), Computation overhead (CO), Overall overhead (OH) and Percentage of overhead (OH_%), w.r.t. number of submitted tasks.

| # Tasks | CT | W (KJ) | TO (KJ) | CO (KJ) | OH (KJ) | OH % | TO_CH (KJ) | TO_CM (KJ) | CO_CH (KJ) | CO_CM (KJ) |
|---|---|---|---|---|---|---|---|---|---|---|
| 150 | 148 | 118.21 | 3.37 | 0.21 | 3.58 | 3.03 | 0.1037 | 0.0128 | 0.0085 | 0.0002 |
| 300 | 299.96 | 245.45 | 6.54 | 0.21 | 6.75 | 2.75 | 0.2062 | 0.0233 | 0.0085 | 0.0002 |
| 600 | 590.24 | 472.06 | 12.82 | 0.43 | 13.25 | 2.81 | 0.4097 | 0.0442 | 0.0145 | 0.0012 |
| 1200 | 979.32 | 642.76 | 25.40 | 0.64 | 26.03 | 4.05 | 0.8165 | 0.0859 | 0.0210 | 0.0020 |
| 2400 | 1098.26 | 656.18 | 50.54 | 1.92 | 52.46 | 7.99 | 1.6302 | 0.1694 | 0.0625 | 0.0062 |

#### *4.2.5. EA overhead evaluation*

The EA scheduling strategy, in addition to being effective, is also an efficient strategy, because it reaches the desired objective with limited energy overhead, as demonstrated by the experimental results presented in this section. In particular, we evaluated both the overhead for computation (i.e. scheduling and allocation) and for communication (transmission) among nodes in order to implement the EA strategy. The overhead introduced by EA is the sum of these two cost factors. For measuring the EA computation overhead, we performed the allocation strategy steps on a HTC Hero smartphone with Android OS, whereas the communication overhead was analytically computed on the basis of the energy model described in Section 2.1.

We evaluated the EA overhead considering an increasing number of submitted tasks. Table IV shows the overhead costs, specifying the overhead incurred for transmission ($TO$), the one for computation ($CO$) and the overall overhead ($OH$) detailed also in terms of percentage value on the total amount of energy consumed by the network of devices. Moreover, the overhead is also specified with respect to the node type, either cluster-head ($TO\_CH$ and $CO\_CH$) or cluster member ($TO\_CM$ and $CO\_CM$). Node overhead values are expressed as average values. The results show that the overhead is very small and the computation term ($CO$) is negligible. Precisely, one can note that the EA overhead constitutes a very small percentage of the energy consumed by the smartphones to execute the submitted tasks (workload). In fact, the additional work required by the scheduling strategy remains around 3% by submitting a number of tasks less than or equal to 600. The overhead reaches approximately 8% when 2400 tasks are submitted; however it is actually quite small given the amount of work that is being done. It is important to note that the overhead refers to the total energy depletion incurred by the entire network of devices; thus, the energy consumed by each single node can be considered as negligible. In fact, we can see that the cluster member nodes present on average really a negligible overhead cost both in terms of transmission (from 0.0128 to 0.1694 KJ) and mostly that of computation (from 0.00052 to 0.0062 KJ).

Figure 17 highlights the comparison between the overhead (showing the overall overhead, as well as its computation and communication components) and the workload, pointing out that both increase with the number of submitted tasks. As already underlined, it is evident that the EA overhead is really marginal compared to the workload. Note that the workload remains constant after the submission of 1200 tasks, while the overhead keeps a rising trend.

This is because not all submitted tasks are completed, as shown in Table IV, while for each task, the assignment process is started. Anyway, even when the number of submitted tasks is very high, the overhead remains negligible.

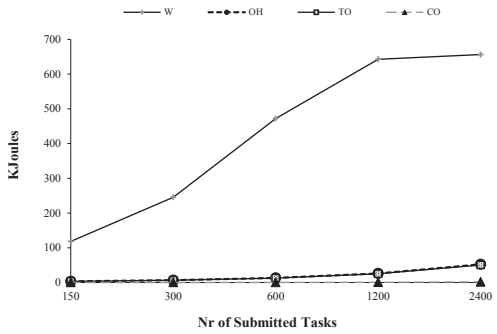
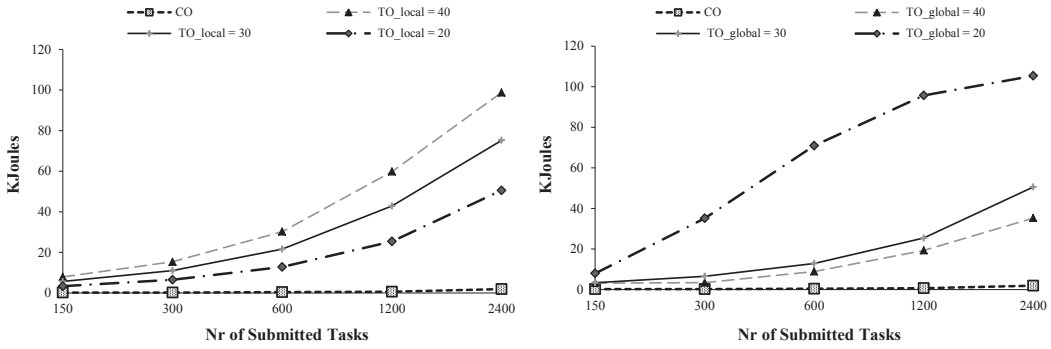



Figure 17. Overhead (OH) versus Workload (W) w.r.t. number of submitted tasks.

### *4.2.6. Parameters selection*

As discussed in Section 3.4, two parameters are used in the task allocation algorithm: $local_{thr}$ in the allocateTask method, and $global_{thr}$ in the localAssignment method. In order to determine the most suitable values for such parameters, we conducted some simulation experiments which show how the thresholds influence EA overhead.




(*a*) (*b*)

Figure 18. (a) Local threshold and (b) Global threshold w.r.t. number of submitted tasks.

Figures 18(a) and (b) show how the computation overhead ($CO$) and the transmission overhead ($TO$) vary using different values of $local_{thr}$ and $global_{thr}$, respectively. In particular, Figure 18(a) shows the results for three values of $local_{thr}$ (20, 30 and 40%) with $global_{thr}$ fixed to 30%. As expected, the computation overhead is not influenced by $local_{thr}$. In contrast, the

communication overhead is significantly affected by the value of this threshold, and the lowest value is obtained with $local_{thr} = 20\%$. This is due to the fact that the lower the threshold, the higher the number of local assignments, which in turn reduces the communication overhead between cluster. For this reason, we chose $local_{thr} = 20\%$ in our simulations.

Regarding $global_{thr}$, which is used in the global assignment phase, its impact on the overhead is shown in Figure 18(b). Differently from $local_{thr}$, the higher the value of the threshold, the lower the number of global assignments, and therefore the lower the communication overhead. However, we can note that the difference between 30% and 40% is very low, particularly when the number of submitted tasks is below 600. Therefore, in our simulation we chose $global_{thr} = 30\%$ as it is a trade-off that allows keeping low the communication overhead without overloading local groups.

## 5. Related Work

Most of the existing research work in the area of energy-aware systems concerns hardware-based techniques focused on reducing the energy consumption of the processor. Among those techniques, turning off idle components is widely adopted [26]. Dynamic Voltage Scaling (DVS) is another hardware-based technique for energy conservation, allowing for simultaneously varying the processor voltage and frequency as per the energy performance level required by the tasks [27, 32, 31]. Accordingly, several energy-saving scheduling (e.g. [28, 29]) algorithms for multi-processor distributed systems are focused on architectures with dynamic voltage scaling or dynamic power management capabilities. Another important work in this direction is [30] where the authors formulate a multi-processor energy-aware scheduling problem for certain architectures of embedded systems and propose a heuristic to solve it.

Among software-based energy-aware techniques, remote execution provides that a device with limited energy transfers a computational task to a nearby device which is more energy powerful [3, 2]. Software-based energy management emerged as a key topic for resource management in distributed cluster-based systems. There are, thus, several works focusing on energy optimization in multiprocessor environments, such as [4, 6, 7]. The use of virtualization for consolidation is presented in [8], which proposes an approach for power optimization in virtualized server clusters through an algorithm that dynamically manages the virtualized servers. Following the same idea, [9] aims to reduce the power consumption in virtualized data center by supporting task migration and task placement optimization. Verma et al. [10] also propose a virtualization aware adaptive consolidation approach, measuring energy costs executing a given set of applications and using correlation techniques to predict usage. A system that is partially related to ours is PARM [11]. It is a distributed power-aware middleware framework consisting of several distributed servers (service providers), proxy servers, meta-data repositories (directory services) and mobile clients. The framework adapts to the diminishing power availability of the devices over time, by dynamically offloading expensive middleware components to a proxy.

Energy-aware task scheduling is another software method where the scheduling policy aims at optimizing the energy consumption. To the best of our knowledge, little work has been done on energy-aware scheduling over a mobile network. Alsalih et al. [33] proposed an energy-aware

dynamic task allocation algorithm over mobile networks. However, this work is different from ours in terms of the underlying architecture and cost function to be optimized. We clustered the devices to promote local cooperation among nearby devices and minimize the transmission energy. This issue is particularly relevant because we have found that the transmission energy highly impacts on the overall energy consumption. In contrast to ours, the solution proposed in [33] does not address the communication aspects of the system. Furthermore, we adopt a different objective function: we maximize the network lifetime rather than minimizing the energy consumption. Using the network lifetime parameter we are able to actually consider the real energy consumption rate of single devices, single clusters and the overall network. Conversely, [33] does consider only the local computation issues and works at a node level without taking into account the workload in the rest of the network.

We conclude the section by discussing some interesting solutions for job scheduling in mobile ad hoc grids, which broadly relate to our system, but do not focus on energy aspects.

Hummel and Jelleschitz [36] introduced a decentralized job scheduler for mobile peers forming an ad-hoc grid. The scheduling approach is based on a first come first serve strategy executed locally by each peer. Coordination between mobile peers is based on job queues shared within a distributed virtual shared memory. To achieve robustness, the system implements proactive and reactive fault tolerance mechanisms by means of job replication and system recovery. As mentioned earlier, this system does not focus on energy. In addition, it focuses on mobile computers (notebooks), in contrast to our EA schedulers that focuses on small devices (smartphones).

Bhagyavati and Kurkovsky [37] proposed a grid-based environment that embraces mobile devices within a wireless cell to share their computational power. The goal of the system is to enable resource-limited devices to solve computationally-expensive problems by redistributing parts of the problem onto other devices. Following the multi-agent paradigm, each mobile device is viewed as an autonomous intelligent agent, capable of performing independent tasks, sharing resources with other agents, and communicating with other agents in the grid.

Phan et al. [39] proposed an approach to integrate wireless devices into a computational grid. In order to compensate the inherent limitations of wireless devices to successful utilize them in a grid, the work suggests a proxy-based, clustered system architecture. In this architecture, wireless devices (from PDAs to laptops) are grouped in clusters. Each cluster is centered around a proxy (called interlocutor) that can be either another wireless device, a server within the grid, or a dedicated middleware server. The interlocutor runs appropriate grid middleware to publish itself as a node that can contribute a certain amount of computational resources, which is the aggregate total of the resources of the wireless devices in the cluster headed by the interlocutor. Since the focus of this system is on architectural aspects, it does not address job scheduling and energy efficiency issues.

Finally, Park et al. [38] addressed the same problem of integrating mobile devices into a grid. The grid system is divided into three parts: static grid sites, a group of mobile devices, and a gateway interconnecting static and mobile resources. A central queuing server runs on the gateway to implement job distribution on the available mobile devices, with a focus on reliable job management. Different to this work, our solution adopts a decentralized scheduling approach that is more suitable to a network including only wireless devices.

## 6. Conclusions

Distributed computing over mobile environments demands for effective scheduling strategies addressing both the energy constraints of battery-operated wireless devices and the decentralized nature of mobile networks. The Energy-Aware (EA) scheduling strategy proposed in this paper was designed to address this two-fold need. To maximize network lifetime and the number of alive devices, the EA scheduler implements a heuristic algorithm that balances the energy load among all the devices exploiting a cluster-based architecture. A performance analysis has been made to assess the efficiency of the EA strategy in different network scenarios. The performance results showed that by using the proposed energy-aware task allocation approach, the network lifetime can be extended and the number of alive devices can be significantly higher compared to alternative scheduling strategies, while meeting application-level performance constraints. Based on these results, we conclude that the EA approach combined with a cluster-based architecture is an effective strategy to support distributed task execution over small devices in wireless scenarios. As a possibile future work, the EA strategy could be extended to take into account computational heterogeneity of the devices.

## REFERENCES


1. F. Bsching, S. Schildt, L.C. Wolf: DroidCluster: Towards Smartphone Cluster Computing - The Streets are Paved with Potential Computer Clusters. *ICDCS Workshops 2012*, pp. 114-117, (2012).
2. Z. Li, C. Wang, and R. Xu. "Computation offloading to save energy on handheld devices: a partition scheme". *ACM International Confenrence Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 238-246, (2001).
3. A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. "Saving portable computer battery power through remote process execution". *SIGMOBILE Mobile Computing and Communication Review*, 2(1):19-26, (1998).
4. R. Bianchini and R. Rajaniony. "Power and energy management for server systems". *Computer*, 37(11):68–76, (2004).
5. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. SIGKDD Explor. Newsl., 11(1):10-18, November 2009.
6. C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller. "Energy Management for Commercial Servers". Computer, 36(12):39–48, (2003).
7. E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. "Load balancing and unbalancing for power and performance in cluster-based systems". *Workshop on Compilers and Operating Systems for Low Power*, volume 180, pp. 182–195, (2001).
8. V. Petrucci, O. Loques, B. Niteroi, and D. Moss. "Dynamic configuration support for power-aware virtualized server clusters". *21th Euromicro Conference on Real-Time Systems*, 2009.
9. L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen. "GreenCloud: a new architecture for green data center". *6th international conference on Autonomic computing and communications*, pp. 29–38, (2009).
10. A. Verma, P. Ahuja, and A. Neogi. "Power-aware dynamic placement of hpc applications". *International Conference on Supercomputing*, pp. 175–184, (2008).
11. S. Mohapatra, N. Venkatasubramanian, "PARM: Power Aware Reconfigurable Middleware". *23rd International Conference on Distributed Computing Systems*, pp. 312–319, (2003).
12. J. Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile applications". *Symp. on Operating Systems Principles*, pp. 48–63, (1999).
13. S. Gurun and C. Krintz. "Addressing the energy crisis in mobile computing with developing power aware software". *UCSB, Computer Science Department, Technical Report*, (2003).
14. L.D. Fife and L. Gruenwald, "Research Issues for Data Communication in Mobile Ad-Hoc Network Database Systems". *ACM SIGMOD RECORD*, 32(2):42–47, (2003) .

15. C. Comito, D. Talia, and P. Trunfio. “An Energy-Aware Clustering Scheme for Mobile Applications’. *IEEE Scalcom’11*, pp. 15–22, (2011).
16. T.S. Rappaport. “Wireless Communications: Principles and Practices”. 2nd Edition, Prentice Hall, (2002).
17. J.-H. Ryu, S. Song, and D.-H. Cho. “New Clustering Schemes for Energy Conservation in Two-Tiered Mobile Ad-Hoc Networks”. *IEEE ICC’01*, 3:862-66, (2001).
18. M. Chatterjee, S. Das, and D. Turgut. “Wca: A weighted clustering algorithm for mobile ad hoc networks”. *Cluster Computing Journal*, 5(2):193-204, (2002).
19. A. B. McDonald and T. F. Znati. “A Mobility-based Frame Work for Adaptive Clustering in Wireless Ad Hoc Networks”. *IEEE JSAC*, 17:1466-87, (1999).
20. M. Lichman. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
21. C. R. Lin and M. Gerla. “Adaptive Clustering for Mobile Wireless Networks”. *IEEE JSAC*, 15:1265-75, (1997).
22. R. Garey, D. Johnson. “Bounds for multiprocessor scheduling with resource constraints”. *SIAM J. Computing*, 4:187–200, (1975).
23. L. M. Feeney. “An energy-consumption model for performance analysis of routing protocols for mobile ad hoc networks”. *Mobile Networks and Applications Journal*, 6(3):239-250, (2001).
24. L. M. Feeney, M. Nilsson. “Investigating the energy consumption of a wireless network interface in an ad hoc networking environment”. *INFOCOM* 2001: 1548-1557, (2001).
25. H. W. D. Chang, W. J. B. Oldham. “Dynamic task allocation models for large distributed computing systems”. *TPDS*. 6:1301–1315, (1995).
26. K. Li, R. Kumpf, P. Horton and T. Anderson. “A Quantitative Analysis of Disk Driver Power Management in Portable Computers”. *USENIX* conference, pp. 279-292, (1994).
27. J. Zhuo, C. Chakrabarti. “An efficient dynamic task scheduling algorithm for battery powered DVS systems”. *ASP-DAC’05*, pp. 846–849, (2005).
28. Y. Zhang, X. Hu and D. Chen. “Task scheduling and voltage selection for energy minimization”. *DAC’02*, pp. 183–188, (2002).
29. J. Luo, N. K. Jha, “Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems”. *ICCAD*, (2000).
30. J. Hu, R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints”. *ASP-DAC*, (2003).
31. K. Seth, A. Anantaraman, F. Mueller, E. Rotenberg. “FAST: Frequency-Aware Static Timing Analysis”. IEEE RTSS, pp.40-51, Dec. 2003.
32. H. Aydin, R. Melhem, D. Moss, P. Mejia-Alvarez. “Power-Aware Scheduling for Periodic Real-Time Tasks”. *IEEE Transaction on Computers*, 53(5):584–600, (2004).
33. W. Alsalih, S. G. Akl, H. S. Hassanein. “Energy-Aware Task Scheduling: Towards Enabling Mobile Computing over MANETs”. *IPDPS’05*, pp. 242a, (2005).
34. D. Talia, P. Trunfio. “Service-oriented distributed knowledge discovery” Chapman and Hall/CRC, 2012
35. C. Comito, D. Falcone, D. Talia, P. Trunfio, ”Energy Efficient Task Allocation over Mobile Networks”. Proc. of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2011), Sydney, Australia, pp. 380–387, December 2011.
36. K. A. Hummel, G. Jelleschitz: A Robust Decentralized Job Scheduling Approach for Mobile Peers in Ad-hoc Grids. CCGRID 2007: 461-470
37. Bhagyavati, S. Kurkovsky. Wireless Grid Enables Ubiquitous Computing. Proceedings of The 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2003), Reno, NV.
38. S.-M. Park, Y.-B. Ko, J.-H. Kim. Disconnected Operation Service in Mobile Grid Computing. In 1st Int. Conference on Service-Oriented Computing,pages 499-513, 2003.
39. T. Phan, L. Huang, C. Dulan. Challenge: Integrating Mobile Wireless Devices into the Computational Grid. In 8th Int. Conference on Mobile Computing and Networking, pages 271-278, 2002.