Analyzing Power Decisions in Data Center Powered by Renewable Sources

Igor Fontana de Nardin, Patricia Stolf, Stéphane Caux

Analyzing Power Decisions in Data Center Powered by Renewable Sources

Igor Fontana de Nardin
Laplace UMR5213, IRIT,
Université de Toulouse, CNRS, Toulouse INP
Toulouse, France
igor.fontana@irit.fr

Patricia Stolf
IRIT, Université de Toulouse,
CNRS, Toulouse INP, UT3
Toulouse, France
patricia.stolf@irit.fr

Stephane Caux
Laplace UMR5213,
Université de Toulouse
Toulouse, France
stephane.caux@laplace.univ-tlse.fr

***Abstract*—Both academics and industry have engaged their efforts in reducing greenhouse gas (GHG) emissions of Information and Communications Technology (ICT). Data centers are one of the most electricity-expensive ICT actors due to their uninterrupted service. Reducing the usage of brown energy or migrating to green energy using renewable sources (RES) is a way to reduce these emissions. However, this migration is not straightforward due to intermittence from these sources. This work is part of Datazero 2 ANR project. This project aims to design a data center powered only by RES production and storage elements. This architecture requires several decisions at different levels of management. This project divides the problem into two groups: offline and online. On the offline side, it uses renewable and workload predictions to prepare an offline plan with a power envelope (power delivered to the servers) and hints on how to manage the storage (batteries and hydrogen). Given the offline plan, the article's contribution is how to deal with real and dynamic power constraints online while keeping the planned storage level at the end. So, this article proposes policies to modify the plan according to the changes in predictions. We evaluate these policies in a homogeneous and heterogeneous data center. The results demonstrate that our policies could approach the storage level and improve Quality of Service (QoS) independently of data center infrastructures.**

***Index Terms*—Power decisions, Data center, Renewable sources**

I. Introduction

Information and Communications Technology (ICT) emits around 1.8-2.8% of the world's total greenhouse gas (GHG) [1], [2]. Due to its uninterrupted operation, the data centers sector is one of the most electricity-expensive ICT actors [3]. A report from 2015 revealed that Google data centers consumed the same amount of energy as the entire city of San Francisco [4]. In addition, the situation tends to get even worse due to the reduction of the improvements in processor technologies and the predicted expansion of internet usage [5], [2]. The IT community has started investigating brown energy's replacement with renewable energy, aiming to reduce the impact of data center providers' emissions [6]. Renewable sources (RES), such as wind and sun irradiance, can provide clean energy. However, they introduce several uncertainties, mainly due to weather conditions. Big cloud providers (e.g., Google and Amazon) smooth these uncertainties by inserting grid connections [7]. Therefore, they are not fully clean data centers. This work is part of Datazero 2 Project, which designed a feasible data center architecture operated only by RES without any link to the grid [8].

A clean-by-design data center must introduce several elements to provide energy to the IT servers, such as Wind turbines, Solar panels, Batteries, and Hydrogen tanks. Consequently, a manager must predict the weather conditions. It is also important to predict the IT power demand to deal with the future workload. This project divides the problem into two decision levels: offline and online. On the *offline* level, it uses both predictions (weather and workload) to create a plan for a defined next time window (e.g., the next 3-days). This offline plan consists of the power envelope and expected storage (battery and hydrogen) level. The power envelope is the estimated power given to the IT servers. It is the sum of expected renewable production and planned storage usage. Then, the *online* side applies the offline plan. However, the predictions are not exactly the real values, so the manager must react to the online events. For example, workload prediction uses stochastic models to estimate the mass of the load. However, in *online* mode, it receives the precise jobs to execute. Also, renewable production will be different, demanding modifications in the storage usage plan to keep the Quality of Service (QoS). However, *online* must finish the time window with the storage levels as close to the plan end as possible.

As QoS, we have considered slowdown and number of jobs killed. The slowdown is the response time (or time each job stays in the system) normalized by the running time [9]. These metrics are highly impacted by power decisions. For example, maintaining a few servers running uses less power but increases the waiting time, which directly affects the slowdown. Also, putting a server to sleep during a low renewable production could kill the jobs running on it. So, *online* must find a balance between the power decisions and QoS impact. Besides, a heterogeneous data center infrastructure complicates the decision-making even more. Since all servers have different characteristics, the choice of the server on which the job will run is important. Therefore, the server choice could impact QoS and energy consumption differently. This article

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

focuses on online plan adaptations. The main contributions are twofold. First, it presents an online model detailing the the problem to solve. This model implements three power compensation policies to react to power fluctuations. The objective is to finish the time window with the storage levels as close as possible to the planned, improving QoS when it is possible. The model also takes into account the servers' heterogeneity in the decision-making. Second, this work details experiments showing the improvements of the proposed model compared to an offline baseline and two reactive executions. The offline baseline only follows the power plan. The reactive executions do not use the plan, just reacting to the events.

This paper is organized as follows. Section II presents the related work, highlighting the gap in state-of-the-art. Then Section III shows an overview of the problem. Section IV addresses the proposed online model. Section V presents the experiments and the discussion about the results. Finally, Section VI concludes the article.

II. Related work

Several works pursue ways to deal with RES uncertainties. However, they introduce, in many cases, grid (brown) connections or just one level of management (online or offline) [10], [11], [12], [13], [14], [15]. In work [14], the authors created an offline optimization framework using a model to capture the randomness of the RES. The authors in [10] propose an offline mechanism to vary the price according to the grid and RES price. They modeled a Stackelberg game to stimulate users to join the Demand-Response's load shift. [15] proposes two online renewable-aware schedulers to maximize RES usage. The main objective is to maximize the jobs running when there is more solar irradiation, using the grid and batteries to deal with the intermittence. The authors in [13] describe two offline scheduling algorithms: a Genetic Algorithm (GA) and a Similar Mathematical Morphology (SMM). Both algorithms use Dynamic Voltage-Frequency Scaling (DVFS) technique to reduce the processor's frequency, using less energy. SMM is similar to GA but uses frames to compare and improve the best solution. In [11], the authors create an online heuristic to assign the user requests to the data centers. Their objective is to reduce the makespan, energy consumption, and overall cost.

Finally, the authors of [12] proposed a Mixed Integer Linear Programming (MILP) to optimize the commitment of a data center powered by only wind turbines, solar panels, batteries, and hydrogen storage systems. It is offline and based on prediction. They use weather forecasts to find the optimal long-term decisions to supply a data center's demand. However, the work lacks real-time events, such as scheduling and fluctuations in power usage. Therefore, to the best of our knowledge, no work makes online decisions using offline plans for data centers powered by only renewable energy.

III. Problem statement

This project designs a feasible architecture to use only renewable sources to provide power to the servers in a data center. This architecture includes several power elements to maintain the servers, such as Wind turbines, Solar panels, Batteries, and Hydrogen tanks. Since all power comes from RES, it must estimate the future renewable production and power demand from IT. These estimations allow it to find a match between delivered and demanded power. However, the actual power demand and renewable production could vary from the predictions. Therefore, it must react to the actual events. This project divides the problem into two levels: offline and online. Figure 1 shows this integration. Since this article focuses on the online side, we will not detail offline. The main *offline* objective is to match the workload demand with the power delivered to the servers. The power delivered to the servers is also named the power envelope. The power envelope is the sum of estimated renewable production and storage usage.

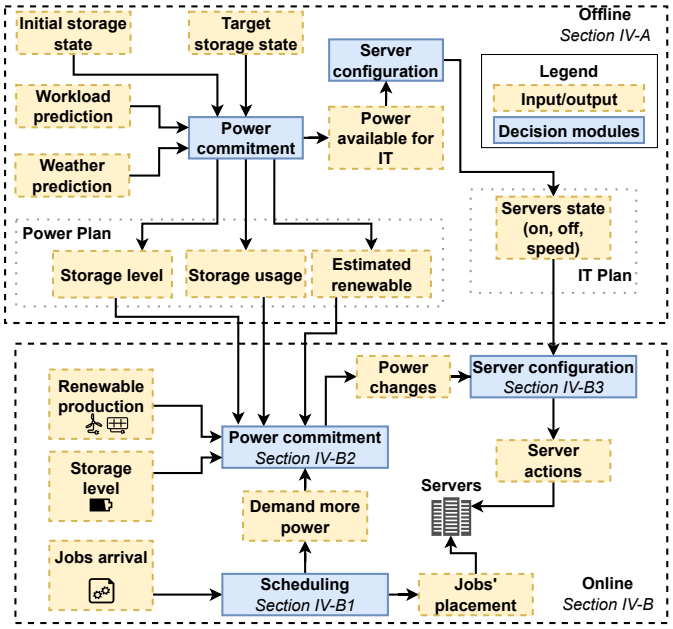

Figure 1. Online and offline integration with the input/output of each decision module. The figure indicates each section that will detail each decision module.

The *online* input consists of three time series: storage level (e.g., batteries' state of charge), storage usage, and estimated renewable production. As mentioned before, the two last time series are the power envelope. The power envelope is decomposed into two time series because the *online* handles both differently. *Online* can change storage usage, dealing with renewable power fluctuations. The time series have one time window size divided into several time steps. Figure 2 demonstrates the time division. Given each time step t with a length of 300 seconds, a 3-day time window has 864 steps. The entire data center life management has several time windows. The main goal of online is to have the storage level as close to the plan as possible at the end of the time window. So, it reacts to the events in real-time, improving the QoS when possible. We have considered slowdown and jobs killed as

QoS metrics because both describe the impact of the power decisions on the jobs. The jobs' waiting time directly impacts slowdown. Thus, maintaining a few servers running or at a minimal frequency will reduce the power consumption but will affect the slowdown. Also, if the *online* puts a server to sleep, this could kill the jobs running on it.

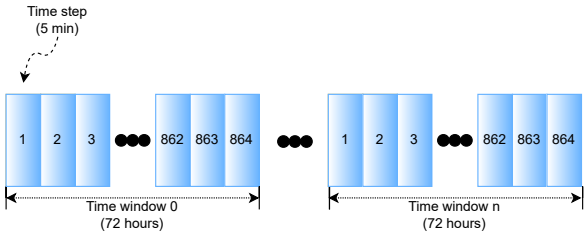

Figure 2. Time window definition.

Figure 1 shows that the online decision is composed of three decision modules. The scheduling module receives the actual jobs submitted by the users and defines the job placement in the servers. It is its responsibility also to evaluate QoS and improve it. For example, it verifies if the power envelope is sufficient to maintain the jobs running. If it is not, the scheduling module requests more power. The power commitment module has two responsibilities. First, it evaluates if power changes in the offline plan violate the battery and hydrogen levels. For example, if the data center uses more power than the offline plan, it estimates if this change will dry the batteries now or in the future. So, it recalculates the state of charge for all future steps. The second responsibility is to verify if it is possible to compensate for power changes. It compensates in a future step for any modification in the power plan. This compensation is crucial to assure that the storage levels will be close to the planned at the end of the time window. Finally, the server configuration module translates the power changes in server states. For example, if there is more power to use now, this module can turn on more servers or speed up the running ones. Since it runs in online mode, all online decisions must be quick. Thus, these *online* modules can not find the optimal solution for the time window. The following sections will detail the modules, presenting the proposed solution for each one.

IV. Proposed Model

A. Offline Plan

As described in Section III, the *online* receives an offline plan with three time series: storage level, storage usage, and estimated renewable production. They are composed as follows:

- Storage level time series:
  - SoC_t: Batteries' state of charge (SoC);
  - LoH_t: Level of Hydrogen (LoH);
- Storage usage time series:
  - $Pdch_t$: Power to discharge from batteries;
  - Pch_t: Power to charge the batteries;
  - Pfc_t: Power to discharge from Hydrogen;
  - Pez_t: Power to charge the Hydrogen;
- Estimated renewable production time series:
  - Pr_t: Estimate power from RES;

All time series have the same number of steps t. As presented before, with a time window of 3 days and a step length of $L = 300$ seconds (or 5 minutes), the time series have $T = 864$ steps. The power envelope $Pprod_t$, used by the servers, is calculated as follows:

$$Pprod_t = Pr_t - Pch_t - Pez_t + Pdch_t + Pfc_t \quad (1)$$

Since the model has an estimated power envelope, it can define a pre-planned server configuration. This server configuration translates the $Pprod_t$ into server states. Then, it finds the data center's fastest speed (in flops [16]) for the available power. The server configuration can change each server's processor's speed due to the DVFS technique. This technique allows decreasing the processor's frequency to use less energy. The CPU frequency range is discrete, although some works define it to be continuous [17]. So, it must find the best combination of servers off and on at some speed that uses equal or less energy than the power envelope. Thus, given a data center with S servers, each server s has a list of states D_s. Each state d in D_s has two values: $F_{s,d}$ means the flops per second of the server s at state d, and $P_{s,d}$ symbolizes the power needed to run the server s at state d at maximum load. $D_{s,d,t}$ is the boolean decision variable that indicates that the server s is at state d at step t. So, Equation 2 tries to maximize the flops possible, while Equation 3 is a constraint to maintain the power lesser or equal to the envelope. Equation 4 assures that the server s will be at only one state d at time t.

$$\textbf{maximize} \sum_{t=0}^{T} \sum_{s=0}^{S} \sum_{d=0}^{D_s} D_{s,d,t} \times F_{s,d} \quad (2)$$

$$\sum_{s=0}^{S} \sum_{d=0}^{D_s} P_{s,d} \times D_{s,d,t} \leqslant Pprod_t, \forall t \quad (3)$$

$$\sum_{s=0}^{S} \sum_{d=0}^{D_s} D_{s,d,t} = 1, \forall t \quad (4)$$

We have solved it using Integer programming. The *online* module receives this pre-planned server configuration and uses it as an initialization. This *online* module maintains the server configuration plan if there is no need to change the power usage.

B. Online Model

1) Scheduling: The main objective of a data center is to run jobs coming from users. The scheduler component defines the action of running a job j in a server s. The scheduling model is applied to scientific High-Performance Computing (HPC) data centers. The user of these data centers submits an expected job duration $wall_j$ (named walltime) with the job. If the job takes more time than the walltime, the scheduler kills the job. Usually, the scheduler is driven by some QoS metric. Our QoS metrics are twofold: number of jobs killed

and slowdown. The former indicates how many jobs have started but, for some reason, did not finish their execution. Equation 5 presents the latter QoS metric [18]. The slowdown indicates the relationship between the time that the job spends in the system and its size. The best slowdown value is 1, where the scheduler puts the job to run as soon as it arrives. It increases according to the waiting time.

$$slow_j = \frac{wait_j + wall_j}{wall_j} \tag{5}$$

The model deals with both metrics in two ways. First, the scheduler decides the placement in the available servers given by $D_{s,d,t}$ plan. We chose as the scheduling algorithm the heuristic EASY-Backfilling [19]. This heuristic naturally improves the slowdown due to its backfilling behavior, but we also sort the job waiting list by the slowdown. Due to a possible heterogeneous data center, our model tries to take first the servers with higher speed for the jobs at the beginning of the waiting list. This behavior helps in the slowdown and maximizes the usage of faster machines. Regarding the killed jobs, the scheduler tries to maintain the servers at the minimum speed to avoid killing the jobs using Equation 6, where $elapTime_j$ is the actual execution time of job j and $elapFlop_j$ is how many flops the job j has processed. Equation 7 estimates the job size $jobFlop_j$ (in floating-point operations) using the walltime $wall_j$, a fixed server's speed $F_{s',d'}$, and an error ϵ_u. In [20], the authors introduced the error ϵ_u calculated by each user. They claimed that the users usually overestimate the execution time by up to 5 times. Therefore, Equation 6 estimates how many flops the server must deliver to finish the job before the deadline. 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 $elapTime_j$ will increase, but the flops $jobFlop_j$ will not, reducing the time horizon ($wall_j - elapTime_j$) and demanding a higher speed ($D_{s,d,t} \times F_{s,d}$). Equation 6 also compensates for slower speeds for a job at the beginning with faster states in the end and vice-versa. The scheduler asks for more power to maintain the server at least at the lowest speed. However, this modification respects the rules of the online power commitment (see the next section).

$$(wall_j - elapTime_j) \times D_{s,d,t} \times F_{s,d} \geqslant jobFlop_j - elapFlop_j \tag{6}$$

$$jobFlop_j = wall_j \times F_{s',d'} \times \epsilon_u \tag{7}$$

2) Power commitment: The power commitment adapts the offline plan according to real-time events. This model focuses on battery decisions since hydrogen is harder to work in online mode since it needs time to turn on/off. The battery power usage can vary from the plan. This variation happens because:

1) Server idleness: The *offline* estimates a worst-case scenario where the servers will spend maximum usage during the time step. However, the server can stay idle for some time or use less energy (the application can fluctuate its usage);
2) Renewable production: Renewable production can be different from the estimated. The batteries will maintain the servers running at the defined speed or absorb the over-production;
3) Scheduling changes: The scheduler can ask for more power to finish the jobs running. Also, it could not use all the power planned because there are fewer jobs or the jobs consume less than expected;

Therefore, the power commitment must verify the impact of these variations on future storage levels. For example, a higher usage now could dry the batteries in the next steps. Equation 8 shows the constraint that the state of charge must be between the higher threshold ($SoCmax$) and the lower threshold ($SoCmin$) in all steps. Equation 9 shows how the SoC_t is calculated, where SoC_{t-1} is the previous state of charge, σ is the natural discharge rate, ηch is the charge efficiency, and ηdch is the discharge efficiency. Pch_{t-1} and $Pdch_{t-1}$ are the power to charge and discharge from the batteries. This module changes Pch_{t-1} and $Pdch_{t-1}$ to modify the power envelope $Pprod_t$ (as presented in Equation 1). Then, when it adjusts the power usage, the power commitment recalculates the state of charge of all future steps (using the plan for future Pch and $Pdch$). During this process, if a future SoC_t violates Equation 8, power commitment adapts Pch and $Pdch$ from the previous steps to solve it. For example, if the $SoC_{t'}$ will reach $SoCmax$ at t', this module must increase $Pdch_{t'-1}$ (or decrease $Pch_{t'-1}$) at $t'-1$.

$$SoCmin \leq SoC_t \leq SoCmax, \forall t \tag{8}$$

$$SoC_t = (SoC_{t-1} \times (1-\sigma)) + (Pch_{t-1} \times \eta ch \times L) - (\frac{Pdch_{t-1}}{\eta dch} \times L) \tag{9}$$

Also, the power commitment can not infinitely increase or decrease Pch, $Pdch$, and $Pprod$. It must respect the boundaries of each variable. Equation 10 shows the $Pprod_t$ boundary, where $Pmin$ is the power to maintain all servers sleeping and $Pmax$ is the power to run all servers at maximum speed. For example, inserting more power than $Pmax$ in a step is useless since the servers can not use this power. Equation 11 presents the boundaries to charge the battery. The maximum possible charge power is the minimum value between the battery physical charge limit ($Pchmax$) and the estimated renewable energy (Pr_t). This module can recharge the batteries using only renewable, so it must limit this value to the estimated production. Finally, Equation 12 demonstrates that for discharging, the value goes from 0 to the battery physical discharge limit ($Pdchmax$).

$$Pmin \leq Pprod_t \leq Pmax, \forall t \tag{10}$$

$$0 \leq Pch_t \leq \min(Pr_t, Pchmax), \forall t \tag{11}$$

$$0 \leq Pdch_t \leq Pdchmax, \forall t \tag{12}$$

All previous constraints guarantee the boundaries of storage levels and power envelope. However, the main power commitment's objective is to ensure that SoC and LoH are close to the plan at the end of the time window. Just applying the previous power changes does not assure that the SoC

and LoH will have the planned values at the end of the last step. For this reason, the power commitment module must compensate these changes in a future step. For example, if the planned SoC at the end is 50% and the power commitment module increases the usage of the battery by 5% now without compensation, the SoC will be 45% at the end. So, the module must reintroduce these 5% (reducing the usage) in a future step. The question now is: in which future time step? To answer this, we propose three policies: *peak*, *next*, and *last*. Figure 3 illustrates an example of these policies. In the figure, if the scheduler demands more power at step 1, it could compensate for it at step 3 (*next*), step 12 (*peak*), or step 15 (*last*). The same happens for server idleness and renewable production. However, since both are unpredictable, their compensation is reactive (after finishing the step). The *next* and *last* policies execute the same search independently of the type of compensation (increase or decrease). The *peak* policy finds the higher peak to reduce and the lower peak to increase. So, for example, if it tries to find a step to increase the usage, it will take the one with lower usage.

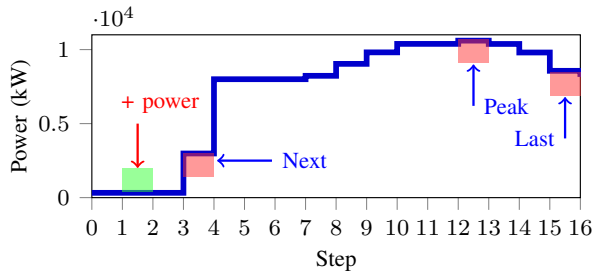

Figure 3. Compensation policies. The blue curve is the usage plan in step 1. The green square is the increased power, and the red squares are where the compensations occur for each policy.

3) Server configuration: The last part is to transform the power change in server configuration. As mentioned in Section IV-A, *online* receives the server states calculated using the power envelope. However, if the compensation increases or decreases the usage in a future step, it must convert this change in the server states, adapting the given by the *offline*. We propose a heuristic to find quick solutions. This heuristic has a list of all power and speed differences between two states, so it can fastly decide which one will impact the most on data center speed. When there is more power than the usage, it will go through the list searching for the highest flops improvement below or equal to the power increment. It will do it in the following order:

1) Find the highest improvement possible for the servers running some job;
2) Find the highest improvement possible for the idle servers.

Table I gives an example of one server. For example, if there are two servers: one with $d = 5$ and another with $d = 10$. If the system has $30W$ to increase, it will increase first the server with $d = 10$ to $d = 3$, because it will increase 8.08 Gflops (against 4.5 Gflops from $d = 5$ to $d = 0$). If a server is sleeping, it also considers the power needed to turn the server on. When the step has less power than planned, it runs the following:

1) Reduce the speed of idle servers;
2) Reduce the speed of the servers running jobs with higher $(wall_j - elapTime_j)$.

The second step will reduce servers' speed with more time to compensate for this reduction in the future. It is better to maintain jobs closer to finish with the maximum speed, granting that they will complete.

Table I
SERVER DEFINITION EXAMPLE. THE VALUES ARE FROM GRID5000'S PARASILO SERVER [21], [22]. STATES 14 AND 15 ARE THE TRANSITION STATES BETWEEN ON AND OFF, AND VICE-VERSA.

State	$P_{s,d}$ (W)	$F_{s,d}$ (Gflops)
0	221.77	38.4
1	216.77	37.78
2	213.58	36.93
3	208.90	36.01
4	204.45	34.72
5	200.62	33.90
6	197.28	32.84
7	192.49	31.72
8	184.26	30.63
9	182.04	29.25
10	179.75	27.93
11	176.70	26.37
12	175.53	25.01
13 (sleep)	4.5	0
14 (on→off)	114.12	0
15 (off→on)	164.14	0

V. SIMULATION EXPERIMENT

A. Experiment environment

This section details the experiments, comparing the three compensation policies (*next*, *peak*, and *last*) with a baseline and two reactive algorithms (power reactive and workload reactive). All executions use the same EASY-Backfilling implementation. The baseline execution just applies the offline plan without any modification. The power reactive algorithm does not use the offline but adapts the server configuration using only the renewable source. It defines the server states using the same algorithm from IV-B3. So, this power reactive algorithm will try to maximize the usage of renewable sources. Finally, the workload reactive algorithm also ignores the offline, allocating servers according to the jobs' arrival. When a new job arrives, this algorithm tries to put it in a running server with the Easy-Backfiling scheduling. If it is not possible, it turns on a new server. It turns off a server using the Dynamic power management (DPM) technique [23]. This technique verifies if the server is idle for a defined time. If so, the algorithm turns the server off. We have calculated the time values using the same equation from [24].

We have simulated two data center infrastructures: homogeneous and heterogeneous. The homogeneous infrastructure uses 400 Parasilo servers from Grid5000 [1]. The heterogeneous

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

has 400 servers also, but for eight different types: Dahu, Grvingt, Parasilo, Chifflet, Grisou, Chetemi, Gros, and Graffiti. They have distinct power consumption and speed. The power usage with all servers at maximum speed on homogeneous configuration is around 88 kW and for heterogeneous is about 111 kW. Regarding the speed, the homogeneous' max speed is 15360 Gflop/s and for the heterogenous is 17680 Gflop/s. Thus, the heterogeneous is faster but more power-hungry. The data center values (power and speed) come from previous experiments in Grid5000 [21], [22]. We have simulated this platform in the BATSIM simulator [25], which operates using the SIMGRID framework [26].

Another input for our model is renewable production. We have generated two profiles with the renewable production of three days in the city of Toulouse from the Renewable ninja website [2]. We have used wind speed and irradiation to calculate renewable production, resulting in the profiles from Figure 4. *Offline* creates the offline plan using profile 1. For the *online*, we have executed two experiments. First, *online* receives profile 1 as renewable production, simulating a good weather prediction and offline plan. Second, the *online* receives profile 2, adding uncertainties in the weather prediction. Regarding the workload, we have selected nine different 3-day workloads from MetaCentrum 2 trace [27]. Figure 5 shows the nine different workloads patterns with the execution of EASY-Backfilling with all servers available all the time. We have used as offline workload prediction the workload W5 pattern from the figure. Then, we have defined the initial storage and target storage as $SoC = 50\%$ and $LoH = 300kg$. Using all these inputs, we have calculated the offline plan using the MILP from [12]. This plan is the input for the policies and baseline execution. After that, we have executed all workloads for all algorithms. We have chosen to run only one time window (3 days) because if the algorithms can finish one time window with a good storage level, it can be applied in a continuous execution (with several time windows, one after another).

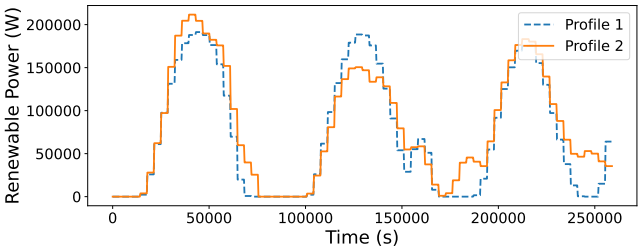

Figure 4. Two different profiles from Toulouse, France.

B. Results

This section presents the results, highlighting some elements of the executions. Our focus is both power and QoS-related. Regarding power, we show the distance from the target level of hydrogen and batteries. The objective is to finish as close to the target as possible. We discuss the QoS metrics presenting the slowdown and number of jobs killed over the different workloads. First, we present the results with a perfect renewable estimation. After, we discuss the execution with different profiles for offline and online.

[2]https://www.renewables.ninja/

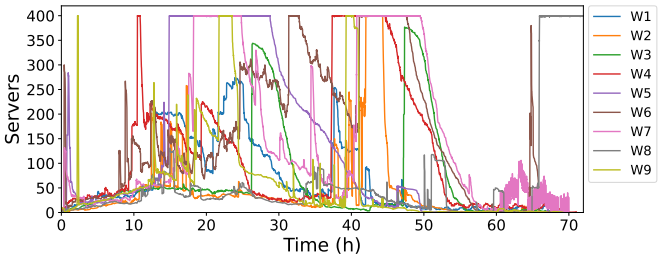

Figure 5. The nine different workloads server usage pattern if all servers are available.

1) Perfect Renewable Estimation: First, this section details the results of a perfect renewable estimation (the estimation is equal to the real). Regarding the storage level, Figure 6 shows the battery and hydrogen level at the end of the time window. The baseline execution has an overall higher storage level than planned since it does not reintroduce the not used planned power. For example, if a server stays idle for some moments, the difference between usage and planned stays in the storage. This behavior explains these results. However, the power reactive execution is even worst. This execution uses mainly renewable power to maintain the servers. The storage is used only when there is no power sufficient to maintain the servers sleeping or going to sleep. The workload reactive algorithm has the second-worst storage management due to its lack of storage awareness. The workload load highly impacts the results of this reactive algorithm. So, it tends to use more storage independently of the power constraints in a high load and low power scenario.

The three policies have better results in storage level. At the Hydrogen level, only the *last* policy has some values different from the target level. The *next* policy has the best storage results. It has almost every workload around 10% of the battery target level and several values with a difference of less than 1%. It achieves these results because it compensates as soon as possible. So, it has time to deal with possible fluctuations. On the other hand, the *last* policy has values quite similar to the baseline since it puts all the compensations in the end. Therefore, it has no time to use this compensation since it needs to run jobs to use power. *Peak* has the second-best storage level at the end of the time window. This policy tends to smooth the peaks, resulting in more constant power over time. However, in the last steps, it could not have enough time to use the power. Regarding homogeneous and heterogeneous infrastructure, it is possible to realize that the results of the *next* and *peak* policies are closer to the target level in the heterogeneous infrastructure. As mentioned before, the heterogeneous infrastructure is more energy-hungry. Therefore, when the policy uses more power to approximate the target

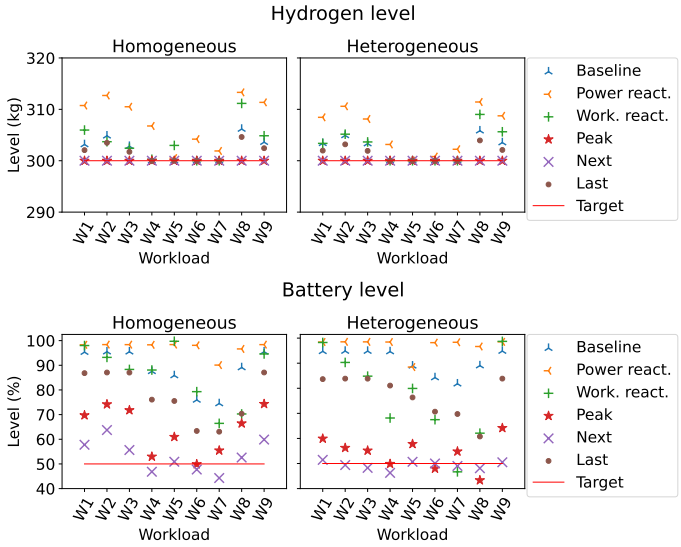

Figure 6. Storage level for each workload at the end of the time window with perfect renewable estimation.

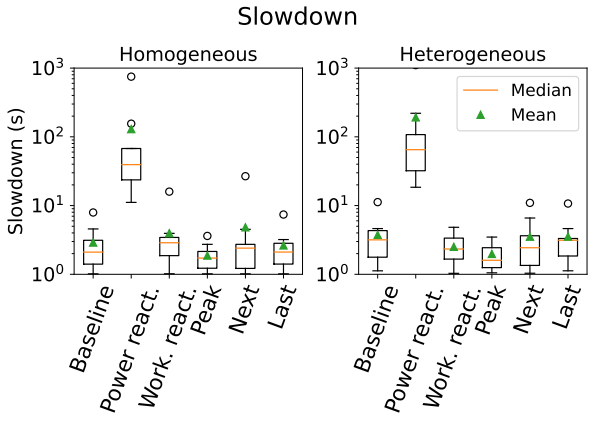

Figure 7. Slowdown over the different workloads with different policies and with perfect renewable estimation.

level, this decision has a higher impact on the battery level in the heterogeneous data center.

Regarding QoS, Figure 7 presents the slowdown over the workloads. The power reactive algorithm has higher slowdown values than all the other algorithms. This algorithm suffers from periods of low renewable production (e.g., at night), putting all servers to sleep during these periods. This behavior makes the jobs wait, increasing the average waiting time and impacting the slowdown. The baseline has good values. However, as mentioned before, it could use more power to improve it. The workload reactive algorithm also has good results. This algorithm is workload-driven, so it will have a good slowdown naturally. Nevertheless, it turns servers according to the job arrival. Thus, sometimes, the jobs must wait until the server finishes the turning-on procedure. The algorithms with offline prediction have some servers waked up and ready to run the jobs. The best slowdown results come from the *peak* policy. It uses the offline estimation to turn on the servers in advance and reintroduce the power difference for improving QoS. As mentioned before, it smooths the power usage curve, which helps to improve the waiting time in the different workloads patterns. The *last* policy has similar results to the baseline because this policy leaves the power at the end. So, it could not improve the QoS so much.

The *next* policy has difficulties with some workloads because it compensates in the first possible step. Figure 8 exemplifies one of these workloads. In the figure highlighted moment, the policy stops all servers because the state of charge is too close to zero. Since the main focus of the policies is to finish with the target SoC, it prefers to stop the servers than start to use more hydrogen. After that, it maintains no server running for a long period, increasing the slowdown. This undesirable effect occurs because the *next* policy tends to compensate near the step when it is demanded. So, if the load is high, the scheduler demands more power, and the power commitment delivers it, compensating for it in the *next* step, where maybe the load continues high. So, when the state of charge is too close to zero, the policy stops all servers, killing the jobs running on it. This behavior affects the QoS, but it is important to notice that, in the end, the SoC is near 50%.

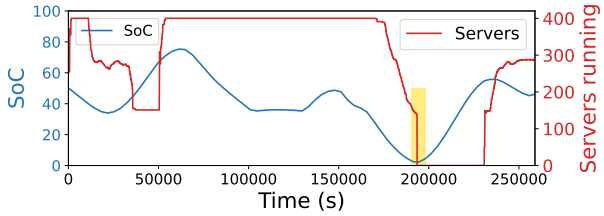

Figure 8. Battery level with *next* policy for workload W4 and heterogeneous server. The highlighted area is the moment when the policy turns off all machines because the SoC is close to a dangerous value.

Finally, Table II shows the total execution of each experiment. The total work to do is calculated using the job size in flops. So, if an algorithm kills a big job, it has a higher impact than a small one. The worst algorithm is the power reactive since it kills the jobs when there is no power to maintain them running. The workload reactive has almost every execution with all jobs finished. It can run more jobs than the others because it does not have power constraints, using as much power as it wants. The policies have quite good results, with several 100% execution. The values below 100% come from the same reason presented in Figure 8. The policies kill jobs and turn off the servers when the SoC is close to a dangerous value. The heterogeneous data center has some values lower than 100% because it can arrive at a low SoC more easily.

2) Different Profiles: While the previous section describes a perfect renewable estimation, this section analyzes the impact of a not-so-good prediction. With a good estimation, the policies can trust in the future values. However, a poor prediction introduces uncertainties in the plan. For example, a policy can define that it will recharge the batteries in a

Table II

TOTAL EXECUTION (IN %) FOR A PERFECT RENEWABLE PREDICTION. EACH COLUMN IS A DIFFERENT WORKLOAD AND INFRASTRUCTURE. THE HIGHER VALUES ARE THE BEST ONES (MAXIMUM OF 100%). WE HIGHLIGHTED THE BEST AND WORST VALUES FOR EACH COLUMN. IN GREEN THE BEST AND RED THE WORST. A KILLED JOB WILL REDUCE THE TOTAL EXECUTION.

	Homogeneous									Heterogeneous								
Execution	W1	W2	W3	W4	W5	W6	W7	W8	W9	W1	W2	W3	W4	W5	W6	W7	W8	W9
Baseline	100	100	100	98.3	100	93.9	97	42.6	100	100	100	100	100	100	99.4	100	47.3	100
Pow.reactive	69.8	66.3	85	57.5	60.9	68.3	70.7	24.6	65.2	75.6	68.4	92.5	64.9	61.1	75.7	70.2	29.3	88.9
Work.reactive	100	100	100	100	100	100	100	62.9	100	100	100	100	100	100	100	100	67.2	100
Peak	100	100	100	100	100	98.4	100	59.4	100	100	100	100	99.6	100	85.8	100	64.3	100
Next	100	100	100	100	100	100	85.6	59.4	100	100	100	100	88.2	100	92.2	99.3	42.9	100
Last	100	100	100	98.9	100	94.2	99.1	59.3	100	100	100	100	100	100	99.7	100	67.5	100

future step. However, this future step could not have enough power incoming. So, it could not recharge as planned. Figure 9 illustrates the results of both storage levels. The battery levels are very high when compared with a perfect estimation. The biggest problem with profile 2 compared with profile 1 is that there is more power in the time window end. So, the policies do not have time to use the difference properly, resulting in a high battery state of charge. However, the *next* and *peak* policies are in general better than the reactive and baseline executions. The baseline execution has almost every battery level at 100% and several hydrogen levels above the target level. Nonetheless, the *next* policy is perfect regarding Hydrogen levels, finishing with 300 kg in every execution. The second-best is the *peak* policy, having only one workload with a slightly high Hydrogen level. The other executions vary according to the workload.

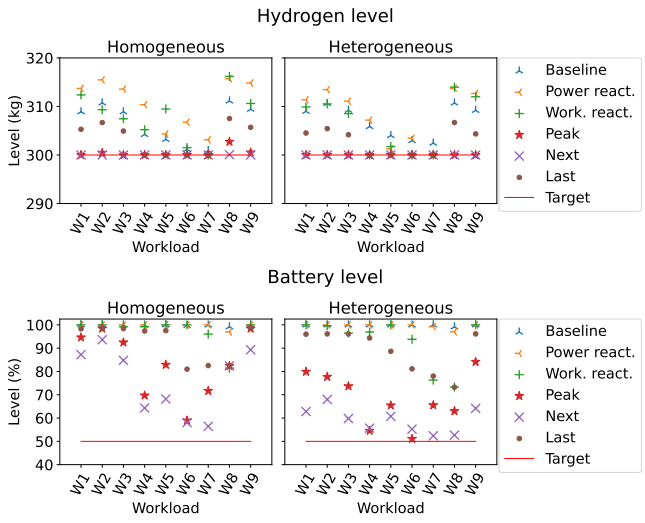

Figure 9. Storage level for each workload at the end of the time window with different online and offline power profiles.

Profile 2 has more energy than profile 1. Table III demonstrates the total work finished using profile 2. The baseline and workload reactive do not change their results since the former only follows the plan independently of the renewable power, and the latter allocates as many servers as possible for the incoming workload. The power reactive improved its overall results because it has more energy to spend. However, it is the worst execution yet. The table also shows that the policies use more energy to increase the total work finished, resulting in 100% in almost every workload (except only W8). These results are very similar to the workload reactive algorithm.

Regarding the slowdown, Figure 10 demonstrates that the results are similar to the execution with a perfect estimation. As the total work finished metric, the baseline and workload reactive algorithms have the same slowdown independently of the power profile. The power reactive algorithm improved the slowdown but is still worst than other values. The *next* policy presents better results now, even better than the *peak* in the homogeneous data center. With more power, the *next* policy could avoid the problem illustrated in Figure 8.

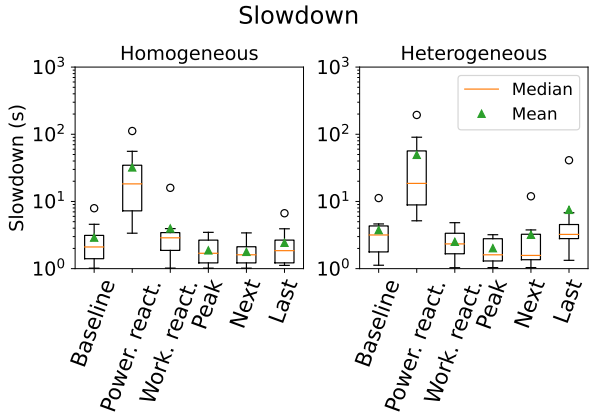

Figure 10. Slowdown over the different workloads with different policies and with a renewable production different from the planned one.

3) Discussion: The previous results demonstrate that using only an offline plan is not enough to deal with power fluctuations. Both workload and renewable production uncertainties demand an online adjustment. The baseline and workload reactive present stable results in QoS, ignoring what is incoming from renewable in their decision-making process. This could seem a good approach since the QoS metrics are good, but it has a huge impact on the power side. When renewable production is superior to the estimation, baseline and workload reactive will not take advantage of it (even if the QoS is not so good). On the other hand, with low renewable production, they will maintain the QoS and use more storage power (maybe drying both batteries and hydrogen). So, in a renewable-only data center, these solutions are not feasible. The power reactive algorithm uses the power only from renewable sources, killing

Table III

TOTAL EXECUTION (IN %) WITH DIFFERENT OFFLINE AND ONLINE POWER PROFILES. EACH COLUMN IS A DIFFERENT WORKLOAD AND INFRASTRUCTURE. THE HIGHER VALUES ARE THE BEST ONES (MAXIMUM OF 100%). WE HIGHLIGHTED THE BEST AND WORST VALUES FOR EACH COLUMN. IN GREEN THE BEST AND RED THE WORST. A KILLED JOB WILL REDUCE THE TOTAL EXECUTION.

	Homogeneous									**Heterogeneous**								
Execution	W1	W2	W3	W4	W5	W6	W7	W8	W9	W1	W2	W3	W4	W5	W6	W7	W8	W9
Baseline	100	100	100	98.3	100	93.9	97	42.6	100	100	100	100	100	100	99.4	100	47.3	100
Pow.reactive	77.4	67.7	86.3	54.3	60.3	75.9	78.7	36.7	59.9	85.7	70.2	93.4	60.9	60.4	82.7	75.7	41.2	71.1
Work.reactive	100	100	100	100	100	100	100	62.9	100	100	100	100	100	100	100	100	67.2	100
Peak	100	100	100	100	100	100	100	59.3	100	100	100	100	100	100	100	100	67.5	100
Next	100	100	100	100	100	100	100	59.3	100	100	100	100	100	100	100	100	67.5	100
Last	100	100	100	100	100	100	100	59.3	100	100	100	100	100	100	100	100	67.5	100

jobs even if it has power in batteries. On the power side, it does not reintroduce the power from servers' idleness. This algorithm is not a good solution either.

The policies present a simple implementation and good balance between the storage target level and QoS, mainly the *peak* policy. This policy achieves good storage levels (not the best ones but around the best values) and the best QoS overall performance. Regarding QoS, it has the best slowdown (see Figures 7 and 10) and has a good percentage of total completed work (see Tables II and III). The policies can perform better than the other algorithms because they awake the servers in advance. The *last* policy puts the compensation at the end, having less time to use the power and resulting in similar QoS performance as the baseline. The *next* suffers from the problem presented in Figure 8. It could arrive at a dangerous SoC level during a high load moment, killing jobs and impacting slowdown. However, if the *next* policy can avoid these moments, it is even better than the *peak* (as demonstrated in the executions with different profiles). Of course, in an unpredictable lower renewable production the policies will degrade QoS to maintain the storage levels. However, this is the expected behavior in a renewable-only data center.

On the power side, *next* has the best results. This policy compensates as soon as possible, which gives time to apply the difference between planned and real. The *last* is the worst policy concerning storage level because it is the opposite of the *next*. *Peak* has mid-term values between *next* and *last*. *Peak* behavior reduces the Figure 8 problem while compensating in a step where there is time to use the power. Then, this policy uses the best behaviors of the two others. The experiments lead us to explore, in the future, other perspectives. For example, we could introduce another policy that compensates in the moments with a higher deficit between the workload demanded load and the provided power. However, this behavior could suffer from wrong workload estimations. While *peak* tries to smooth the curves, this workload policy could put too much power in a step that maybe will not have this much demand. Future experiments will introduce this policy. Finally, the best implementation could be a mix of the previous algorithms, changing the policy according to the system's state.

VI. CONCLUSION

A renewable-only data center introduces several elements, such as batteries, hydrogen, wind turbines, and solar panels. It demands a plan for the following days using workload and weather predictions. However, just following this plan may not be sufficient. This work presented a model for online adaptations to change an offline plan, aiming to improve QoS and deal with power fluctuations. The results demonstrated that simply following the offline plan or only reacting to the online events is not enough due to the variance in workload and renewable production. The experiments also revealed that the presented policies could approximate the target storage level and improve QoS when possible. Future works will include other policies linked to workload estimation (e.g., increase power when workload prediction shows that more jobs would arrive), a mix of policies, more complex QoS improvements (e.g., improve slowdown), and a more complete job description (I/O, network communication, etc).

REFERENCES

[1] F. Bordage, "Empreinte environnementale du numérique mondial," *Paris, greenit. fr*, p. 9, 2019.

[2] 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.

[3] Q. Zhou, M. Xu, S. S. Gill, C. Gao, W. Tian, C. Xu, and R. Buyya, "Energy efficient algorithms based on vm consolidation for cloud computing: comparisons and evaluations," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 489–498.

[4] M. A. Khan, A. P. Paplinski, A. M. Khan, M. Murshed, and R. Buyya, "Exploiting user provided information in dynamic consolidation of virtual machines to minimize energy consumption of cloud data centers," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2018, pp. 105–114.

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

[6] 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

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

[8] 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

[9] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2001, pp. 188–205.

[10] C. Jiang, C.-L. Tseng, Y. Wang, Z. Lan, F. Wen, F. Chen, and L. Liang, "Optimal Pricing Strategy for Data Center Considering Demand Response and Renewable Energy Source Accommodation," *Journal of Modern Power Systems and Clean Energy*, pp. 1–9, 2021, conference Name: Journal of Modern Power Systems and Clean Energy.

[11] 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.

[12] M. Haddad, J. M. Nicod, C. Varnier, and M.-C. Peéra, "Mixed integer linear programming approach to optimize the hybrid renewable energy system management for supplying a stand-alone data center," in *2019 Tenth international green and sustainable computing conference (IGSC)*. IEEE, 2019, pp. 1–8.

[13] X. Liu, P. Liu, H. Li, Z. Li, C. Zou, H. Zhou, X. Yan, and R. Xia, "Energy-aware task scheduling strategies with QoS constraint for green computing in cloud data centers," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, ser. RACS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 260–267. [Online]. Available: http://doi.org/10.1145/3264746.3264792

[14] Y. Lu, R. Wang, P. Wang, Y. Cao, J. Hao, and K. Zhu, "Energy-Efficient Task Scheduling for Data Centers with Unstable Renewable Energy: A Robust Optimization Approach," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Jul. 2018, pp. 455–462.

[15] Y. Li, A.-C. Orgerie, and J.-M. Menaud, "Balancing the Use of Batteries and Opportunistic Scheduling Policies for Maximizing Renewable Energy Consumption in a Cloud Data Center," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, Mar. 2017, pp. 408–415, iSSN: 2377-5750.

[16] R. Hunger, *Floating point operations in matrix-vector calculus*. Munich University of Technology, Inst. for Circuit Theory and Signal ..., 2005.

[17] 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.

[18] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Proceedings. International Conference on Parallel Processing Workshop*. IEEE, 2002, pp. 514–519.

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

[20] 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.

[21] G. da Costa, "MojitO/S," Nov. 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03453537

[22] ——, "Keynote Lecture: Performance and Energy models for modern HPC servers," Jun. 2022. [Online]. Available: https://pdco2022.sciencesconf.org/resource/page/id/5

[23] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 299–316, 2000.

[24] 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.

[25] 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.

[26] 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.

[27] D. Klusáček, Š. Tóth, and G. Podolníková, "Real-life experience with major reconfiguration of job scheduling system," in *Job scheduling strategies for parallel processing*. Springer, 2015, pp. 83–101.