Evaluation of Heuristics to Manage a Data Center Under Power Constraints

Igor Fontana de Nardin, Patricia Stolf, Stéphane Caux

Evaluation of Heuristics to Manage a Data Center Under Power Constraints

Igor Fontana de Nardin*†, Patricia Stolf* and Stephane Caux†
*Institut de Recherche en Informatique de Toulouse (IRIT), Université de Toulouse, Toulouse, France
Email: [igor.fontana,patricia.stolf]@irit.fr
†Laplace UMR5213, Université de Toulouse, Toulouse, France
Email: stephane.caux@laplace.univ-tlse.fr

***Abstract*—In recent years, academics and industry have increased their efforts to find solutions to reduce greenhouse gas (GHG) due to its impact on climate change. Two approaches to reducing these emissions are decreasing energy consumption and/or increasing the use of clean energy. Data centers are one of the most expensive energy actors in Information and Communications Technology (ICT). One way to provide clean energy to Data Centers is by using power from renewable sources, such as solar and wind. However, renewable energy introduces several uncertainties due to its intermittence. Dealing with these uncertainties demands different approaches at different levels of management. This work is part of the Datazero2 Project which introduces a clean-by-design data center architecture using only renewable energy. Due to no connection to the grid, the data center manager must handle power envelope constraints. This article investigates some scheduling and power capping online heuristics in an attempt to identify the best algorithms to handle fluctuating power profiles without hindering job execution. Then, it details experiments comparing the results of the heuristics. The results show that our heuristic provides a well-balanced solution considering power and Quality of Service (QoS).**

***Index Terms*—Power constraints, Data center, Renewable energy, Heuristics**

I. Introduction

The Information and Communications Technology (ICT) sector has a significant impact on the global greenhouse gas (GHG), generating 1.8-2.8% of the global emissions [1]. Despite energy technology improvements, emissions of the sector have risen steadily. In addition, some experts warn that improvements in processor technologies could slow after 2025, creating an even worst scenario [1]. Therefore, internet providers can not overlook its carbon footprint by just increasing the number of resources to deal with the predicted expansion of internet by 2023 [2]. One of the most crucial elements in providing internet services is the large-scale data centers [3]. Data centers are a group of computing resources connected by fast networks and designed to run diverse applications' types. Data centers are a notable power waste actor using around 1% of worldwide electricity [3]. Hence, the ICT community has started to investigate other power supply possibilities, such as renewable energy usage [3].

Renewable energy employs self-renewing resources, such as wind and sunlight, providing clean energy. On the one hand, renewable sources could provide clean energy, but on the other hand, they introduce uncertainties due to weather conditions. Cloud providers, such as Google and Amazon insert grid connection as a way to deal with renewable sources uncertainties [4]. Datazero2 Project [5] designed a data center operated only by renewable sources without any link to the grid. This project aims to provide a feasible architecture to maintain data centers 100% clean. This architecture introduces a global power constraint since all the power usage must be lower than the power target (also named power capping). The power comes from several elements, such as solar panels, wind turbines, batteries, and hydrogen tanks. This article shows how algorithms can deal with fluctuations in global power capping without a significant impact on Quality of Service (QoS). This article investigates scheduling and power capping heuristics to identify the best algorithms to handle fluctuating power profiles without hindering job execution. Power capping heuristics change the machine state (on/off) or speed (using the Dynamic Voltage-Frequency Scaling (DVFS) technique), aiming to meet the power available. This article also presents experiments to evaluate the algorithms' performance with different workloads and renewable power production.

This paper is organized as follows: Section II starts with a brief overview of the state-of-the-art. Then Section III describes the server configuration problem. Section IV details the model. Section V presents the algorithms to solve the problem. Section VI explains the experimental setup. Section VII analyzes the experimental results, and finally, Section VIII concludes the article.

II. Related work

As renewable energy becomes a solution to the data center's carbon footprint reduction, several works deal with this topic in the literature. [6] proposes a scheduler to define the workload placement, considering the renewable sources production, cooling subsystem, and electricity rate structure. The main objective is to reduce energy costs. In [7], the authors create an algorithm to efficiently assign the user requests to the data centers, reducing the makespan, energy consumption, and overall cost. [8] designed an algorithm to group similar service-level-objective (SLO) jobs in the same physical machine group powered by renewable sources. All previous works aim to maximize renewable energy source

This work was partly supported by the French Research Agency under the project Datazero 2 (ANR-19-CE25-0016).

usage but use grid (brown) energy to add reliability. This connection removes the power constraints since they could use the energy from the grid when there is no renewable power available.

A power constraint problem is proposed by [9], but for embedded systems. Their algorithm uses DVFS to improve performance and energy efficiency, linking the harvested energy from the solar panels with the job scheduling. They also propose a way to deal with the overflow energy. This proposal applies a different power constraint for each processor. A renewable-only data center has a global power constraint and must find the best server combination possible to deal with it (e.g., how many servers are on at which speed). In [10], authors defined an energy abstraction for handling intermittent power constraints. This abstraction disables the server when there is no energy available which can be useful for web applications. However, this approach is not so good for other applications types, such as batch. In web applications, the data center must respond quickly to user requests. On the other hand, batches can delay the start of execution to the best time to execute it. The authors in [11] proposed genetic algorithms for job scheduling in data centers with power availability constraints. These algorithms find a near-optimal solution, but they have a high processing time.

Several works deal with power capping techniques to control the energy costs and peak power consumption [12], [13], [14], [15], [16], [17]. However, the majority of them deal with power capping as a local and not a global constraint [13], [14], [15], [16]. These works set a power limit for each processor individually. In [12], the authors proposed a model to define the processor frequency according to a global power constraint. They first improve running jobs as much as possible and then try to run new jobs. Their work has some similarities with this paper but with significant drawbacks. First, their jobs can run indefinitely without any QoS evaluation. So, the jobs do not have any constraint. Secondly, they have a priory step to profile the power and execution time relationship for each job submitted. Finally, they did not present how their algorithm deals with power changes. Their experiments used only one constant power capping (not realistic in a system powered by only renewable sources). Finally, the authors in [17] introduce a dynamic power scheduling with power constraint fluctuations. The power balancing updates the processors' power according to the previous usage. However, their strategy tends to distribute equally between the processors when the power capping is low. Also, they do not take into account the jobs' QoS in the decision-phase. Therefore, to the best of our knowledge, no work manages data centers reactively with fluctuations in global power capping using QoS in the decision-making process.

III. Problem statement

Like in [12] and [17], the problem is divided into two: server configuration and scheduling new jobs. First, the model defines the server configuration to deal with power capping. As mentioned before, the Datazero2 project aims to define a clean-by-design data center. So, the project must introduce several elements, such as solar panels, wind turbines, batteries, and hydrogen tanks. This diversity demands varied levels of management. Therefore, Datazero2 introduces different modules to deal with all aspects of a data center. A power management module defines a power target using power from renewable sources, such as wind turbines and solar panels, and storage, such as batteries and hydrogen. However, this module is not the subject of this article. The power management module defines a different power constraint at each time step. The main objective of this article is to meet the power target (capping) in real-time. Thus, a server configuration module must find the highest speed possible for the nodes below the power available. It does not know future power constraints, just the actual power limit. For this reason, sometimes, the server configuration module can not meet the power constraint. This occurs mainly because the servers need time and spend energy to go off. So, if a previous step has high power and in the next step the power drops, the total energy demanded to put the servers off could be higher than the available. Section VII-B1 presents a discussion about this point.

One way to measure the server speed is by using floating-point operations per second (flop/s) [18]. There are two possible actions for matching the power available. The first one is related to turning on or off a server. Obviously, when a server is off, it can not process any task (flop/s is zero), but it has the lowest power consumption. Nevertheless, both transitions on→off and off→on are not instantaneous and waste energy [19]. During these transitions, the server can not process any job. Also, putting a server off could lead to killing its running jobs, impacting the QoS and requiring to restart them later. The second action is changing the server running speed using the DVFS technique. A server can have different states (named p-state) when it is on. Each p-state has different power consumption and speed. Faster states consume more power than slower states. The CPU frequency range is discrete, although some works define it to be continuous [20]. The different states values are obtained by profiling the data center's servers. Hence, the algorithm needs to find the best combination of actions to meet the new power capping when it arrives, considering the impact of these actions. These actions could be a mix of on/off and DVFS p-state modifications.

After defining the server configuration, the scheduler must assign jobs to the servers. This article focuses mainly on CPU-bound scientific high-performance computing (HPC) jobs [21], [22]. In this application class, the user executes one application and waits for its end [23]. Another characteristic of this job type is that the schedule can delay the processing start [22]. This feature allows choosing the best moment to run it. The main action of the job scheduling is to designate the server or servers to allocate each job that arrives in the data center. Each server can run only one job at a time. Since the server configuration changes the server's state and speed, the scheduling must consider these modifications to define the placement. The scientific HPC applications are submitted with the following characteristics: arrival date, the number

of parallel resources, and walltime defined by the user [21], [22]. The first property gives the job submission time. The second indicates how many servers must process this job in parallel. Walltime is the total execution time allowed. Since this type of application is CPU-dependent, we ignored the communication between the parallel resources. The proposed algorithms react to possible fluctuations in CPU usage (e.g., when the application arrives at a communication-intensive moment). The principal QoS metric here is to avoid killing jobs by putting a server off or reaching the job's walltime.

IV. Proposed Model

Given a data center with S servers, each server s has an list of speeds D_s. For each speed inside D_s, there are two values: $F_{s,d}$ means the flops per second of the server s at speed d, and $P_{s,d}$ symbolizes the power needed to run the server s at speed d at maximum load. This article considers a global power constraint coming from an electrical module dealing with sources' commitment. However, it could also be applied to other global power constraints (e.g., a data center inside a smart city with a mix of power and carbon footprint to respect). The power is a global constraint defined as P_t^{avai}. A power profile consists of all values of P_t^{avai}, but the values are revealed step by step. So, the model only knows the P_t^{avai} of the actual time step. Time is discretized between time $t = 0$ and time $t = L$ with a time step length Δt. Δt can vary according to the specification of the renewable source, but P_t^{avai} does not change during the time step. The power module, which gives the power constraint, deals with batteries that can guarantee stable power even with small fluctuations during the step length. Therefore, global power usage of the data center ($P_{s,t}$ is the power usage for the server s at time t) must be less or equal to the power available, as demonstrated by Equation 1. Equation 2 defines the power for each server ($P_{s,t}$) as the state with higher power usage. $D_{s,d,t}$ is a boolean that indicates that the server s is at state d at step t.

$$\sum_{s=0}^{S} P_{s,t} \leqslant P_t^{avai}, \forall t \tag{1}$$

$$P_{s,t} = \max_{\forall d}(D_{s,d,t} \times P_{s,d}), \forall t, s \tag{2}$$

The server configuration algorithms will decide the binary variable $D_{s,d,t}$ for each time step. The objective is to find the configuration with the highest speed inside the power constraint, as presented in Equation 3. Besides the presented equations, the server configuration also considers the transitions between running and sleeping. These transitions use energy and take time. During the transition, the server is unavailable to execute jobs, so they do not increase the flops in Equation 3. The variable $l_{s,d,t}$ indicates how many seconds the server s is in the state d at step t. At each step t, the server could be only in one final state (running or sleeping). Table I exemplifies some servers' values. For example, just one $D_{s,d,t}$ between states 0 and 6 could be true at any step t, reducing switching operations.

$$\textbf{maximize} \sum_{t=0}^{T} \sum_{s=0}^{S} \sum_{d=0}^{D} D_{s,d,t} \times F_{s,d} \times l_{s,d,t} \tag{3}$$

Table I
Server definition example. States 0-6 are final states.

State (d)	Paravance		Taurus	
	$P_{s,d}$ (W)	$F_{s,d}$ (Gflops)	$P_{s,d}$ (W)	$F_{s,d}$ (Gflops)
0	200.5	38.4	223.7	18.4
1	165.1	34.56	189.03	16.56
2	136.76	30.72	161.28	14.72
3	114.69	26.88	139.67	12.88
4	98.10	23.04	123.43	11.04
5	86.22	19.20	111.79	9.20
6 (sleep)	4.5	0	8.5	0
7 (on→off)	65.7	0	106.63	0
8 (off→on)	112.91	0	125.78	0

With the server configuration plan defined ($D_{s,d,t}$), it is possible to choose the job scheduling. Given J jobs, job j contains a walltime W_j, floating-point operations Fl_j, arrival date Ar_j, and parallel resources Pl_j. In fact, the scheduler does not know the floating-point operations Fl_j. So, it must infer it using the execution time, for example. The execution time could also be estimated by the walltime [22]. The main objective of the scheduling is to find the first possible moment after Ar_j to place job j in Pl_j servers, meeting the constraints from Equations 4 and 5. Et_j is the job's execution time, and Ef_j is the total flops executed. Equation 4 guarantees that the job will finish in less time than the walltime defined by the user. Equation 5 indicates that the job will execute the job's flops entirely.

$$Et_j \leqslant W_j, \forall j \tag{4}$$

$$Ef_j \geqslant Fl_j, \forall j \tag{5}$$

Finally, Equation 6 demonstrates the objective function of the scheduling. The objective is to minimize the Slowdown [21], [24]. This metric shows how long a job waits ($wait_j$) relatively to its size. Values close to 1 are the best since it indicates a small waiting time. Small jobs are very sensitive to waiting time.

$$\textbf{minimize} \sum_{j=0}^{J} \frac{wait_j + Et_j}{Et_j} \tag{6}$$

V. Algorithms

As mentioned before, the model does not know the incoming events (power and jobs). Therefore, the system must deal with these changes in real-time. So, the following sections detail some real-time solutions for both server and scheduling problems.

A. Server configuration

At each time step t, a new power available P_t^{avai} arrives. The heuristics define $D_{s,d,t}$ using P_t^{avai}, trying to assure the constraint 1. As mentioned before, due to a fast power avaiable drop, this constraint maybe could not be achieved. The constraints from Equations 4 and 5 are used to ensure that jobs will be executed respecting their walltime and entirely. The following sections present some different heuristics to deal with this problem. The first two heuristics are quite straightforward. The last one is our main contribution to server configuration.

1) Idle Fewest Server Degradation (IFS): The main idea of this heuristic is maintaining more machines at the fastest speed. First, the algorithm runs only on idle servers, and after, if needed, it runs on the running machines. The servers are sorted by the server's power consumption, generating two variations: high power first (Idle Descending Fewest Server Degradation - IDFS) and low power first (Idle Ascending Fewest Server Degradation - IAFS). After that, it puts all machines in the fastest state possible. Then, the algorithm takes the first machine and reduces its speed one time. It will reduce the speed of this machine until it goes to sleep or Equation 1 is met. Taking the idle first helps to meet Constraints 4 and 5 because it will first reduce the power consumption without impacting the jobs.

2) Idle Servers Balanced Degradation (ISB): This algorithm is similar to the previous one, and it is similar than the proposed by [17]. The objective of ISB is to reduce the speed of the servers equally, maintaining more running machines. Server Balanced Degradation also has two variations: Descending Servers Balanced Degradation (IDSB) and Ascending Servers Balanced Degradation (IASB). This algorithm makes a Round-Robin between the servers, changing the server in each interaction.

3) Highest Flops Lowest Deadline (HFLD): HFLD is a heuristic that tries to find the highest speed for the data center (like Equation 3). The heuristic estimates the changes with the greatest impact on the data center speed inside the power available. Its main idea is presented in Algorithm 1. Unlike IFS and ISB, this algorithm decides the improvements based on the previous state. Line 2 shows this initialization. Line 3 stops the algorithm if the power usage is equal to the power available. Line 6 evaluates if the power usage is lower than the power available. If so, the function more_power() will use two lists in its decision: $F_{s,d,d'}$ is a list of flops correlations between the states of each server and $P_{s,d,d'}$ is a list of power correlations between the states. So, the function will search, for each server s at state d, the state change $d^{'}$ with the highest flop increment ($F_{s,d,d'}$) inside the power available ($P_{s,d,d'}$). The server with the highest possible increment is chosen and improved. This is the same objective as Equation 3. These lists are calculated one time for the available hardware. With all the possibilities mapped on this list, it can fastly decide which one will impact the most on the data center flops. The function more_power() finishes when the remaining power ($P_t^{avai} - \sum_{s=0}^{S} P_{s,t}$) is not sufficient to make any increase. The function will search the highest $F_{s,d,d'}$ with $P_{s,d,d'} \leqslant P_t^{avai} - \sum_{s=0}^{S} P_{s,t}$.

However, if the power usage is greater than the power available, the heuristic will try to reduce it. First, it will put all idle servers to sleep (see line 9). If it is not sufficient (lines 10-12), it will take each running job and put on minimal state to meet constraints 4 and 5 (lines 13-18). It sorts the running jobs by the deadline ($W_j - Et_j$) with the highest values first (line 13). Here, deadline means the time that the job must finish after starting to run. So, if a job starts at $t^{'}$, the job deadline will be the sum of this time and the walltime W_j (so, $t^{'} + W_j$). The algorithm uses Equation 7 to define the minimal speed $F_{s,d}$. The variables Ef_j and Et_j are the flops and execution time so far. Indirectly, this equation will adjust the job speed during its execution. For example, if a job arrives in memory or communication-intensive moment, it will use less CPU. So, the execution time Et_j will increase, but the flops Ef_j will not, reducing the time horizon ($W_j - Et_j$) and demanding a higher speed $F_{s,d}$. Equation 7 also compensates for slower speeds for a job at the beginning with faster states in the end and vice-versa.

Algorithm 1: Highest flops with lowest deadline

```
   input : F_{s,d,d'}, P_{s,d,d'}, D_{s,d,t-1}, and P_t^{avai}
   output: D_{s,d,t}
1  begin
2      D_{s,d,t} ⇐ D_{s,d,t-1};
3      if calculate_power_usage() = P_t^{avai} then
4          break;
5      end
6      if calculate_power_usage() < P_t^{avai} then
7          return more_power();
8      end
9      D_{s,d,t} ⇐ put_idle_servers_sleep();
10     if calculate_power_usage() < P_t^{avai} then
11         return D_{s,d,t};
12     end
13     for j in J.sort() do
14         D_{s,d,t} ⇐ minimal_state(j);
15         if calculate_power_usage() < P_t^{avai} then
16             return D_{s,d,t};
17         end
18     end
19     repeat
20         j ⇐ get_highest_deadline(J);
21         D_{s,d,t}, _sleep ⇐ reduce_speed(j);
22         if _sleep then
23             D_{s,d,t} = kill_job(j);
24         end
25         if calculate_power_usage() < P_t^{avai} then
26             return D_{s,d,t};
27         end
28     until J = ∅;
29     return D_{s,d,t};
30 end
```

$$(W_j - Et_j) \times F_{s,d} \geqslant Fl_j - Ef_j \tag{7}$$

If it is not sufficient (lines 15-17), it will take the job with the highest deadline and reduce its speed. The algorithm maintains closer to finish jobs with a higher speed since they have less time to compensate for a reduction. The algorithm does it until killing all jobs (lines 22-24) or reaching the power (lines 25-27).

B. Scheduling definition

The scheduling algorithms use the output of the previous algorithm ($D_{s,d,t}$) to define, for each new job, the placement. Two well-known algorithms were used: First-Fit (FF) [25] and Easy Backfilling (EBF) [26], [21], [24]. The First-Fit in the heuristics tries to fit the jobs using the jobs' order in the available servers until there is no more job to place [25]. Backfilling algorithm, on the other hand, uses the queue as a priority queue [26]. So, it tries to place all priority jobs on the servers. When it is not possible to place the next priority

job, the algorithm fills the "holes" with other jobs but without delaying the priority one. Both, First-Fit and Easy Beackfilling, use four job sorts, aiming to define the job priority [21], [24]: First-Come-First-Serve (FCFS), Descending size (Also named Largest Area First or Total Resources), Ascending size (Also named Smallest Area First), and Slowdown. FCFS sorts the jobs by the arrival date. Descending and ascending size calculate the job's size as the parallel resources multiplied by the walltime. Finally, the slowdown order calculates the slowdown of the waiting jobs (similarly to Equation 6).

VI. Experimental Environment

A. Platform

The platform consists of 20 simulated machines from GRID5000 [1], 10 Paravance servers, and 10 Taurus servers. The experiments use two different servers, adding heterogeneity to the data center. Therefore, the model's variable $N = 20$. The servers consist of six states, as shown in Table I. These values are representative and quite similar to other works [27], [28], [19]. This platform runs in the BATSIM simulator [29]. This simulator operates using the SIMGRID framework [30].

B. Power profile

Figure 1 demonstrates the three power profiles representing renewable energy collected from real solar panels and wind turbines at Toulouse, France [27]. These profiles represent the P_t^{avai} at each time step. Despite the similarity of the values, the figure shows different patterns. All three profiles describe two days of power generation with a new power available every 5 minutes.

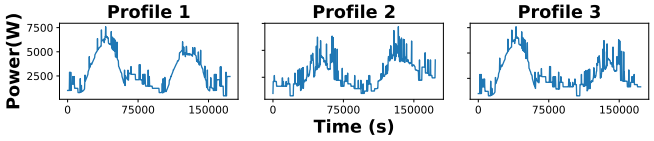

Figure 1. Power profiles from real solar panels and wind turbines at Toulouse, France [27].

C. Workload

As workload, the experiments used the traces from Google [31]. A workload generator based on [32] is used to generate the jobs. The generator creates several jobs varying the amount of work in flops, the arrival, and the number of parallel executions. The workload generator creates 20 different workloads. All workloads consist of 288 jobs, but they vary in size and arrival. The entire time execution is one day, with 12 jobs arriving per hour. The first experiment runs every workload using power profile 3. The first results of Section VII are an average of these executions. Finally, one of these workloads is executed with all profiles to evaluate the influence of the power capping on the algorithms. The results of this multiple profiles execution are discussed in Section VII-B4.

[1]https://www.grid5000.fr/

VII. Results

The following sections detail the metrics used in the comparison of the algorithms and the results themselves. After that, Section VII-B presents a discussion highlighting some aspects of the experiments. The experiments were realized by combining each job scheduling possibility with each server configuration. The next sections describe the results using the following notation: scheduling algorithm (FF / EBF) + jobs' order (Asc / Desc / FCFS / Slow) + server configuration algorithm (IDFS / IAFS / IDSB / IASB / HFLD). For example, EBF Slow HFLD algorithm means Easy-backfilling with Slowdown jobs' order and Highest Flops Lowest Deadline server configuration algorithm.

A. Evaluation metrics

The main objective of the experiments is to evaluate the influence of the power capping on the algorithms. Therefore, three metrics are used to compare the algorithms: Number of power violations, number of jobs killed, and slowdown. The first metric is energy-related, and the last two are job-related. A violation is when the power usage is above the power capping. Even if Equation 1 is a constraint, sometimes the heuristics can not avoid the power violation due to transition on→off. Section VII-B1 will discuss this point. The second metric highlights how many jobs are killed. Usually, a job is killed when there is not enough energy to maintain its server running or if it reaches the walltime. Finally, the last metric is the slowdown (see Equation 6). An algorithm could maintain a low speed to meet the power capping, but it will increase the slowdown because the running jobs will spend more time in the servers, and the waiting time of the queue jobs will increase. So, the idea of the experiment is to show a balance between QoS and power. Figure 2 presents the average result of all executions using Profile 3. The values are inverted (so the $^{-1}$), generating a higher area to lower values (which are the best).

A good option is the algorithm EBF Slowdown HFLD. In fact, this algorithm showed a good balance. IDSB algorithm presented a very good slowdown but with a higher number of power violations. IDFS also resulted in a good balance. However, this algorithm has, in general, more jobs killed than the HFLD. We solved the same problem as the heuristics using Mixed-integer linear programming (MILP). MILP knows the power profile entirely and all job arrivals, calculating the optimal solution. However, MILP takes a long time to find the optimal result. So, the experiments compare with MILP to illustrate how far the heuristics are from the optimal. Table II shows the best algorithms chosen. The table values are the average distance from each algorithm to the best possible result obtained by the MILP for each workload. The table also presents the results of the execution of EBF Slowdown without server configuration, aiming to provide a baseline for the power fluctuations-aware algorithms. This algorithm is often used in the literature. EBF Slowdown without server configuration maintains all servers at maximum speed. The following sections compare the results presented in Table II.

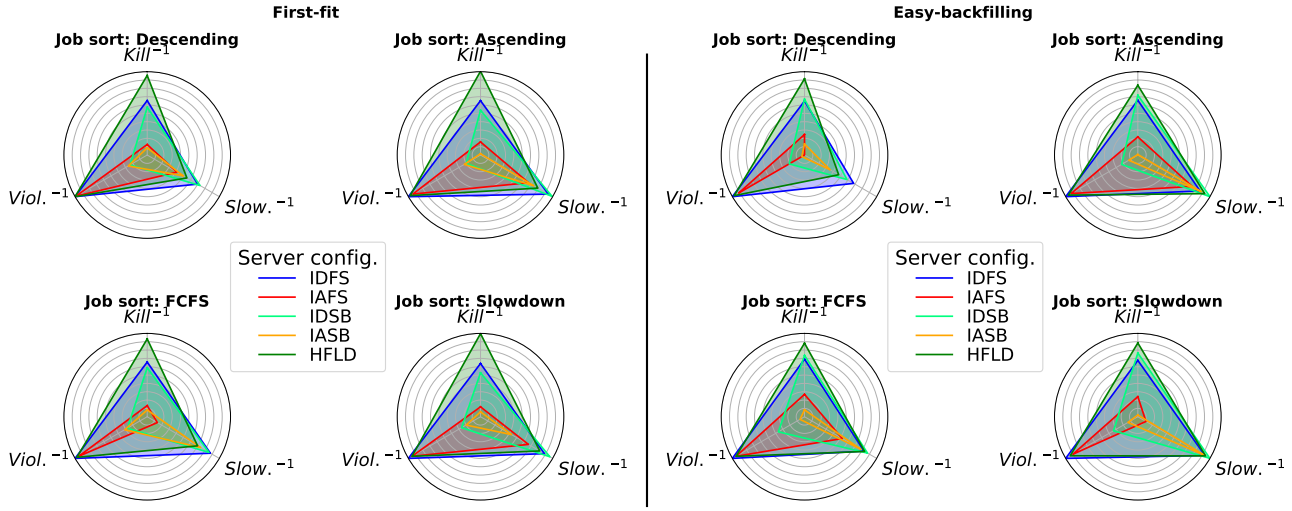

Figure 2. Average results for the 20 different workloads with Profile 3. The best values are the executions with the higher area. The power violation range is between 3 and 7. The number of jobs killed metric is between 14 and 54. Finally, Slowdown values are between 3.5223 and 16.9373.

The experiments run in 3h and 27min on 6 CPUs Core i7-10700, and draw 0.21 kWh. Based in France, this has a carbon footprint of 8.13 g CO2e, which is equivalent to 0.01 tree-months (calculated using green-algorithms.org v2.1 [33]).

Table II
AVERAGE INCREASE COMPARED TO MILP EXECUTION WITH PROFILE 3. THE MILP HAS NO POWER VIOLATION NOR JOB KILLED WITH A SLOWDOWN OF 1.44. SO, FOR EXAMPLE, EBF SLOW NO CONFIG. HAS 1.87 OF SLOWDOWN (1.44 FROM MILP PLUS 0.43 FROM THE DIFFERENCE).

Algorithm	Slowdown	Kill	Viol.
EBF Slow no config.	0.43	0	184
EBF Slow HFLD	2.90	19	3
FF Asc IDFS	2.91	28	3
FF Slow HFLD	4.48	14	3
EBF Slow IDSB	2.12	23	6

B. Discussion

1) Power Violation: The first metric is the power violation. Figure 3 illustrates two different executions from the power violation point of view. MILP could avoid all violations using the entire power profile to decide the configuration in each time step. HFLD provides a good alternative, dealing with the power fluctuations reactively. Figure 3 shows that it is not perfect, having some violations. The main reason for this violation is the highlighted part of the figure. At this moment, the algorithm has some machines running because the previous step has a high power available (so, the heuristic maximizes the flops with the power available). However, the power drops, and the algorithm puts all servers on sleep. However, the transition between on to off is not instantaneous and has considerable power usage. So, the power will not go from a high value to zero instantaneously. In consequence, one violation occurs. So, the heuristics can not completely avoid Equation 1 violation. However, few violations, such as those presented in the figure, could be removed using a prediction technique to analyze the power tendencies.

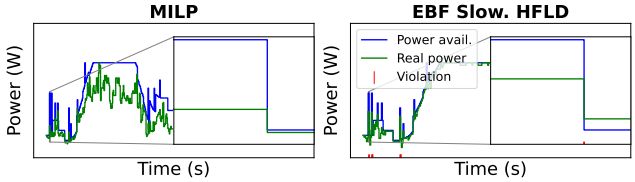

Figure 3. A zoomed moment with a violation. MILP avoids all violations, while HFLD avoids most of them.

2) Jobs Killed: A job is killed when there is not enough power to maintain it running (so the server goes to sleep) or if the execution time is greater than the walltime. This metric is crucial because a high number of jobs killed impacts the energy usage overall (due to the need to rerun killed jobs). Figure 4 shows the number of jobs killed over the 20 executions. The execution with no server configuration does not kill any job since it never changes the server's state. Besides this execution, the best algorithm is First-Fit Slowdown HFLD, having almost every result below 20 jobs killed. This algorithm achieves that because it is idle-aware and reduces the speed of the jobs with a longer deadline. Therefore, it tries to maintain the jobs which are closer to finish with higher speed. The Easy-Backfilling of the same algorithm has a good result also, but with some executions above 30 jobs killed.

First-Fit achieves better results in the jobs killed metric because it runs more small jobs than Easy-Backfilling. Easy-Backfilling reserves some future space for the priority job and just puts small jobs in front of the priority if this allocation would not change priority allocation. Idle Descending Fewest Server Degradation (IDFS) executions have high values of this metric, with their best result between 20 and 25. Finally, Idle Descending Servers Balanced Degradation (IDSB) results are

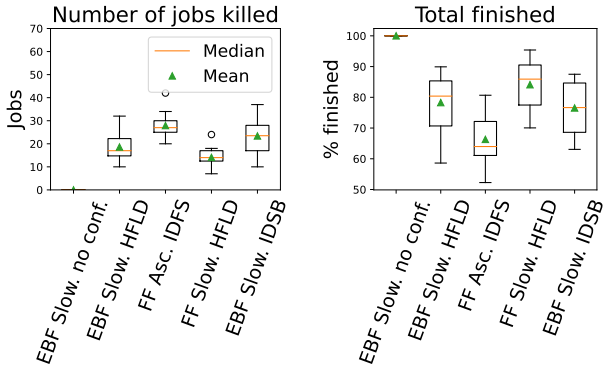

Figure 4. The number of jobs and total of finished work over the 20 executions for Profile 3.

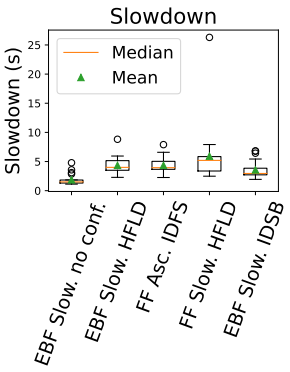

Figure 5. Slowdown over the 20 executions for Profile 3.

very different between each execution, demonstrating that this algorithm struggles to deal with varied workloads. Figure 4 illustrates the total finished work over the 20 different workloads. The percentage is related to the size of each workload. Easy-backfilling without server configuration executes all the jobs, so it has 100% of finished flops. Each job impacts differently on this value since they have different sizes. The size of a job is calculated by the total flops to execute multiplied by the number of parallel resources. Both HFLDs have a good value of total finished work compared with IDFS and IDSB. HFLD achieves these results since it gives more speed to the closer jobs to finish. So, the other algorithms can kill jobs that are almost finished.

3) Slowdown: Finally, the last metric is the slowdown. Figure 5 details the value of each execution. IDSB has good results, because it maintains more machines running than the others which allows running more jobs in parallel. However, it has more violations and more jobs killed than the others. IDFS good results come from the ascending order that prefers allocating small jobs first. So, the jobs more waiting time-sensitive are the priority. First-Fit Slowdown HFLD is worst than the other heuristics. It presents an outlier above 25, which is the worst of all executions. The reason for this behavior is linked with the job placement algorithm. First-Fit ignores the priority job, allocating all the possible jobs from the queue. In contrast to this algorithm, Easy-Backfilling will prioritize the job with a higher slowdown. Nevertheless, the HFLD heuristics have a slightly higher average value since they run more jobs, increasing the waiting time. But their values are quite near the best ones.

4) Multiple profiles: This section presents the experiments using one workload with the three profiles from Figure 1 to evaluate the behavior under different power constraints. Figure 6 demonstrates the results. We have run only the best executions from the previous section with multiple profiles. The figure shows that IDFS kills more jobs than the other algorithms for profiles 1 and 3. In profile 2, it is the second-worst. The HFLD has the best number of jobs killed using both Easy Backfilling and First-Fit for all profiles. However, the First-Fit has the highest slowdown on profile 2. As mentioned before, First-fit will place as many jobs as possible. In a strict

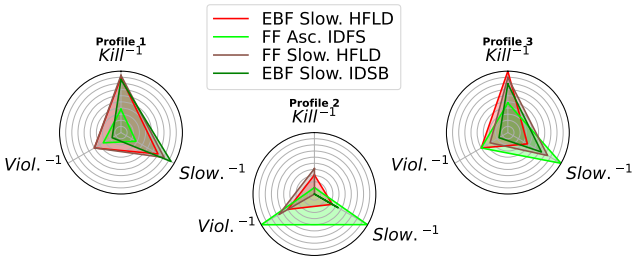

Figure 6. One workload execution on different power profiles. The power violation range is between 0 and 6. The number of jobs killed metric is between 15 and 44. Finally, Slowdown values are between 4.04 and 11.60.

power profile, jobs with higher demand will stay more time in the queue. Therefore, the slowdown of these jobs will increase. These results indicate that Easy Backfilling Slowdown HFLD delivers a quite robust implementation independently of the power profile and workload.

Table III
OVERALL RESULTS CONSIDERING ALL EXPERIMENTS.

Algorithm	Slowdown	Kill	Viol.
Easy Backfilling Slowdown Highest Flops Lowest Deadline	+	++	++
First-Fit Ascending Idle Descending Fewest Server Degradation	++		+++
First-Fit Slowdown Highest Flops Lowest Deadline	+	+++	++
Easy Backfilling Slowdown Idle Descending Servers Balanced Degradation	+++	+	

Table III presents a simplified view of the overall results, aggregating the previous results from the multiple workloads and multiple profiles. The table highlights that the Highest Flops Lowest Deadline presents a well-balanced implementation, independently of the scheduling algorithm. This heuristic is well-balanced because it considers the state of the jobs in the decision-making. Therefore, it is more likely to finish jobs than the other heuristics, even with more restrictive power profiles. The slowdown is slightly higher than the other heuristics. However, this is understandable as it executes more jobs.

VIII. CONCLUSION

Datazero2 is a project that aims to model a green by-design data center powered by only renewable sources. Renewable

sources introduce uncertainties due to their intermittence. Therefore, it is vital to design algorithms to deal with these uncertainties. This article investigates a number of scheduling and power capping heuristics, in an attempt to identify the best algorithms to handle fluctuating power profiles without hindering job execution. Combining Easy-Backfilling job scheduling with Slowdown metric to sort the jobs' queue and the Highest Flops Lowest Deadline as the server configuration algorithm provided the best balance between power violations, killed jobs, and slowdown. The results also show that it is essential to consider the impact on the running jobs in the power capping decisions. Future works will include other workload types, such as services. Also, we will introduce more flexibility in the decision-making changing in the power capping to improve QoS. Finally, we will include power predictions in the decision-making.

REFERENCES

[1] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. Blair, and A. Friday, "The climate impact of ict: A review of estimates, trends and regulations," 2021.

[2] U. Cisco, "Cisco annual internet report (2018–2023) white paper," 2020.

[3] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020. [Online]. Available: https://science.sciencemag.org/content/367/6481/984

[4] S. Kwon, "Ensuring renewable energy utilization with quality of service guarantee for energy-efficient data center operations," *Applied Energy*, vol. 276, p. 115424, 2020.

[5] J.-M. Pierson, G. Baudic, S. Caux, B. Celik, G. Da Costa, L. Grange, M. Haddad, J. Lecuivre, J.-M. Nicod, L. Philippe, V. Rehn-Sonigo, R. Roche, G. Rostirolla, A. Sayah, P. Stolf, M.-T. Thi, and C. Varnier, "DATAZERO: DATAcenter with Zero Emission and RObust management using renewable energy," *IEEE Access*, vol. 7, p. (on line), juillet 2019. [Online]. Available: http://doi.org/10.1109/ACCESS.2019.2930368

[6] K. Haghshenas, S. Taheri, M. Goudarzi, and S. Mohammadi, "Infrastructure aware heterogeneous-workloads scheduling for data center energy cost minimization," *IEEE Transactions on Cloud Computing*, 2020.

[7] S. K. Nayak, S. K. Panda, S. Das, and S. K. Pande, "An efficient renewable energy-based scheduling algorithm for cloud computing," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2021, pp. 81–97.

[8] J. Gao, H. Wang, and H. Shen, "Smartly handling renewable energy instability in supporting a cloud datacenter," in *2020 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2020, pp. 769–778.

[9] S. Liu, J. Lu, Q. Wu, and Q. Qiu, "Harvesting-aware power management for real-time systems with renewable energy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1473–1486, 2012.

[10] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: managing server clusters on intermittent power," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, 2011, pp. 185–198.

[11] A. Kassab, J.-M. Nicod, L. Philippe, and V. Rehn-Sonigo, "Assessing the use of genetic algorithms to schedule independent tasks under power constraints," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 252–259.

[12] B. Wang, D. Schmidl, C. Terboven, and M. S. Müller, "Dynamic application-aware power capping," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, 2017, pp. 1–8.

[13] C. Hankendi, S. Reda, and A. K. Coskun, "Vcap: Adaptive power capping for virtualized servers," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ser. ISLPED '13. IEEE Press, 2013, p. 415–420.

[14] T. M. Ha, M. Samejima, and N. Komoda, "Power and performance estimation for fine-grained server power capping via controlling heterogeneous applications," *ACM Trans. Manage. Inf. Syst.*, vol. 8, no. 4, aug 2017.

[15] H. Zhang and H. Hoffmann, "Podd: Power-capping dependent distributed applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.

[16] J. Park, S. Park, and W. Baek, "Rppc: A holistic runtime system for maximizing performance under power capping," in *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '18. IEEE Press, 2018, p. 41–50.

[17] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz, "Dynamic power sharing for higher job throughput," in *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2015, pp. 1–11.

[18] J. E. Smith, "Characterizing computer performance with a single number," *Communications of the ACM*, vol. 31, no. 10, pp. 1202–1206, 1988.

[19] I. Raïs, A.-C. Orgerie, M. Quinson, and L. Lefèvre, "Quantifying the impact of shutdown techniques for energy-efficient data centers," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 17, p. e4471, 2018.

[20] S. Saha and B. Ravindran, "An experimental evaluation of real-time dvfs scheduling algorithms," in *Proceedings of the 5th Annual International Systems and Storage Conference*, 2012, pp. 1–12.

[21] E. Gaussier, J. Lelong, V. Reis, and D. Trystram, "Online tuning of easy-backfilling using queue reordering policies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2304–2316, 2018.

[22] S. Takizawa and R. Takano, "Effect of an incentive implementation for specifying accurate walltime in job scheduling," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2020, pp. 169–178.

[23] M. Labidi, O. Lodygensky, G. Fedak, M. Khemakhem, and M. Jemni, "Xtremdew: a platform for cooperative tasks and data schedulers," *International Journal of High Performance Computing and Networking*, vol. 16, no. 1, pp. 55–66, 2020.

[24] D. Carastan-Santos, R. Y. De Camargo, D. Trystram, and S. Zrigui, "One can only gain by replacing easy backfilling: A simple scheduling policies case study," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019, pp. 1–10.

[25] D. E. Knuth, "The art of computer programming. vol. 1: Fundamental algorithms. second printing," 1969.

[26] D. A. Lifka, "The anl/ibm sp scheduling system," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1995, pp. 295–303.

[27] S. Caux, P. Renaud-Goud, G. Rostirolla, and P. Stolf, "It optimization for datacenters under renewable power constraint," in *European Conference on Parallel Processing*. Springer, 2018, pp. 339–351.

[28] V. Villebonnet, G. Da Costa, L. Lefèvre, J.-M. Pierson, and P. Stolf, "Energy aware dynamic provisioning for heterogeneous data centers," in *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2016, pp. 206–213.

[29] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, "Batsim: a realistic language-independent resources and jobs management systems simulator," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2015, pp. 178–197.

[30] H. Casanova, "Simgrid: A toolkit for the simulation of application scheduling," in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 2001, pp. 430–437.

[31] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, vol. 1, 2011.

[32] G. Da Costa, L. Grange, and I. De Courchelle, "Modeling, classifying and generating large-scale google-like workload," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 305–314, 2018.

[33] L. Lannelongue, J. Grealey, and M. Inouye, "Green algorithms: Quantifying the carbon footprint of computation," *Advanced Science*, vol. 8, no. 12, p. 2100707, 2021.