Computers and Electrical Engineering 97 (2022) 107630

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Energy-aware task scheduling in data centers using an application signature☆

Juan Carlos Salinas-Hilburg [a,*], Marina Zapater [b], José M. Moya [d,c], José L. Ayala [a,c]

[a] DACYA, Complutense University of Madrid, Madrid, Spain
[b] School of Management and Engineering Vaud (HEIG-VD), University of Applied Sciences Western Switzerland (HES-SO), Switzerland
[c] Center for Computational Simulation, Technical University of Madrid, Madrid, Spain
[d] Integrated Systems Laboratory, Technical University of Madrid, Madrid, Spain

ARTICLE INFO

Keywords:
Energy efficiency
Data centers
Application signature
Task scheduling

ABSTRACT

Data centers are power hungry facilities. Energy-aware task scheduling approaches are of utmost importance to improve energy savings in data centers, although they need to know beforehand the energy consumption of the applications that will run in the servers. This is usually done through a full profiling of the applications, which is not feasible in long-running application scenarios due to the long execution times. In the present work we use an application signature that allows to estimate the energy without the need to execute the application completely. We use different scheduling approaches together with the information of the application signature to improve the makespan of the scheduling process and therefore improve the energy savings in data centers. We evaluate the accuracy of using the application signature by means of comparing against an oracle method obtaining an error below 1.5%, and Compression Ratios around 39.7 to 45.8.

1. Introduction

Data centers are facilities composed by a set of computer systems working in a non-stopping way, hence they consume too much power. In the year 2014, U.S data centers consumed about 70 billion kWh which represented the 1.8% of the total U.S electricity consumption, and it is estimated that in 2020 they will consume approximately 73 billion kWh [1]. Moreover, the computing resources of the data centers are going to double within the next 3 to 4 years, hence, the need to improve energy efficiency in data centers will be required to manage energy growth [2]. The major contributor to the data center power is the computing (or IT) power [3]. Energy-efficient techniques, such as *resource management*, are used to reduce the IT power consumption by applying an energy aware task scheduling approach to allocate efficiently the tasks in the servers [4]. This technique is usually used proactively and assume the existence of a full dynamic power profiling of the applications obtained through a complete execution of the application. In scenarios of long-running applications (applications that run for hours) performing a full dynamic power profiling is not viable, since doing a full profile of a large batch of long-running applications is not energy-efficient. These long-running applications can be found in many scientific applications (e.g. climate modeling) [5] and have the characteristic to be iterative, data-intensive, and often they are formed by computational intensive kernels (e.g. matrix multiplications). Additionally, these types of applications can be executed with multiple instances of the same application in a single-threaded or sequential way, and also,

☆ This paper is for regular issues of CAEE. Reviews were processed by the Editor-in-Chief Dr. Manu Malek.

* Corresponding author.
E-mail addresses: jcsalinas@ucm.es (J.C. Salinas-Hilburg), marina.zapater@heig-vd.ch (M. Zapater), josem@die.upm.es (J.M. Moya), jayala@ucm.es (J.L. Ayala).

https://doi.org/10.1016/j.compeleceng.2021.107630
Received 5 June 2020; Received in revised form 5 September 2021; Accepted 22 November 2021
Available online 8 December 2021

can be executed in a multi-threaded way using multiple cores as occur in the High Performance Computing (HPC) scenario. Finally, long-running applications are executed in large-scale data centers, for example, the *Barcelona Supercomputing Center*[1] or in small-scale data centers such as *Madrid Supercomputing and Visualization Center*.[2]

In our previous work [6], we developed a fast energy framework for long-running applications that uses an application signature. The *application signature* is defined as a reduced version, in terms of execution time, of the original application and it is used to estimate the energy of long-running applications without the need for a full dynamic profiling of the applications. The goal of this work is to use the information provided by the application signature to apply different energy-efficient task scheduling approaches in order to reduce the **makespan** of the original batch, and therefore improve the energy efficiency in data centers. The **makespan** is defined as the total execution time of the batch of applications that will run in the data center. It should be noted that the main goal of this work is to show that the information of the application signature can be used to apply in a proactive way energy-aware scheduling approaches. Without the application signature those energy-aware scheduling approaches would require a full dynamic profiling of the applications preventing them to be applied in an efficient way. Hence, it is not the main goal of this work to propose new or better energy-aware scheduling approaches, but to expand and improve these thanks to the use of the application signature. Finally, our contributions are:

- We validate the usefulness of the application signature by applying energy-efficient task scheduling approaches using the energy information provided by the application signature (execution time and mean power). We use three different task scheduling approaches: (i) an optimal approach using a Mixed Integer Linear Programming (MILP) technique, (ii) an energy-aware heuristic, and finally, (iii) we propose an implementation of a metaheuristic using a Simulated Annealing process. The resulting overall data center energy consumption from each task scheduling approach is compared against a Round-Robin (RR) approach.
- We evaluate the energy savings obtained through the task scheduling approaches in a large-scale and small-scale scenario. We compare the energy savings from the application signature and the energy savings from the real energy values of the applications, obtaining a difference below 1.5%.
- We define the **Compression Ratio** (CR) as the ratio of the makespan of the batch using the Round-Robin task scheduling approach against the total execution time of the application signatures of the applications within the batch. We obtain CR values around 39.7 to 45.8. This means that the application signature is able to estimate the energy of the whole batch 45.8 times faster than performing a full dynamic profiling of the whole batch.

The remainder of the paper is organized as follows. Section 2 presents the related work. Section 3 shows the use of the information provided by the application signature to apply task scheduling techniques in data centers. A detailed explanation of each of the task scheduling approaches are shown Section 4. The experimental setup and results are presented in Section 5 and Section 6, whereas Section 7 concludes the paper.

2. Related work

In order to apply efficient energy-aware task scheduling approaches the value of either power or performance must be known beforehand. Previous works proposed different techniques to predict either power or performance for long-running and scientific applications using collected data through the complete execution of the applications. The work by Sirbu et al. [7] uses a data-driven model to predict the power consumption of through a dedicated monitoring framework. Sadjadi et al. [8] presents a performance model for long-running scientific applications. The performance model is built by performing a full profile of the execution of the application without using intrusive techniques such as instrumentation or code inspection. In our present work we use the estimated energy of the applications using the application signature obtained through the fast energy estimation framework developed in our previous works [6]. This allows to implement different energy-aware task scheduling approaches without the need to fully execute the applications.

There are works that present a methodology to predict the performance by using an application signature or a partial execution of the applications. In the work presented by Wong et al. [9] the application signature is used to predict the performance of parallel applications. The application signature is extracted by executing the whole application on a platform *A*. Then, the application signature is used to predict the performance on a different platform *B*. Yang et al. [10] uses a partial execution of the original application to predict the performance. They execute the application completely in order to the predict the performance in a different platform with the same approach as the previously commented work (Wong et al.). These works relay on the complete execution of the original application in order to either build the application signature or to predict the performance. As far as we know, the application signatures used on those works do not estimate energy values. In our work, we use the value of the estimated energy using the application signature to apply energy-aware task scheduling approaches.

There is an extended research on using energy efficient task scheduling approaches for energy savings in data center. The work by Auweter et al. [11] presents an energy-aware task scheduler to improve energy savings of supercomputers. They introduce a prediction model that forecast performance and power of large-scale applications. In the work by Mämemelä et al. [12] is shown an energy-aware scheduler that can be applied to HPC data centers. They used energy-aware variations of the FIFO (First In First Out)

[1] https://www.bsc.es/.

[2] https://www.cesvima.upm.es/.

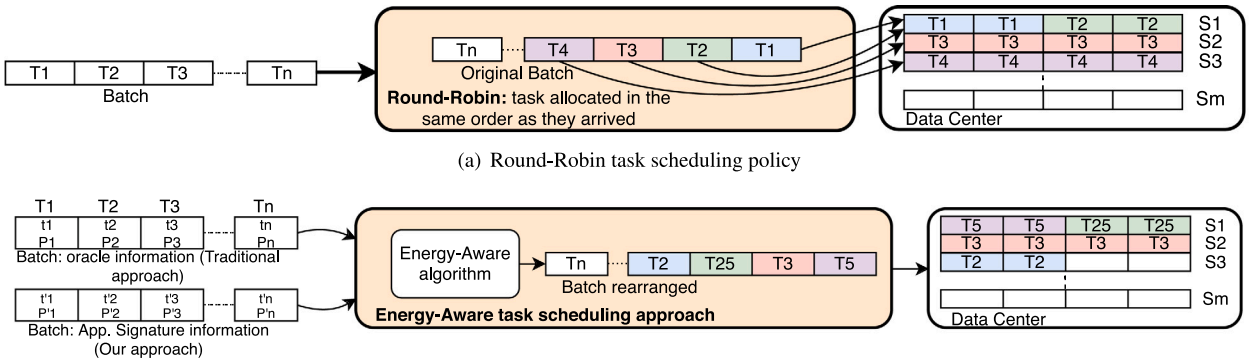


Fig. 1. Comparison between task scheduling approaches: Round-Robin vs. Energy-Aware.

and Backfilling schedulers and also, presents a very detailed power consumption model. In all of these works there is an assumption of the existence of either energy, power or performance of the tasks that will be executed in the data center. Whereas, in our work we use the information of the application signature to estimate the energy and apply a scheduling approach.

The task scheduling approaches can be implemented in the form of Integer Linear Programming, or by using metaheuristics or heuristics methodologies. In the case of Integer Linear Programming based approach there is a great amount of research. Goldman et al. [13] presents a Mixed Integer Linear Programming task scheduling approach for parallel independent tasks. They present the MILP formulations for either fragmented or non-fragmented systems. A fragmented system is one where each thread of the task does not need to be using a continuous set of resources, i.e the threads of the parallel task does not need to be running on the same processor. The work developed by Chretien et al. [14] shows a task scheduling approach using successive Linear Programming approximations. They used an iterative Linear Programming scheme to find the optimal makespan. Metaheuristics approaches can find near optimal solutions in much less computation time than Integer Linear Programming scheduling approaches. Lei et al. [15] proposed a scheduling approach based on a co-evolutionary algorithm for green data centers. In the work by Kashani et al. [16] shows a task scheduling method based on Simulated Annealing to minimize the makespan in distributed systems. Finally, heuristic methods allows to find good solutions with much less computation time than Integer Linear Programming and metaheuristic approaches. The work by A. Beloglazov et al. [17] propose an energy-aware heuristic to allocate efficiently virtual machines in a cloud oriented scenario. The work presented by Garefalakis et al. [18] shows a cluster scheduler for long-running applications. They implement both an Integer Linear Programming and a heuristic based scheduling approach. In our present work we use and implement three different scheduling approaches based on Mixed Integer Linear Programming, Simulated Annealing and a heuristic approach based on the Longest Task First method. The main goal of this work is to validate the use of the application signature with different scheduling approaches.

3. Task scheduling with the application signature

Task scheduling techniques to improve energy efficiency are widely used in today's data centers. As we previously commented in the Related Work section, there are energy-efficient proactive task scheduling approaches that require some information of the tasks that will be executed in the data center. This information can be the execution time or even the power that the tasks will consume during the execution. Traditionally, the previous information can be obtained through a full profiling of the tasks. Nevertheless, this process is not feasible in long-running tasks scenarios where the process to gather the tasks information is not efficient. A Round-Robin policy is applied when is not possible to obtain any information of the tasks. Fig. 1(a) shows the task allocation of a batch of T_n tasks using the Round-Robin policy, where each task is allocated to an available server in the same order as they arrived. The example shows a data center with m servers and each server has 4 cores. The first T_1 task of the batch requires 2 cores and is allocated to server S_1, similarly task T_2 requires 2 cores and since server S_1 has enough resources available the task T_2 is allocated there. When a task cannot be allocated in a server it goes to a queue and waits until there is enough resources available to be executed. The Round-Robin policy is not efficient in terms of energy since the tasks are not allocated aiming to reduce the makespan or the power consumption of the servers. Nonetheless, this policy does not need previous information of the applications, hence it can always be used in any scheduler. Furthermore, a Round-Robin policy is present in schedulers such as SLURM. This scheduler is widely used in several data centers around the world. In this work we use the Round-Robin policy as a baseline to evaluate the energy savings of energy-aware task scheduling approaches.

As we have previously mentioned, efficient proactive task scheduling approaches can be used to reduce the energy consumption in data centers, for example by reducing the makespan of the original batch. Fig. 1(b) shows that an energy-aware task scheduling approach can be used to rearranged the original batch and therefore minimize the makespan. In the example we can see that the tasks are allocated in a different manner when compared to the Round-Robin policy. This change in the order of execution of the

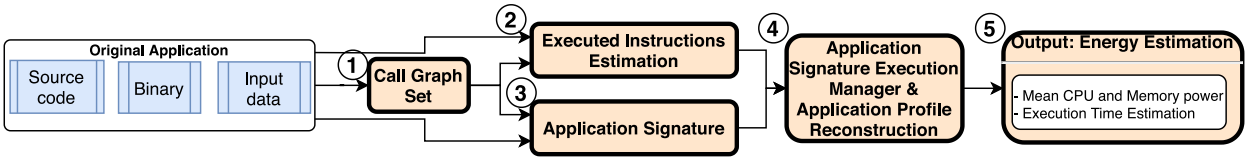


Fig. 2. Overview of the fast energy estimation framework using the application signature.

original batch is the result of applying an energy-aware algorithm to minimize the makespan of the original batch. The energy-aware algorithm needs information (execution time t_i of the mean power consumption P_i) from the tasks that will be executed. Traditionally, this information is obtained through a full dynamic profiling of each task. In the present work we call this knowledge as the **oracle** information of the tasks. We propose the use of an application signature to leverage the oracle information. By executing the application signature we can obtain the information of execution time t_i' and mean power consumption P_i' without the need to perform a full profiling of each task of the batch. It should be noted that the oracle information is our gold-standard and it allows to evaluate the accuracy of the energy savings when using the information of the application signature.

Finally, as we have previously commented the batch is formed by data-intensive, iterative long-running applications. This type of application has the characteristic to present long execution times that can last for hours. The scheduling inside the batch is static, this means that when a batch arrives we apply an energy-aware task scheduling approach to the whole batch and then each task of the batch is sent to the servers. Each task of the batch is executed in a non-preemptive way.

3.1. Application signature

In this section we summarize the application signature concept. A detailed explanation of the implementation of the application signature can be found in our previous work [6]. Fig. 2 shows an overview of the fast energy estimation framework using the application signature. The goal of the fast energy estimation framework is to take information from the original application (source code, binary and the input dataset of the application) and estimate the mean CPU and memory power, and the execution time without the need to execute the original application.

1. The **Call Graph Set** takes as ***input*** the source code of the original application and the ***output*** is the Call Graph Set (CGS) which is a set of Call Graphs for each independent execution path of the original application. An ***independent execution path*** is defined as a path from the Call Graph obtained through the following process: (i) start the path search at the main function, (ii) if an edge (a function call) can be followed, do so; (iii) if not, stop the path search.
2. The **Executed Instructions Estimation module** estimates the number of executed instructions for each independent execution path via a static profiling approach (without executing the whole application). The module takes as ***inputs*** the source code, binary and the input dataset of the original application, and the Call Graph Set. The ***output*** of this module is the estimated executed instructions of each independent execution path. The instruction estimation is done with: (i) upper bounds of each loop of the original application (from the source code), and (ii) the CPU instructions from each independent execution path (from the binary). This module takes into account the input dataset (e.g. the size of a matrix, or the number of timesteps) of the original application since this information affects the number of the estimated executed instructions.
3. The **Application Signature module** creates the application signature. The ***inputs*** are the source code of the original application and the Call Graph Set. The ***output*** is the application signature and is composed by the binaries of each independent execution path.
4. The **Application Signature Execution Manager and Application Profile Reconstruction module** takes as ***inputs*** the estimated executed instructions of each independent execution path, the application signature and the input dataset of the original application. The binaries from the application signature are executed (taking as input to each binary the input dataset of the original application) and a set of hardware counters profiles are collected for each executed binary. The reconstructed application profile (***output***) is composed by the reconstructed hardware counters profiles of each independent execution path. The reconstructed application profile is equivalent, in terms of energy, to the original application profile obtained through a full profiling of the original application.
5. The **Energy Estimation module** is the ***output*** of the whole framework. The mean CPU and memory power, and the execution time are estimated using the reconstructed application profile as ***input***.

3.2. Using the application signature for energy-aware task scheduling

Algorithm 1 shows the process of using the application signature information for energy-aware task scheduling in data centers. The process starts when a batch formed by n tasks arrives to the data center (2). Each task of the batch is sent to an available server of the data center using a Round-Robin policy to extract and execute the application signature and therefore, estimate the execution time and the mean power (4). The information (execution time and mean power) obtained from the application signature of each

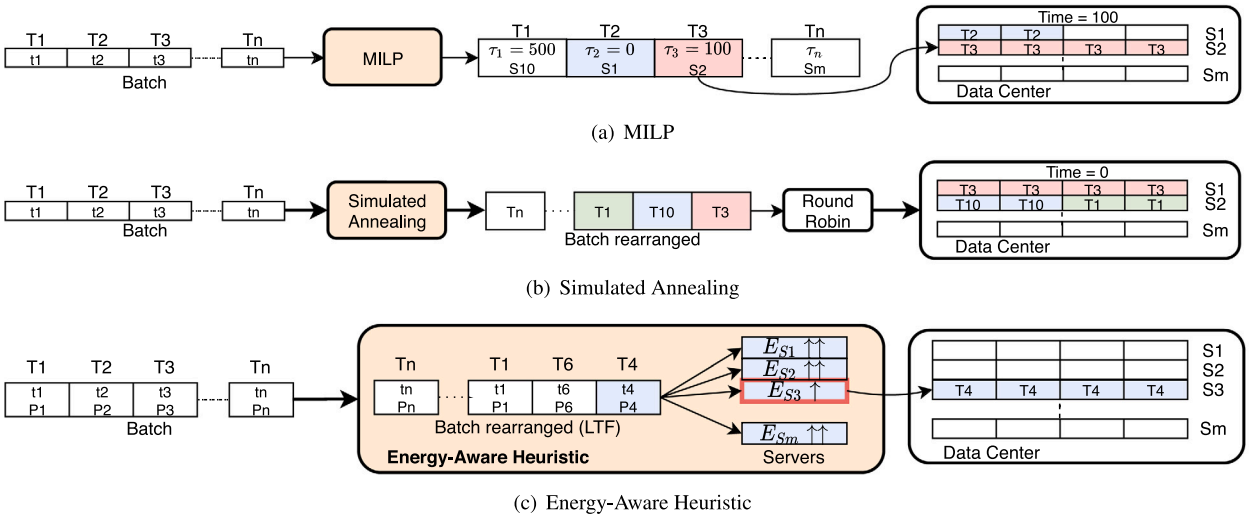


Fig. 3. Energy-Aware task scheduling approaches: input, output and task allocation.

task is saved in a list (5). Furthermore, an energy-aware task scheduling approach can use the applications signature information saved in the list to rearrange the batch (changing the order of execution of each task) aiming to reduce the makespan (7). Finally, the rearranged batch is executed resulting in a lower makespan than the execution of the original batch arrange (8). In Section 4 we will show how an energy-aware approach can use the tasks information to reduce the energy consumption in data centers.

Algorithm 1 Application Signature for Energy-Aware Task Scheduling

Require: batch of n tasks, set of batches, m servers
1: **while** set of batches not empty **do**
2: batch= New batch arrives
3: **while** batch not empty **do**
4: [Exec. Time_n, Mean Power_n]=app_signature(task_n,server_m)
5: App. Signature Info List = append values of Exec. Time_n and Mean Power_n
6: **end while**
7: rearrange_batch=energy_aware_task_scheduling_rearrange(App. Signature Info)
8: allocation→ energy_aware_task_scheduling_execute(rearrange_batch,m servers)
9: **end while**

3.3. Compression ratio

The Compression Ratio measures the acceleration of the execution time and mean power information extraction of the whole batch using the application signature. The CR is calculated as the ratio of total execution time or makespan of the original batch using the Round-Robin approach (T_{RR}) to the total execution time of extracting the execution time and mean power with the application signature (T_{AS}) of the whole batch, as shown in Eq. (1).

$$CR = \frac{T_{RR}}{T_{AS}} \tag{1}$$

4. Task scheduling approaches

In this section we explain the three task scheduling approaches to validate the use of the application signature. The task scheduling approaches presented rearrange the execution order of the tasks from a batch and allocate the workload to the servers in order to improve the energy efficiency in data centers. Three scheduling approaches were selected and implemented based on: Mixed Integer Linear Programming, a metaheuristic approach with Simulated Annealing and an energy-aware heuristic. These scheduling approaches were selected because they can be efficiently used with the real (oracle) values of either execution time or mean power, and this would not be possible without a full dynamic profiling of the applications. The goal of our work is not to outperform existing energy-aware task scheduling approaches, but to show that with the information provided by the application signature we open the possibility to perform energy optimization in data centers with scheduling techniques more powerful than applying reactive heuristics. The three task scheduling approaches covers in great detail different options of searching the optimal value. The

optimal value of energy savings is found with the Mixed Integer Linear Programming approach, and this is the baseline of the best result of energy savings we can expect against the other scheduling approaches. Finally, the following approaches assume that the number of copies or threads that each task will execute is a known information and is not considered as an information from the oracle nor the application signature.

4.1. Mixed integer linear programming formulation

A mixed integer linear programming (MILP) formulation is presented for the task scheduling problem of a batch execution in a data center. In the present work the MILP formulation for the task scheduling of sequential and parallel tasks is based on the work from Goldman et al. [13]. We use this formulation since is a generic MILP task scheduling formulation and it solves our task scheduling problem very efficiently. The original MILP formulation does not consider that all the resources of the data center are distributed along different servers. Therefore, we propose a modification of the original MILP formulation in order to send each task to a different server. The batch B_{MILP} is formed by a number of n independent parallel tasks T_i $(1 \leq i \leq n)$. Each task requires c_{T_i} cores and has an execution time equal to t_i. Each parallel task can only be executed in one server S_i, and there is m number of servers. Each server has c_{server} cores.

Fig. 3(a) shows the task scheduling process using the MILP formulation. The **input** for the MILP problem is the execution time t_i of each task T_i of the batch B_{MILP}. The **output** for the MILP task scheduling approach is the batch of tasks with their respective starting time (τ_i) and the assigned server (S_i) where the task is going to be executed. For example, at the time instant of 100 the task T_3 should begin its execution. Therefore, the 4 copies or threads (c_{T_3}) of task T_3 are allocated in the server S_2. The 2 copies or threads (c_{T_2}) of task T_2 were already running in the server S_1 since its starting time is equal to 0. The task T_1 needs to wait until the time instant 500.

The objective of the MILP optimization process is to minimize the makespan (C) of the whole batch B_{MILP} of tasks. The complete MILP formulation is as follows:

Minimize C
Subject to:

1. $\tau_j \geq 0 \ \forall T_j$
2. $x_{kj},\ y_{kj} \in \{0,1\} \ \forall T_j, T_k$
3. $z_{ji} \in \{0,1\} \ \forall T_j, S_i$
4. $\sum_{i=1}^{s} z_{ji} = 1 \ \forall T_j, S_i$
5. $\tau_j \leq \tau_k + t_k + (3 - x_{kj} - z_{ji} - z_{ki})X \ \forall T_j, T_k, S_i$
6. $\tau_k + t_k \leq \tau_j + t_j + (3 - x_{kj} - z_{ji} - z_{ki})X \ \forall T_j, T_k, S_i$
7. $\tau_k + t_k + d \leq \tau_j + (2 + y_{kj} + x_{kj} - z_{ji} - z_{ki})X \ \forall T_j, T_k, S_i$
8. $\tau_j + t_j + d \leq \tau_k + t_k + (3 - y_{kj} + x_{kj} - z_{ji} - z_{ki})X \ \forall T_j, T_k, S_i$
9. $\sum_{\substack{j=1 \\ j \neq k}}^{n} c_{T_j} \times x_{kj} \leq c_{server} - c_{T_k} \ \forall T_k$
10. $C \geq \tau_k + t_k \ \forall T_k$

Constraint (1) indicates that the starting time (τ_i) of each task must be a positive number. Constraints (2)–(3) show the decision variables of the MILP formulation. The decision variables x_{kj} and y_{kj} indicate the condition when a task starts the execution before or after another task. The decision variable z_{ji} indicates the condition when a task is running on server S_i. Constraint (4) guarantees that each parallel task runs in one server. Constraints (5)–(8) allows to calculate, with no overlaps, the starting time (τ_i) of each task. The constraint (9) assures the validity of the whole task scheduling process. This is done by checking that all the tasks that are concurrently running in each server do not consume more than c_{server} cores. Finally, constraint (10) shows the objective function as given by $C = max\{\tau_j + t_j\}$. The value of X is a constant and must follow the rule: $X > \sum_{k=1}^{n} t_k$. The constant d is a delay that guarantees the no presence of strict inequalities in the MILP formulation. The value of d is selected such as: $d < min_{1 \leq i \leq n}\ t_i/2$.

As we previously commented the output of the MILP task scheduling approach is the starting time τ_i and assigned server S_i of each task T_i of the original batch B_{MILP}. The value of the starting time τ_i of each task is directly obtained at the end of the MILP calculation. For the assigned server S_i we use the final state of the z_{ji} decision variable, where each row represent a task T_i and each column represent a server S_i. A value of 1 in a column indicates that the task is assigned to that server.

4.2. Simulated annealing

A metaheuristic approach is proposed to minimize the makespan (C) of the task scheduling process based on the Simulated Annealing technique. As opposed to the MILP technique, the Simulated Annealing process does not guarantee to find the global optimum whereas is used to find an approximate global optimum in large search spaces. Furthermore, the Simulated Annealing technique has the advantage, like other metaheuristics, to be more scalable than the MILP technique. The Simulated Annealing technique works by modeling the physical process whereby a solid material is slowly cooled until it reaches a frozen state, which happens at a minimum system energy.

We propose the use of a Simulated Annealing algorithm to rearrange the tasks T_i from a batch of n tasks in order to minimize the makespan (C) of the task scheduling process. Fig. 3(b) shows the Simulated Annealing task scheduling approach. The **input** for the Simulated Annealing algorithm is the execution time t_i of each task of the batch B_{SA}. The **output** is the batch rearranged, where the tasks from the batch are executed in a different order from the original batch using a Round-Robin approach. In the example, we can see that after the Simulated Annealing process is finished the positions for tasks T_3, T_{10} and T_1 are different from the original batch. Each task of the rearranged batch is allocated to the server using a Round-Robin process. Since task T_3 is the first in the batch its 4 copies or threads (c_{T_3}) are allocated in first available server which is server S_1. The 2 copies or threads ($c_{T_{10}}$) of task T_{10} are allocated in the server S_2, followed by the 2 copies or threads (c_{T_1}) of task T_1. Tasks T_{10} and T_1 are allocated in the server S_2 since there is enough available cores for both of them. The rest of the tasks of the rearranged batch are allocated in the rest of available servers.

Algorithm 2 Simulated Annealing Algorithm

Require: batch: B_{SA}
1: set initial temperature $Temp = Temp_0$
2: **while** k ≤ max_iterations **do**
3: k=k+1
4: temperature_iteration=0
5: **while** temperature_iteration≤n **do**
6: $B_{rearranged}$=annealing_function(B_{SA},Temp)
7: C=makespan_function($B_{rearranged}$)
8: $\Delta = C\text{-}C_{old}$
9: **if** $\Delta < 0$ **then**
10: $B_{SA}=B_{rearranged}$
11: **else**
12: $B_{SA}=B_{rearranged}$ with probability $P = \frac{1}{1+exp(\frac{\Delta}{Temp})}$
13: **end if**
14: temperature_iteration=temperature_iteration+1
15: **end while**
16: $Temp = Temp_0 \times 0.95^k$
17: **end while**

Algorithm 3 Annealing Function (*annealing_function*)

Require: batch: B_{SA}, Temp
1: **for** i=1:Temp **do**
2: r1=random_integer(1,length(B_{SA}))
3: r2=random_integer(1,length(B_{SA}))
4: $B_{tmp}=B_{SA}$
5: $B_{SA}(r1)=B_{tmp}(r2)$
6: $B_{SA}(r2)=B_{tmp}(r1)$
7: **end for**

The Simulated Annealing algorithm is shown in Algorithm 2. The input of the algorithm is the original batch, which has the information of the execution times t_i of each task T_i. The algorithm starts by setting an initial temperature $Temp_0$. The temperature variables $Temp$ and $Temp_0$ are defined in the context of the Simulated Annealing process. They should not be confused with the temperature values of the servers in the data center. The algorithm has two while loops: (i) the inner while loop ((5)–(15)) rearranges the original batch B_{SA} to improve the makespan C. This is done with the same temperature $Temp$ and a number of n iterations, and (ii) the outer while loop ((2)–(17)) decrease the temperature $Temp$ value until a number of iterations equal to *max_iterations*. Inside the inner while loop a random trial point is generated using the annealing function (6). In this process the concept of point refers to changes in the tasks T_i positions from the batch B_{SA}, generating a new rearranged batch $B_{rearranged}$. Next, the algorithm determines if the new point is better or worse (in terms of minimizing the makespan C) than the current point with a probability of acceptance P, as shown from lines (7) to (13). To calculate the makespan C the algorithm uses the objective function *makespan_function* taking as input the rearranged batch $B_{rearranged}$. Finally, the temperature changes according to a function (16). Where, $Temp_0$ is the initial

Algorithm 4 Objective Function (*makespan_function*)

Require: Batch: B_{SA}
1: C=0
2: num_servers=m
3: num_cores=c_{server}
4: servers=ones(num_servers,num_cores) /*A matrix of ones. Saves the current temporal state of each core of the server*/
5: used_cores=zeros(num_servers) /*A vector of zeros. Saves the available cores of each server*/
6: flag=0 /*Flag to label if a task is allocated to a server*/
7: **while** true **do**
8: **if** length(B_{SA})>0 **and** $B_{SA}(1,1) \leq$ C **then**
9: **for** k=1:length(B_{SA}) **do**
10: **if** $B_{SA}(k,1)$ >C **then**
11: break
12: **end if**
13: **for** i=1:length(servers) **do**
14: **if** used_cores(i)+$B_{SA}(k,3) \leq$ num_cores **then**
15: available_cores=find_empty_cores(servers(i,:)≤0);
16: **for** j=1:$B_{SA}(k,3)$ **do**
17: servers(i,available_cores(j))=$B_{SA}(k,2)$;
18: **end for**
19: flag=1
20: break
21: **end if**
22: **end for**
23: **if** flag=1 **then**
24: flag=0
25: $B_{SA}(k,:)$=[]
26: break
27: **end if**
28: $B_{SA}(k,1)$=C+k
29: **end for**
30: **end if**
31: C=C+1
32: servers=servers-1 /*Reduce in one the current temporal state of each core of each server*/
33: used_cores=sum(servers>0) /*Update the available cores of each server*/
34: **if** length(B_{SA})<1 **and** servers<0==ones(num_servers,num_cores) **then**
35: break
36: **end if**
37: **end while**

temperature and k is the iteration number until reannealing. The two important functions of the Simulated Annealing algorithm are the annealing function and the objective function. The annealing function (*annealing_function*) process is shown in Algorithm 3. The function takes as input the batch B_{SA} and the temperature $Temp$. The goal of the function is to change the tasks T_i positions within the batch B_{SA}. The number of tasks position changes are proportional to the actual temperature $Temp$ state and are done in a random manner (*random_integer*).

The objective function (*makespan_function*) returns the makespan C of the task scheduling process, as shown in Algorithm 4. The input for the objective function is the batch B_{SA},a matrix with n rows (number of tasks in the batch B_{SA}) and three columns. The first column ($B_{SA}(i,1)$) is the task T_i starting time (or the task position), the second column ($B_{SA}(i,2)$) is the execution time t_i of the task T_i and the third column ($B_{SA}(i,3)$) is the number of cores (c_{T_i}) requested by the task T_i. Lines (7) through (37) are the core of the algorithm, where each task is assigned to a server until the batch is empty. The whole process is stopped when the batch is empty and all the tasks have completed their execution ((34)–(36)). The condition in line (8) through (30) guarantees that a task is scheduled if there are any task in the batch and the task starting time is less or equal than the makespan C. Then, for each task in the batch a task is assigned to a server ((9)–(29)). Lines (10) through (12) indicates that the task allocation process is temporarily stopped until the makespan C is equal or greater to the starting time of the next task on the batch. Lines (13) through (22) shows that a task is assigned to the first available server (17). The batch is updated by removing the task that has already been assigned to a server, as shown in lines (23) through (27). Line (28) shows the update of the starting time for the tasks that are in queue waiting for execution. Line (31) shows that the makespan C is updated every iteration and this is the final output of the algorithm.

4.3. Energy-aware heuristic

We leverage the use of an energy-aware heuristic to improve the energy efficiency in Data Centers. The energy-aware heuristic is based from the work of A. Beloglazov et al. [17]. In that work the authors propose an algorithm to allocate in an energy efficient

manner virtual machines in a cloud oriented scenario. We adapted their algorithm to our non-cloud oriented scenario and added an heuristic to sort the original batch of tasks. Fig. 3(c) shows the overall energy-aware heuristic approach for task scheduling in data centers. The **input** is the batch B_{EA} of tasks T_i. From the batch B_{EA} we have the information of execution times t_i and the mean CPU and memory power (both represented as P_n) of each task of the batch. As opposed to the MILP and Simulated Annealing approach the energy-aware heuristic is both proactive and reactive. The first step of the process is the proactive part of the approach, where the heuristic called Longest Task First (LTF) is used to sort the tasks of the original batch in a descending way according to the execution time t_i of each task to improve the overall makespan C. This means that the tasks with higher execution times t_i are executed first. In our example the task T_4 has a higher execution time than the rest of the tasks, therefore it will be executed first. The second step of the process is the reactive part, where the selected task will be send to the server where the energy consumption is increased the least. Following our example from Fig. 3(c) the values of mean power P_4 and execution time t_4 of task T_4 are used to estimate the increase of energy in each server of the data center. The server that shows the minimum energy increase is server S_3, thus the 4 copies or threads of task T_4 are allocated in S_3 of the data center.

Algorithm 5 Energy-Aware Heuristic

Require: Batch: B_{EA}
1: B_{LTF} = LTF($B_{EA}(:,1)$)
2: **for** i=1:length(B_{LTF}) **do**
3: minEnergy=MAX
4: allocatedServer=NULL
5: **for** j=1:length(servers) **do**
6: **if** servers(j) has enough resources for $B_{LTF}(i,3)$ **then**
7: energy=estimateEnergy(servers(j),$B_{LTF}(i,1)$,$B_{LTF}(i,2)$)
8: **if** energy < minEnergy **then**
9: minEnergy=energy
10: allocatedServer=server(j)
11: **end if**
12: **end if**
13: **if** allocatedServer ≠ NULL **then**
14: allocate $B_{LTF}(i)$ to allocatedServer
15: **end if**
16: **end for**
17: **end for**

The energy-aware heuristic algorithm is shown in Algorithm 5. The input of the algorithm is the batch B_{EA}, a matrix with a row number equal to the number of tasks and 3 columns. The first column $B_{EA}(i,1)$ is the execution time t_i, the second column $B_{EA}(i,2)$ is the values of mean power from both CPU and memory and the third column $B_{EA}(i,3)$ is the number of cores (c_{T_i}) requested by the correspondent task. The first step of the algorithm is to sort the original batch in a descending way according to the execution time of each task using the LTF heuristic (1). Then, from line (2) to line (17) each task of the sorted task list is allocated to a server where the energy is increased the least. If a server has enough resources for the task selected from the rearranged batch B_{LTF} (6) then the energy of that task running on that server is estimated (7). To estimate the energy we take into account the current power state of the server and update the mean dynamic CPU and memory power with the mean power values from the selected task. In practical terms, the mean dynamic CPU and memory power values from the selected task are added to the $P_{CPU,dyn}$ and $P_{Mem,dyn}$ of the Eq. (3) (Experimental Setup section) to update the server power. Additionally, the leakage power shown in Eq. (4) is updated by adding the mean dynamic CPU power of the selected task to the current $P_{CPU,dyn}$ value in Eq. (5). Once the current power state is updated the energy is estimated taking into account the execution time t_i of the selected task. The variables *minEnergy* and *allocatedServer* are updated every time a lower energy value is found ((9)–(10)). The process is repeated for every server that has enough resources ((5)–(16)) and the selected task is allocated to the server that shows the minimum increase in energy consumption (14).

5. Experimental methodology and setup

This section explains the experimental methodology and setup used. Fig. 4 shows the overall experimental methodology followed in this work. First, we generate randomly a batch of tasks. From each task of this batch we collect, in two different ways, the execution time and the mean power information. The oracle information is obtained by a time-consuming full dynamic profiling of the applications from the batch, while the application signature information is obtained by applying the fast energy estimation framework to the shorter version of the tasks. In order to calculate the overall data center power consumption, we use a data center simulator called SFIDE [19]. We implement the three energy-aware task scheduling approaches using the Allocator module inside the simulator. We execute the simulator for each task scheduling approach and for both the oracle and application signature case, and obtain the overall data center power profile. From the power profile we can calculate the energy consumption of the whole data center. The same process is applied using a Round-Robin approach for the original batch. In this case, there is no need to have the execution time and power information since the Round-Robin approach is not an energy-aware task scheduling policy. We compare

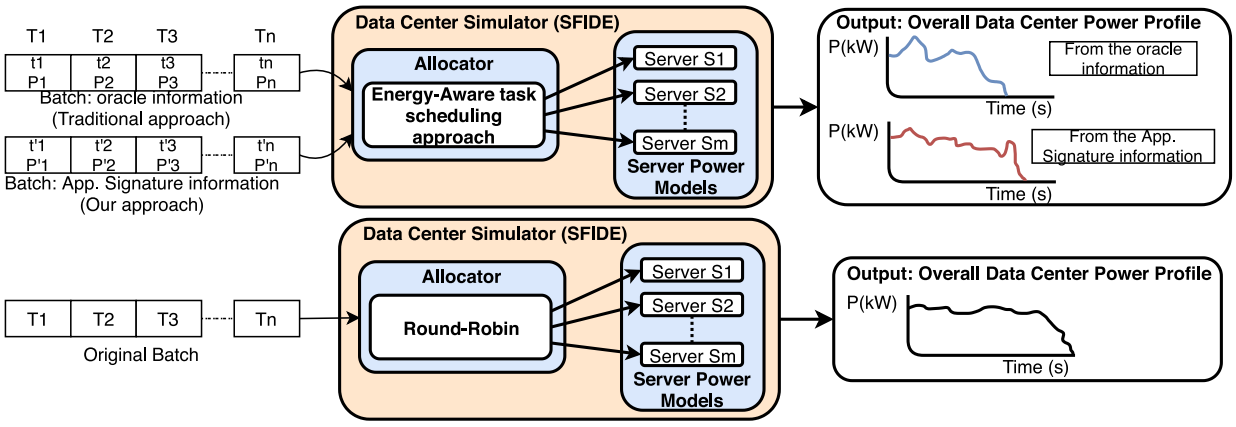


Fig. 4. Experimental methodology: validation of the app. signature information using a data center simulator (SFIDE).

the data center energy consumption from the energy-aware approaches and the energy consumption from the Round-Robin process to measure the energy savings, as shown in Eq. (2). Finally, we study two scenarios of long-running applications: (i) a small scale scenario using a small number of servers and, (ii) a large scale scenario that simulates a data center with a great number of servers.

$$Energy\ Saving\ (\%) = \frac{Energy_{Round\ Robin} - Energy_{Task\ Sched\ Approach}}{Energy_{Round\ Robin}} \times 100 \tag{2}$$

In the following sections we will show an overview of the data center simulator together with the server power models used in this work. Additionally, we will describe the type of applications that composed the batches from both the small and large scale scenarios.

5.1. Data center simulator

The SFIDE data center simulator [19] is a simulator based on discrete event system modeling (DEVS). The modular architecture of the simulator allows an easy implementation of the different task scheduling approaches. The SFIDE simulator was validated against real server and data center traces of the same type of workload we use in this work. The SFIDE data center simulator is composed by two main modules: Room and Cooling. The Room module is where the task allocation and processing takes place. The Cooling module regulates the temperature of the data center according to a cooling model. We do not implement a cooling model and therefore to calculate the overall data center power we assume a fixed Power Usage Effectiveness (PUE) value. The Room module is formed by the following modules: Allocator, In Row Cooling Units (IRC), Rack and Server. The Allocator module takes a batch of tasks and assign each task to a server. In the present work we implement different Allocator modules for each task scheduling approaches. The IRC module groups a set of Racks and each Rack contains a set of Servers. The IRC module computes the overall status of all the servers from all the racks. For the Server module we use an specific server model, the Decathlete server model since this is the same model we used in our previous work to implement and evaluate the applications signatures. The Decathlete (Intel S2600GZ) server model defined in the Server module is composed by 2 SandyBridge-EP processors, both with 6 cores and therefore each server has a total of 12 cores (c_{server}). Additionally, the server has 16 4 GB memory modules, 4 hard disk drives, 5 fans and 2 power supply units. For both the Simulated Annealing and the Energy-Aware heuristic approaches we implemented the algorithms as an independent modules in the SFIDE data center simulator. The MILP formulation was implemented using IBM ILOG CPLEX, version 12.8.

5.1.1. Server power model and overall data center power

The server power model defined in the simulator was built and validated in our previous work [20]. In that work, we use Grammatical Evolution techniques to predict dynamic CPU and memory power together with a temperature-dependent leakage power model to obtain the overall server power model. The overall server power model was validated with real traces of a heterogeneous set of workloads. The overall server power model has a prediction error below 12 W, which represents 7.3% of the overall server power. The server power model is shown in Eq. (3). Where P_{Idle} is the server power when there is no task execution and is the aggregated idle power of the CPU's, memory subsystems and the power of the other components of the server (disks units, motherboard, etc.). The $P_{CPU,dyn}$ and $P_{Mem,dyn}$ values are the dynamic CPU and memory power from a task being executed in the server. Finally, the $P_{Leakage}$ value represents the temperature-dependent leakage power, which has a dependence with the

Table 1
Real and estimated mean power and execution time values of the tasks.

Apps	Threads	Oracle			Application signature		
		$P_{CPU,dyn}$ (W)	$P_{Mem,dyn}$ (W)	Exec. time (s)	$P_{CPU,dyn}$ (W)	$P_{Mem,dyn}$ (W)	Exec. time (s)
BT-D	1	7.45	7.37	55 669	7.94	7.85	58 740
	2	10.12	8.98	27 982	11.04	9.92	29 411
	4	15.27	12.17	14 256	16.94	14.01	14 915
	6	20.06	15.31	9717	22.54	18.03	10 108
SP-D	1	8.65	7.70	39 627	8.91	7.80	36 997
	2	12.25	9.55	20 572	12.79	9.76	19 032
	4	18.87	13.10	10 822	19.87	13.46	9986
	6	22.72	15.37	8438	24.92	15.95	7698
Dgemm1	1	5.46	10.19	1721	5.31	11.07	1674
Dgemm2	1	7.53	8.87	41 420	7.57	8.92	40 727
Stream1	1	7.40	7.64	1493	7.42	7.52	1691
Stream2	1	7.44	7.67	15 396	7.43	7.65	15 441
Linpack1	1	6.38	5.95	1420	7.68	5.24	1033

CPU temperature as shown in Eq. (4). Furthermore, the CPU temperature (T_{CPU}) has a dynamic CPU power dependence (Eq. (5)). Finally, the value of P_{fan} is the power from the server fans, which has a cubic dependence with the fan speed.

$$P_{server} = P_{Idle} + P_{CPU,dyn} + P_{Mem,dyn} + P_{Leakage} + P_{fan} \tag{3}$$

$$P_{Leakage} = \alpha_0 + \alpha_1 \times T_{CPU} + \alpha_2 \times T_{CPU}^2 \tag{4}$$

$$T_{CPU} = k_0 + k_1 \times P_{CPU,dyn} \tag{5}$$

The value of P_{Idle} is equal to 120 W. The simulations are done with a fixed fan speed for all servers equal to 6000 RPM, resulting in a fan power equal to 14.4 W. The coefficients for the CPU temperature are: $k_0 = 44.3$ and $k_1 = 0.7353$, which are derived from a LUT using a fixed ambient temperature value equal to 22 °C. The coefficients for the leakage power $P_{Leakage}$ are: $\alpha_0 = 27.5$, $\alpha_1 = -1.016$ and $\alpha_2 = 0.0112$. Finally, the dynamic CPU and memory power values ($P_{CPU,dyn}$ and $P_{Mem,dyn}$) depends on the CPU and memory power of the tasks being executed on the server. The SFIDE simulator is able to calculate the overall data center power, including IT power and cooling power. We established a fixed PUE equal to 1.5 to obtain the overall data center power P_{DC}, as shown in Eq. (6). The value of P_{IT} is the power from all the servers of the data center. The PUE values for data centers around the world are in the range of 1.1 to more than 1.5 with an average PUE value of 1.58 in 2018 [21]. Therefore, a PUE value of 1.5 is a reasonable value for a current energy-efficient data center. The simulator outputs the data center power P_{DC} each time there is an event (a task is send to a server, a task ends its execution, etc.), allowing to obtain a power profile of the overall data center power consumption. Therefore, the energy consumption of the data center can be calculated using the data center power profile and the makespan of the executed batch of tasks.

$$\begin{aligned} P_{DC} &= PUE \times P_{IT} \\ where \quad P_{IT} &= \sum_{i=1}^{m} P_{server_i} \end{aligned} \tag{6}$$

5.2. Simulation scenarios and task batch composition

We validate our results using two data center scenarios: (i) a small scale scenario formed by 1 rack and 5 servers, which creates a scenario with a total of 60 available cores; (ii) a large scale scenario formed by 5 racks and 10 servers per rack, generating a total of 600 available cores. To validate the usefulness of the application signature for energy savings we use an heterogeneous set of long-running tasks composed of: the applications *BT-D* and *SP-D* (input: Class D) from the NAS Parallel suite [22], *Stream* [23], *Dgemm* [24] and *Linpack* [25] benchmarks. Table Table 1 shows the CPU and memory power values together with the execution times of each application. Moreover, these power and execution time values were defined in the SFIDE simulator. The oracle values are the real values obtained from the full execution of the applications and the application signature values are the estimated power and execution times values. We can see the long execution times of the applications considered in this work. The overall estimated total execution time error, from the application signature process, is below 14.0% except for the *Linpack1* application where the estimated total execution time error is equal to 27.9%. The Compression Ratios of the workload set are in the range of 10.1 to 191.2, indicating that the application signature process estimates mean power and total execution time 191.2 times faster than executing the whole original application (oracle). The differences between Dgemm/Stream 1 and 2 come from the input parameters. The input parameters for Dgemm are a matrix size of 1024 and 2048, for Dgemm1 and Dgemm2 respectively. The input parameters for Stream are an array size of 1.0^9 and 1.05^9, for Stream1 and Stream2 respectively. The input parameters for Linpack1 are a number of time steps equal to 20 000 and a size of matrices equal to 300. A more detailed explanation of the data presented in Table Table 1 can be found in our previous work [6].

Table 2
Energy savings results for the small and large scale scenario when compared to the baseline Round-Robin policy.

	Small scale scenario				
	Energy saving (%)			Makespan (s)	
	Oracle	Application signature	Diff.	Oracle	Application signature
MILP	19.4	18.3	**1.1**	58 467	59 090
Metaheuristic	17.7	16.3	**1.4**	59 591	62 188
Heuristic	17.0	15.5	**1.5**	61 548	64 117
	Large scale scenario				
	Energy saving (%)			Makespan (s)	
	Oracle	Application signature	Diff.	Oracle	Application signature
Metaheuristic	13.3	12.5	**0.8**	74 119	75 815
Heuristic	8.6	8.2	**0.4**	82 899	82 979

The applications *BT-D* and *SP-D* are parallel and the rest of the applications are sequential. For the sequential applications we also consider the execution with a number of copies higher than 1, specifically 2, 4 and 6 copies. For example, Dgemm1-4 would represent the execution of 4 copies of the application Dgemm1. In the simulator we adapted each sequential application to be executed with more than 1 copy by escalating the CPU and memory power values. The execution times for more than 1 copy remain the same since is not a parallel execution. The batch for both scenarios are generated randomly with sequential (1 thread) and parallel tasks (2, 4 and 6 threads or copies) from the workload set previously commented. Finally, the batch size for both the small and large scale scenario, are equal to 66 and 900 tasks respectively.

6. Results

In this section we present the results of applying the three different task scheduling approaches using the information from the application signature. We compare the energy savings against a Round-Robin task scheduling approach (baseline) of the original task list. Additionally, we evaluate the error of the energy savings values from the application signature with energy savings values obtained from the oracle values of task execution times and mean CPU and memory power. The energy savings are calculated as shown in Eq. (2).

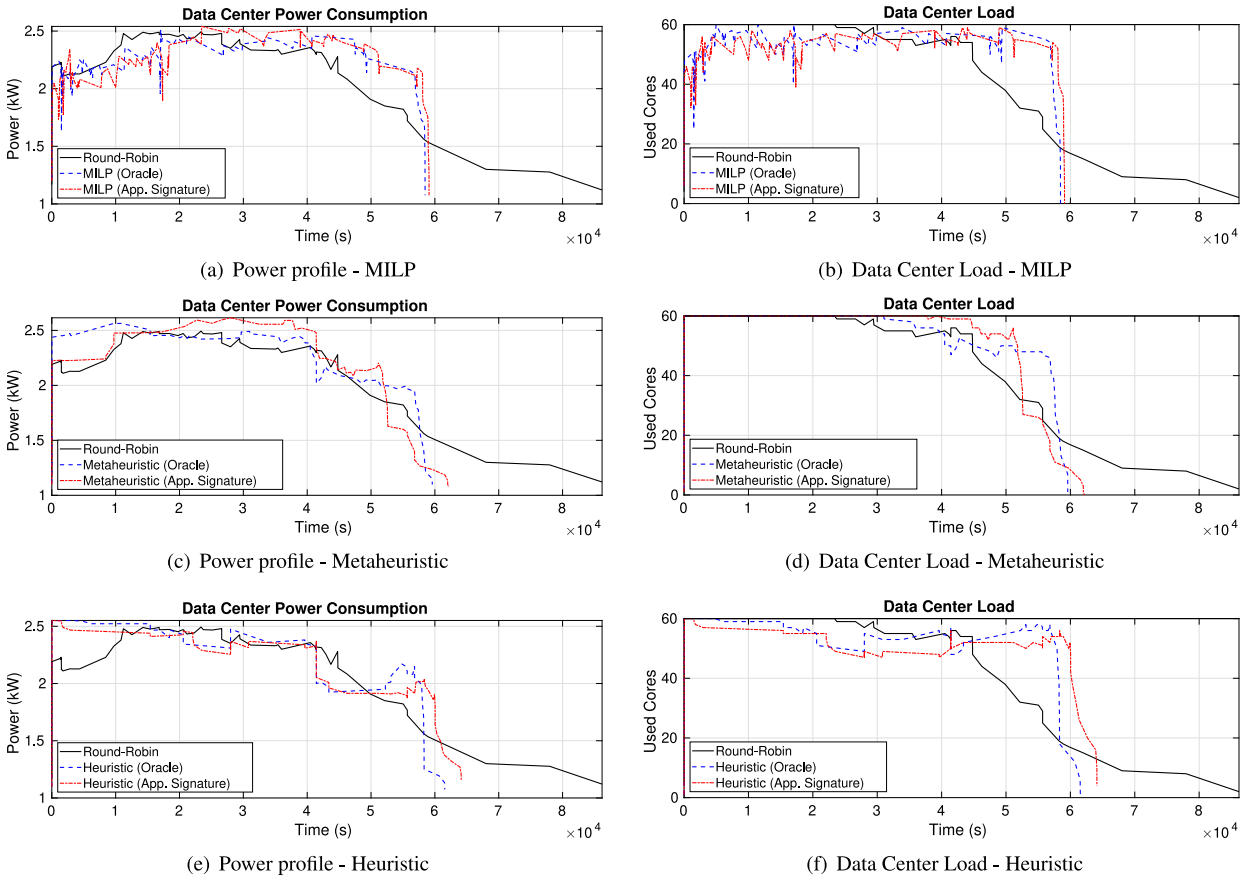
6.1. Small scale scenario

Table 2 shows the energy savings and makespan of every task scheduling approach for the small scale scenario when compared to the baseline Round-Robin policy. The results presented in Table Table 2 are calculated taking into account that the application signature of all the tasks in the batch is already extracted. This means that both energy savings and makespan values do not include the energy and makespan of the application signature calculation process. This is a fair approach since we are comparing the results against an oracle that extract the information (execution time and mean power) from a full profiling of the original applications.

The MILP technique offers the highest energy savings when compared with the baseline task scheduling approach (Round-Robin), both for the application signature and the oracle. The metaheuristic and heuristic energy savings are below the MILP technique, although presenting energy savings higher than 15% when compared with the Round-Robin approach. The result is expected since the MILP technique searches the global optimum while the metaheuristic and the heuristic find an approximate global optimum. Furthermore, we can see the energy savings values obtained by using the information from the application signature are close to the energy savings values by using the information of the original application (oracle). The difference of the energy savings between the values from the application signature and the oracle is below 1.5%. Additionally, Table 2 shows the makespan values for all the task scheduling approaches. As expected, when using the application signature information, the lowest makespan value is from the MILP approach with a value equal to 59 090 s.

Figs. 5(a), 5(c) and 5(e) show the power profile of each task scheduling approach when compared with the Round-Robin approach. As we can see, the overall energy savings between all the task scheduling approaches comes from the makespan optimization since every batch ends before the Round-Robin policy. The power profiles from each task scheduling approach are different since the tasks are scheduled in different order. For example, the MILP approach (Fig. 5(a)) shows, at the beginning of power profile, an average power lower than the metaheuristic and heuristic approach. The heuristic approach (Fig. 5(e)) shows a power peak at the beginning of the power profile indicating that this approach is scheduling high power consuming tasks at the beginning of the batch execution.

Figs. 5(b), 5(d) and 5(f) show the load profile, or the number of used cores, of each task scheduling approach. In case of the MILP approach, the number of used cores are between 50 and 60 cores during the whole batch execution and it almost never gets to a value equal to 60 cores. The metaheuristic approach shows the opposed behavior, as shown in Fig. 5(d). The number of used cores is equal to 60 cores during almost all the batch execution, therefore all the data center resources are being used during the execution. In the case of the heuristic approach (Fig. 5(f)), the number of used cores are close to 60 at the beginning of the batch execution, it slightly decreases until the end of the execution where the number of used cores peaks again.



(a) Power profile - MILP (b) Data Center Load - MILP

(c) Power profile - Metaheuristic (d) Data Center Load - Metaheuristic

(e) Power profile - Heuristic (f) Data Center Load - Heuristic

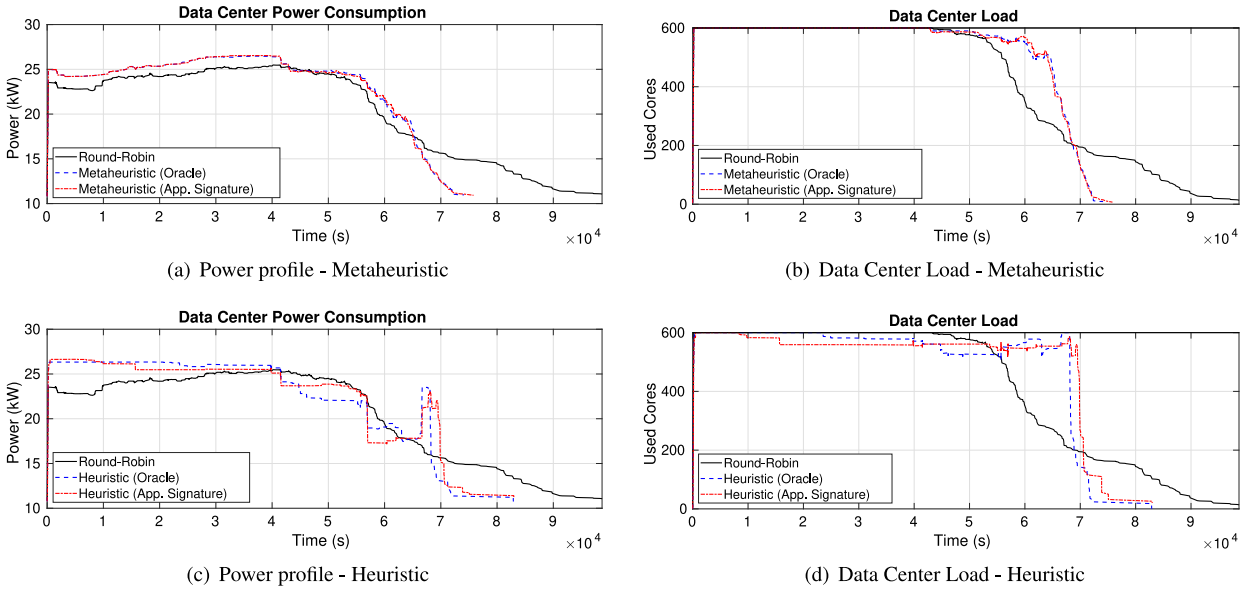
Fig. 5. Small scale scenario: Power and load profiles.

The differences between the load profiles resides on the task scheduling process of each approach. In the output of the MILP approach each task of the batch has an initial execution time together with a small delay (parameter d of the MILP formulation) respect of the previous executed task. This explains the noisy form of the load profile seen in Fig. 5(b). As opposed to the MILP approach, the output of the metaheuristic only rearranges the original batch and then applies a Round-Robin approach to send the tasks to available servers, the tasks from the batch do not have the small delay shown in the MILP approach. This leads to a more stable load profile (Fig. 5(d)), where during almost all the batch execution the data center is completely loaded because the Allocator (from SFIDE) is assigning each task to a server once a server is available. The heuristic approach has a similar stable load profile as the metaheuristic approach, as seen in Fig. 5(f). In this case, the data center is not completely loaded during the batch execution. The task scheduling process of the heuristic is allowing tasks with low number of threads (for example, 1 or 2 threads) get assigned at the same time to different servers not allowing tasks with high number of threads getting assigned to an available server. Thus, leaving a number of cores unused during the execution of the batch.

6.2. Large scale scenario

Table 2 shows the energy savings and makespan of every task scheduling approach for the large scale scenario. In this scenario the MILP approach is not used since is not scalable and the process to find the global optimum would take too much time. We obtain energy savings higher than 8% when compared with the Round-Robin task scheduling approach. The metaheuristic approach presents the highest value of energy saving and is equal to 12.5% when the information of the application signature is used. The difference of the energy savings between the values from the application signature and the oracle is below 0.8%. Additionally, the makespan is shown in Table 2 for the metaheuristic and heuristic approaches. As expected, the metaheuristic present better results than the heuristic since it can find a closer approximate value to the global optimum (minimize the makespan C).

Figs. 6(a) and 6(c) show the power profile of each task scheduling approach when compared with the Round-Robin approach for the large scale scenario. The power profile of the metaheuristic approach shows a stable signal during the whole execution of the batch, when using the information from both the oracle and the application signature. The main difference is that the batch execution ends earlier when the information from the oracle is used. The heuristic task scheduling approach power profiles (Fig. 6(c))



(a) Power profile - Metaheuristic

(b) Data Center Load - Metaheuristic

(c) Power profile - Heuristic

(d) Data Center Load - Heuristic

Fig. 6. Large scale scenario: Power and load profiles.

for both the oracle and the application signature are very similar. Both power profiles show a rise of the overall data center power around the time 7×10^4 s indicating that a set of power consuming tasks were waiting to be assigned to available servers.

The load profiles for both the metaheuristic approach and the heuristic approach are shown in Figs. 6(b) and 6(d). The loads profiles are similar to those obtained from the small scale scenario. In the metaheuristic approach the load profile obtained when using the information from the application signature is very similar to the load profile when using the information of the oracle. The load profile from the heuristic task scheduling approach when using the information of the application signature is slightly different from the oracle, since the load profile from the oracle shows a more loaded data center until around the time 4×10^4 s. This indicates the task assigning process when using the application signature is different from the oracle. Nonetheless, the results show that using the information of the application signature will result in energy savings similar to the energy savings obtained when using the oracle information.

6.3. Compression ratio

In this section we compare the execution time of the application signature extraction process against the execution time of the whole batch using the Round-Robin approach. For the small scale scenario the execution time of the application signature extraction of the whole batch (66 tasks) takes 1881 s, a 2.18% of the execution time of the whole batch using the Round-Robin approach (86 187 s). This leads to a Compression Ratio of the whole batch equal to 45.8, when we compare the execution time of application signature process to the execution time of the whole batch using the Round-Robin task scheduling approach. The large scale scenario has similar results with an execution time of the application signature process equal to 2483 s, a 2.51% of the execution time of the whole batch using the Round-Robin task scheduling approach (98 667 s). Resulting in a Compression Ratio equal to 39.7. These results show that using the application signature process is viable, when compared to the Round-Robin approach, since the Compression Ratios have high values.

6.4. Overall results

The overall results shows that the energy savings difference between the use of the oracle information and the use of the application signature information is less than 1.5% and Compression Ratios around 39.7 to 45.8. This indicates that the information provided by the application signature allows to apply the different energy-aware task scheduling approaches with similar results from the oracle information.

We use three methodologies that cover in a general way different energy-aware task scheduling approaches for energy saving in data center and obtained energy savings around 8.2% to 19.4%. Nonetheless, these results show that we can use or develop many other sophisticated energy-aware scheduling approaches that use the information provided by the application signature. Before our application signature proposal, these approaches were only possible to be used with a priori full dynamic profiling of the applications. Furthermore, we can use the information provided by the application signature to optimize energy-aware scheduling algorithms and thus, improve the energy efficiency in data centers.

7. Conclusions

In this work, we presented the use of an application signature for energy-aware task scheduling approaches. The application signature is a reduced version (in terms of execution time) of the original application and allows to estimate mean power and execution time without the need to fully execute the application. We use the information given by the application signature together with energy-aware task scheduling approaches to obtain energy savings in data centers in the scenario of long-running applications. These type of applications run for hours and therefore is not viable to perform a full dynamic profiling. We validate our results by implementing three energy-aware scheduling approaches based on: Mixed Integer Linear Programming, Simulated Annealing and an energy-aware heuristic. We use a data center simulator together with an heterogeneous set of long-running applications to evaluate the results in a small and large scale scenario. The energy savings from each scheduling approach were obtained by comparing the energy values of a batch execution against a baseline scheduling approach based on a Round-Robin policy. The energy savings values obtained with the application signature were compared against the energy savings obtained through the real (oracle) energy values of the applications. The results showed a difference between the energy savings obtained with the application signature and the oracle values below 1.5%, indicating that using the application signature for energy-aware task scheduling approaches is useful. We obtained Compression Ratios around 39.7 to 45.8, showing that using the application signature is an efficient method to be used for energy savings in data centers by energy-aware task scheduling approaches. Finally, although it was not the main goal of this work, we showed that the three selected energy-aware task scheduling approaches provide high energy savings when compared with a Round-Robin policy, hence we obtained energy savings around 8.2% to 19.4%.

CRediT authorship contribution statement

Juan Carlos Salinas-Hilburg: Methodology, Data curation, Writing – original draft. **Marina Zapater:** Conceptualization, Writing – review & editing. **José M. Moya:** Conceptualization, Writing – review & editing. **José L. Ayala:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially founded by a research grant from Complutense University of Madrid and Banco Santander, Spain, under grant CT45/15-CT46/15. We also acknowledge support from the Spanish Ministry of Science and Innovation under project PID2019-110866RB-I00.

References

[1] Shehabi A, Smith SJ, Sartor DA, Brown RE, Herrlin M, Koomey JG, Masanet ER, Horner N, Azevedo IL, Lintner W. United States data center energy usage report. Tech. rep., Lawrence Berkeley National Laboratory; 2016.
[2] Masanet E, Shehabi A, Lei N, Smith S, Koomey J. Recalibrating global data center energy-use estimates. Science 2020;367(6481).
[3] Barroso LA, Clidaras J, Hölzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synth Lect Comput Archit 2013;8(3):1–154.
[4] Aikema D, Kiddle C, Simmonds R. Energy-cost-aware scheduling of HPC workloads. In: 2011 IEEE international symposium on a world of wireless, mobile and multimedia networks. IEEE; 2011, p. 1–7.
[5] Haidar A, Jagode H, Vaccaro P, YarKhan A, Tomov S, Dongarra J. Investigating power capping toward energy-efficient scientific applications. Concurr Comput: Pract Exper 2019;31(6):e4485.
[6] Salinas-Hilburg JC, Zapater M, Moya JM, Ayala JL. Fast energy estimation framework for long-running applications. Future Gener Comput Syst 2021;115:20–33.
[7] Sîrbu A, Babaoglu O. Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer. In: Euro-par 2016: Parallel processing. Cham: Springer International Publishing; 2016, p. 117–30.
[8] Sadjadi SM, Shimizu S, Figueroa J, Rangaswami R, Delgado J, Duran H, Collazo-Mojica XJ. A modeling approach for estimating execution time of long-running scientific applications. In: 2008 IEEE international symposium on parallel and distributed processing. 2008, p. 1–8.
[9] Wong A, Rexachs D, Luque E. Parallel application signature for performance analysis and prediction. IEEE Trans Parallel Distrib Syst 2015;26(7):2009–19.
[10] Yang LT, Ma X, Mueller F. Cross-platform performance prediction of parallel applications using partial execution. In: SC'05: Proceedings of the 2005 ACM/IEEE conference on supercomputing. IEEE; 2005, p. 40.
[11] Auweter A, Bode A, Brehm M, Brochard L, Hammer N, Huber H, Panda R, Thomas F, Wilde T. A case study of energy aware scheduling on supermuc. In: International supercomputing conference. Springer; 2014, p. 394–409.
[12] Mämmelä O, Majanen M, Basmadjian R, De Meer H, Giesler A, Homberg W. Energy-aware job scheduler for high-performance computing. Comput Sci Res Dev 2012;27(4):265–75.
[13] Goldman A, Ngoko Y. A MILP approach to schedule parallel independent tasks. In: 2008 international symposium on parallel and distributed computing. IEEE; 2008, p. 115–22.
[14] Chrétien S, Nicod J-M, Philippe L, Rehn-Sonigo V, Toch L. Job scheduling using successive linear programming approximations of a sparse model. In: European conference on parallel processing. Springer; 2012, p. 116–27.
[15] Lei H, Wang R, Zhang T, Liu Y, Zha Y. A multi-objective co-evolutionary algorithm for energy-efficient scheduling on a green data center. Comput Oper Res 2016;75:103–17.

[16] Kashani M, Jahanshahi M. Using simulated annealing for task scheduling in distributed systems. In: International conference on computational intelligence, modelling and simulation. 2009, p. 265–9.
[17] Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 2012;28(5):755–68.
[18] Garefalakis P, Karanasos K, Pietzuch P, Suresh A, Rao S. Medea: scheduling of long running applications in shared production clusters. In: Proceedings of the thirteenth EuroSys conference. 2018; p. 1–13.
[19] Penas I, Zapater M, Risco-Martín JL, Ayala JL. SFIDE: a simulation infrastructure for data centers. In: Proceedings of the summer simulation multi-conference. 2017; p. 1–12.
[20] Salinas-Hilburg JC, Zapater M, Risco-Martín JL, Moya JM, Ayala JL. Unsupervised power modeling of co-allocated workloads for energy efficiency in data centers. In: Design, automation & test in Europe conference & exhibition (DATE). 2016, p. 1345–50.
[21] Koronen C, Åhman M, Nilsson LJ. Data centres in future European energy systems—energy efficiency, integration and policy. Energy Effic 2020;13(1):129–44.
[22] Bailey D, Barszcz E, Barton J, Browning D, Carter R, Dagum L, Fatoohi R, Frederickson P, Lasinski T, Schreiber R, Simon H, Venkatakrishnan V, Weeratunga S. The nas parallel benchmarks. Int J Supercomput Appl 1991;5(3):63–73.
[23] McCalpin JD. A survey of memory bandwidth and machine balance in current high performance computers. 1995.
[24] Luszczek PR, Bailey DH, Dongarra JJ, Kepner J, Lucas RF, Rabenseifner R, Takahashi D. The HPC challenge (HPCC) benchmark suite. In: Proceedings of the 2006 ACM/IEEE conference on supercomputing, Vol. 213. Citeseer; 2006, p. 1188455–677.
[25] Dongarra J. The LINPACK benchmark: An explanation. In: Proceedings of the 1st international conference on supercomputing. London, UK, UK: Springer-Verlag; 1988, p. 456–74.

Juan Carlos Salinas-Hilburg received his Ph.D. degree in Computer Science at the Complutense University of Madrid (2021). Additionaly, he received the M.Sc. degree in Telecommunication Engineering from the Public University of Navarre (2013) and also, the M.Sc. degree in Electronic Systems Engineering from the Technical University of Madrid (2014).

Marina Zapater received her Ph.D. degree in Electronic Engineering from the Technical University of Madrid in 2015 and she is currently an associate professor in the School of Management and Engineering Vaud (HEIG-VD), University of Applied Sciences Western Switzerland (HES-SO).

José M. Moya received the M.Sc. and Ph.D. degrees in Telecommunication Engineering from the Technical University of Madrid, Spain, in 1999 and 2003, respectively. He is currently an associate professor with the Department of Electronic Engineering, Technical University of Madrid.

José L. Ayala received the M.Sc. and Ph.D. degrees in telecommunication engineering from the Technical University of Madrid, Spain, in 2001 and 2005, respectively. He is currently an Associate Professor with the Department of Computer Architecture and Automation, Complutense University of Madrid.