Scheduling independent tasks in parallel with power constraint

Ayham KASSAB, Jean-Marc NICOD, Laurent PHILIPPE, Veronika REHN-SONIGO
FEMTO-ST Institute / CNRS – Univ. Bourgogne Franche-Comté
25000 Besançon, France
Email:{ayham.kassab,jean-marc.nicod,laurent.philippe,veronika.sonigo}@ubfc.fr

***Abstract*—Energy consumption has become a major concern in the recent years and Green computing has arisen as one of the challenges in order to reduce CO_2 emissions in the computing domain. Many efforts have been made to make hardware less energy consuming, reduce cooling energy of data and computing centers by relocating those facilities to cool regions and other. A novel approach to make the computing domain greener is to add renewable energy sources for the power supply. The challenge of this work is to consider computing facilities which are solely run by renewable energy sources such as solar panels and wind turbines. In this work we tackle the problem of scheduling independent tasks within a predicted power envelope that varies during the time. First we evaluate different instances of the problem from a theoretical point of view. Then we propose several heuristics for the case of multi-core architectures and we assess their performance through extensive simulations.**

***Index Terms*—Energy**

I. INTRODUCTION

Improving energy efficiency is one of today's major concerns as it is one possible solution to minimize global warming. In computer science and information technology, lots of research works tackle this problem as computers and devices in computing or data centers are known to be one of the big energy consumers, at least they will be in the future if we continue in the same direction. There are however several ways to reduce the energy footprint of a consumer, either reducing its consumption or using energy sources that less impact the environment. In the case of IT or computing resources lots of effort is put on reducing the energy consumption of the resources, from the processor to the cooling. Nevertheless, as low as their consumption will be, they will still consume power. An alternative solution to further reduce their impact is to use green sources such as solar panels, wind turbines, or fuel cells as these devices do not produce CO_2. The challenge of this work is to consider computing facilities which are solely run by renewable energy sources.

Efficiently powering a computing or data center means to deliver the correct energy level to each of the center components. Renewable energy sources however provide a variable energy provisioning depending on solar and wind conditions so that they must be completed by batteries or other energy storage systems to guarantee a continuous work. In this problem, the most central part is taken by the computers. If they do not have enough power to run tasks then it is useless to supply energy to the rest of the center. On the other hand at some points of the power supplier life-cycle the energy storage components become full and there is no need to spare energy that should rather be consumed than lost. For these reasons, we concentrate in this work on processing tasks depending on the available power.

This article tackles the problem that we can formally solve in scheduling with green energy optimization. Our approach is different from traditional energy aware scheduling approaches in that it does not target energy minimization itself but it rather targets not to use energy sources other than renewable energy. The optimization problem is thus rather to limit the energy waste, the produced energy that cannot be used, than finding ways to decrease the consumed energy. We tackle on the one hand computing center oriented problems where the optimization objective is the makespan to finish sets of jobs as soon as possible and, on the other hand, data center oriented problems where the optimization objective is the flowtime to reduce the mean waiting time. To concentrate on the scheduling problem, this first work considers scheduling a set of tasks on shared memory machine as, in that case, we do not need to take machine power on/off into account for our optimizations.

The presented contributions are as follows:

- We provide formal results on the complexity of three out of four scheduling problems. We show that most power constraint scheduling problems are complex. The results are proven for one machine problems and can be extended to more general parallel problems.
- Second, a simulation based performance study of several heuristics proposed to efficiently solve the problem. These simulations show that depending on the weight of the processing time compared to the power need of the tasks the best algorithm

The paper is organized as follows: related work is presented in Section II, then a system model is proposed in Section III. We present formal results on the complexity of the related optimization problems in IV and propose several heuristics to solve the considered problems in Section V. Experiments to assess the heuristic performance are presented in Section VI and we conclude in Section VII.

II. RELATED WORK

Energy saving is a major concern in the computing domain and there exists a large variety of research that tackles the problem of reducing the energy consumption. Several surveys

give an interesting overview of the research done in the field of green computing: i.e., limiting the energy consumption of computing or data center. For instance [1]–[3] give a wide survey on all the technologies and tools that can be used in data centuries to lower the energy consumption.

One possible technique for energy reduction is the usage of Dynamic Voltage and Frequency Scaling (DVFS), which allows to run processors and servers at a lower speed at the price of increased execution times for tasks. Wu et al. [4] use this technique to reduce the energy consumption in Cloud data centers. They use a two step approach to allocate tasks to servers and then determine the voltage/frequency combination to run the servers. Kim et al. [5] use DVFS to provision virtual machines for real-time Cloud services. Garg et al. [6] include Dynamic Voltage Scaling (DVS) in their scheduling algorithm for HPC applications on Cloud-oriented data centers. Results show that their solution allows to reduce up to 25% the energy usage in comparison to profit based scheduling policies with even higher profit. It seems however that the DVFS tuning tends to be more and more embedded directly inside the processor with less control actions leave to the user as explained in [7].

Another approach to reduce the energy consumption is the consideration of a shutdown model, where processors have an additional state to on and off, which is called sleep state. In a one machine offline setting with jobs of unit processing times, release dates, and deadlines, a dynamic programming approach allows to minimize the number of idle time periods in polynomial time [8]. In follow up work of Baptiste et al. [9], [10] the result is extended to heterogeneous preemptive jobs. Albers and Antoniadis [11] combine speed scaling with the sleep state model. They prove NP-completeness of the energy minimization problem for heterogeneous tasks with release dates and deadlines.

Often works on energy efficiency imply a bi-criteria objective. Indeed, minimizing the energy consumption in any setting without another complementary objective simply leads to stop all executions, shut down all devices and have zero energy consumption. Beloglazov et al. [12] for example propose energy efficient solutions for Cloud data centers via virtual machine migration while respecting quality of service (QoS) requirements. They ensure quality of service by respecting service level agreements (SLA) which can be minimal ensured throughput, maximal response time or latency and other.

In [13] the authors tackle the theoretical complexity of energy-efficient scheduling algorithms. They schedule independent tasks on parallel identical and uniform machines and give optimal solutions for divisible loads minimizing the makespan. They also show that the non-divisible case of tasks is NP-hard. The optimization criteria is the makespan and they optimize in one step the makespan and the consumed energy by considering the energy consumed by the machine when they are idle. They however have no constraint on power and can always use the available machines.

In recent years an other trend has arisen: Data and Computing centers integrate renewable energy sources as power supply. An early work on green energy utilization in data centers by Aksanli et al. [14] shows the importance of power prediction. They propose an adaptive data center job scheduler that reduces the number of aborted jobs while improving the green energy utilization.

In [15], [16] the authors propose GreenSlot, a batch scheduler for parallel tasks, that aims to reduce the brown power consumption of a data center which is partially powered by solar panels. In GreenSlot the jobs have deadlines and the scheduler first reserves resources for the jobs with lower slack (distance from latest possible start time to current time). Based on weather forecasting and power prediction GreenSlot schedules the tasks on time slots. However the authors do not try to optimize their schedules just reduce the consumption and costs while meeting as much deadlines as possible.

Similarly [17] presents an holistic approach where the energy cost that includes incomes from running batch jobs and outcomes to buy brown energy is optimized. The paper also provides a proposition for net zero scheduling batch jobs. It is however based on virtualization and is not bounded by the number of resources.

Wang et al. [18] propose a green-aware virtual machine migration method for data centers. They consider data centers which are powered by hybrid energy, i.e., they have access to renewable energy sources as well as grid energy. Their approach uses a heuristic approach and stochastic search to maximize the profit of the data center while minimizing the operational cost and energy.

However none of the afore mentioned work deals with data centers and computing facilities that are 100% provisioned with renewable energy. In this work we condiser computing facilities that are solely run by green energy sources. Therefore the avaiable power is constraint at any moment by the actual power production and our scheduling solutions have to cope with the available power envelope. We thus tackle the classical mono criteria optimization objectives makespan and flow time under power constraints.

III. Model

The first step of the problem study is to define a model for the considered system. In this section we define the models of machines, power and tasks that we use in the addressed optimization problem.

The considered computing platform is parallel which means that several execution units are available to process the tasks. Parallelism can be achieved thanks to either cores (shared memory parallelism) or nodes (distributed memory parallelism). In the model we do not differentiate the parallelism type and the platform consists of a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of m machines M_j that represent execution units (cores) either on the same machine or not.

As the power provisioning of the platform solely comes from green energy sources such as solar panels or wind turbines, its production is not stable and varies over time. The evolution of the available power is represented at each time t by a curve $\Phi^{available}(t)$. To be able to optimize the usage of the power, this power production must be approximated. We assume that the available power $\Phi^{available}(t)$ is a constant value $\Phi_x^{available}$ over an interval of time Δ_x. Over the considered period, the available power is thus modeled by X given intervals Δ_x

whose length is δ_x over a given time horizon $\mathcal{H}$, such that $\sum_{x=1}^{X} \delta_x = \mathcal{H}$. This power is shared by all the machines of the platform.

Each machine M_j consumes a static power Φ_j^{stat}, i.e., the minimum power that this machine needs when it is powered on but does not process any tasks. Since this static power has a constant value for the entire considered horizon of time, and since it is useless to run a machine without processing tasks, we only consider useful available power $\Phi_x = \Phi_x^{available} - \sum_{j=0}^{\mathcal{M}} \Phi_j^{stat}$ for the period of time Δ_x in the following. If the platform consists of only one machine, $\Phi(t) = \Phi_x$.

For the task model we relay on the usual definitions: We consider a set $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ of n tasks T_i which are characterized by their processing time p_i. These tasks are sequential independent tasks. Running a task on one machine generates an extra power consumption [19]. This power consumption varies over time depending whether the task intensively computes or accesses input/output devices and it has to be approximated to be used in an optimization problem. We assume here that each task T_i has a constant power demand over its lifetime which is equals to its largest power need φ_i. By taking the larger power consumption of the task, we guarantee that the resulting schedule will fit in the power envelop. Other more accurate models could be used, as to cut the different consumption phases of the tasks in time periods for instance, but this complicates the solving of the problem.

To schedule a given task T_i we need to guarantee that T_i can be completed before the available power becomes lower than its need. To be able to exhibit such time slots we define the set $\mathcal{E}_j(\varphi_i) = \{\mathcal{E}_{1,i}, \mathcal{E}_{2,i}, \ldots, \mathcal{E}_{K_i,i}\}$ of K_i eligible time slots. Let $b_{k,i}$ and $f_{k,i}$ be respectively the beginning of the slot $\mathcal{E}_{k,i}$ and its finish time. Then, for $\mathcal{E}_{k,i} = [b_{k,i}, f_{k,i}[$, the available power must be greater than φ_i, with $b_{k,i} \leq t < f_{k,i}$ and $\Phi_k(t) \geq \varphi_i$. Formally, it exists two integer values x and s such that the kth time slot $\mathcal{E}_{k,i}$ is defined by $\mathcal{E}_{k,i} = \Delta_x \cup \Delta_{x+1} \cup \ldots \cup \Delta_{x+s}$ $(x + s \leq X)$ where at any time $t \in \mathcal{E}_{k,i}$, $\Phi(t) \geq \varphi_i$ and at any time $t \in \Delta_{x-1}$ $(x > 1)$ or $t \in \Delta_{x+s+1}$ $(x + s + 1 \leq X)$ $\Phi(t) < \varphi_i$. So $b_{k,i} = \sum_{x'=1}^{x-1} \delta_{x'}$ and $f_{k,i} = \sum_{x'=x+s}^{X} \delta_{x'}$ (see Figure 1). If, considering already scheduled tasks, it remains enough time to perform T_i in the duration $l_{k,i} = f_{k,i} - b_{k,i}$ that time slot is an option to run T_i. When a time slot is chosen to schedule a task, the corresponding power is substracted from available power in the intervals that compose this time slot.

Finally, we consider an allocation function $A(i, j) = k$ that returns in which time interval $\mathcal{E}_{k,i}$ the task T_i is scheduled on machine M_j. Let $\mathcal{T}_{k,j}$ be a subset of task set $\mathcal{T}$ that contains the tasks scheduled in the time slot $\mathcal{E}_{k,i}$ on M_j. For every task $T_i \in \mathcal{T}_{k,j}$, we set $A(i, j) = k$. Note that $\sum_{i|T_i \in \mathcal{T}_{k,j}} p_i \leq f_{k,i} - b_{k,i} = l_{k,i}$.

Note that, in order to make the reading more understandable, the index j is removed from previous notations for the one-machine problems, i.e., $\mathcal{E}_{k,j}$ becomes $\mathcal{E}_k$. Table I summarizes the notations used in the remainder of the paper.

IV. Optimization Problems

Using the preceding model we consider static optimization problems where the number, the consumption, the duration of the tasks and the available power are known in advance. Note that the model defined in Section III covers a larger set of optimization problems than we tackle here. Static problems are sometimes far from real practical cases and their solutions do not always give interesting results for real life. Tackling such static problems is however necessary to prove the complexity of the optimization problem in the simpler cases and then deduce the complexity of more complex ones.

To better define and characterize the tackled problems, we first extend the Graham notation in this section. Then we assess the complexity of basic problems to later proof the complexity of more general ones.

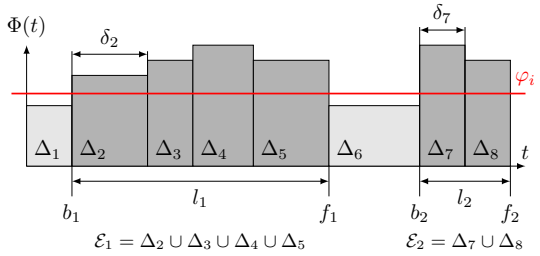


Figure 1: Illustrative example for intervals $(\Delta_1, \ldots, \Delta_8)$, available power on time and time slots $(\mathcal{E}_2, \mathcal{E}_2)$ in which one task T_i could be scheduled when its power need is φ_i onto a one machine platform.

Table I: Summary of the notations

variable	definition
$\mathcal{T}$	set of tasks
T_i	task i
n	number of tasks
p_i	processing time of T_i
φ_i	power needed by T_i
$\mathcal{M}$	set of machines
M_j	jth machine of $\mathcal{M}$
m	number of machines
Φ_x	useful power on M_j at interval Δ_x
Δ_x	interval with constant power
δ_x	length of Δ_x
X	number of consecutive intervals Δ_x
$\mathcal{E}_j(\varphi_i)$	time slot: set of eligible time intervals
$\mathcal{E}_{k,i}$	kth eligible interval of $\mathcal{E}_j(\varphi_i)$
$b_{k,i}$	beginning of time slot $\mathcal{E}_{k,i}$
$f_{k,i}$	finish time of time slot $\mathcal{E}_{k,i}$
K	number of time slots in $\mathcal{E}_j(\varphi_i)$
$l_{k,i}$	length of time slot $\mathcal{E}_{k,i}$

A. Notations and objective

Graham [20] defined the $\alpha|\beta|\gamma$ notation that characterizes a scheduling optimization problem. We shortly recall here the main notations used in parallel scheduling optimization problems. In this notation the α value stands for the characteristics of the execution platform: 1 for one machine, P for parallel identical machines, Q for parallel uniform machines and R for parallel unrelated machines. The β value stands for the tasks characteristics and/or constraints: $p_i = p$ is set when all the tasks are of the same size, $prec$ when there are precedence constraints between the tasks, $pmtn$ if the tasks can

be preempted, etc. The γ value gives the criteria to be optimized as, for instance the makespan $C_{\max}$, the maximum tardiness $T_{\max}$ or the flowtime, $\sum C_i$, where C_i is the completion time of task T_i.

In this paper we consider the problem of scheduling tasks under limited available power. To express this constraint we propose to add $\varphi_i \leq \Phi_k$ for one machine problems and $\sum \varphi_i \leq \Phi_k$ for parallel machine problems to the Graham notation, with φ_i is the power needed by a task. This enforces that the power needed by one (φ_i) or several tasks ($\sum \varphi_i$) must be lower than the power provided by the energy sources. For example the problem $1|\varphi_i \leq \Phi_k|C_{max}$ is a one machine problem where we target makespan optimization for independent tasks. If the Φ_k variable is set to Φ then the available power is constant over the considered period and if the φ_i variable is set to φ then all the tasks need the same power to run.

Considering computing and data centers, two main criteria can be minimized to improve the efficiency. The makespan ($C_{\max}$) is a classical criterion that targets the minimization of the running time for a set of tasks. This criterion is relevant for computing centers where applications are composed of a set of tasks. In this case there is no need to finish one task or another earlier, the user just wants its task set to be finished as soon as possible. In the case of several tasks launched by different users then the flowtime ($\sum C_i$) is more relevant as minimizing this criterion leads to minimizing the mean finish time. It enforces a fair share of the resources between users. Minimizing this criterion is thus more useful in data centers where incoming user requests have to be processed.

In the following we first tackle the one machine problems. Showing that these problems are NP-Complete proves that the more general parallel problems are NP-Complete as well.

B. One Machine Problems

We consider here one machine problems for both objectives makespan and flowtime and for the cases with or without preemption. These cases are simple when there is no power constraint. We recall that all no delay schedules (i.e., schedules without delay between the tasks) are optimal solutions for the makespan objective and that the Shortest Processing Time (SPT) algorithm gives optimal solutions for the flowtime objective. We show here that, with power constraints, these problems are polynomial in the case of identical tasks (i.e., tasks with same p_i) and that the problems where tasks have different processing times are actually NP-Complete if preemption is not allowed.

In the one machine problems the static power Φ_j^{static} consumed by the machine $M_j \in \mathcal{M}$ is constant over time. Then the machine needs at least this amount of power to run and it cannot run at any period of time t where the power provided by the sources is lower than this value. If there is no machine running there is no scheduling problem as no task can be run. For that reason we assume that the static power needed by the machine is at least available in each interval and there is no need to take the static power consumption in consideration in these problems. In the remainder of the paper we set that the available power in each period of time is only the useful power for running tasks and we note $\Phi_x = \Phi_x^{available}$

Note that, when the available power $\Phi_x = \Phi$ is constant over the considered time horizon, the problem is simple. In this case task, that need more power than Φ to be performed cannot be processed by the machine. For the remaining tasks the optimization problem are the same as the one machine optimization problem without power constraint. For that reason in the remainder of the paper the only Φ_x case is considered. Note that using DVFS may help to schedule over consuming tasks. In that case reducing the voltage of the processor will limit the task consumption and may lead to being able the run the task even with limited available power. The resulting problem in this case is the same with different values of p_i for the tasks.

We now consider different cases for the task processing time and the objective function.

1) Problems without preemption: In computing centers the nodes are usually dedicated to the users and no preemption is applied to the tasks when they are running, both for efficiency and synchronous reasons. We assess here the complexity of the scheduling problem in this context.

a) Identical tasks $p = p_i$ and $\varphi_i \leq \Phi$: The most simple problems for our two objectives, flowtime and makespan, are the cases where every task has the same computing time $p_i = p$. In both cases, to optimize our objective, we just have to put as many tasks as possible in each time slot, starting with the tasks with the largest power need φ_i. Obviously this solution is optimal for the makespan objective as every task is interchangeable with another changing the order will not give a better solution and we do not leave empty places where a task can be put. For the flowtime, as every task has the same processing time, none of them has a larger weight in the final sum and there is no need to order the tasks in a specific manner.

b) Non-identical tasks: In the case of non-identical tasks problems, denoted respectively $1|\varphi_i \leq \Phi|C_{\max}$ for the makespan optimization and $1|\varphi_i \leq \Phi|\sum C_i$ for the flowtime optimization, are NP-Complete. In the following, these complexity results are proven in a row.

Theorem 1: Minimizing the makespan of the schedule of a set of tasks ($1|\varphi_i \leq \Phi|C_{\max}$) to run in a set of intervals is NP-Complete in the strong sense if the tasks have different processing times p_i.

Proof: First note that, in the case where all tasks need the same power to run $\varphi_i = \varphi$, a time interval Δ_x either provides enough power to run a task or not. The real amount of power provided during this interval is not important as it is just a binary question of enough power or not. The NP-Completeness of the makespan minimization problem will be demonstrated by proving first the problem where each task needs a power φ ($1|\varphi_i = \varphi \leq \Phi|C_{\max}$) to be executed and where the set $\mathcal{E}(\varphi)$ defines time slots in which it is possible to schedule tasks.

Let us consider the following decision problem: given a time Z, is there a schedule where the last task is completed before Z ? We assume that the allocation respects the constraints of the problem such that every task allocated to one time slot has enough time to be completed before the end of this time slot and the power available into this time slot is greater or equal

to the sum of power needed by the tasks scheduled in the time slot.

The problem is in NP: given a schedule it is easy to check in polynomial time whether it is valid or not before the time Z. The NP-Completeness is obtained by reduction from 3-PARTITION [21] which is NP-Complete in the strong sense.

Let us consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3H$ positive integers $a_1, a_2, \ldots, a_{3H}$ such that for all $i \in \{1, \ldots, 3H\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^{3H} a_i = HB$, does it exist a partition $I_1, \ldots, I_H$ of $\{1, \ldots, 3H\}$ such that for all $h \in \{1, \ldots, H\}$, $|I_h| = 3$ and $\sum_{i \in I_h} a_i = B$?

We build the following instance $\mathcal{I}_2$ of our problem (see Section III) with $\mathcal{E}(\varphi) = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_H\}$ the set of qualified time slots $\mathcal{E}_h$ to run tasks (i.e., the available power is greater than φ) and whose length are all equals to $f_h - b_h = l_h = l = B$. There are $3H$ tasks $T_i \in \mathcal{T}$ such that each T_i needs a power of φ to be executed and its processing time is $p_i = a_i$ for all $1 \leq i \leq 3H = n$. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. For $1 \leq h \leq H$, task T_i is assigned to time slot $\mathcal{E}_h = [b_h, f_h[$ with $i \in I_h$ within the period and $p_i = a_i$. Then, we have $\sum_{i|A(i)=h} p_i = l = \sum_{i \in I_h} a_i = B$ and therefore the constraint on the processing time is respected for the H slots. We have a solution to $\mathcal{I}_2$.

Suppose that $\mathcal{I}_2$ has a solution. Let $\mathcal{T}_h$ be the set of tasks allocated to the slot $\mathcal{E}_h$ (We recall that if $T_i \in \mathcal{T}_h$, $A(i) = h$) such that for all tasks $T_i \in \mathcal{T}_h$ with $i \in I_h$, $\sum_{i \in I_h} p_i = l = B$. Because of $p_i = a_i$, $|\mathcal{T}_h| = |I_h| = 3$. The length of the time slot l in which the available power is φ has to be fully filled for all H periods to be sure to complete the last task within the slot $\mathcal{E}_H = [b_H, f_H[$ at time $t = f_H = Z$. Otherwise, an other slot has to be used to complete unprocessed tasks. Thus the solution is a 3-PARTITION.

We have proven that minimizing the makespan $C_{\max}$ of scheduling a set of tasks with different processing time which need the same amount of power φ to be performed on one machine is NP-Complete in the strong sense.

Since this problem $1|\varphi_i = \varphi \leq \Phi|C_{\max}$ is a special case of $1|\varphi_i \leq \Phi|C_{\max}$, it is sufficient to prove the NP-Completeness of $1|\varphi_i \leq \Phi|C_{\max}$.

This concludes the proof. ■

Theorem 2: Optimizing the flowtime of the schedule of a set of tasks ($1|\varphi_i \leq \Phi|\sum C_i$) to run in a set of intervals is NP-Complete in the strong sense if the tasks have different processing times p_i.

Proof: Let us consider the following decision problem: given a time Z is there a schedule where the sum of the task completion times is less than Z ? We assume that the allocation respects the constraints of the problem.

The problem is in NP: given a schedule it is possible to confirm in polynomial time whether this schedule is valid or not and the sum of the task completion times is less than Z. The NP-Completeness is obtained by reduction from the $1|\varphi_i = \varphi \leq \Phi|C_{\max}$ problem that is proven NP-Complete in the strong sense in Theorem 1.

Let us consider an instance $\mathcal{I}_1$ of $1|\varphi_i = \varphi \leq \Phi|C_{\max}$ described within the paper: given $\mathcal{E}(\varphi) = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_H\}$ the set of H qualified time slots $\mathcal{E}_h$ to run tasks and whose length are all equal to $f_h - b_h = l_h = l = B$ ($1 \leq h \leq H$) and given $3H$ tasks $T_i \in \mathcal{T}$ such that each T_i needs the same power φ to be executed and its processing time is p_i for all $1 \leq i \leq 3H = n$ such that for all $i \in \{1, \ldots, 3H\}$, $B/4 < p_i < B/2$ and with $\sum_{i=1}^{3H} p_i = HB$. Does there exist a schedule $\mathcal{T}_1, \ldots, \mathcal{T}_H$ such that, for all $h \in \{1, \ldots, H\}$ and for all $T_i \in \mathcal{T}_h$, T_i is scheduled in $\mathcal{E}_h$ ($A(i) = h$) and $C_{max} = f_H$? Obviously, $|\mathcal{T}_h| = 3$ with $1 \leq h \leq H$ considering p_i.

We build the following instance $\mathcal{I}_2$ of the problem addressed in the beginning of the proof: $1|\varphi_i = \varphi \leq \Phi|\sum C_i$ with the set $\mathcal{E}'(\varphi) = \mathcal{E}(\varphi) \cup \mathcal{E}_{H+1}$ of $H+1$ qualified time slots ($\mathcal{E}$ described for $\mathcal{I}_1$), the same set $\mathcal{T}$ of $3H = n$ tasks T_i with $1 \leq i \leq n = 3H$. $\mathcal{E}_{H+1}$ is defined as a valid time slot ($\varphi \leq \Phi(t)$ with $b_{H+1} \leq t < f_{H+1}$) such that $b_{H+1} = n \times f_H$. Considering this problem instance, does there exist a schedule with $Z = n \times f_H$?

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. Let us show now that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. For $1 \leq h \leq H$, task T_i is assigned to time slot $\mathcal{E}_h = [b_h, f_h[$ if $T_i \in \mathcal{T}_h$. Then, we have $\sum_{i|T_i \in \mathcal{T}_h} p_i = l = B$ and therefore the constraint on the processing time is respected for the H slots and $|\mathcal{T}_h| = 3$. Considering the schedule given by $\mathcal{I}_1$, it is possible to minimize the flowtime within $\mathcal{E}_h$ ($F_h = \sum_{i|T_i \in \mathcal{T}_h} (C_i - b_h)$ with C_i the completion time of T_i) by sorting the 3 tasks by increasing processing time order. Then each time slot $\mathcal{E}_h$ has its own flowtime F_h. As $f_h - b_h = l_h = l = B$ for all $1 \leq h \leq H$, it is possible to exchange task allocations from one time slot $\mathcal{E}_{h1}$ to another time slot $\mathcal{E}_{h2}$ ($h1 \neq h2$ and $1 \leq h1, h2 \leq H$) without changing the value of the makespan. Consequently, by sorting F_h in increasing order and by reallocating tasks $T_i \in \mathcal{T}_h$ to the right time slot regarding its rank given by the sort, the obtained flowtime for the whole task set is the smallest possible. We have a solution to $\mathcal{I}_2$.

Suppose now $\mathcal{I}_2$ has a solution. If the flowtime of the schedule is less than $Z = n \times f_n$, $\mathcal{T}_{H+1} = \emptyset$, otherwise since $b_{H+1} = n \times f_n$, if one task T_i is in $\mathcal{T}_{H+1}$, the flowtime is not able to be less than $n \times f_n$ because the completion time of T_i is at least $C_i = b_{H+1} + p_i = n \times f_n + p_i$ which is greater than Z. Thus, all tasks are scheduled within $\mathcal{E}$. Since $\sum_{i|T_i \in \mathcal{T}} p_i = HB$ and since $f_h - b_h = l = B$ for all $1 \leq h \leq H$ and $|\mathcal{E}| = H$, the completion time of the last task is $f_H = C_{max}$. We have a solution to $\mathcal{I}_1$.

By using the same valid arguments than within the proof of Theorem 1, we can confirm that we have proven that minimizing the flowtime of scheduling a set of tasks with different processing times which need the same amount of power φ to be performed on one machine is NP-Complete in the strong sense.

Since this problem $1|\varphi_i = \varphi \leq \Phi|\sum C_i$ is a special case of $1|\varphi_i \leq \Phi|\sum C_i$, it is sufficient to prove the NP-Completeness of $1|\varphi_i \leq \Phi|\sum C_i$.

This concludes the proof.

■

2) Problems with preemption: In the special case of data centers the requests that are processed are not as constrained as parallel tasks in a computing center and they can thus be preempted. We consider here the impact of preemption on the scheduling problem complexity for both makespan and flowtime objectives.

We do not present the special case for identical tasks as we show that the more general problem with different processing time is polynomial.

The $1|\varphi \leq \Phi, pmtn|C_{\max}$ problem, the problem where all tasks need the same power to run, accepts a polynomial solution. Remember that without power constraint non delay schedules are optimal. With power constraints it is however not possible to always have non delay schedules as some of the intervals Δ_x may not provide enough power Φ_x to schedule a task.

The general idea is to avoid leaving intervals empty when there are still unscheduled tasks. For this purpose we schedule tasks with the following policy: At the beginning of a new interval or when a task is finished, we schedule the task (or the remaining part of a task) which wastes the less power ($\min(\Phi_x - \varphi_i)$) next. If another task than the current running task is selected, the running task is preempted and rescheduled later. We call this algorithm Less Wasting Remaining Task (LWRT).

Theorem 3: Algorithm LWRT gives an optimal solution for the $1|\varphi_i \leq \Phi, pmtn|C_{\max}$ problem.

Proof: The optimality of the LWRT algorithm is demonstrated by contradiction.

Let C^* be the optimal makespan. In the optimal schedule S^* we do not have a guarantee that at each interval, starting from $t = 0$ we always run the LWRT task (the task T_i that minimizes $\Phi_x - \varphi_i$ in the set of tasks that are scheduled from s_i, the start time of T_i). Let assume that interval x is the first interval such that it includes task T_i ($S^*(T_i) = t$) which is not the LWRT task and such that T_i', the LWRT task, runs later ($S^*(T_i') = t', t' > t$). As T_i is not the LWRT task then we have $\Phi_x - \varphi_i > \Phi_x - \varphi_i'$ and $\varphi_i < \varphi_i' \leq \Phi_{k_1}$. Since the power consumed by T_i' is higher than the power consumed by T_i and since T_i' fits in interval Δ_x because it is the LWRT tasks for this interval then we can swap T_i and T_i' (or at least part of them). Moreover, since T_i needs less power than T_i' it could be scheduled before t' in an interval that was not exploited by T_i' which more power. After this step the resulting schedule it as least the same but it could also have been improved by moving T_i. This result is a contradiction with the assumption that S^* is optimal and given any schedule we can do better if we respect the LWRT order. This implies that the LWRT algorithm gives an optimal schedule which concludes the proof. ■

Figures 2 and 3 illustrate the case where the LWRT task is not scheduled at each interval change or task end. On Figure 2 task T_2 is not preempted at the end of interval Δ_2. As a result task T_4 is scheduled later because of its large power need and interval Δ_5 is not used. On Figure 3 task T_2 is preempted at the end of interval Δ_2 and Task T_4 is executed instead. As Task T_2 needs less power to run it can be executed in interval Δ_5 which improves the makespan.

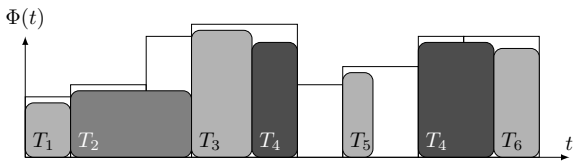


Figure 2: Illustrating example for the LWRT algorithm, T_2 is not the LWRT task for interval Δ_3, T_4 must be run here.

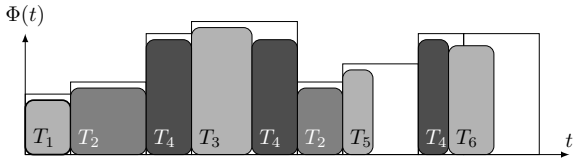


Figure 3: Illustrating example for the LWRT algorithm, part of T_4 has been swapped with T_2 which can be executed sooner than T_5, the makespan is optimal.

We now consider the $1|\varphi_i \leq \Phi, pmtn|\sum C_i$ problem. The flowtime objective is more tricky to solve than the makespan as we must take the order of the tasks into account. We recall that, without power constraint, the SPT (Shortest Processing Time) algorithm gives the optimal schedule. This algorithm also solves the $1|\varphi = \Phi, pmtn|\sum C_i$ problem as we can use preemption to stop a task at the end of an interval, if the available power is too low in the next interval, then resume it when the power is again above the φ power need. In the general case however, this algorithm does not work as we can have to schedule longer tasks before short ones do to the power constraints as shown in Figure **??**. The tradeoff is hence to balance between the need to schedule short tasks at the beginning as they will more impact the flowtime and the risk of scheduling tasks with higher power needs in distant time slots. We do not have any proof for the moment but we suspect this problem to be NP-Complete. The complexity of this problem is hence open.

C. Parallel Problems

We consider here the problem of scheduling a set of tasks on a set of machines. Considering problems with power constraints several sub-problems can be identified from the general parallel problem aside from the classical P, Q, R cases. Shared memory problems are indeed different from distributed memory problems. In the shared memory problems only one machine is used and the tasks are processed by the different cores of the machine (one task per core). As just one machine must be powered on, we do not need to take static power into consideration as when there is not enough power to run the machine there is no problem. The remaining power is dedicated to the task's execution. The considered problem is then a parallel machine problem where we only take the task power consumption into account. Note also that, in this case, the machines are cores and we can limit the study to identical machines (P in the Graham notation) as the cores of a same node are usually identical. In the distributed memory problems several machines are used and, when have less tasks than the

number of machines we have to take a shutdown model into account.

to be removed for the paper too risky

Note that we do not consider the power consumed by cores independently contrary to that cores could be activated or not on next generation processors. This problem could be related to the Agrawal model [13] where machines are not stopped and have a working consumption $\mu(C_i)$ and an idle consumption $\gamma(C_i)$. However on the one hand the model where a core always consumes the same power when active is not realistic and on the other hand in our case all the cores have the same $\mu(C_i)$ and $\gamma(C_i)$. In this case the available power Φ_k is shared between the cores and the power constraint becomes that the sum of the power φ_i of the running tasks must be less than Φ_k.

From the previous complexity results we can deduce that $P|\sum \varphi_i \leq \Phi|C_{\max}$ and $P|\sum \varphi_i \leq \Phi|\sum C_i$ problems are NP-Complete as parallel problems are generalizations of one machine problems. Problems with preemption must however be investigated.

For the $P|\sum \varphi_i \leq \Phi, pmtn|C_{\max}$ problem we have to schedule several tasks at the same time such that the sum of their power needs $\sum \varphi_i$ is lower than the available power Φ_k in each interval.

If the power needed by the tasks is the same, the $P|\sum \varphi_i \leq \Phi, \varphi_i = \varphi, pmtn|C_{\max}$ problem, then the problem is simple: in a given interval execute as much as possible tasks at the same time provided that the power Φ_k and number of cores P constraints are respected. Then, at the end of a task, schedule another one and, at the end of the interval, schedule less tasks if there is less power and schedule more if there is an idle core and more power.

If the power needed by the tasks is different, the $P|\sum \varphi_i \leq \Phi, pmtn|C_{\max}$ problem, then the problem is NP-Complete.

Theorem 4: Minimizing the makespan of the schedule of a set of tasks that do not consume the same power to run in set intervals ($P|\sum \varphi_i \leq \Phi, pmtn|C_{\max}$) is NP-Complete in the strong sense if the tasks can be preempted.

Proof: The NP-Completeness of this problem will be demonstrated by proving that the special case where the processing time of each task is 1 unit of time (ut), is NP-hard in the strong sense. The remainder of the proof is build on a similar pattern than used within the proof of the theorem 1.

Let us consider the following decision problem: given a horizon of K intervals of time Δ_k ($1 \leq k \leq K$) where their length δ_k is equals to 1 unit of time and where the available power is $\Phi(t) = \Phi_k = \Phi$ ($1 \leq k \leq K$) and given a processor with 3 cores that share the available power, is there a schedule that allocates tasks over time such that the power needed by the cores never exceeds φ for every time intervals Δ_k ($1 \leq k \leq K$)? In other words, if $\mathcal{T}_k \subset \mathcal{T}$ is the set of tasks that are scheduled within the time interval Δ_k, $\forall k \leq K$, is $\sum_{i|T_i \in \mathcal{T}_k} \varphi_i \leq \Phi_k = \Phi$? The problem is in NP: given a schedule of K time intervals, it is easy to check in polynomial time whether this schedule is valid or not. The NP-Completeness is obtained by reduction from 3-PARTITION [21] which is NP-Complete in the strong sense.

Let us consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3K$ positive integers $a_1, a_2, \ldots, a_{3K}$ such that for all $i \in \{1, \ldots, 3K\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^{K} a_i = KB$, does exist a partition $I_1, \ldots, I_K$ of $\{1, \ldots, 3K\}$ such that for all $k \in \{1, \ldots, K\}$, $|I_k| = 3$ and $\sum_{i \in I_k} a_i = B$?

We build the following instance $\mathcal{I}_2$ of our problem with K time intervals, each interval Δ_k having a length of time $\delta_k = 1$ and with an available power $\Phi_k = \Phi = B$ for $1 \leq k \leq K$. There are $3K$ tasks T_i in $\mathcal{T}$ with $p_i = 1\,ut$ and $\varphi_i = a_i$ for all $1 \leq i \leq 3K = m$. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. For $1 \leq k \leq K$, task T_i is assigned to $\mathcal{T}_k$ within the period k with $i \in I_k$ and $\varphi_i = a_i$. Then, we have $\sum_{i|T_i \in \mathcal{T}_k} \varphi_i = \phi_k = \sum_{i \in I_k} a_i = B$ and therefore the constraint on the demand is respected for the K time intervals. We have a solution to $\mathcal{I}_2$.

Suppose that $\mathcal{I}_2$ has a solution. Let $\mathcal{T}_k$ be the set of machines allocated to the period k such that for all tasks $T_i \in \mathcal{T}_k$ with $i \in I_k$, $\sum_{i \in I_k} \varphi_i = \Phi_k = \Phi = B$. Because of φ_i, $|\mathcal{T}_\parallel| = |I_k| = 3$. Since the available power Φ has to be consumed for the K time intervals to process the scheduled tasks, the solution is a 3-PARTITION.

We have proven that the problem where $\Phi(t) = \Phi_k = \Phi$ for every time interval Δ_k ($1 \leq k \leq K$) and $p_i = 1$ for every Task $T_i \in \mathcal{T}$ ($1 \leq i \leq n$) is NP-Complete in the strong sense. Since this problem is a special case of the more general problem where available power Φ_k during each time intervals Δ_k is different from each other and where processing time p_i of each task T_i is also different from each other, it is sufficient to prove the NP-Completeness of this general problem. This concludes the proof. ■

Note that the proof highlight that the problem is NP-Complete even if the tasks are of the same size, $p_i = p$ ($P|\sum \varphi_i \leq \Phi, p_i = p, pmtn|C_{\max}$ problem).

For the flowtime objective, the $P|\sum \varphi_i \leq \Phi, pmtn|\sum C_i$ problem, we can differentiate the particular case where tasks have the same power need $\varphi_i = \varphi$ which is simple from the more general case where tasks have different power needs. In the $\varphi_i = \varphi$ case the SPT algorithm, completed to take the available power Φ_k and the number of cores P constraints into account, gives an optimal solution even if the tasks are of different size. Then the case where the tasks have different power needs is NP-Complete as the problem $P|\sum \varphi_i \leq \Phi, p_i = p, pmtn|\sum C_i$ is equivalent to $P|\sum \varphi_i \leq \Phi, pmtn|C_{\max}$ since the tasks do not need to be ordered as they are of the same size. This implies that the more general case $P|\sum \varphi_i \leq \Phi, pmtn|\sum C_i$ is NP-Complete.

In the following parts we propose and assess solutions for some of the NP-Complete problems.

V. Heuristics

In this part we propose scheduling algorithms to optimize tasks execution under power constraints. As for the previous section we concentrate here on multi-core parallelism as distributed parallelism implies to take a shutdown model into

account. As the cores of a same node are usually identical it is more logical to concentrate on identical machine (P) problems.

We propose here heuristic algorithms to solve the general problems $P|\varphi \leq \Phi|C_{\max}$ and $P|\varphi \leq \Phi|\sum C_i$. Most of them are adapted from classical scheduling algorithms to introduce the power constraints and time slots. Remember that time slots are sets of consecutive intervals (see III). A time slot is considered available for scheduling a task if, in all the intervals that compose this time slot, the available power is higher than the task power need and there is at least one available core. The task to time slot matching is done by the $Place\_Task()$ function that is used by all the algorithms. $Place\_Task()$ function is illustrated in Algorithm 1.

Algorithm 1: $Place\_Task(p_i, \varphi_i, x)$

```
   Data:
     Φ_x: useful power at Δ_x
     δ_x: length of Δ_x
     s_x: start time of Δ_x
     e_x: end time of Δ_x
     nc_x: number of available cores in Δ_x
   Result:
     exstart_i: execution start time of T_i
1  if (φ_i ⩽ Φ_x) ∧ (nc_x > 0) then
2      if p_i ⩽ δ_x then
3          Remove Δ_x from the interval list
4          Add new intervals in the interval list:
5          Δ_x' = {[s_x, s_x + p_i], Φ_x − φ_i, nc_x-1}
6          Δ_x'' = {[s_x + p_i, e_x], Φ_x, nc_x}
7          return s_x
8      else
9          found ← true
10         repeat
11             Take next Δ_x
12             if (Φ_x < φ_i) ∧ (nc_x > 0) then
13                 found ← false
14         until Σ δ_x ≥ p_i
15         if found then
16             for Intervals Δ_x in the time slot except the last one
               do
17                 Δ_x = {[s_x, end_x], Φ_x − φ_i, nc_x-1}
18             Remove the last interval Δ_y from interval list
19             Add new intervals in the interval list:
20             Δ_y' = {[s_y, s_y + p_i − Σ_{z=x}^{y-1} δ_z], Φ_x − φ_i nc_x-1}
21             Δ_y'' = {[s_y + p_i − Σ_{z=x}^{y-1} δ_z, e_y], Φ_x, nc_x}
22             return s_x
23 return False
```

The $Place\_Task()$ takes a task, its computing time p_i and its power need φ_i, and a starting interval x. When the function is called it first checks that the proposed interval still have enough available power and cores (lines 1). Then, if the task fits in the interval then the interval is removed from the list and two new intervales are added (lines 2-7), one for the part where the task is scheduled and one for the other part, to keep a correct count of the available power and of the free cores. If the task does not fit in the interval the algorithm checks that there is enough available resources (cores and power) in the following intervals until the end of the task (lines 9-15). If such a time slot is foud then the corresponding power and one free core are substrated (lines 16-19). The last interval is cut in two as previsouly (lines 20 to 24). Note that cutting the intervals allow that a task can always be scheduled at the begining of an interval. The function return true is the task is placed, false otherwise.

A. List Algorithms

For both cases where we target the makespan and flowtime minimization it is logical that our first solution tries to greedily schedule tasks in the earliest available interval, seeking to minimize the task completion time C_i, as all the tasks are available at before computing the schedule. This is the list algorithm approach and its advantage is its implementation simplicity as we can see on the 2. Then by ordering the task list the algorithm may foster one or another task type. Often a random task order is used to compare other algorithms with a non smart solution. This algorithm is named $Random$ in the performance study.

The performance of the list algorithm however often depends on the order in which the tasks are processed. In our case their are two data associated with a task, namely its processing time p_i and the power it needs to run φ_i. Using decreasing p_i as the task ordering criteria (Largest Processing Time, or LPT, algorithms) fosters long tasks which are more difficult to place and usually gives good results for the makespan minimization on parallel identical machines when the number of tasks exceed 50, as shown in [22]. For this reason we assess the performance of this algorithm, named LPT in the performance study. On the other hand, when the flowtime minimization is targeted an increasing p_i as the task ordering criteria (Shortest Processing Time, or SPT, algorithm) must be preferred. We assess the performance of this algorithm, named SPT in the makespan performance study.

The task power need φ_i is also a significant criteria for the task choice as the tasks are constrained by the available power. As for large tasks, tasks with large p_i, tasks with large power need are difficult to place in the time slots and scheduling them first may avoid using later slots. For this reason we assess the performance of this algorithm, named LPN for Largest Power Need, in the performance study.

As already said, both the processing time p_i and the power need φ_i are important values for scheduling the tasks and taking them independently only fosters one and ignore the second. To take both into account we propose to use the product of the two values to order the tasks in the task list. The LPTPN algorithm, for Largest Processing Time Power need, sorts the tasks by decreasing values of $p_i \times \varphi_i$.

Note that these two last algorithms are rather makespan oriented and they do not produce appropriate solutions for the flowtime minimization.

To demonstrate the mechanism of the list algorithm, we summarize it in the following steps:

B. Dual Approximation Algorithms

It is worthwhile noting that the list based solutions fail to take into account how much extra power there is in a time slot compared to the task consumption. In fact it can only determine if the time slot is long (for the time) or large (for the

Algorithm 2: List algorithm

Data: Task list: $\mathcal{T}$
Result: A schedule
Order $\mathcal{T}$ in the chosen order
for $T_i \in \mathcal{T}$ **do**
  repeat
    $\Delta_x \leftarrow next(Intervals)$
  until $Place\_Task(p_i, \varphi_i, x)$

power) enough for executing a given task or not. Consequently, the use of time slots with high power levels to execute tasks that do not need a lot of power, might cause power waste if the remaining power in that time slot is not enough to execute another task on an other core, or if there are no more core to exploit the remaining power. The next logical step was hence to take the power availability and power consumption into account when placing tasks. Statistically, It is very likely that the task may have more than one available time slot that they can be scheduled in. We aim so to reduce the power waste by placing each task in the time slot that has the closest available power level to its power demand, rather than placing it in the earliest one possible, as the list based solution does.

A key problem here is how to avoid scheduling a task towards the end of the runtime, just because a time slot over there produces less power waste than many earlier ones. In other words, how to find the best fit for each task, without decreasing the quality of our solution (increasing $C_{\max}$). Applying the dual approximation technique presented by Hochbaum and Shmoys in [23] propose an interesting method to solves this problem. The Dual Approximation approach use a time horizon to limit the searching area. It is possible to schedule the tasks everywhere before the time horizon which allows to place tasks in the time slot where the power waste is the lowest as illustrated on figure 4. This solution has the advantage to take into consideration both power and performance constraints at the same time. At the beginning of the algorithm two time limits are set: the lower one, which must be lower or equal to the shortest possible schedule, and the higher one which is usually chosen such that we are sure that every algorithm will fit in the given time horizon. Here the lower value is set to $(\sum_{i=0}^{N} p_i)/NB$, where NB is the number of cores, as it is a lower bound. The higher value is set to $\mathcal{H}$, the time horizon. Then the algorithm attempt to reduce the time horizon on a binary search manner trying to fit all tasks in the new shorter time horizon. If they all fit, it re-reduces the searching area, if not, it increases it. The dual approximation algorithm is illustrated by Algorithm 3.

The dual approximation algorithm also works with a task list which order impacts the algorithm performance. We thus use all the variants of task ordering used with the list algorithm: largest processing time, shortest, largest power need, aso. The algorithm are named by prefixing the task ordering method by the method name DAPW for Dual Approximation Power Waste. A random ordering hence becomes DAPW-R and the largest processing time becomes DAPW-LPT.

Note that this method is assessed for both objectives to be minimized. In its design it however better supports the makespan objective as it targets the reduction of the time horizon and does not directly takes the time ordering into account.

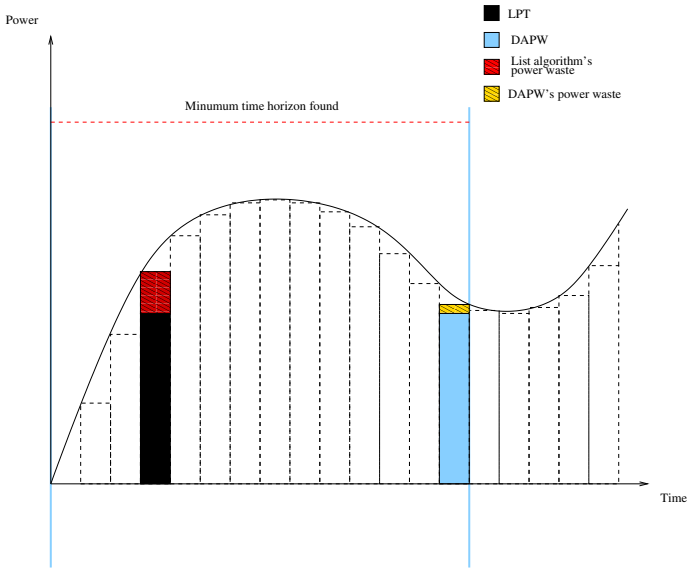


Figure 4: Comparison of the power waste resulting from scheduling the same task using greedy algorithm and BSPW

VI. Experiments

In this section, we present the experiments conducted in order to assess the performance of the algorithms mentioned in the previous section. The presented experiments were realized on a simulator rather than on a real platform as running lots of real life experiments is costly and hence does not allow to explore a wide range of parameters.

A. Simulator

simulator -¿ Python + Turtle + R We have developed a simulator[1] in python to assess the algorithm performance. Our simulator reads simple CSV files as input, these files contain the configuration values desired for tasks and intervals. Then, using Python, it randomly generates the necessary tasks lists and time slots lists before running the algorithms to compute the schedules. Finally, results are presented in different diagrams using R, in order to provide a clear comparison between different performances of the algorithms. The resulting schedule of each single execution can optionally be produced using Python's Turtle graphical tool to better understand where an algorithm operates correctly and where it does not. An example of schedule presentation with turtle is given on Figure 5.

RRR

B. Settings

The simulator takes a list of tasks as input. We did not used real data as input for the simulator but we tried to to use in our simulations data sets that are as close to real data

[1]This simulator is available on GitHub at http://github.com

Algorithm 3: Dual Approximation Power Waste algorithm

Data:
Task list: $\mathcal{T}$
hl: high limit, init to $(\sum_{i=0}^{N} p_i)/NB$
ll: low limit, init to $\mathcal{H}$
$schedule$: a schedule
Result: A schedule
Order $\mathcal{T}$ in the chosen order
while $hl > ll$ **do**
  $midpoint \leftarrow \frac{(hl\text{-}ll]}{2}$
  $placed \leftarrow true$
  repeat
    Take the next task T_i in $\mathcal{T}$
    repeat
      $\Delta_x \leftarrow$ next(Intervals from ll to $midpoint$, ordered by power waste)
      $placed \leftarrow Place\_Task(p_i, \varphi_i, x)$
      if $placed$ **then**
        $schedule \leftarrow schedule \cup \{T_i, result\}$
    until $placed \vee endofintervallist$
    if $placed$ **then** $finihed \leftarrow true$ **else** $finished \leftarrow false$
  until $finihed \vee endoftasklist$
  if $placed$ **then**
    $hl \leftarrow midpoint$
  else
    $ll \leftarrow midpoint$
return $schedule$

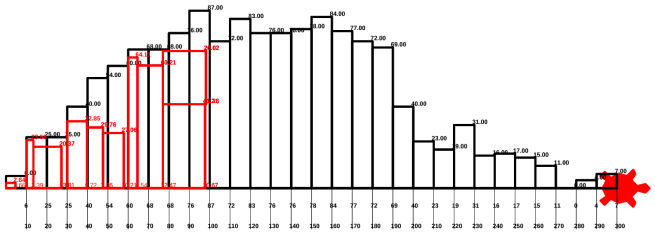


Figure 5: Turtle example

collected in other experiments as possible. For instance, p_i values of the generated task lists were chosen randomly using hyper-gamma law suggested by Lublin and Feitelson in [24] for some experiments. We also use randoms values that follow an exponential law for p_i in parts of the experiments to simplify them. When using the exponential law based task generation we define an upper bound $p_{i_{\max}}$ and we set the mean value used by the exponential law to half of the $p_{i_{\max}}$ value. In the experiments $p_{i_{\max}}$ ranges from 10 to 100, by steps of 10.

As we were lacking values for the power consumption of the tasks is given in power units and we choose to use a random generation of φ_i with a uniform law between 0.1 and $\varphi_{i_{\max}}$. In the experiments $\varphi_{i_{\max}}$ ranges between 4 and 40 power units, by steps of 4.

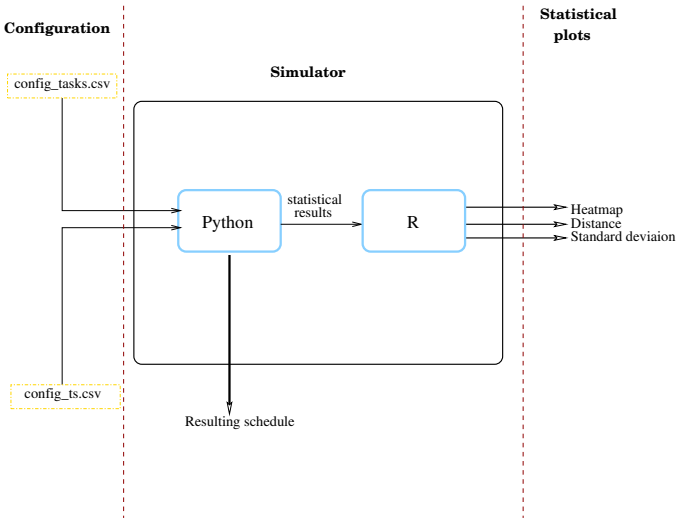


Figure 6: Comparison of the power waste resulting from scheduling the same task using greedy algorithm and DAPW-LPPN

For the experiments we choose intervals of equal length as it seems more realistic that the available power will be discretized on time with a constant period. The length of the intervals is 10 time units. To explore solar panels like power generation we generate sets of interval with a bell shape, as shown on figure 7. For the interval generation we define 5 levels and we randomly generate the available power for each interval inside the level. To generate bell shape we give a higher probability to increase the level when we are in an increasing phase and a higher probability to decrease the level when we are in a decreasing phase. The used random law in the levels is uniform, the maximum power that can be provided by the sources is 80 and each level has a height of 16.

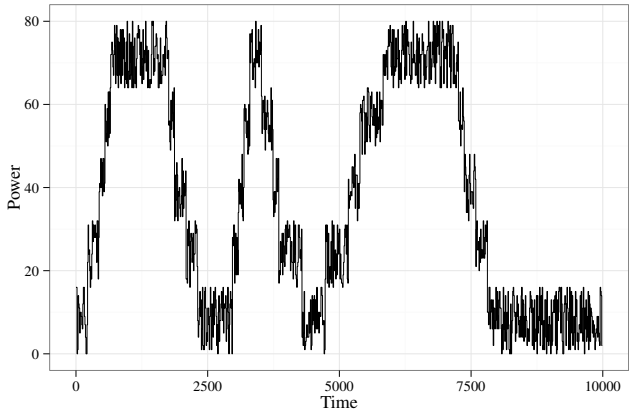


Figure 7: Interval generation on a bell shape to ...

For the experiments where the exponential law based task generation is used, we generate 250 intervals per set and we generate 600 intervals per set for the experiments that use the hyper-gamma law based task generation. Note that there is not guarantee, when we use a set of intervals, that a schedule can be found. For that reason we use large numbers of intervals.

Also interval from tasks: leave that for the RR

For the experiments we generate 100 different sets of intervals but with the same parameters and , for each couple of $p_{i_{\max}}$ and $\varphi_{i_{\max}}$ values, we generate 100 different sets of tasks. 10 000 experiments where thus performed, 100 for each $\{p_{i_{\max}}, \varphi_{i_{\max}}\}$ couple, each with a different task set. The same interval set is used for each $\{p_{i_{\max}}, \varphi_{i_{\max}}\}$ couple.

The number of cores is set to 8 for all the experiments which means that up to 8 tasks can be scheduled on the same interval. To assess the impact of the available power on the algorithm performance we use two values for the available power, 40 and 80. As the $\varphi_{i_{\max}}$ value ranges from 4 to 40 this means that the tasks may require up to 320 power unit to run without constraint in the case where $\varphi_{i_{\max}} = 40$.

To compare the results we need normalized values. Raw makespan or flowtime values cannot be compared as they depend on the considered set of tasks and intervals. A set of larger tasks will always give a longer makespan than a set of shorter ones. Therefore we use the following values to compare the schedules. We define the makespan performance $PERMAK$ as $(makespan - useless)/\sum p_i$, where makespan is the makespan obtained by the schedule and useless is the sum of the length of the intervals, between 0 and the end of the schedule, where no task can be scheduled because of too low available power. in a same way we define $PERFLOW$ as the flowtime performance as $(\sum(C_i - useless_i))/\sum p_i$, where C_i is the completion time of task T_i and $useless_i$ is the sum of the length of the intervals, between 0 and C_i, where no task can be scheduled because of too low available power.

C. Results

In this section we present the results of the experiments done. The figures are generated using R statistical environment.

Note that all the upper right corners of the figures are missing. This is because the computations were conducted on 250 intervals and, for the values of p_i and φ_i, not all the algorithms find a solution so that the mean cannot be computed. A new set of simulation has been started but was not finished at the deadline of paper submission. They will be included in the camera ready version.

We first assess algorithms regarding the makespan performance $PERMAK$. Figure 8 presents the best algorithm for each value of p_i and φ_i. The best algorithm is defined as the algorithm that has the best mean makespan on the 100 simulation runs for a couple of values $\{p_i, \varphi_i\}$. As we can see on the figure the best algorithm depends on the values of the processing time p_i and of the need power φ_i. Unsurprisingly when the power need of the tasks is low, then the power is not strongly constrained and the LPT algorithm that fosters long jobs give the best results. We are, in this case, close to the classical $P||C_{\max}$ problem which is efficiently solved by LPT. On the other hand, when the φ_i of the tasks is higher then the algorithms that takes power need into consideration get better results. Different algorithms however get the best results depending on the power need. When the processing time is small then the LPN algorithm is the best but when it increases then the LPTPN which takes both processing time and power need into account is better. For the case of a medium power need and small processing times, the DAPW family of algorithms find good schedules.

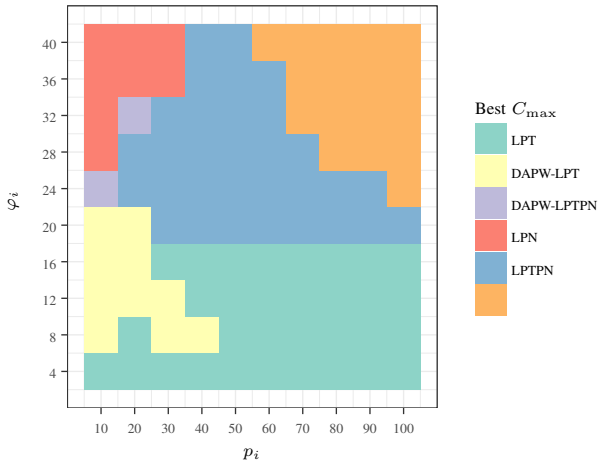
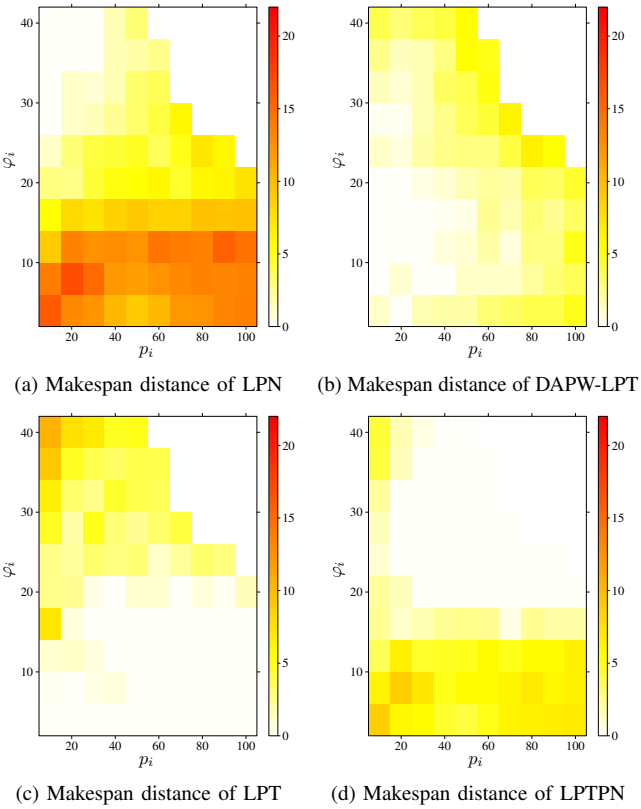


Figure 8: Heat map of the best average makespan performance $PERMAK$ for values of p_i ranging from 10 to 100 and values of φ_i ranging from 4 to 40



(a) Makespan distance of LPN (b) Makespan distance of DAPW-LPT

(c) Makespan distance of LPT (d) Makespan distance of LPTPN

Figure 9: Distance of the mean makespan performance $PERMAK$ of the LPV, DAWP-LPT, LPT, LPTPN algorithms to the best makespan performance

In an attempt to search for an algorithm that gives a solution that might not be the best solution, but not far from the best in most cases, the distances between the makespan performances

and the best performance for the four algorithm that gives more often the best performance, DAPW-LPN, DAPW-LPT, LPT and LPTPN, are presented in Figure 9. From Figures 9b and 9d we can see that the DAPW-LPT algorithm generates schedules with makespan never more far than 6% from the best one. This makes them good candidate for a global solution. Between both the LPTPN algorithms gives more ofen the best makespan. Unsurprisingly, the LPT algorithm, which gives the best makespan in the cases where the power need is low, generates its worst schedules in the cases where the power need is high and the processing time of the tasks small. It is however never worst than 11 %. On the opposite we note from Figure 9a that the LPN algorithm gives its best solutions when the power is constrained and the tasks of small size.

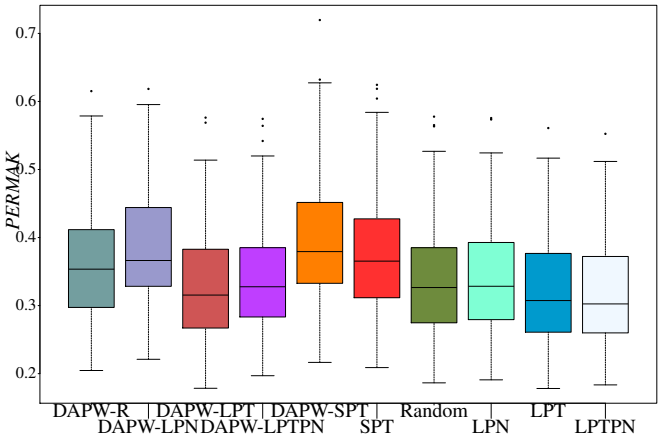


Figure 10: standard deviation of the makespan performance $PERMAK$for p_i = 100 and φ_i i = 20

Figure 13 gives the standard deviation of the makespan performance $PERMAK$ for a pair of values for p_i and φ_i. As can be seen on on the figure the varaiation is low, ranging between 0.3 and 0.38. Other measure realized shows that the variation may increase up to 0.8 but is always low.

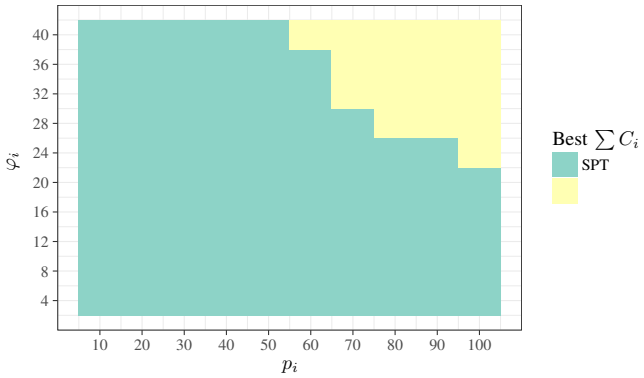
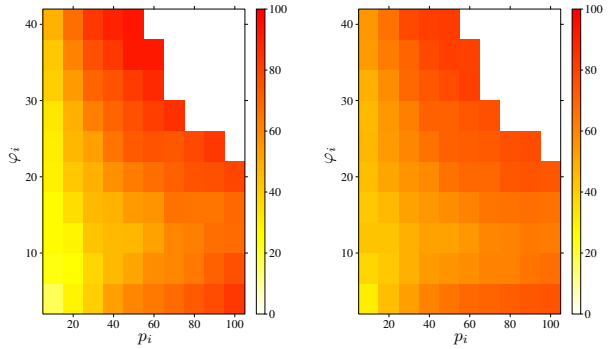
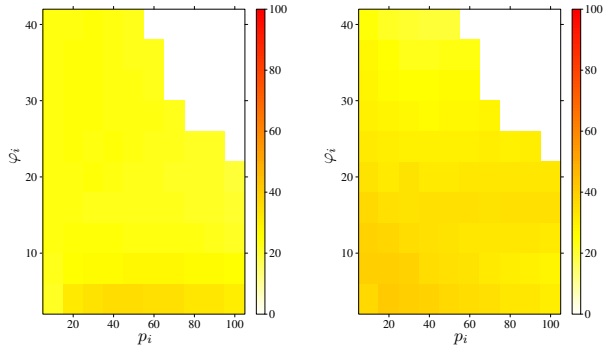


Figure 11: Heat map of the best average flowtime performance $PERFFLOW$ for values of p_i ranging from 10 to 100 and values of φ_i ranging from 4 to 40

Second, we assess algorithm performance regarding flowtime performance $PERFLOW$. Figure 11 proves that a list algorithm with SPT order would always produce the least flowtime.



(a) Flowtime distance of DAPW-SPT (b) Flowtime distance of DAPW-R



(c) Flowtime distance of Random (d) Flowtime distance of LPN

Figure 12: Distance of the mean flowtime performance $PERFLOW$ of the Random LPN, DAWP-LPN and DAWP-SPT algorithms to the best makespan performance

Similarly to the makespan performance assessment, we analyze the distance between the flowtime performance $PERFLOW$ of each algorithm from the best performance. The distances between the flowtime performances of DAPW-SPT, DAPW-R, Random, LPN and the best performance are presented on Figure 12. Figure 12 illustrates that ordering the task list according to the power need of tasks does not produce a good flowtime. Furthermore, even in DAPW which is not designed to minimize C_i, using the shortest processing time first order gives good results when the tasks are small enough. However, a list algorithm with random tasks order, has better probability to produce a good flowtime than DAPW algorithms or list algorithms with power need based task list order.Finally, we assess algorithms performance regarding their compute time, Figure 14 shows a clear gap between the compute time of list and DAPW algorithms, and it is observed that the gap expands for bigger tasks, which indicates that increasing pi has more negative effect on DAPW algorithms than on list algorithms regarding the compute time. Which is justified by the higher complexity of DAPW. In addition, our findings would seem to show that the complexity of both DAPW and list algorithms increases with the increase in the number of intervals, thus longer compute times are generated, which presents a challenge when conducting a big number of simulations.

Figure 13 gives the standard deviation for the flowtime

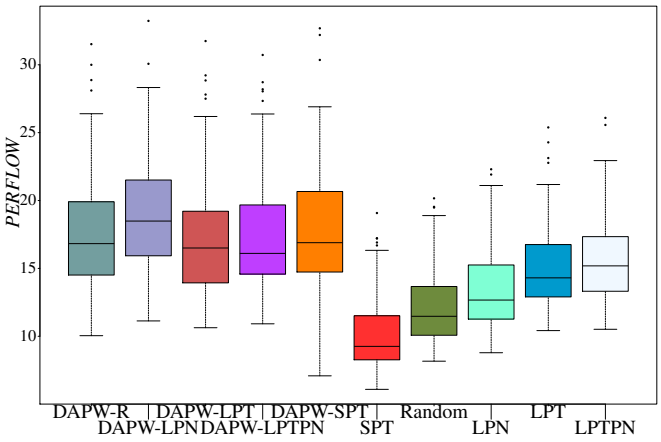


Figure 13: standard deviation of the flowtime performance $PERFLOW$ for p_i = 100 and φ_i i = 20

for all the algorithms used in the experiments. The SPT algorithm gives the lower deviation, lower than 10%. Globally the list based algorithms give lower variations than the Dual Approximation onces. Note that, as for the $PERFLOW$ performance measure, the RANDOM algorithm less than 12% is ranked second after SPT which means that the proposed algorithms are inefficient in this case.

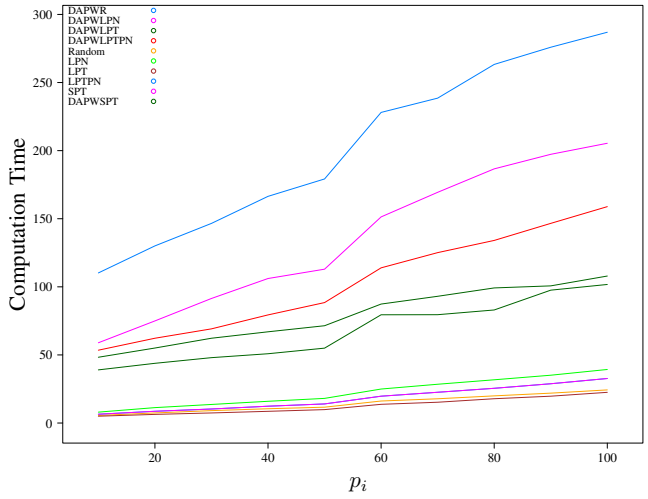


Figure 14: Compute time of the algorithms for $\varphi_i = 20$

Figures 14 and 15 give the mean compute time for the 100 runs done with one given value of the processing time p_i (20) and power need φ_i (20). Note that, on both figures, the LPTPN algorithms is barely visible on the plot as it gets the same running times as SPT and the curves overlap. Globally, if we except the case of the DAWP-R algorithm, we can see that the power need value only slightly impacts the compute time, the computing tiem actually slightly decrease when the φ_i value increases, while the processing time value causes longer computing times when increasing. From these results it is also clear that the DAPW family of algorithms have a higher running time than the list based family. This is not surprising as they iterates on the horizon value and, at each

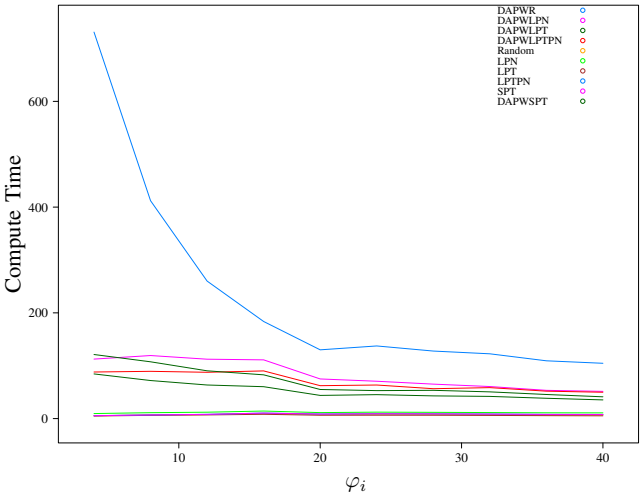


Figure 15: Compute time of the algorithms for $p_i = 20$

iteration, apply the same type of algorithm than the list based algtorithms. From the plots we can see that the DAPW-R, Dual Approximation with random list, generates the longest execution times, 10 times more than the faster algorithm, i.e. LPT.

It is worthwhile noting that discretizing the time in intervals increases the complexity of the algorithms. The worst case complexity of the $Place\_Task()$ function depends on X, the number of intervals. As the aglorithms iterate on the intervals to schedule a task then their complexity depends on X^2. Considering that the complexity of a list based algorithm usually depends the list ordering which is in $Nlog(N)$, where N is the number of tasks, the complexity of the list based algorithms with power constraint is $X^2Nlog(N)$. It turns out that the complexity of placing the tasks in the intervals heavily weighs on the computation times. As an exemple, running the whole set of experiements to compute the heatmaps takes 2 days when we run it with 250 intevals while it takes more than one week with 500 intervals.

As a synthesis of this performance assessment, LPTPN generates the best performance for the makespan objective on many values and it generates schedules never worst than 5% of the best ones while keeping reasonable computing time. This make it a could candidate in the general case. On the other hand a multi-policy algorithm that differently order the tasks in the list depending on the p_i and $varphi_i$ values could also be implemented to improve the performance.

VII. Conclusion

Acknowledgments

This work was supported in part by the ANR DATAZERO (contract “ANR-15-CE25-0012”) project and by the Labex ACTION project (contract “ANR-11-LA BX-01-01”). Computations have been performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté. – Besançon.

REFERENCES

[1] E. Oró, V. Depoorter, A. Garcia, and J. Salom, "Energy efficiency and renewable energy integration in data centres. strategies and modelling review," *Renewable and Sustainable Energy Reviews*, vol. 42, 2015.

[2] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 47:1–47:31, 2014.

[3] T. Kaur and I. Chana, "Energy efficiency techniques in cloud computing: A survey and taxonomy," *ACM Comput. Surv.*, vol. 48, no. 2, 2015.

[4] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141 – 147, 2014.

[5] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of virtual machines for real-time cloud services," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 13, pp. 1491–1505, Sep. 2011.

[6] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732 – 749, 2011, special Issue on Cloud Computing.

[7] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS*, 2015.

[8] P. Baptiste, "Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, ser. SODA '06. Society for Industrial and Applied Mathematics, 2006, pp. 364–367.

[9] P. Baptiste, M. Chrobak, and C. Dürr, *Polynomial Time Algorithms for Minimum Energy Scheduling*. Springer Heidelberg, 2007, pp. 136–150.

[10] P. Baptiste, M. Chrobak, and C. Dürr, "Polynomial-time algorithms for minimum energy scheduling," *ACM Trans. Algo.*, vol. 8, no. 3, 2012.

[11] S. Albers and A. Antoniadis, "Race to idle: New algorithms for speed scaling with a sleep state," *ACM Trans. Algorithms*, vol. 10, no. 2, 2014.

[12] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. IEEE Computer Society, 2010.

[13] P. Agrawal and S. Rao, "Energy-efficient scheduling: Classification, bounds, and algorithms," *CoRR*, vol. abs/1609.06430, 2016. [Online]. Available: http://arxiv.org/abs/1609.06430

[14] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. ACM, 2011.

[15] I. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: Scheduling energy consumption in green datacenters," *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, vol. 00, 2012.

[16] I. Goiri, M. E. Haque, K. Le, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Matching renewable energy supply and demand in green datacenters," *Ad Hoc Networks*, vol. 25, Part B, 2015.

[17] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 175–186, Jun. 2012.

[18] X. Wang, Z. Du, Y. Chen, and M. Yang, "A green-aware virtual machine migration strategy for sustainable datacenter powered by renewable energy," *Simulation Modelling Practice and Theory*, vol. 58, P 1, 2015.

[19] Y. Georgiou, D. Glesser, and D. Trystram, "Adaptive resource and job management for limited power consumption," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015.

[20] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete mathematics*, vol. 5, no. 2, 1979.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman & Co, 1979.

[22] S. Chrétien, J. Nicod, L. Philippe, V. Rehn-Sonigo, and L. Toch, "Using a sparse promoting method in linear programming approximations to schedule parallel jobs," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 14, pp. 3561–3586, 2015.

[23] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems theoretical and practical results," *J. ACM*, vol. 34, no. 1, pp. 144–162, Jan. 1987.

[24] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *J. Parallel Distrib. Comput.*, vol. 63, no. 11, pp. 1105–1122, 2003.