Compas'2022 : Parallélisme/ Architecture/ Système
MIS/UPJV - Amiens France, 5-8 juillet 2022

Smart Heuristics for Power Constraints in Data Centers Powered by Renewable Sources

Igor Fontana de Nardin[1 2], Patricia Stolf[1], Stephane Caux[2]

Université de Toulouse, Toulouse
[1]Institut de Recherche en Informatique de Toulouse (IRIT)
[2]Laplace UMR5213
{igor.fontana,patricia.stolf}@irit.fr, stephane.caux@laplace.univ-tlse.fr

---

Résumé
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).

Mots-clés : Power constraints, Job Scheduling, Data center, Renewable energy, Heuristics

---

1. Introduction

The information and Communications Technology (ICT) sector has a significant impact on the global greenhouse gas (GHG), generating 2.1-3.9% of the global emissions [1, 8]. 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 [8]. Therefore, Internet providers can not overlook its carbon footprint by just increasing the number of resources to deal with the predicted expansion of Internet usage by 2023 [4]. One of the most crucial elements in providing Internet services is the large-scale data centers [17]. Data centers are a notable power waste actor using around 1% of worldwide electricity [17]. Hence, the IT community has started to investigate other power supply possibilities, such as renewable energy utilization [17]. Renewable energy employs self-renewing resources, such as wind and sunlight, providing clean energy, but introducing uncertainties due to weather conditions. Cloud providers, such as Google and Amazon insert grid connection as a way to deal with renewable sources uncertainties [14]. As renewable energy becomes a solution to the data center's carbon footprint reduction, several works deal with this topic in the

literature. Some works aim to maximize renewable energy source usage but use grid (brown) energy to add reliability [11, 18]. This connection removes the power constraints since they could use the energy from the grid when there is no renewable power available. Several works deal with processor power capping to control energy costs [12, 10, 26, 19]. 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). Few articles detail global power constraints [25, 7]. However, they have limitations such as the need to profile jobs' power and execution time relationship, no power fluctuations, equal power distribution independently of jobs state, or no QoS evaluation. Datazero2 Project [20] designed a data center operated only by renewable sources without any link to the grid. 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 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. Finally, it also presents experiments to evaluate the algorithms' performance with different workloads and renewable power production. This paper is organized as follows : Section 2 presents the model and algorithms. Section 3 explains the experimental setup. Section 4 analyzes the experimental results, and Section 5 concludes the article.

2. Model and algorithms

2.1. Model

Given a data center with S servers, each server s has an list of states D_s which 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. This article considers a global power constraint coming from an electrical module dealing with sources' commitment. 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. The model only knows the P_t^{avai} of the actual time step. 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. 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, which 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.

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

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

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

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 [23]. The scheduling finds 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. Finally, Equation 6 demonstrates the objective function of the scheduling. The objective is to minimize the Slowdown [9, 2]. 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.

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

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

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

2.2. Server configuration algorithms

2.2.1. Idle Fewest Server Degradation (IFS)

The main idea of this heuristic is maintaining more machines at the fastest speed. 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). 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 the power needed to maintain this state is lower or equal to the available.

2.2.2. Idle Servers Balanced Degradation (ISB)

This algorithm is similar to the previous one, and it is similar than the proposed by [7]. The objective of ISB is to reduce the speed of the servers equally, maintaining more running machines. Server Balanced Degradation also has two variations : Idle Descending Servers Balanced Degradation (IDSB) and Idle Ascending Servers Balanced Degradation (IASB). This algorithm makes a Round-Robin between the servers, changing the server in each interaction.

2.2.3. Highest Flops Lowest Deadline (HFLD)

HFLD is a heuristic that tries to find the best speed for the data center improving first the running machines. The heuristic estimates the changes with the greatest impact on the data center flops inside the power available. Its main idea is presented in algorithm from Appendix B. Differently from IFS and ISB, this algorithm decides the improvements based on the previous state. HFLD has a list of all transitions power and flops, so it can fastly decide which one will impact the most on the data center flops. The function more_power() will go through the list searching the highest flops improvement inside the power available. However, if the power usage is greater than the power available, it first puts the idle servers to sleep. Then, it reduces the processor speed ($F_{s,d}$), while $(W_j - Et_j) \times F_{s,d} \geqslant Fl_j - Ef_j$ is true, taking jobs with a higher distance from walltime first. If these changes are not sufficient to match the power available, the algorithm reduces the speed of the servers running the jobs.

2.3. Scheduling algorithms

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) [13] and Easy Backfilling (EBF) [16, 9, 2]. 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 [13]. Backfilling algorithm, on the

other hand, uses the queue as a priority queue [16]. 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 [9, 2] : First-Come-First-Serve (FCFS), Descending size, Ascending size, and Slowdown.

3. Experimental Environment

The platform consists of 10 Paravance servers and 10 Taurus servers from GRID5000 [1]. Appendix C presents the values for each server state. The parameters are representative and quite similar to other works [3, 24, 21]. This platform runs in the BATSIM simulator [6]. Appendix D demonstrates the three power profiles representing renewable energy collected from solar panels and wind turbines at Toulouse. We have used these values as P_t^{avail} for each step. Regarding the workload, we have used a generator from [5]. This generator is based on Google trace [22]. The workload generator creates 20 different workloads with 288 jobs, but they vary in size and arrival. The first experiment runs every workload using power profile 3. The first results of Section 4 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 4.1.4.

4. Results

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.

4.1. Discussion

We have used three metrics to compare the algorithms : power violations, jobs killed, and slowdown. These metrics are discussed individually in the following sections. Appendix E presents the average result of all executions using Profile 3. From these executions, we have selected four algorithms : EBF Slowdown HFLD, FF Slowdown HFLD, FF Asc IDFS, and EBF Slow IDSB. Both HFLD algorithms have a good balance between the three metrics. FF Asc IDFS also has a good balance but with a higher number of jobs killed. Finally, EBF Slow IDSB has the best slowdown but with high violations. We solved the same problem as the heuristics using Mixed-integer linear programming (MILP). It fully knows both power profile and job arrivals, maximizing Equation 3. 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 1 shows the average distance from each algorithm to the result obtained by the MILP for each workload. The table also presents the results of the execution of EBF Slowdown without server configuration, maintaining all servers at maximum speed.

4.1.1. Power Violation

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.

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

TABLE 1 – 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

MILP could avoid violations by using the entire power profile to decide the configuration in each time step. The heuristics provide a good alternative, dealing with the power fluctuations reactively. However, it is not perfect, having some violations. These violations come from a high power available in one step and a drop in the next one. With high power, they put several servers to run. But if the power drops, the servers demand time and energy to go to sleep. Therefore, one violation occurs. Appendix F shows it with MILP and HFLD. However, few violations could be removed using a prediction technique to analyze the power tendencies.

4.1.2. 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 overall energy usage. The user could reintroduce these jobs into the system using more energy (our scheduling does not do this automatically). Figure 1 shows the number of jobs killed over the 20 workloads. The execution with no server configuration does not kill any job since it maintains all servers at maximum speed. 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 workloads above 30 jobs killed. First-Fit achieves better results in the jobs killed metric because it runs more small jobs than Easy-Backfilling, due to the Easy-Backfilling priority awareness. IDFS execution has high values of this metric, with its best result between 20 and 25. Finally, IDSB results are very different, demonstrating that this algorithm struggles to deal with varied workloads. Figure 1 also illustrates the total finished work over the 20 different workloads. Each job killed impacts differently on this value since they have different sizes. The size is 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.

4.1.3. Slowdown

Finally, the last metric is the slowdown. Figure 1 details the value of each execution. This metric is important since 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. This metric shows the impact of the power capping in QoS. IDSB has good results, because it maintains more servers 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

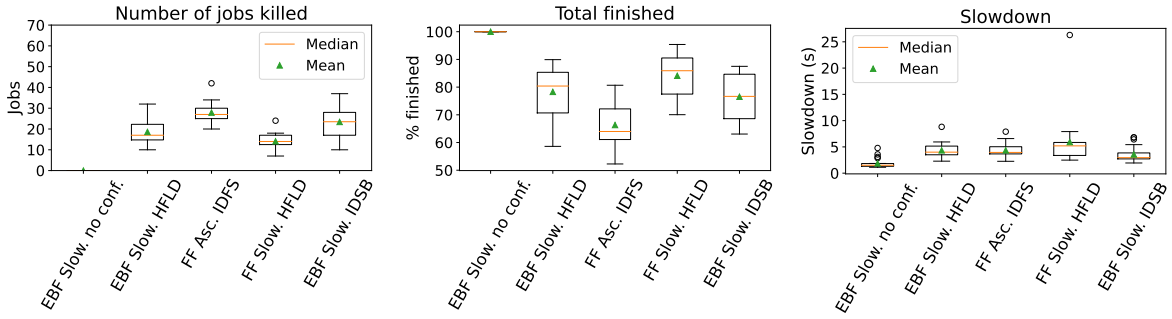

FIGURE 1 – The number of jobs, total of finished work and slowdown over the 20 executions for Profile 3.

small jobs first. So, the jobs more waiting time-sensitive are the priority. First-Fit Slowdown HFLD 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.

4.1.4. Multiple profiles

This section presents the experiments using one workload with the different profiles from Appendix D to compare the robustness of our algorithms. Figure 2 illustrates the results. IDFS kills a large number of jobs. The HFLD has the best number of jobs killed using both Easy Backfilling and First-Fit for all profiles. However, the First-Fit has a greater slowdown on profile 2. Easy Backfilling Slowdown HFLD is a quite robust implementation, independent of the profile.

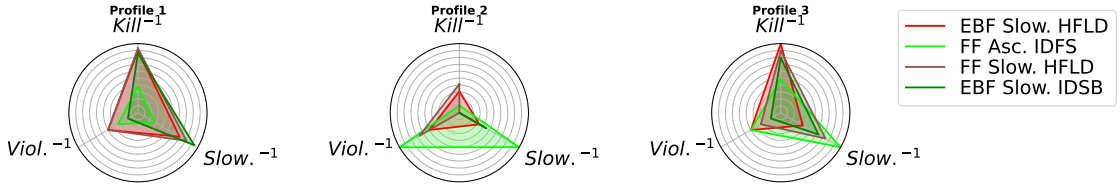

FIGURE 2 – One workload execution on different power profiles. The power violation range is between 0 and 6. The number of jobs killed metric is between 15 and 44. Finally, Slowdown values are between 4.04 and 11.60.

5. 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.

6. Acknowledgment

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

Bibliographie

1. Bordage (F.). – Empreinte environnementale du numérique mondial. *Paris, greenit. fr*, 2019, p. 9.
2. Carastan-Santos (D.), De Camargo (R. Y.), Trystram (D.) et Zrigui (S.). – 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)*, pp. 1–10. IEEE, 2019.
3. Caux (S.), Renaud-Goud (P.), Rostirolla (G.) et Stolf (P.). – It optimization for datacenters under renewable power constraint. – In *European Conference on Parallel Processing*, pp. 339–351. Springer, 2018.
4. Cisco (U.). – Cisco annual internet report (2018–2023) white paper, 2020.
5. Da Costa (G.), Grange (L.) et De Courchelle (I.). – Modeling, classifying and generating large-scale google-like workload. *Sustainable Computing : Informatics and Systems*, vol. 19, 2018, pp. 305–314.
6. Dutot (P.-F.), Mercier (M.), Poquet (M.) et Richard (O.). – Batsim : a realistic language-independent resources and jobs management systems simulator. – In *Job Scheduling Strategies for Parallel Processing*, pp. 178–197. Springer, 2015.
7. Ellsworth (D. A.), Malony (A. D.), Rountree (B.) et Schulz (M.). – Dynamic power sharing for higher job throughput. – In *SC'15 : Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11. IEEE, 2015.
8. Freitag (C.), Berners-Lee (M.), Widdicks (K.), Knowles (B.), Blair (G.) et Friday (A.). – The climate impact of ict : A review of estimates, trends and regulations, 2021.
9. Gaussier (E.), Lelong (J.), Reis (V.) et Trystram (D.). – Online tuning of easy-backfilling using queue reordering policies. *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, n10, 2018, pp. 2304–2316.
10. Ha (T. M.), Samejima (M.) et Komoda (N.). – Power and performance estimation for fine-grained server power capping via controlling heterogeneous applications. *ACM Trans. Manage. Inf. Syst.*, vol. 8, n4, aug 2017.
11. Haghshenas (K.), Taheri (S.), Goudarzi (M.) et Mohammadi (S.). – Infrastructure aware heterogeneous-workloads scheduling for data center energy cost minimization. *IEEE Transactions on Cloud Computing*, 2020.
12. Hankendi (C.), Reda (S.) et Coskun (A. K.). – Vcap : Adaptive power capping for virtualized servers. – In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, *ISLPED '13*, ISLPED '13, p. 415–420. IEEE Press, 2013.
13. Knuth (D. E.). – The art of computer programming. vol. 1 : Fundamental algorithms. second printing, 1969.
14. Kwon (S.). – Ensuring renewable energy utilization with quality of service guarantee for energy-efficient data center operations. *Applied Energy*, vol. 276, 2020, p. 115424.
15. Lannelongue (L.), Grealey (J.) et Inouye (M.). – Green algorithms : Quantifying the carbon footprint of computation. *Advanced Science*, vol. 8, n12, 2021, p. 2100707.
16. Lifka (D. A.). – The anl/ibm sp scheduling system. – In *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 295–303. Springer, 1995.

17. Masanet (E.), Shehabi (A.), Lei (N.), Smith (S.) et Koomey (J.). – Recalibrating global data center energy-use estimates. *Science*, vol. 367, n6481, 2020, pp. 984–986.
18. Nayak (S. K.), Panda (S. K.), Das (S.) et Pande (S. K.). – An efficient renewable energy-based scheduling algorithm for cloud computing. – In *International Conference on Distributed Computing and Internet Technology*, pp. 81–97. Springer, 2021.
19. Park (J.), Park (S.) et Baek (W.). – 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, CCGrid '18*, CCGrid '18, p. 41–50. IEEE Press, 2018.
20. Pierson (J.-M.), Baudic (G.), Caux (S.), Celik (B.), Da Costa (G.), Grange (L.), Haddad (M.), Lecuivre (J.), Nicod (J.-M.), Philippe (L.), Rehn-Sonigo (V.), Roche (R.), Rostirolla (G.), Sayah (A.), Stolf (P.), Thi (M.-T.) et Varnier (C.). – DATAZERO : DATAcenter with Zero Emission and RObust management using renewable energy. *IEEE Access*, vol. 7, juillet 2019, p. (on line).
21. Raïs (I.), Orgerie (A.-C.), Quinson (M.) et Lefèvre (L.). – Quantifying the impact of shutdown techniques for energy-efficient data centers. *Concurrency and Computation : Practice and Experience*, vol. 30, n17, 2018, p. e4471.
22. Reiss (C.), Wilkes (J.) et Hellerstein (J. L.). – Google cluster-usage traces : format+ schema. *Google Inc., White Paper*, vol. 1, 2011.
23. Takizawa (S.) et Takano (R.). – 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*, pp. 169–178, 2020.
24. Villebonnet (V.), Da Costa (G.), Lefèvre (L.), Pierson (J.-M.) et Stolf (P.). – Energy aware dynamic provisioning for heterogeneous data centers. – In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 206–213. IEEE, 2016.
25. Wang (B.), Schmidl (D.), Terboven (C.) et Müller (M. S.). – Dynamic application-aware power capping. – In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, pp. 1–8, 2017.
26. Zhang (H.) et Hoffmann (H.). – Podd : Power-capping dependent distributed applications. – In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–23, 2019.

Appendices

A. Experiments carbon footprint

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 [15]).

B. Highest flops with lowest deadline algorithm

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: $D_{s,d,t}$

begin
  $D_{s,d,t} \Leftarrow D_{s,d,t-1}$;
  if *calculate_power_usage()* $= P_t^{avai}$ **then**
    break;
  end
  if *calculate_power_usage()* $< P_t^{avai}$ **then**
    return more_power();
  end
  $D_{s,d,t} \Leftarrow$ put_idle_servers_sleep();
  if *calculate_power_usage()* $< P_t^{avai}$ **then**
    return $D_{s,d,t}$;
  end
  for *j in J.sort()* **do**
    $D_{s,d,t} \Leftarrow$ minimal_state(j);
    if *calculate_power_usage()* $< P_t^{avai}$ **then**
      return $D_{s,d,t}$;
    end
  end
  repeat
    j $\Leftarrow$ get_highest_deadline(J);
    $D_{s,d,t}$, _sleep $\Leftarrow$ reduce_speed(j);
    if _sleep **then**
      $D_{s,d,t}$ = kill_job(j);
    end
    if *calculate_power_usage()* $< P_t^{avai}$ **then**
      return $D_{s,d,t}$;
    end
  until $J = \emptyset$;
  return $D_{s,d,t}$;
end

C. Server example

TABLE 2 – 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

D. Power profiles

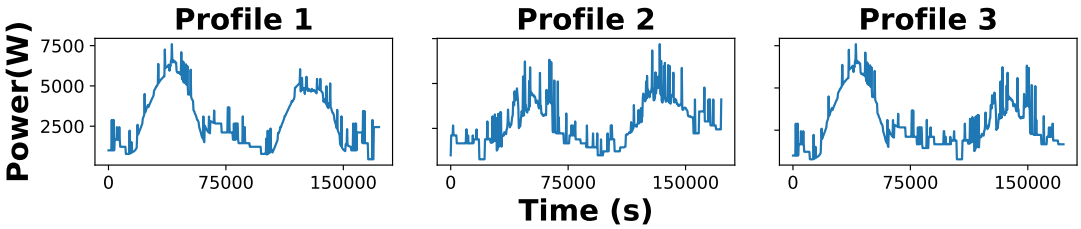

FIGURE 3 – Power profiles representation of the experiments.

E. Experimental results

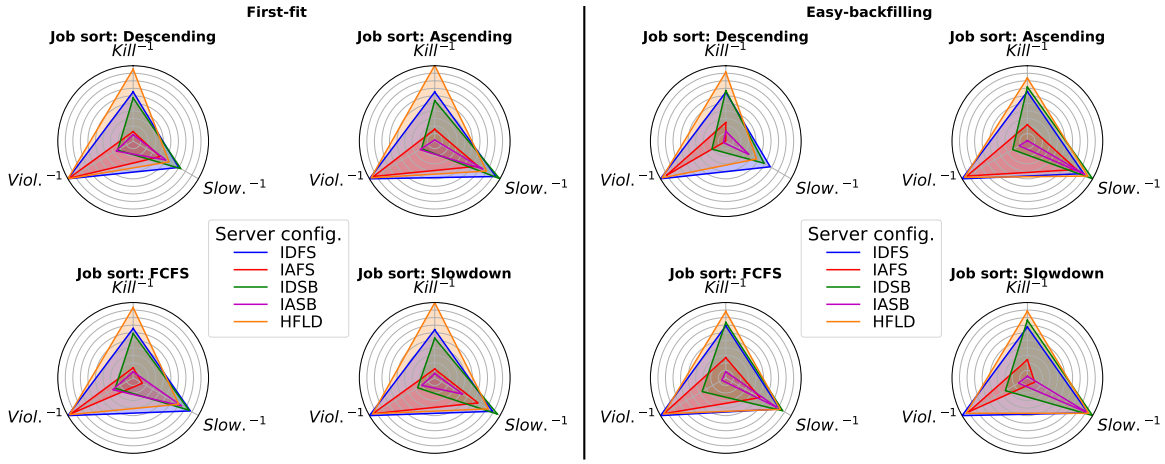

FIGURE 4 – Average results for the 20 different workloads with Profile 3. The values are inverted (so the $^{-1}$), generating a higher area to lower values (which are the best). The power violation range is between 3 and 7. The number of jobs killed metric is between 14 and 54. Finally, Slowdown values are between 3.5223 and 16.9373.

F. Power violation

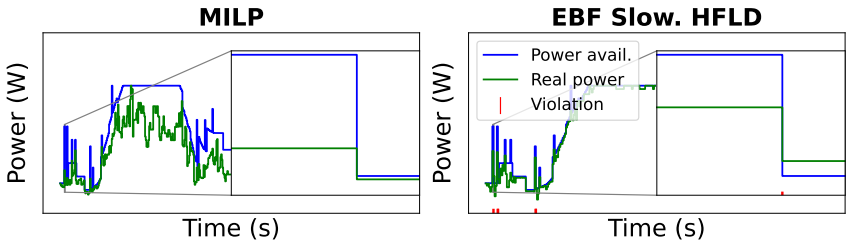

FIGURE 5 – A zoomed moment with a violation. MILP avoids all violations, while HFLD avoids most of them.