Future Generation Computer Systems 78 (2018) 257–271

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Dynamic energy-aware scheduling for parallel task-based application in cloud computing

Fredy Juarez [a,b], Jorge Ejarque [a,*], Rosa M. Badia [a,c]

[a] Barcelona Supercomputing Center (BSC), Workflows and Distributed Computing Group, Barcelona, 08034, Spain

[b] Instituto Tecnológico Superior de Álamo Temapache, Xoyotitla, Veracruz, 92730, Mexico

[c] Artificial Intelligence Research Institute (IIIA), Spanish National Research Council (CSIC), Spain

HIGHLIGHTS

- Energy-aware run-time scheduler for task-based applications.
- Model for estimating the application Energy consumption.
- Methodology to automatically generate the required power consumption profile.
- Multi-heuristic resource allocation algorithm to get solutions in polynomial time.
- Energy saving/performance trade-off evaluation for different scenarios.

ARTICLE INFO

Article history:
Received 1 December 2015
Received in revised form
20 April 2016
Accepted 23 June 2016
Available online 5 July 2016

Keywords:
Distributed computing
Cloud computing
Green computing
Task-based applications
Energy-aware scheduling
Multi-heuristic resource allocation

ABSTRACT

Green Computing is a recent trend in computer science, which tries to reduce the energy consumption and carbon footprint produced by computers on distributed platforms such as clusters, grids, and clouds. Traditional scheduling solutions attempt to minimize processing times without taking into account the energetic cost. One of the methods for reducing energy consumption is providing scheduling policies in order to allocate tasks on specific resources that impact over the processing times and energy consumption. In this paper, we propose a real-time dynamic scheduling system to execute efficiently task-based applications on distributed computing platforms in order to minimize the energy consumption. Scheduling tasks on multiprocessors is a well known NP-hard problem and optimal solution of these problems is not feasible, we present a polynomial-time algorithm that combines a set of heuristic rules and a resource allocation technique in order to get good solutions on an affordable time scale. The proposed algorithm minimizes a multi-objective function which combines the energy-consumption and execution time according to the energy-performance importance factor provided by the resource provider or user, also taking into account sequence-dependent setup times between tasks, setup times and down times for virtual machines (VM) and energy profiles for different architectures. A prototype implementation of the scheduler has been tested with different kinds of DAG generated at random as well as on real task-based COMPSs applications. We have tested the system with different size instances and importance factors, and we have evaluated which combination provides a better solution and energy savings. Moreover, we have also evaluated the introduced overhead by measuring the time for getting the scheduling solutions for a different number of tasks, kinds of DAG, and resources, concluding that our method is suitable for run-time scheduling.

1. Introduction

Recent studies [1,2] have estimated that around 1.5%–2.0% of the total energy consumption is consumed by data centers, and this energy demand is growing extremely fast due to the popularization of Internet services and distributed computing platforms such as clusters, grids, and clouds. Regarding the efficiency of data centers, studies have concluded that, in average, around 55% of the energy consumed in a data center is consumed by the computing system and the rest is consumed by the support system such as cooling, uninterrupted power supply, etc. For that reason, green cloud

* Corresponding author.
E-mail address: jorge.ejarque@bsc.es (J. Ejarque).

http://dx.doi.org/10.1016/j.future.2016.06.029
0167-739X/

computing is essential for ensuring that the future growth of cloud computing is sustainable [3].

There are several ways to reduce the energy consumed by an application when executed on a distributed platform: It includes the usage of low-power processor architectures or dynamic voltage frequency scaling (DVFS) [4], re-design of algorithms using energy-efficient patterns in compilers [5] or changing the scheduling policies for task-based applications on the available resources [6]. Traditionally, scheduling techniques have tried to minimize the total execution time of an application(makespan—C_{max}) [7] without worrying about the energy consumed. However, there is a trade-off between energy consumed and the execution time, and sometimes increasing the performance for a faster execution implies a higher energy consumption.

The aim of our work is to offer resource providers and end-users more options for executing task-based applications in an energy conscious manner, giving the possibility of reducing energy consumption without a significant increase in total execution time or reducing the total execution time without a significant increase in energy consumption. In this paper, we present an energy-aware scheduling system for task-based applications. To decide which is the best scheduling solution according to the energy consumed, we propose a model for estimating the energy consumed by the application for a given application and a resource power consumption profile.

Since task allocation on distributed computing resources is a well known NP-hard problem in the general form [8], due to the time limitation required for run-time schedulers, the implementation of large-time optimization algorithms is not suitable. For a real time scheduler it is convenient to develop heuristic techniques for sub-optimal solutions, in order to build a scheduling algorithm that runs in polynomial-time without performing exhaustive search. Therefore, we propose the use of a multi-heuristic resource allocation (MHRA) that is essentially a faster local search algorithm for partial solutions. The algorithm is divided into two phases: the first phase combines a set of heuristic rules for ranking an eligible group of parallel tasks for a given Direct Acyclic Graph (DAG), based on the amount of data transfers, number of task predecessors or successors and execution time. The second phase combines a set of importance factors for the resource allocation algorithm that are used to determine which is the best position in the cloud for a specific task that minimizes energy (E_{flow}) and makespan (C_{max}). The algorithm provides good real-time scheduling solutions in an affordable time scale.

The proposed scheduler has been designed to be applied to the COMP Superscalar (COMPSs) framework [9,10]. It provides an infrastructure-agnostic task-based programming model, which facilitates the development of parallel applications in distributed computing platforms. Developers can program their applications in a sequential fashion and without caring about the details of the underlying infrastructure. They just need to identify the tasks, which are the methods of the applications, to be executed in the distributed platform. At run-time, COMPSs detects data dependencies between tasks creating a DAG. Once the DAG is created, the COMPSs runtime will use the energy-aware scheduler for allocating and executing the tasks on the available computing resources in order to minimize energy or makespan.

The scheduler has been tested with different kinds of DAGs generated at random as well as on real COMPSs applications: embarrassingly parallel (EB), parallel reduction (PR), parallel increase/reduction (PIR), matrix multiplication (MT). Using three different size instances and importance factors. We have evaluated which combination of MHRA provides a better solution and energy savings and the execution time in each case, and the effect on the cloud elasticity. Moreover, we have also evaluated the introduced overhead by measuring the time for getting the scheduling solutions for a different number of tasks, kinds of DAG, and resources, concluding that it is suitable for run-time scheduling.

The rest of the paper is organized as follows: First, Section 2 presents the related work in energy-aware scheduling and Section 3 gives an overview of the COMPSs framework. Afterwards, Section 4 formulates the energy-aware scheduling problem, and the model used to estimate the application energy consumption, the profiling methodology and the multi-heuristic resource allocation algorithm are presented in Sections 5–7, respectively. In Section 8, we present the experiments performed to evaluate the proposed scheduler. Finally, Section 9 draws the conclusions and proposes guidelines for future work.

2. Related work

Traditional task scheduling algorithms for distributed platforms such as clusters, grids, and clouds, focus in minimizing the execution time [11,12] without considering energy consumption. Regarding specific work on energy-aware scheduling two main trends can be found in the literature: (1) pure scheduling software and (2) combined scheduling hardware/software. For combined scheduling, a commonly used technique is taking profit of the Dynamic Voltage Frequency Scaling (DVFS) feature which enables processors to reduce the energy consumption. By using DVFS, processors can run at different voltage, impacting on the frequency and energy consumption.

In [13] the authors present a scheduling heuristic for reducing power consumption of precedence-constrained parallel tasks in a cluster with DVFS, the model is proposed for applications that have slack time for non-critical jobs while they scale down supply voltage for reducing energy consumption and extend execution time of jobs. The model considers homogeneous nodes as processing elements (PEs) with the same processing speed and uses green SLA negotiation in order to accept a tolerable performance loss. In our work, we take into account makespan and energy consumption in a similar way, but in addition we take into account the cloud environment and setup times for creating and destroying VMs; in contrast for reduced energy consumption, we do not use DVFS, because our model minimizes energy consumption while striving to maintain application performance.

Another approach for combining DVFS in scheduling is proposed in [14]. In this case, the authors propose a scheduling algorithm in order to reduce power consumption by applying DVFS for enabling processors to run at low frequencies and low voltages. It is only applicable for those applications in which the performance is not important or can run under certain threshold frequency. The algorithm complies with the Service Level Agreement and obeys the SLA to assign resources for the job given to the consumers. The algorithm takes into account the maximum $Job(F_{max})$ and minimum $Job(F_{min})$ frequencies given for each job and the multiple servers S_i that are running at maximum $S_i(F_{max})$ and minimum $S_i(F_{min})$ frequencies. For specific jobs, the algorithm selects a server that runs between, (F_{min}, F_{max}) and guarantees the execution performance of the job while ensuring that the job does not overuse resources. In contrast, our algorithm detects parallelism at run-time by analyzing the task dependencies of a job (application) and allocates these tasks on specific VMs that have impact on the total energy and makespan.

Beloglazov et al. [6] propose an energy-aware allocation heuristic for client applications over the data center resources, considering QoS expectations. The green Cloud architecture proposed here, it is a power model that divides the energy level consumption in an idle server and running a server with the CPU utilization controlled by DVFS for different frequency and voltage utilization. This model proposes VM allocation divided in

two parts: the first part performs VM provisioning and placing the VMs on hosts like a bin packing problem with different bin sizes and prices, and the second part performs the optimization of the current VM allocation similarly to [15] the authors use a modification of the Best Fit Decreasing (MBFD) algorithm [16]. To optimize the current VM allocation first selects VMs that need to be migrated, then the chosen VMs are placed on the hosts using an MBFD algorithm. The idea is to migrate VMs according to upper and lower utilization thresholds of the hosts in order to consolidate resources.

In [17] the authors propose a cloud controller for performance-base pricing; the cloud controller is in charge of determining actions to allocate and manage the VMs, taking into account CPU frequency scaling, pricing and geo-temporal inputs such as real-time electricity prices [18] and temperature-dependent cooling [19], the problem is balancing the trade-off between energy savings and revenue loss for performing actions by scaling CPU frequency, for creating and migrate, suspend or resume VMs that have to pay a high cost in energy. The proposed controller can determine the optimal CPU frequency between energy savings and workload performance, and can reevaluate control actions at run-time when considering geo-temporal inputs, electricity prices, and temperatures. The difference with our work is that we do not only take into account the allocation of the VMs, also we take into account the energy consumption of these actions and of the data transfer in task-based applications.

However the DVFS feature is not always available in computing systems, especially when nodes are shared by different application users. In those cases, we can only use traditional software for scheduling task on resources. For instance in [7], the authors propose an energy-constrained provisioning for scientific workflow ensembles, focused on the provisioning of resources for scientific workflow ensembles and address the problem of meeting energy constraints along with either budget or deadline constraints. The application model is focused on workflow based applications modeled as a DAG and do not take into account the time for data transfers. The cloud resources assume that jobs do not run concurrently on VMs and offer different VM instance types with CPU, memory, and hard disk combinations, but only a single VM instance is used for simplicity. As for our work, the problem considered take into account task-based applications with fine granularity, also take into account the time and energy for data transfer, VM setup, VM shutdown and our resource profiler can model VMs with different architectures, different number of CPUs and memory.

Other examples of how to perform application scheduling with multi-objective criteria can be found in [20]. The authors propose a Bi-criteria workflow task allocation and scheduling in cloud computing environments, in order to minimize the cost incurred by using a set of resources and total execution time. The bi-criteria function takes into account the time execution and data transfer. They proposed three algorithms to schedule workflows. The first algorithm focuses on the cost incurred in execution time and transfer time of the workflow. The second algorithm aims to minimize the total execution time by makespan criterion. The third algorithm is based in the Pareto solution [21] of the first and second approach. This work is similar to our work in the use of a bi-criterion objective function, but our combination is between energy consumption and total execution time and takes into account the time and energy consumption for more elements such as cores, VMs, and nodes.

A similar approach is proposed in [22], where the authors present a cost-efficient task scheduling for executing large programs in the cloud, using two heuristic strategies. The first strategy dynamically maps tasks to the most cost-efficient VMs, based on the concept of Pareto dominance. The second strategy, a complement to the first strategy, reduces the monetary costs of non-critical tasks. This algorithm shows that we can reduce monetary costs while producing a good makespan like [17] does, but without using the DVFS technique.

Finally, in [23] the authors propose criticality-aware dynamic task scheduling on heterogeneous architectures with OmpSs [24,25]. The paper proposes scheduling policies in order to reduce the total execution time of a task represented by DAG. The algorithm takes into account the critical path (CP) and task prioritization, by keeping two ready queues, one for critical and one for non-critical tasks respectively, and insert each task in the corresponding queue according to its priority. To execute a task in a heterogeneous platform, the critical tasks in the critical queue are assigned to fast processors and non-critical tasks in the non-critical queue are assigned to slow processors, in order to minimize the total execution time. Compared with our work, both seek to minimize the total execution time (Makespan), but our work also takes into account the energy consumption plus the cloud infrastructure.

3. COMP Superscalar overview

COMP Superscalar (COMPSs) is a framework that provides a programming model for the development of task-based applications for distributed environments and a runtime to efficiently execute them on a wide range of computational infrastructures such as clusters, grids and clouds. The aim of COMPSs is to provide an easy way to develop parallel applications, while keeping the programmers unaware of the execution environment and parallelization details. The programmers do not require prior knowledge about the underlying infrastructure, they are only required to create a sequential application and specify which methods of the application code will be executed remotely. This selection is done by providing an annotated interface where these methods are declared with some metadata about the directionality of their parameters.

3.1. Runtime

Once the application is developed with the COMPSs Programming Model, developers can run their task-based applications on the different distributed platforms without requiring to do any change in the code. The COMPSs runtime, shown in Fig. 1, is in charge of transparently detecting the inherent parallelism and of executing the application exploiting the available resources in the distributed computing platform. To achieve this, the COMPSs runtime intercepts any method call defined as tasks and substitutes them to calls to a Task Analyzer. The Task Analyzer creates a task-dependency graph (DAG) by analyzing the data-dependencies between the different invoked tasks.

Once a task is free of dependencies, the Task Scheduler is responsible of assigning a infrastructure resource to execute the task taking into account data locality, task constraints and the workload of each resource. The information required for the scheduling is provided by the Data Info provider, which tracks the locations of the different versions and replicas of the application data, and the Resource Manager, which provides the information of the available infrastructure resources and performs elasticity actions to adjust the number of resources with the current workload. When the number of tasks is higher than the available cores, the resource manager takes into account the offers of the cloud providers to determine the one that offers the resources that better fit the requirements of the application with lower economic cost. Symmetrically, when the runtime detects an excess of resources, it powers off unused instances in a cost-efficient way [26].

Finally, the Job Manager is in charge of submitting the task execution in the selected resource and of monitoring it during

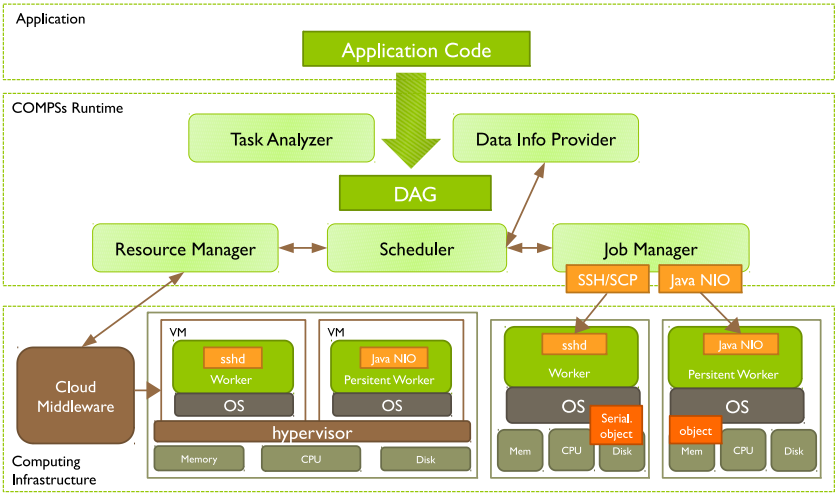


Fig. 1. COMPSs runtime architecture.

its execution life-cycle. As part of this task execution monitoring, the COMPSs runtime keeps track of the duration of the different tasks executions in the different resource. With this historical data, the runtime can generate a regression model for each type of task to estimate the duration of future executions in the available resources. Once a task execution finishes, the tasks duration model is updated with the duration of the new task, the DAG is updated, possibly resulting in new dependency-free tasks that can be scheduled.

4. Problem formulation

As explained in the previous section, COMPSs translates a sequential program S into a directed acyclic graph (DAG) by evaluating data and task dependencies between parts of code in real time. The generated DAG, represented by $G = (T, D)$, contains a set of vertexes T, which represents the task invocations, and a set of edges D, which represents dependencies between task invocations. For two tasks $i, j \in T$ an edge of the form $(i \rightarrow j)$ denotes a relation task dependency between tasks i, j, where the task j should be executed after task i is completed. In this sense i is called parent and j is called child. While a child j may have many parents i, j is only ready when all parents i have been completed.

On the other hand, a Cloud platform can be described as a set of nodes $N = \{n_1, n_2, \ldots, n_m\}$, where node m is represented by n_m. Each node n_m is responsible for managing a set of virtual machines (VM) $V_m = \{v_{m1}, v_{m2}, \ldots, v_{mk}\}$, where VM k of node m is represented by v_{mk}. Each VM v_{mk} has a set of processing cores $C_{mki} = \{c_{mk1}, c_{mk2}, \ldots, c_{mki}\}$, where each core i of VM v_{mk} is represented by c_{mki}.

Finally, the scheduling problem consists on looking for a task scheduling S, which represents the execution order of each task j on the set of available resources. So, the scheduling for c_{mki} (represented by S_{mki}) is given by the expression (1), which represents the order that n task is executed on the ith core of the kth VM of mth node ($j_n \rightarrow c_{mki}$), and the complete scheduling solution S can be represented by expression (2), as the set of schedulings for all the resources of the Cloud.

$$S_{mki} = \{j_1 \rightarrow c_{mki}, j_2 \rightarrow c_{mki}, \ldots, j_n \rightarrow c_{mki}\} \quad (1)$$

$$S = \{\forall_{m,k,i} S_{mki}\}. \quad (2)$$

Finally, we will also refer to task j, with $j \in T$, scheduled at core i of VM k at node m by the term $task^j_{mki}$, and the set of tasks scheduled at c_{mki} as T_{mki} and the set of tasks scheduled at VM v_{mk} as T_{mk}.

In our scheduling problem, the process of looking for the proper task scheduling is driven by two goals:

1. To improve the energy efficiency by looking for the best positions of tasks in resources which minimize the energy consumed by the execution of the whole set of tasks T.
2. To improve the execution performance by searching for resource allocations that minimize the total execution time.

However, there is a trade-off between the optimization of the energy consumption and the total execution time. A solution which minimizes the execution time can be inefficient with regards of the energy consumption, and a solution which minimizes the energy consumption may have larger execution times. The aim of our scheduling strategy is to find a good trade-off for both problems. To achieve this goal, we propose a bi-objective cost function which combines the temporal efficiency, defined by the makespan ($C_{\max}$), and the energy efficiency, defined by the total energy flow (E_{flow}) with a normalized weight factor (α) that indicates what is more important for the users and resource providers: the energy-efficiency or the execution time. Therefore, the proposed scheduling problem can be modeled as an optimization problem which looks for a solution S that minimizes the bi-objective cost function, as represented in Eq. (3).

$$minimize \left(\alpha \frac{E_{flow}(S)}{E_{sf}} + (1 - \alpha) \frac{C_{\max}(S)}{C_{sf}} \right). \quad (3)$$

The cost function is composed of two main terms $E_{flow}(S)$ and $C_{\max}(S)$. $E_{flow}(S)$ is the overall energy flow which estimates the energy consumed by the execution of solution S, and $C_{\max}(S)$ is the makespan which is defined as the maximum completion time of the latest task to leave the system [27].

To balance the bi-objective function, we apply the adaptive weighted sum method [28], which introduces a weight factor α, where $0 \leq \alpha \leq 1$, to indicate which term is more important for the user. However, makespan and total energy flow have different units. Makespan is calculated in milliseconds, and energy is calculated in millijoules, hence, to make these metrics comparable we have introduced two factors E_{sf} and C_{sf} to normalize the energy consumption and execution time. As explained above, the makespan ($C_{\max}$) is basically defined by the latest task (C_n) to leave the system. However, the estimation of the E_{flow} term is not easy. Next section provides more details about the model used to estimate E_{flow} for a given scheduling S.

5. Energy model

The bi-objective cost function presented in the previous section requires the calculation of the makespan ($C_{\max}$) and the overall

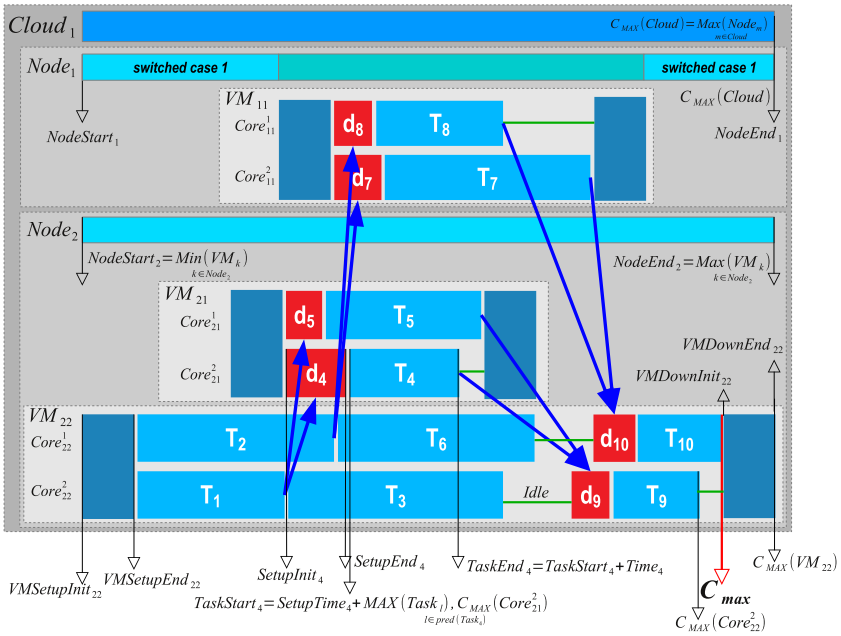


Fig. 2. Schedule times for all elements considered.

energy flow (E_{flow}) for a given a scheduling solution, such as the one described in Fig. 2. From this scheduling solution, the energy-aware scheduler can extract the following application timings to estimate the total execution time and the energy consumption:

- Start and end time of tasks' executions in each specific core.
- Start and end time of data transfers required by each task.
- Start and end time of VMs setup and VMs shutdown.
- Start and end time of overall running time for each VM.
- Start and end time of overall running time for each node.

Another key information to compute the application energy estimation, is the power profile for the different infrastructure resources. For each resource, we need to calculate the following values. The procedure to build energy profiles is explained in Section 6.

1. $P_{c_{mki}}$ is the proportional mean power consumed by using the core i of VM k in node m.
2. $P_{up_{v_{mk}}}$ and $P_{down_{v_{mk}}}$ are the mean power consumed by VM when the hypervisor creates and destroys VM k in node m.
3. $P_{idle_{n_m}}$ is the mean power consumed by a node m in an idle state, in other words, the power consumed when the hypervisor is not handling VMs and only the operating system services are running.
4. $P_{net_{n_m}}$ is the mean power needed for transferring data through the cloud infrastructure.
5. *PUE* (Power Usage Effectiveness) is the proportional mean power consumed by the support system (UPS, cooling, etc.).

With the application timings and the energy profiles we can estimate the energy consumed by each element by multiplying the duration of the different events by the corresponding mean power values. In the following paragraphs, we provide more details about how to calculate the energy consumed by the different events. First we show, how to calculate the energy consumed per task and how they are accumulated per VM and node in order to calculate the total energy flow (E_{flow}).

5.1. Energy consumption per task

To calculate the energy consumed by a task j allocated on core i of VM k in node m ($task^j_{mki}$), we have to calculate start time ($time_{init_j}$), the setup time for transferring files ($time_{setup_j}$), and the end time ($time_{end_j}$). The following paragraph describes how these times and energy is calculated.

5.1.1. Task start time

Each task j has a set of l predecessor tasks (T_{pre_j}), which must finish before starting task j. Therefore, the minimum starting point for task j depending on the predecessors ($time_{pre_j}$) must be the largest end time of the l predecessor tasks which is calculated as indicated on Eq. (4), In case that task j does not have predecessors, the $time_{pre_j}$ time is zero.

$$time_{pre_j} = \max\left(\underset{l \in T_{pre_j}}{time_{end_l}}\right). \tag{4}$$

In the case that a task j is the first task assigned into a core, the start time will directly set to the end time determined by the task predecessors ($time_{pre_j}$). However, if other tasks have already been assigned to the same core, the start time will also depend on the end time of those tasks denoted by $time_{end_t}$. In this case, the task start time will be determined by the maximum value between the largest predecessors end time ($time_{pre_j}$) and the largest end time of previously assigned tasks t at the same core c_{mki}, expressed by $\max(time_{end_t})$ where $t \in T_{mki}$. Finally, in the case that j is the first assigned task to virtual machine k of node m (v_{mk}), the tasks start time will also depend on the time to setup of v_{mk}. If this time is larger than the largest end time of predecessor tasks, the tasks start time ($time_{init_j}$) will be equal to $setup_{v_{mk}}$. In summary, Eq. (5) describes the way to calculate the start time of a task j.

$$time_{init_j} = \max\left(time_{pre_j}, \max_{t \in T_{mki}}(time_{end_t}), setup_{v_{mk}}\right). \tag{5}$$

5.1.2. Task setup time

Before starting the task execution, the data required by a task must be transferred to the resource which is going to execute the task. We refer to this time as setup time, which is represented by $time_{setup_j}$. This time depends on the proposed scheduling solution

because the amount of data to transfer will depend on the resource where predecessor tasks l have been assigned. Therefore, the setup time for task j, described in Eq. (6), is given by the sum of all the transfer times ($time_{transfer_{v_{mk}}}(f_l)$) of those required data which is out of the VM file system.

$$time_{setup_j} = \sum_{l \in T_{pre_j}, l \notin v_{mk}} time_{transfer_{v_{mk}}}(f_l). \quad (6)$$

5.1.3. Task end time

The time when a task is finished ($time_{end_j}$) is easily calculated as indicated on Eq. (7), by adding the task start time ($time_{init_j}$), the task setup time between and the processing time of a task j carried out on core i of VM k of node m ($time_{exec_j^{c_{mki}}}$). This execution time is estimated by the COMPSs runtime based on historical data as introduced in Section 3.1.

$$time_{end_j} = time_{init_j} + time_{setup_j} + time_{exec_j^{c_{mki}}}. \quad (7)$$

5.1.4. Task energy consumption

The energy consumption by a task j, represented by ($e_{task_{mki}^j}$), is estimated by the addition of the energy consumed by data file transfer and the energy consumed by the task processing. The energy consumed by file transfers is estimated by multiplying the time spent to transfer the required data by the mean power of transferring data across the network infrastructure (P_{net}), while the energy consumed by task processing is estimated by multiplying the processing time of the task j by the proportional mean power of using the assigned core in the VM ($P_{c_{mki}}$), as indicated on Eqs. (8) and (9) respectively.

$$e_{transfer_{mk}^j} = time_{setup_j} \times P_{net_{n_m}} \quad (8)$$

$$e_{task_{mki}^j} = e_{transfer_{mk}^j} + \left(time_{exec_j^{c_{mki}}} \times P_{c_{mki}}\right). \quad (9)$$

5.2. Energy consumption per VMs

The second set of parameters that we have to calculate to get the total energy flow is the energy consumed by each VM used in the cloud (v_{mk}). This overall energy amount consumed by the virtual machine, represented by $e_{v_{mk}}$, can be calculated as indicated on Eq. (10). It includes the energy consumption by setup and down of the virtual machine k of node m, which are estimated by multiplying the time spent for setup and down of the VM ($setup_{v_{mk}}$ and $down_{v_{mk}}$) by the proportional mean power ($P_{up_{v_{mk}}}$ and $P_{down_{v_{mk}}}$), and the energy consumption by the different tasks j executed in the virtual machine v_{mk} and it is calculated as shown on Eq. (11).

$$e_{v_{mk}} = setup_{mk} \times P_{up_{v_{mk}}} + e_{tasks_{mk}} + down_{mk} \times P_{down_{v_{mk}}} \quad (10)$$

$$e_{tasks_{mk}} = \sum_{i \in C_{mk}, j \in T_{mki}} e_{task_{mki}^j}. \quad (11)$$

5.3. Energy consumption per node

The energy consumed by a node (e_{n_m}) is basically composed by two terms. The first term is the aggregated energy consumption by the different VMs of a node, which can be calculated as indicated on Eq. (12).

$$e_{V_m} = \sum_{v_{mk} \in V_m} e_{v_{mk}}. \quad (12)$$

The other term is the energy spent by the background OS services which can be estimated by a product of the time that a node is running and the mean power of a node in idle state(P_{idlen_m}). The calculation of the time that a node is running varies depending on the features provided by the infrastructure for powering off nodes. We have considered two scenarios:

Scenario 1: When the cloud infrastructure allows to power off nodes, the running time is determined by the duration of virtual machines running on node m. We can calculate the node running time as indicated on Eq. (13), which is the difference between, the maximum time of the last job to be assigned to any VMs of a node ($time_{end_j}$) plus the time to shutdown the corresponding VM ($down_{mk}$), and the minimum time of the first job to be assigned to any VM of a node ($time_{init_j}$) minus the time to setup of corresponding VM ($setup_{mk}$).

$$run_{n_m} = \max(\max_{j \in v_{mk}}(time_{end_j}) + \min_{k \in node_m}(down_{mk})) - \min(\min_{j \in v_{mk}}(time_{init_j}) - \min_{k \in node_m}(setup_{mk})). \quad (13)$$

Scenario 2: When the cloud infrastructure does not allow to power off nodes, all nodes are running during the same time, and it corresponds to the duration of the whole application, which is also defined by the Makespan (C_{max}). Therefore, the running time of a node in this scenario can be described as in Eq. (14).

$$run_{n_m} = C_{max}. \quad (14)$$

Once the node running time is estimated, we can estimate the total energy consumption per node as indicated by Eq. (15)

$$e_{n_m} = run_{n_m} \times P_{idle_{n_m}} + e_{V_m}. \quad (15)$$

5.4. Makespan and energy flow

Once we have calculated the times and partial energy consumptions for tasks, VMs and nodes, we are ready to calculate the Makespan (C_{max}) and the overall energy flow (E_{flow}). The Makespan of the system can be calculated as shown in Eq. (16). It is basically determined by the end time of the last task j executed in a cloud and the down time of the VM where this last task is executed.

$$C_{max} = \max_{j \in T}(time_{end_j} + down_{v_{mk}}). \quad (16)$$

In the case of the overall energy flow (E_{flow}), it can be calculated as indicated in Eq. (17) by aggregating the energy consumed by the different used node, multiplied by the proportional part of energy consumed by the data center infrastructure (cooling system, UPS, ...) which is given by the Power Usage Effectiveness (PUE).

$$E_{flow} = PUE \times \left(\sum_{n_m \in N} e_{n_m}\right). \quad (17)$$

6. Power profiling

As explained in the previous section, a power profile of the host used in the cloud is required to estimate the overall energy flow. The process of profiling consist of extracting the mean power values for the possible events of an application. In this section, we explain the methodology used to extract this power profile energy, and the values obtained for our testbed which is composed by AMD and INTEL architectures.

6.1. Power profiling procedure

The overall procedure to get the power profile is depicted in Fig. 3. The energy provided to the cloud is manager by Power

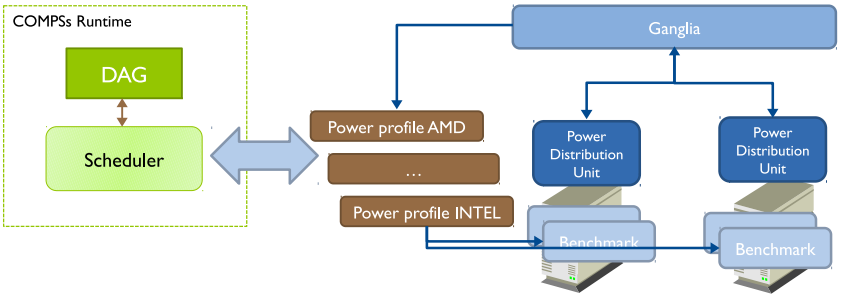


Fig. 3. Profiling for getting power consumption by all elements.

Distribution Units (PDU) which are capable of providing the power provided to the different nodes of the Cloud. The information provided by the PDUs is aggregated by a monitoring system (Ganglia and hsflow in our case) which already provides the resource consumption (network, memory, CPU) per Node and VM. With this system we can run a set of benchmarks which stress the different parts of the system. When the benchmark process has finalized, the power usage information is extracted from Ganglia logs, and it is analyzed to make the power profiles. This power profile is automated by a set of scripts and performed in a setup phase.

The easier part of the power profile to be estimated is the mean idle power $P_{idle_{nm}}$. In this case, we measure the mean power of the nodes when the node is only running the background OS service (idle state), and there are no VMs and tasks or transfers running in the node.

Then, the first benchmark we can perform is the creation of a VM. In this case, for each VM template available in the system, we run a VM creation and measure the time required to create this VM $setup_{v_{mk}}$ and the mean power $P_{setup_{v_{mk}}}$ during the execution of this operation. A similar benchmark is performed destroying a VM and getting the $down_{v_{mk}}$ time and the mean power $P_{down_{v_{mk}}}$.

A second benchmark is designed to stress cores of a VM. The main idea is to run one process per each core of VMs at the same time, starting with one process and finishing with n processes corresponding with the total number of cores distributed through the VMs. Each process consumes 100% of a core. With this benchmark, we can extract the increment of power per core used $P_{c_{mki}}$.

Finally, the last benchmark profiles the power when performing data transfers. For this benchmark, we create two VMs allocated in different nodes with one process in each VM. These processes exchange a large file between them. Measuring the mean power during the time that a VM is sending or receiving the file, we can extract the value for $P_{net_{nm}}$. The difference observed when sending or receiving a file is also neglectable.

We have performed the same benchmark with different VM configurations and directly to the bare metal of the host and we have measured the mean power in the different cases. Analyzing the results, we have observe different interesting facts. The first fact is that the power increment by a VM at idle is neglectable. Once the hypervisor finished with the VM creation process, the VM and Host are in an idle state. The mean power measured when running a VM at idle state is the same than the mean power with the Host at idle state. For this reason, we do not consider VM idle state in the energy model. The other important fact was observed when running the CPU stress tests. In terms of power consumption, the values obtained when running with different VM configurations were almost the same. So, as expected the power consumption depends basically on the real resource usage performed by the processes running in the VMs are doing in the physical host. Therefore, the extra power consumption introduced by the hypervisor mainly appears when a VM is created and destroyed. At operation, the overhead is just observed when

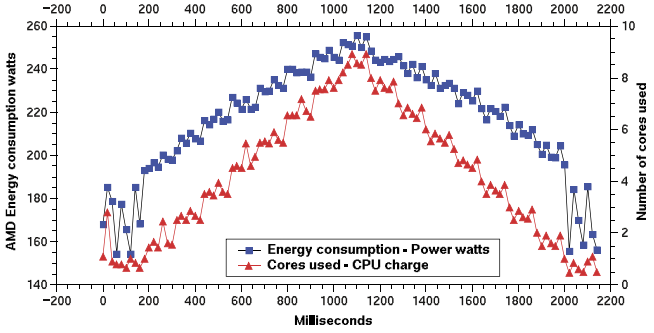


Fig. 4. Power profile for getting energy consumption from AMD nodes.

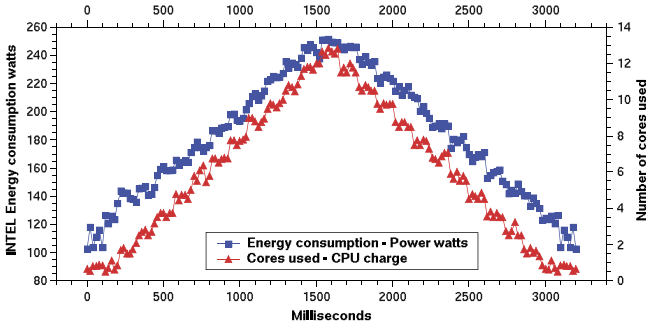


Fig. 5. Power profile for getting energy consumption from Intel nodes.

measuring the time to execute a task, and it is already taken into account when the COMPSs runtime monitors and estimates tasks duration. It means that the hypervisor does not have an observable overhead in terms mean power but it exists in terms of energy which is mainly calculated by the mean power multiplied by the execution time.

6.1.1. Testbed power profile

We have setup the profiling system and we have run the benchmarks for the different type of hosts we have in the testbed. Figs. 4 and 5 show the figures of the power usage for AMD and Intel nodes respectively. These figures show the relationship between the power and cores used over the time as well as the idle times.

In Fig. 4 we can see the power in the idle state, before and after stressing cores, for an AMD host. The estimated mean power used in this case is around 175.67 W. Afterwards, when the use of cores starts, the incremental power usage is around of 9.73 W per core. The same have been done for the Intel nodes (Fig. 5), where we can observe a mean power of 115.30 W and an incremental power of 11.02 W per core used. These values have been estimated with a linear regression as shown in Fig. 6.

Running the other benchmark, we can also estimate the power usage per data transfer, the transfer rate is around of 42.03 MB/s and the time and power usage for creating and destroying VMs in

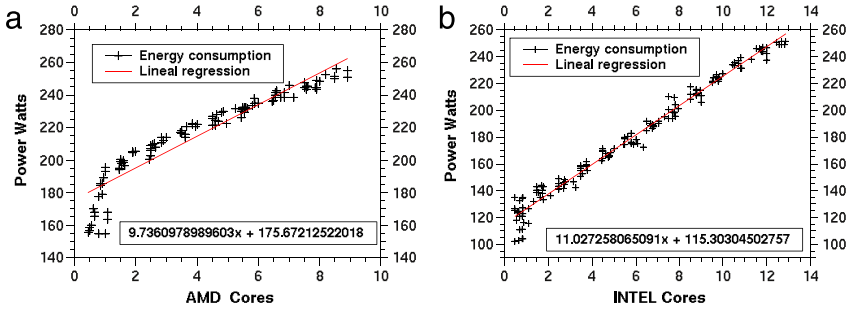


Fig. 6. Linear regression to estimate the incremental power profiling.

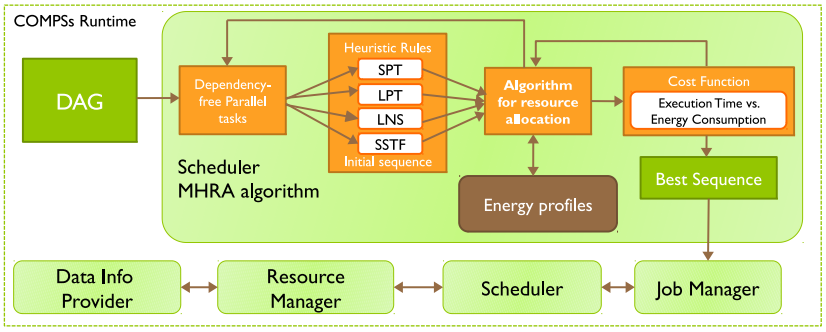


Fig. 7. COMPSs scheduler.

Table 1
Estimated mean power consumption for the different elements of a cloud platform.

Element	AMD mean power	INTEL mean power
$P_{c_{mki}}$	9.73 W	11.02 W
$P_{setup_{v_{mk}}}$	9.49 W	18.24 W
$P_{down_{v_{mk}}}$	9.49 W	18.24 W
$P_{idle_{nm}}$	175.67 W	115.30 W
$P_{net_{nm}}$	30.04 W–42.03 MB/s	27.56 W–42.03 MB/s
PUE	1.2	1.2

the AMD and Intel nodes. The power profile obtained for both types of nodes is summarized in Table 1. Finally, due to the fact that both types of servers are located in the same data center the PUE will be the same.

7. Multi-heuristic resource allocation

In this section, we explain a multi-heuristic resource allocation algorithm (MHRA), which generates scheduling solutions of good quality in near real-time. It is essentially a fast local search algorithm for partial solutions. MHRA builds the solution step by step, taking one task at a time and determining which is the next best location on the infrastructure. Fig. 7 shows an overview of the MHRA policy. The MHRA receives as input a DAG that contains the set of tasks to be carried out on the cloud. The algorithm automatically analyzes the DAG and obtains a subset of tasks free of dependencies that can be executed in parallel. Then, for each subset, the algorithm prioritize the tasks them based on different heuristics rules and applies the resource allocation process which seeks the best resource for each task that minimizes the bi-objective cost function according to the importance factors specified by the user. This cycle between the resource allocation and the objective function evaluation is repeated for each subset of the DAG and heuristic rule. Finally, the scheduler selects the scheduling sequence generated with the heuristic rule which minimizes this cost function. This solution is sent to the COMPSs runtime to get the tasks executed following this sequence. In case that an unexpected event occurs (node failure, etc.) which invalidates the current solution, the runtime updates the DAG and resource configuration and recalculates the scheduling repeating the same process.

The pseudo-code of MHRA algorithm is described in Algorithm 1. It can be divided into two phases:

1. The first phase applies a set of heuristic rules for ranking an eligible subset of a parallel tasks for a given DAG, based on the amount of data transfers, number of task predecessors or successors and time execution.
2. The second phase determines which is the best position (time and resource) in the cloud for each of the specific ranked tasks that minimizes energy and makespan.

7.1. Ranking initial sequences

With the objective of starting from an initial sequence of the tasks that can give a good initial solution, we first analyze the DAG in search of those tasks that do not have precedence constraints or whose tasks predecessors have already been executed. This subset of the DAG tasks can run in parallel in the cloud resources.

For this subset T, we apply heuristic rules for ranking its tasks taking into account the characteristics of the tasks. The list shown below describe the used heuristics.

1. LPT (Longest Process Time). This rule provides greater priority to the tasks whose processing time is greater.
2. SPT (Shortest Processing Time). Selects the tasks whose processing time is smaller.
3. LNS (Longest Number of Successors). Selects the tasks that have a large number of successors.
4. LSTF (Longest Setup Times First). Selects the tasks that have a longer sequence setup time.

In Algorithm 1, lines 2 and 3 show that for each heuristic used, it applies a different energy factor of importance, lines 5 and 6 show that for each subset T of tasks of the DAG that can run in parallel, a ranking based on each heuristics selected is used.

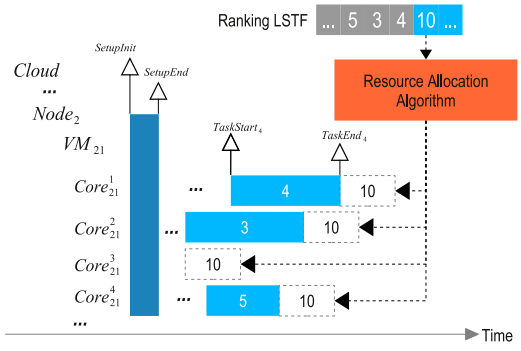


Fig. 8. Initial ranking and resource allocation.

7.2. Task resource allocation

After getting an initial set of sequences by applying the rules described above, we proceed to determine which is the best resource for each task. To achieve this, the resource allocation process is applied as shown in Fig. 8. It inspects the available resources and selects the best position by evaluating the objective function. The metrics used to evaluate the objective function have been described in Section 5. The Algorithm 1 shows, in line 7, after ranking the subset T, how it finds the best resource for each task j that will run in parallel which minimizes the objective function based on the selected heuristics k. The cycle of line 9 tries to find a VM for the current task which improves the objective function. Line 10 builds a new solution S' assigning to the current task a new VM and line 11 evaluates the objective function for the new solution.

Algorithm 1 Multi-heuristic Resource Allocation algorithm

1: **function** $MHRA(DAG, E_{profile}, R_{cloud}, H_{heuristic}, F_{values})$
2: **for** (each $k \in H_{heuristic}$) **do**
3: **for** (each $\alpha \in F_{values}$) **do**
4: $S = \{\}$
5: **for** (each $T \in parallel(DAG)$) **do**
6: $S_{rank} = Rank(k, T)$
7: **for** (each $task^j \in S_{rank}$) **do**
8: $S+ = (first(core_{mki} \in R_{cloud}), task^j)$
9: **for** (each $core_{mki} \in R_{cloud}$) **do**
10: $S' = S + (core_{mki}, task^j)$
11: $f(S') = \alpha \frac{E_{flow}(S')}{s_{f1}} + (1-\alpha)\frac{C_{max}(S')}{s_{f2}}$
12: $S' = VM\_Down(S, R_{cloud})$
13: **if** ($f(S') < f(S)$) **then**
14: **if** (VM of $core_{mki}$ is Down) **then**
15: $S = (VMSetup_{mk}, time_{init_j}) + S'$
16: **end if**
17: $S = S'$
18: **end if**
19: **end for**
20: **end for**
21: **end for**
22: $S_{factor_k^\alpha} = S$
23: **end for**
24: $S_{heuristic_k} = Best(S_{factor_k^\alpha})$, with $\alpha \in F_{values}$
25: **end for**
26: $S_{best} = Best(S_{best_k})$, with $k \in H_{heuristic}$
27: return (S_{best})
28: **end function**

Another important aspect estimated by the algorithm is when a new VM must be created or destroyed, and the effect on time and energy consumption. In line 12 the function $VM\_Dowm()$ call to algorithm 2, to determine if there are VMs that require to be switched off. The algorithm looks in all the VMs, calculating for each VM, if its downtime is greater than the time required to turn it off and turn it on, if so, it inserts into the solution S, the VM identifier and the time in which the VM must be switched off. In contrast, when a resource is selected for running a task, the algorithm, in line 14, determines if a new VM is required and inserts the VM and the time in which the VM must be set up.

Algorithm 2 Stop virtual machines

1: **function** $VM\_Down(S, R_{cloud})$
2: $C_{max} = C_{max}(S)$
3: **for** (each $VM_{mk} \in R_{cloud}$) **do**
4: $VMidle_{mk} = C_{max} - C_{max}(VM_{mk})$
5: **if** ($VMidle_{mk} > down_{mk} + setup_{mk}$) **then**
6: $S+ = (VMdown_{mk}, C_{max}(VM_{mk}))$
7: **end if**
8: **end for**
9: return (S)
10: **end function**

Finally, in lines 22–26, the best solution S for a given initial heuristics k and α importance factor is assigned to $S_{factor_k^\alpha}$. When all heuristics and the factors are evaluated, the solution that produces the best objective function is selected.

8. Experimental evaluation

This section presents the experiments carried out to evaluate the proposed energy-aware scheduler. The first part of the section describes the configuration used for the evaluation including the machine used to run the scheduling and the cloud infrastructure, the benchmark applications and the heuristic used by the scheduler. The second part presents the results of running the energy-aware scheduling for the different benchmarks, showing the effect of the different factors and heuristics in the consumed energy, makespan, the use of resources and the cloud elasticity. The section is finalized with a set of experiments to evaluate the performance of the scheduling algorithm evaluating how the time to get the scheduling solution grows with the number of tasks and resources.

8.1. Experiments

To carry out the experiments, we have implemented a prototype of the proposed energy-aware scheduler and we have installed it in a DELL Notebook with Intel i7-2760QM CPU 2.40 GHz with 8 cores and 8 GB of memory. We have implemented several benchmarking applications with the COMPSs programming model and we have extracted the DAG generated by the runtime which is the input for the scheduler evaluation. We have simulated the scheduling solutions proposed by the energy-aware scheduler for running the applications in an private cloud infrastructure at the Barcelona Supercomputing Center (BSC).

8.1.1. Cloud infrastructure

The private cloud hosted by the BSC consist of two types of computing nodes, AMD and Intel, which are summarized in Table 2. Each node AMD contains 8 cores, a physical memory of 32 GB and a storage capacity of 800 GB, while each Intel node contains 12 cores, a physical memory of 32 GB and a storage capacity of 800 GB. For both nodes is considered a VM representative integrated of four cores, 4 GB of memory and 200 MB.

In each node AMD we can create up to 2 VMs without having a significant performance degradation. The time required to set-up the VM in these nodes is 175 s and three seconds to turn it off. For the Intel nodes we can create up to three VMs. The time required to set-up the VM is 85 s and two seconds to turn it off. The energy profile for these two types of nodes has been already presented in Section 6. For the first experiments where we evaluate the effect of the different factors, we have reserved four nodes (2 AMD, 2 Intel) of this private cloud, resulting in a maximum of 10 VMs and a total of 40 cores. For the performance experiments, we have considered

Table 2
Simulation cloud infrastructure.

Element	AMD node	Intel node
Nodes	2–32	2–32
Physical, Memory, Storage	8, 32 GB, 800 GB	12, 32 GB, 800 GB
VMs	2	3
Cores, Memory, Storage	4, 4 GB, 200 MB	4, 4 GB, 200 MB
Setup, Down	175 s, 03 s	85 s, 02 s

Table 3
DAG size instances.

DAG size	Tasks number
Small	10, 20, 30, 40, 50,60, 70, 80, 90
Medium	100, 200, 300, 400, 500, 600, 700, 800, 900
Large	1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10 000

Table 4
Range of values for task duration and resource consumption.

Element	Range values
Process time (ms)	(85 000, 1 000 000)
Number of cores	(1, 4)
Physical storage (MB)	(100, 1000)
Physical memory (MB)	(100, 1000)

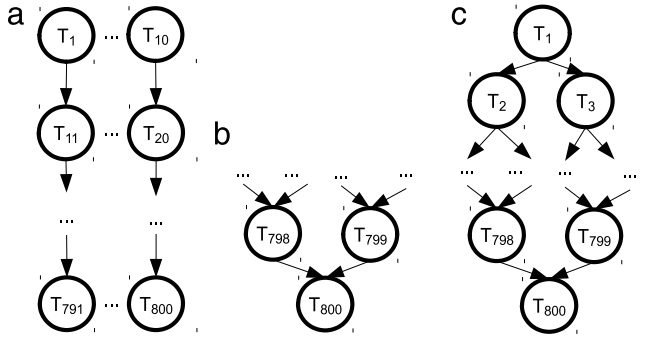


Fig. 9. DAG types MT, GT and SG of 800 tasks.

64 nodes (32 AMD, 32 Intel), resulting in a maximum of 160 VMs and a total of 640 cores.

8.1.2. Benchmark applications

As introduced before, we have implemented four benchmarking applications with the COMPSs programming model. We have executed them in the cluster and the runtime has detected the data dependencies and generated for each application a DAG of tasks. The four types of generated DAGs are enumerated below and Fig. 9 depicts an example of how the different type of DAG looks like.

1. EP—Embarrassingly Parallel. Composed by n independent parallel tasks.
2. MT—Matrix multiplication. Composed by n tasks in parallel with a chain of dependencies.
3. GT—Gather. Composed by n tasks that perform a reduction with different predecessors and successors.
4. SG—Scatter–Gather. Composed by n tasks that perform an expansion and a reduction with different predecessors and successors.

For each type of DAG, we have generated three different sizes (small, medium and large) with different number of tasks as shown in Table 3. Moreover, each task has different duration, consumes a set of different resources, and has a different amount of data dependencies. Table 4 shows the range of duration and resource consumption for the different DAG tasks.

8.1.3. Scheduler configuration

To generate the scheduling solutions the importance factor has to be selected, to indicate the importance of the energy versus performance and vice versa, and the heuristic rules to rank the priority of a task. The importance factor (α) can take any value between [0, 1]. For the evaluation experiments, we have selected different importance factors from 0 to 1 with intervals of 0.1. Table 5 shows the relationship between the makespan (C_{max}) and energy flow (E_{flow}), for different importance factors.

Regarding the heuristic rules, four heuristics have been selected to classify the initial order of how the scheduler is going to allocate the tasks.

1. The LPT rule selects those tasks with longer processing time when we have a subset of parallel tasks ready to be executed. This rule tends to balance their workloads, because it is often advantageous to delay tasks with shorter processing, because these tasks will be useful at the end to balance the machines workload.
2. The SPT rule is opposite to LPT rule. It first selects those tasks with shorter process time, in order to prioritize the maximum number of executed tasks.
3. The LNS rule selects those tasks with longer number of successors. This rule is useful when the tasks are subjected to precedence constraints, aimed at ensuring that the end of the current task will fire a maximum number of parallel tasks.
4. The LSTF rule selects the task with longer sequence setup time. This rule allows to prioritize tasks that require a longer setup time for data transfer. This situation occurs when we have a large number of predecessor tasks, where each predecessor task needs to transfer to the successor tasks its input data.

8.1.4. Experiments size

In order to take into account the size of the experimentation, the total combinations are summarized in 11 energy factors, four heuristic rules, four types of DAGs with nine small and medium runs and ten large runs as shown in Table 3, resulting a total of 4928 executions.

8.2. Impact of the importance factor in the energy/makespan trade-off

The first topic we have studied in the experimentation is the trade-off between energy and time established by the variation of the importance factor. An example of this trade-off is depicted in Fig. 10. In that figure, we can see the estimated makespan and energy consumption for the solutions provided by the scheduler for a matrix multiplication (MT) DAG of 800 tasks with different importance factor and applying two different heuristic rules LSTF and LPT. For a factor 1.0 of energy, which corresponds to a factor 0.0 in makespan, we are obtaining the minimum values of energy consumption and the maximum values of the makespan for both heuristics. Subsequently, when the energy importance tends to 0.0, and the makespan factor tends to 1.0, we observe that for both the heuristic rules, the energy consumption is growing to the maximum values and the makespan is decreasing to the minimum values of the makespan.

Table 6 shows the energy consumption and makespan for all combinations of importance factors α and the four heuristic rules for a run of MT DAG. Despite the behavior of the importance factor is similar for all of the heuristics, the values are different and depending on the DAG and resource configuration an heuristic could provide us a better solution. Moreover, we can note that the improvement in the makespan or the energy saving is associated with the heuristic used and the importance factor. So, a final user or a infrastructure provider could set the importance factor to 1 or 0 depending if it only wants to consider performance or

Table 5
Energy/time importance factor—α.

Func	Importance factor—α										
E_{flow}	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
C_{max}	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.0

Table 6
Energy consumption—Makespan per importance factor (α) and heuristic rule.

α	SPT		LPT		LNS		LSTF	
	Wh	C_{max}	Wh	C_{max}	Wh	C_{max}	Wh	C_{max}
1.0	3140.3	**37,834.4**	**3113.6**	37,138.0	3128.5	37,527.4	3125.9	37,458.8
0.9	3140.3	37,834.4	3120.6	31,744.2	3128.5	37,527.4	3125.9	37,458.8
0.8	3140.3	37,834.4	3140.5	26,598.5	3181.7	24,208.1	3177.7	36,675.3
0.7	3195.7	32,390.9	3122.6	19,039.0	3133.5	23,325.6	3165.0	37,440.2
0.6	3153.1	35,640.5	3289.7	17,666.9	3132.8	19,013.9	3163.0	19,215.1
0.5	3183.3	19,607.3	3616.1	13,990.5	3578.9	14,469.8	3338.6	17,842.1
0.4	3375.4	18,399.9	3683.6	13,633.8	3755.8	13,067.8	3764.6	13,319.0
0.3	3282.1	18,753.7	3869.9	11,629.1	3837.1	11,659.3	3954.4	12,198.3
0.2	3827.4	13,701.0	3837.4	11,482.8	3811.0	11,651.2	3917.0	12,124.5
0.1	3985.7	12,349.4	3834.1	11,458.0	3848.8	11,569.1	3895.8	12,101.6
0.0	**4024.8**	12,521.1	3833.6	**11,425.6**	3838.8	11,453.3	3958.7	12,076.5

Table 7
Energy savings—Makespan degradation relationship per importance factor (α) and heuristic rule.

α	SPT		LPT		LNS		LSTF		AVG	
	Wh%	C_{max}	Wh%	C_{max}	Wh%	C_{max}	Wh%	C_{max}	Wh%	C_{max}
1.0	−21.9	+231.1	**−22.6**	**+225.0**	−22.2	+228.4	−22.3	+227.8	**−22.3**	**+228.1**
0.9	−21.9	+231.1	−22.4	177.8	−22.2	+228.4	−22.3	+227.8	**−22.2**	**+216.3**
0.7	−20.6	+183.5	−22.4	+66.6	−22.1	+104.1	−21.3	+227.6	**−21.6**	**+145.4**
0.5	−20.9	+71.6	−10.1	+22.4	−11.0	+26.6	−17.0	+56.1	**−14.8**	**+44.2**
0.3	−18.4	+64.1	−3.8	+1.7	−4.6	+2.0	−1.7	+6.7	**−7.1**	**+18.6**
0.1	−0.9	+8.0	−4.7	+0.2	−4.3	+1.2	−3.2	+5.9	**−3.3**	**+3.8**
0.0	−0.00	+9.5	−4.7	+0.0	−4.6	+0.2	−1.6	+5.7	**−2.7**	**+3.8**

energy (degrading considerably the other term) or select one of the intermediate values, where he can achieve energy saving within an acceptable makespan degradation.

To illustrate it, Table 7 shows the energy savings and the makespan degradation for the different importance factor—heuristic combination. If we decide to save the maximum amount of energy in range $0.7 \leq \alpha \leq 1.0$, the scheduler can provide energy savings between −21% up to −22%, but paying a makespan increases between +145% up to +228%. If we decide by an average factor in range $0.3 \leq \alpha \leq 0.7$, the energy savings average is between −7% up to −21% with a smaller makespan increment (+18%). Finally, if we decide to not save energy in range $0.0 \leq \alpha \leq 0.3$, we could also get a small energy savings average is between −2% up to −7% by the selection of the correct rule, with a negligible makespan increment.

8.3. Impact of the heuristic rule on the energy savings

Table 8 shows the best energy savings obtained for several runs of the different DAG types (EP, MT, GT and SG) and the heuristic which has provided the best result. For instance the scheduling for a run of the EP DAG with 500 tasks achieves a reduction of −20.61%. The values of maximum and minimum energy consumption are 2357.54 and 1871.78 Wh and the LPT rule provides a better value than SPT, LNS, and LSTF. The energy saving is calculated as the difference between the solutions when applying the importance factor 0.0 and 1.0.

In that table, we can also observe that heuristic rules are associated to instance type to solve, thus, for instances type EP and MT, we observed that the heuristic rule which gives better results is LPT. For instances type GT is LNS rule, and for instances type DG is a combination between SPT rule and LNS rule. With this observation, we could improve the performance of our algorithm by reducing the search space by just evaluating the adequate heuristic rule according to the DAG characteristics.

8.4. Importance factor and resource usage

Another observation we want to highlight is the relationship between the importance factor and the use of resources which is shown in Fig. 11. The figure shows the number of VMs and which type of node is used for different important factors. When we have configured the scheduler without energy importance ($\alpha = 0.0$), it tends to use the maximum number of resources. In the case of the figure, the scheduler uses all the nodes estimating an energy consumption of $E_{flow} = 3853.56$ Wh (equivalent to 13,800,802 J) and a makespan of $C_{max} = 11,425$ s. When we increase the energy importance, the scheduler tends to prioritize the efficient nodes. In the case of $\alpha = 0.7$, the scheduler only uses the Intel nodes, estimating an energy consumption of $E_{flow} = 3122.61$ Wh (equivalent to 11,241,380 J) and a makespan of $C_{max} = 19,039$ s. Finally, when the energy factor is the only priority ($\alpha = 1.0$), the trend is to use the minimum number of nodes with the lowest power consumption. In the case of the figure, the scheduler only uses one of the Intel nodes, estimating an energy consumption of $E_{flow} = 3113,55$ Wh in $C_{max} = 37,137$ s which is equivalent to 11,208,794 J.

This is due to the energy cost of keeping the nodes running with low load. The idle power consumption of the node is very important (it can consume up to half of the total energy), and having an unused node produces an important waste of energy. Fig. 12 shows the power consumption of the AMD and Intel nodes in an idle state and the power consumption when we use the different cores of the node. As you can see, in the figure the Intel

Table 8
Energy savings results per DAG type and heuristic rule.

T	DAG type EP				DAG type MT				DAG type GT				DAG type SG			
	Max Wh	Min Wh	Saving	Rule	Min Wh	Max Wh	Saving	Rule	Min Wh	Max Wh	Saving	Rule	Min Wh	Max Wh	Saving	Rule
100	566.98	386.71	−31.80%	LPT	386.71	1054.20	−51.54%	LNS	1054.20	510.85	−40.02%	LNS	510.85	829.77	−46.81%	SPT
200	1022.77	752.57	−26.42%	LPT	752.57	1861.35	−55.27%	SPT	1861.35	832.61	−34.12%	SPT	832.61	1369.37	−40.44%	LNS
300	1446.04	1120.09	−22.54%	LPT	1120.09	2056.81	−40.88%	SPT	2056.81	1215.91	−30.80%	LNS	1215.91	1837.24	−34.19%	SPT
400	1966.65	1511.62	−23.14%	LPT	1511.62	2432.36	−35.49%	SPT	2432.36	1569.23	−26.21%	SPT	1569.23	2336.27	−30.97%	SPT
500	2357.54	1871.68	−20.61%	LPT	1871.68	2886.17	−32.09%	LNS	2886.17	1959.92	−25.55%	LNS	1959.92	2827.19	−29.89%	LNS
600	2821.23	2242.54	−20.51%	LPT	2242.54	3164.18	−25.65%	LNS	3164.18	2352.46	−24.69%	LNS	2352.46	3334.45	−29.35%	SPT
700	3284.60	2614.56	−20.40%	LPT	2614.56	3632.46	−24.76%	LPT	3632.46	2732.92	−23.90%	LNS	2732.92	3821.87	−26.30%	LSTF
800	3785.64	3014.62	−20.37%	LPT	3014.62	4024.85	−22.64%	LPT	4024.85	3113.55	−22.73%	LNS	3113.55	4261.20	−26.21%	SPT
900	4183.82	3383.89	−19.12%	LPT	3383.89	4502.43	−21.98%	LPT	4502.43	3512.95	−21.47%	LNS	3512.95	4750.86	−26.02%	SPT
1000	4708.49	3791.76	−19.47%	LPT	3791.76	4955.27	−21.36%	LPT	4955.27	3897.05	−21.33%	LSTF	3897.05	5284.20	−24.88%	LNS

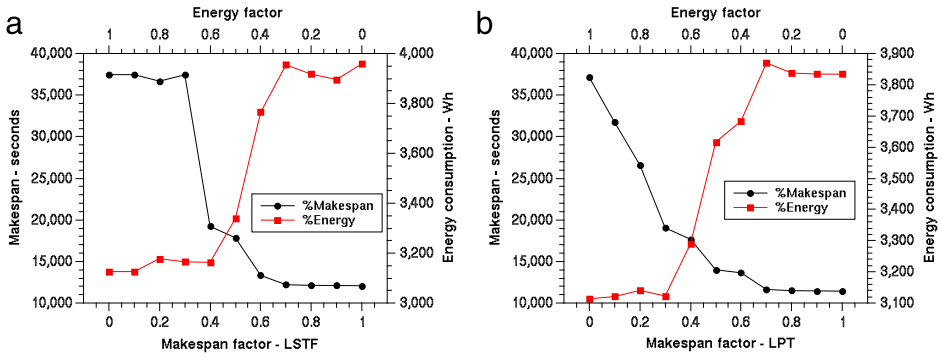


Fig. 10. Trade-off between energy and makespan.

nodes has less power usage, so this is the reason why the scheduler has prioritized the Intel nodes.

A similar behavior is observed with the VM elasticity. In this case, each VM has a cost in time (setup and down time) and its corresponding energy consumption. The MHRA algorithm will decide to use an additional VM depending on how important is the VM management cost (in time and energy) according to the objective function. To illustrate the VM, we use a SG DAG because in the scatter phase the number of parallel tasks increases, in the gather phase the number of parallel tasks is reduced.

Fig. 13 shows when the scheduler decides to create and destroy a VM for the SG DAG of 800 tasks for different importance factors. In the first case ($\alpha = 0.0$) the energy consumption is not considered and the scheduler algorithm is looking for resource allocation to execute tasks in any VM of any node. Therefore, while the number of tasks grows, the MHRA algorithm of the scheduler starts to select new nodes and create new VMs in increasing order, using all available resources on the cloud in order to minimize the makespan. In the second case, when looking for a balance between time and energy ($\alpha = 0.7$), the scheduler is less aggressive in the VM creation deciding to create few VMs and use it more time. This is because using less VMs, reduce the energy consumed by the hypervisor in the VM management. Moreover, using less nodes during more lime reduces the idle time, and using less resources increase the data locality and reduces the number of transfers. Finally, in the latest case ($\alpha = 1.0$). it only creates 3 VMs in the most energy-efficient node and uses them during all the time.

8.5. Performance and overhead

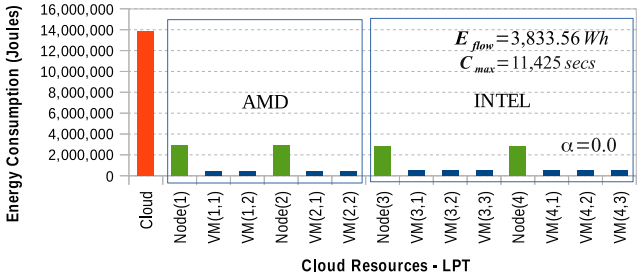
To finalize the experimentation, we have analyzed the performance of the proposed scheduling system by measuring the time spent by the scheduler to get the scheduling solution (Scheduling Time) for a selected importance factor depending on the number of tasks and resources. Fig. 14 show the results of these time measurements. The first part of the figure shows how the time to get the scheduling solution is growing when we increase the number of tasks. In the second part of the figure, we have kept the number of tasks fixed in a relatively big amount of tasks (8000 tasks) and we have measured the scheduling time increasing the number resources to consider in the scheduling problem (10–160 VMs).

To illustrate how important is this scheduling overhead, we have compare this scheduling time with application execution time (makespan). For instance, with the largest measured scheduling time (2717 ms.) for 8000 Tasks and 160 VMs, we can compare to the execution time of an EP DAG of this number of tasks with 10 s duration tasks which is around 130 s. It gives an overhead of 2%. Note that we have consider the time to get the best scheduling solution for the overhead evaluation. It includes the evaluate the whole graph for the different heuristics. However, the scheduling solution calculation could run in background in the runtime thread during the tasks are running. When all resource are full, the runtime remains at idle state until the end of the running tasks and can use this time to improve the solution by using backtracking techniques.

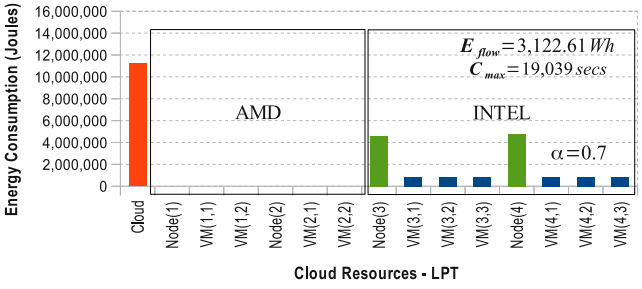
9. Conclusion and future work

In this paper, we have presented an energy-aware scheduling system for task-based applications. The scheduler aims at minimizing a normalized bi-objective function which combines the energy consumption and the makespan (total execution time). Those metrics are combined by an importance factor which enables users and service providers to indicate which is more important for their purposes: save energy or performance.

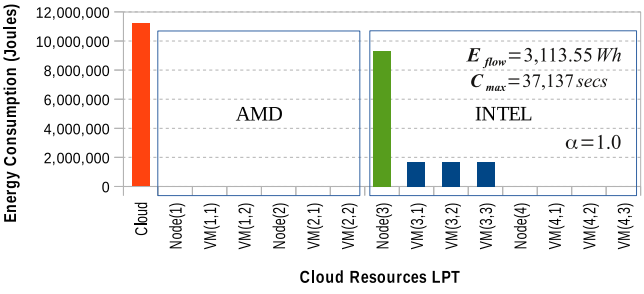
We have proposed a model for estimating the energy consumed by the execution of a given application in a set of resources. This model estimates the application energy consumption by



(a) Importance factor $\alpha = 0.0$.



(b) Importance factor $\alpha = 0.7$.

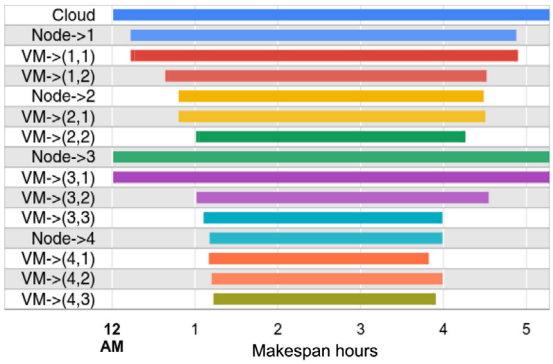


(c) Importance factor $\alpha = 1.0$.

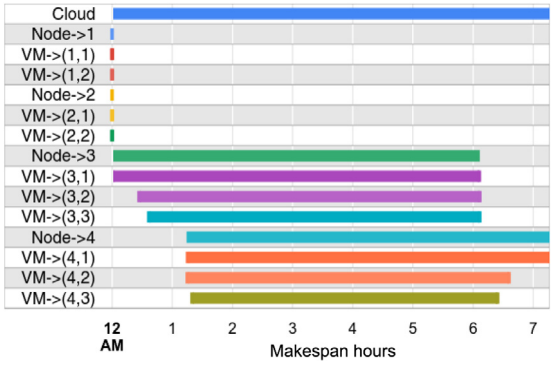
Fig. 11. Resource usage distribution for MT DAG (800 tasks).

aggregating the consumption of the different elements of the application tasks execution, data transfers, VM management and the node background services. In addition to this model, we have proposed a methodology to automatically extract the resource power profile required to calculate the energy estimations.

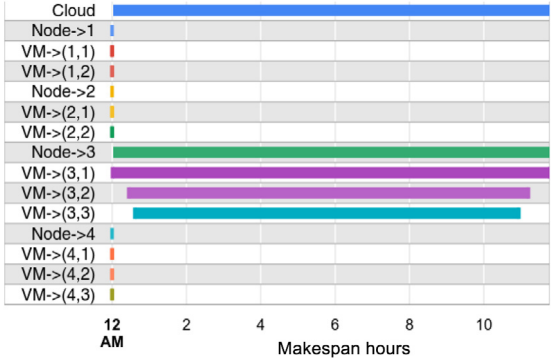
The scheduler has been designed to be part of the COMP Superscalar (COMPSs) runtime scheduler. Due to this constraint, we have proposed a Multi-heuristic Resource Allocation (MHRA) algorithm to get the best scheduling solution in polynomial time. Applications in COMPSs are represented as Directed-Acyclic-Graph (DAG) of tasks dependencies which will be the input of the MHRA algorithm. For the different tasks graph are ranked by a set of heuristic



(a) Importance factor $\alpha = 0.0$.



(b) Importance factor $\alpha = 0.7$.



(c) Importance factor $\alpha = 1.0$.

Fig. 13. VM elasticity for the SG DAG (800 tasks) with different importance factors.

rules (such as SPT, LPT, LNS and LSTF) which decides the order in which the resource allocation algorithm is going to schedule them by selecting the resource which minimize the cost function.

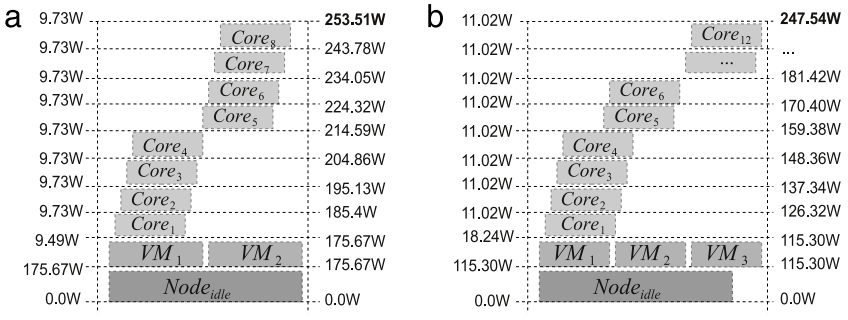
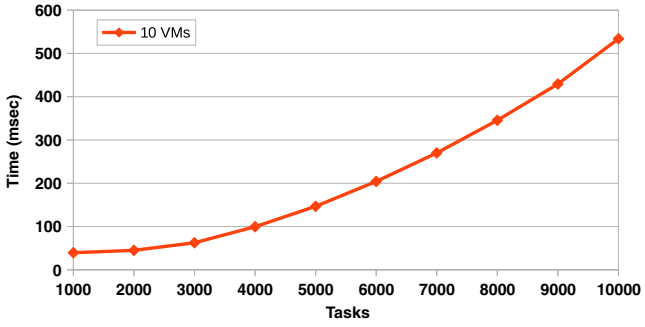
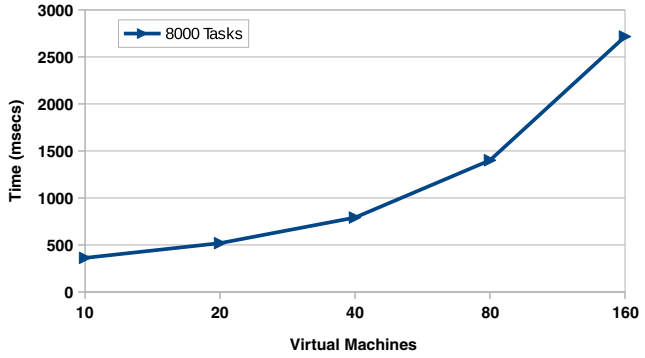


Fig. 12. Energy spend distribution AMD and Intel nodes.



(a) Scheduling time vs number of tasks (fixed VMs).



(b) Scheduling time vs number of resources (fixed tasks).

Fig. 14. Scheduling performance depending on the number of tasks and resources considered in the scheduling problem.

We have implemented a prototype of this scheduler and we have evaluated with different types of DAGs such as EP, MT, ST and GT. We have seen how the scheduler behaves depending on the selected importance factor and its relationship with the makespan and energy consumption. The outcomes of the algorithm show that a considerable energy amount can be saved, depending on the size of the instance and type of DAG. For instance in the case of an energy importance configuration, it allows to save an average of −22.44%, −33.17%, −27.08% and −31.50% for EP, MT and WP, SG respectively, with respect to the makespan importance configuration.

With regards the resource usage, when a makespan importance configuration is set, the scheduler decides to use the maximum number of VMs that the infrastructure can provide, having an aggressive VM creation response. In contrast, when a high energy importance factor is set, the algorithm tries to use the least number of nodes as possible, and a maximum number of VMs per node in order to save energy which leads to increasing execution times.

Future work associated with the integration of the energetic model proposed with COMPSs, also the addition of formulation to calculate the monetary cost associated with the use of VMs and delivery times, with the aim of increasing the rates of return and profits for service providers.

Acknowledgments

This work has been supported by the Spanish Government (contracts TIN2015-65316-P, TIN2012-34557, CSD2007-00050, CAC2007-00052 and SEV-2011-00067), by Generalitat de Catalunya (contract 2014-SGR-1051), by the European Commission (Euroserver project, contract 610456) and by Consejo Nacional de Ciencia y Tecnología of Mexico (special program for postdoctoral position BSC-CNS-CONACYT contract 290790, grant number 265937).

References

[1] J. Koomey, Growth in data center electricity use 2005 to 2010, A report by Analytical Press, completed at the request of The New York Times, 2011, p. 9.
[2] C. Pettey, Gartner estimates ict industry accounts for 2% of global co2 emissions, 14, 2007, p. 2013. Dostupno na: https://www.gartner.com/newsroom/id/503867.
[3] R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges, 2010. arXiv preprint arXiv:1006.0308.
[4] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, R. Rajkumar, Critical power slope: understanding the runtime effects of frequency scaling, in: Proceedings of the 16th International Conference on Supercomputing, ACM, 2002, pp. 35–44.
[5] V. Tiwari, S. Malik, A. Wolfe, Compilation techniques for low energy: An overview, in: Low Power Electronics, Digest of Technical Papers., IEEE Symposium, IEEE, 1994, pp. 38–39.
[6] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (2012) 755–768.
[7] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, R. Sakellariou, Energy-constrained provisioning for scientific workflow ensembles, in: Third International Conference on Cloud and Green Computing (CGC), IEEE, 2013, pp. 34–41.
[8] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of np-Completeness, Freeman, San Francisco, LA, 1979.
[9] E. Tejedor, R.M. Badia, Comp superscalar: Bringing grid superscalar and gcm together, in: 8th IEEE International Symposium on Cluster Computing and the Grid, IEEE, 2008, pp. 185–193.
[10] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Alvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, R.M. Badia, Servicess: An interoperable programming framework for the cloud, J. Grid Comput. 12 (2014) 67–91.
[11] W. Chen, R.F. da Silva, E. Deelman, R. Sakellariou, Using imbalance metrics to optimize task clustering in scientific workflow executions, Future Gener. Comput. Syst. 46 (2015) 69–84.
[12] F. Zhang, J. Cao, K. Li, S.U. Khan, K. Hwang, Multi-objective scheduling of many tasks in cloud platforms, Future Gener. Comput. Syst. 37 (2014) 309–320.
[13] L. Wang, G. Von Laszewski, J. Dayal, F. Wang, Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs, in: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2010, pp. 368–377.
[14] C.-M. Wu, R.-S. Chang, H.-Y. Chan, A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters, Future Gener. Comput. Syst. 37 (2014) 141–147.
[15] A. Beloglazov, R. Buyya, Energy efficient allocation of virtual machines in cloud data centers, in: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2010, pp. 577–578.
[16] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (2009) 599–616.
[17] D. Lucanin, I. Pietri, I. Brandic, R. Sakellariou, A cloud controller for performance-based pricing, in: IEEE 8th International Conference on Cloud Computing (CLOUD), IEEE, 2015, pp. 155–162.
[18] R. Weron, Modeling and Forecasting Electricity Loads and Prices: A Statistical Approach, Vol. 403, John Wiley & Sons, 2007.
[19] H. Xu, C. Feng, B. Li, Temperature aware workload management in geo-distributed datacenters, in: ACM SIGMETRICS Performance Evaluation Review, Vol. 41(1), ACM, 2013, pp. 373–374.
[20] K. Bessai, S. Youcef, A. Oulamara, C. Godart, S. Nurcan, Bi-criteria workflow tasks allocation and scheduling in cloud computing environments, in: IEEE 5th International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 638–645.
[21] M. Voorneveld, Characterization of pareto dominance, Oper. Res. Lett. 31 (2003) 7–11.
[22] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient task scheduling for executing large programs in the cloud, Parallel Comput. 39 (2013) 177–188.
[23] K. Chronaki, A. Rico, R.M. Badia, E. Ayguadé, J. Labarta, M. Valero, Criticality-aware dynamic task scheduling for heterogeneous architectures, in: Proceedings of the 29th ACM on International Conference on Supercomputing, ACM, New York, NY, USA, 2015, pp. 329–338.
[24] A. Duran, E. Ayguadé, R.M. Badia, J. Labarta, L. Martinell, X. Martorell, J. Planas, Ompss: a proposal for programming heterogeneous multi-core architectures, Parallel Process. Lett. 21 (2011) 173–193.
[25] E. Ayguadé, R.M. Badia, P. Bellens, D. Cabrera, A. Duran, R. Ferrer, M. González, F. Igual, D. Jiménez-González, J. Labarta, et al., Extending openmp to survive the heterogeneous multi-core era, Int. J. Parallel Program. 38 (2010) 440–459.
[26] D. Lezzi, F. Lordan, R. Rafanell, R.M. Badia, Execution of scientific workflows on federated multi-cloud infrastructures, in: Euro-Par 2013: Parallel Processing Workshops, Springer, 2013, pp. 136–145.
[27] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, Springer Science & Business Media, 2012.
[28] I.Y. Kim, O. De Weck, Adaptive weighted-sum method for bi-objective optimization: Pareto front generation, Struct. Multidiscip. Optim. 29 (2005) 149–158.

Fredy Juarez has an engineering degree on Computing Systems at the Technical Institute of Ciudad Maduro (ITCM) a M.Sc. in Distributed Systems at Centro Nacional de Investigación *y* Desarrollo Tecnológico (CENIDET) and a Ph.D. in combinatorial optimization for resource allocation by the Universidad Autónoma del Estado de Morelos (UAEM). He worked in the Oil industry and Instituto de Investigaciones eléctricas deploying HPC clusters and Grids and he has been an part-time professor at Universidad Autónoma del Estado de Morelos and Universidad Politécnica Metropolitana de Hidalgo. Currently, he has a postdoc position at Barcelona Supercomputing Center in the Workflows and Distributed Computing group.

Jorge Ejarque has an engineering degree on Telecommunications (2005) and a M.Sc. on Computer Architecture Network and System (2009) at the UPC. In 2005, he worked as IT consultant in Better Consulting, and at the end of 2005 he joined the Grid Computing group at BSC. During his career at the BSC, he has contributed in the design and development of different tools and programming models for HPC in distributed platforms. He has been involved in several National and International R&D projects (CoreGRID, BREIN, NUBA and OPTIMIS) and he was a member of the experts' board of the Spanish National Grid Initiative and he has been a reviewer of several journals and part of the program committee in different conferences. His current research interests are focused on three areas: semantic interoperability of distributed computing platforms, which is his current Ph.D. topic; parallel programming models for easydevelopment in distributed platforms; and energy-efficient execution of distributed applications. Regarding this topic, it is currently working on two EU funded project ASCETIC and Euroserver.

Rosa M. Badia holds a Ph.D. on Computer Science (1994) from the Technical University of Catalonia (UPC). She is a Scientific Researcher from the Consejo Superior de Investigaciones Científicas (CSIC) and team leader of the Workflows and Distributed Computing research group at the Barcelona Supercomputing Center (BSC). She was involved in teaching and research activities at the UPC from 1989 to 2008, where she was an Associated Professor since year 1997. From 1999 to 2005 she was involved in research and development activities at the European Center of Parallelism of Barcelona (CEPBA). Her current research interest is programming models for complex platforms (from multicore, GPUs to Grid/Cloud). The group lead by Dr. Badia has been developing StarSs programming model for more than 10 years, with a high success in adoption by application developers. Currently the group focuses its efforts in two instances of StarSs: OmpSs for heterogeneous platforms and COMPSs/PyCOMPSs for distributed computing including Cloud. For this last case, the group has been doing efforts on interoperability through standards, for example using OCCI to enable COMPSs to interact with several Cloud providers at a time. Dr. Badia has published more than 150 papers in international conferences and journals in the topics of her research. She has participated in several European projects, for example BEinGRID, Brein, CoreGRID, OGF-Europe, SIENA, TEXT and VENUS-C, and currently she is participating in the project Severo Ochoa (at Spanish level), ASCETIC, Euroserver, The Human Brain Project, EU-Brazil CloudConnect, the BioExcel CoE, NEXTGenIO, and trasnPLANT and is a member of HiPEAC2 NoE.