PADS: Power Budgeting with Diagonal Scaling for Performance-Aware Cloud Workloads

Mehmet Savasci*, Abel Souza†, David Irwin*, Ahmed Ali-Eldin‡ and Prashant Shenoy*
*University of Massachusetts Amherst, USA
†University of California Santa Cruz, USA
‡Chalmers University of Technology, Sweden

***Abstract*—Cloud platforms' rapid growth raises significant concerns about their electricity consumption and resulting carbon emissions. Power capping is a known technique for limiting the power consumption of data centers where workloads are hosted. Today's data center computer clusters co-locate latency-sensitive web and throughput-oriented batch workloads. When power capping is necessary, throttling only the batch tasks without restricting latency-sensitive web workloads is ideal because guaranteeing low response time for latency-sensitive workloads is a must due to Service-Level Objectives (SLOs) requirements. This paper proposes PADS, a hardware-agnostic workload-aware power capping system. Due to not relying on any hardware mechanism such as RAPL and DVFS, it can keep the power consumption of clusters equipped with heterogeneous architectures such as x86 and ARM below the enforced power limit while minimizing the impact on latency-sensitive tasks. It uses an application-performance model of both latency-sensitive and batch workloads to ensure power safety with controllable performance. Our power capping technique uses diagonal scaling and relies on using the control group feature of the Linux kernel. Our results indicate that PADS is highly effective in reducing power while respecting the tail latency requirement of the latency-sensitive workload. Furthermore, compared to state-of-the-art solutions, PADS demonstrates lower P95 latency, accompanied by a 90% higher effectiveness in respecting power limits.**

***Index Terms*—Diagonal Scaling, Power Cap, Demand Response, DVFS.**

I. Introduction

The increasing power consumption of data centers is a major global concern due to its financial and environmental impacts. Currently, data centers and networking account for nearly 3% of the world's electricity consumption [1], with forecasts predicting a threefold increase by the end of this decade [2], [3]. In order to amortize costs, data center operators must carefully plan for long-term timeframes when planning new deployments. As such, they need to consider both the power delivery infrastructures and the future computational demands. In addition, many data centers still rely on energy from polluting sources, which generate greenhouse gas emissions and contribute to climate change [3], [4]. While meeting certain environmental regulations, operators must also account for the substantial electricity cost variations necessary to run their equipment during the lifespan of the data center. Considering these issues' importance and contemporary relevance, it is essential to control and optimize the power consumption of data centers flexibly.

Data center providers have employed power oversubscription to improve efficiency and lower costs. Power budgeting is a well-known technique for managing the electricity consumption of data centers by capping its computing demand [5]. Since advancements in the design of data centers have significantly improved the efficiency of cooling and power distribution infrastructures, the majority of the electricity used in modern data centers is consumed by servers. As such, most techniques implement power budgeting through capping mechanisms that consolidate, throttle [6], migrate (to and across different locations), and scale [7] workloads to meet the power budget limits at the data center level.

Controlling power consumption requires both system and workload flexibility. This flexibility can be realized in multiple ways, though most techniques utilize hardware mechanisms such as Running Average Power Limit (RAPL) [8] and Dynamic Voltage and Frequency Scaling (DVFS) [9], as well as software techniques like power-aware scheduling [10] and cluster-level power management [11]. Research has underscored the limited efficacy in reducing server power consumption across many modern architectures with hardware mechanisms. Although they provide an application-agnostic approach to reduce power consumption, the primary limitation of hardware techniques is that power limits are applied to entire CPU sockets rather than individual applications. This results in non-optimal power allocation, especially impacting multi-tier services distributed across multiple servers. Other recent research points out that aggressive power optimization can increase the risk of higher tail latency and degrade application performance. This is because power optimization techniques are typically SLO-unaware [7], and workloads tend to be highly sensitive to dynamic changes in power allocation, which indirectly affect both performance and power consumption, making the simultaneous control of both parameters a complex task [8]. Certain techniques assume batch workloads are best-effort – i.e., utilize every resource available to it – and can better handle performance variations. For instance, Thunderbolt [6] uses CPU bandwidth control to prioritize latency-sensitive workloads by reducing the throughput of batch workloads, effectively allocating more power to the former at the expense of the latter.

Although batch workloads have good flexibility in terms of Service-Level Objectives (SLOs), it is important to prioritize and minimize the impact that resource allocation causes on the tail latency of web workloads. One solution for managing dynamic web workloads is to use elastic scaling, where resources allocated to the web service are dynamically adjusted to match workload variations [12]. Although many cloud providers support elastic scaling [13], [14], [15], methods tend to *overprovision* resources because they consider only a single dimension when scaling, often through horizontal scaling. The target is to handle the peak workload seen in a provisioning step and to reduce SLO violations while avoiding frequent re-provisioning decisions. If variations in workloads is not effectively captured, this results in over-provisioning that accounts for forecasting errors. When vertical scaling is employed, it allows for precise resource provisioning, though at significantly lower levels, to accommodate unexpected peak workload demand. This also results in overprovisioning, as new application replicas consume time to be instantiated, potentially leading to SLO violations and high power usage due to the non-proportionality of computing power consumption [16].

The most significant challenges related to server throttling to implement power budgeting and that requires further investigation involve power-performance tradeoffs. In addition, there is still a lack of models describing the relationship between server configurations and application performance, together with software techniques necessary for optimizing multiple co-located applications. To address these issues, we propose `PADS` (**P**ower-**A**ware **D**iagonal **S**caler), a power budgeting technique that combines horizontal and vertical scaling—referred to as *diagonal scaling*—along with an application power-performance model to cap the total power consumption of servers while respecting application SLOs. Unlike prior work that has used diagonal scaling, `PADS` utilizes application information such as workload rate or computational stage to estimate the precise level of resource bandwidth to allocate to applications. By leveraging these application models, we can meet the SLO targets of high-priority web services and effectively minimize the performance degradation impact on lower-priority, batch jobs. The key idea of `PADS` is that similar CPU bandwidth configurations can result in *different* application performances and *varying* power consumption levels, revealing a new search space to be exploited in implementing power cap policies. In implementing `PADS`, we make the following contributions:

- We showcase the empirical effects of CPU bandwidth allocation when provisioning resources on power consumption and latency (§II).
- We introduce `PADS`'s design that combines diagonal scaling with application power-performance models to cap the power consumption of servers. Our approach leverages analytical profiling power and performance models to jointly meet the workload levels, application SLOs, and power budgets (§III).
- We implement a prototype of `PADS` (§IV) and evaluate it using realistic applications and workloads. Our results show that power efficiency can reach near-optimal levels of 89% while avoiding power cap violations. (§V).
- In addition, we showcase `PADS`'s adaptive properties for Demand-Response use-cases that can be robust to power cap changes without violating the limit while achieving lower response time by 20% compared to baseline.
- We open-source `PADS` for reproducibility[1].

II. Background

This section provides background on latency-critical web services, diagonal scaling techniques, power budgeting and capping techniques, and the motivation of our system.

A. Interactive Services

Interactive workloads are low latency services that require user round-trip times – i.e., the time from request submission to response – to be within certain levels of operation. For applications such as web apps or mobile apps, the round-trip performance of requests is directly experienced by an end user. As such, keeping the latency of requests under specific target – i.e., SLO requirement – is one of the key challenges in increasing user satisfaction. Several studies have shown that high tail latency can significantly increase customer abandonment rates [17], [18], [19]. For instance, one study found that even small increases in response times can lead to a one percent reduction in e-commerce sales [19]. One of the primary causes of high tail latency is the complexity of modern software stacks and the resource management of intricate workflows. Scheduling delays [20], multi-tenancy [21], energy optimizations [22], and bad resource allocation configurations [17] can all introduce significant and random delays in request execution, causing requests to be served orders of magnitude slower than average [20]. To avoid pitfalls when controlling the response time of latency-sensitive workloads, it is common to over-provision applications' resource needs [14].

B. Resource Scaling

Managing the performance within a data center depends primarily on allocating resources and power to applications. While statically over-provisioning resources can mitigate tail latency, it's not a financially sustainable approach, especially for services with millions of users. Thus, cloud data centers continuously seek cost-efficient techniques to maintain low tail latency despite dynamic request patterns. Auto-scaling is a key technique to reduce the wastage of resources. Several factors related to the auto-scaling mechanism can impact the performance of cloud services. One critical factor is the configuration of CPU utilization thresholds, which governs the activation of auto-scaling policies for adding or removing resources. Another essential consideration is the scaling step size, representing the number of instances configured during each provisioning process to accommodate workload spikes. Significant research efforts have been dedicated to examining various aspects, including investigations into vertical and horizontal resource scaling [6], and the tradeoffs between power

[1]The code available at: https://github.com/umassos/PADS

and performance in systems operating under power constraints or aiming to reduce overall footprint.

C. Power Budgeting

The exponential trends in energy needs have generated growing scrutiny regarding the energy consumption of datacenters. Consequently, numerous cloud providers and data center operators have heightened their focus on fostering energy-efficient and sustainable practices such as power budgeting, where server power capping has emerged as a key solution. Power capping is used to limit a server's power consumption to stay within a specific power budget. This approach allows data center operators to reduce peak power consumption at the cost of potential performance degradation for hosted applications. Traditionally, Dynamic Voltage and Frequency Scaling (DVFS) is used to reduce CPU power consumption by lowering the voltage and frequency. Recently, RAPL [23] has been proposed as an alternative that enables direct control over the power consumption of a server's CPU and memory. Extensive research has explored the implementation of power capping through resource scaling – both vertically and horizontally –, as well as through optimizations involving the trade-offs between power allocation and application performance [6].

Performance degradation often occurs when the power consumption of servers is throttled. The extent of performance degradation varies between applications and also depends on the workload experienced at the throttling time. Additionally, different workloads may hold varying levels of importance to the data center operator, who prioritizes certain groups of applications when power budget is insufficient to run servers at full capacity. Schedule mechanisms can use operating system tooling to regulate the amount of processing power applications access from available system resources. For instance, the Linux cgroups [24] feature can be used to limit and prioritize certain groups of processes running in a server while controlling the CPU bandwidth they have. In addition, it can vertically scale a process by dynamically scaling the resources applications can access. Surprisingly, there has been relatively limited exploration into the interactions between scaling parameters, specifically the simultaneous coordination of vertical and horizontal resource allocation and scaling with power allocation and application performance.

D. Motivation

The key insight of our research is that resource bandwidth can lead to application underperformance and increased power consumption, even when allocations are equivalent. Conversely, power budgeting mechanisms used by cloud providers are typically agnostic to the application workload, with caps enforced through resource profiling that may result in budget violations (§V). This creates an opportunity to optimize power budgeting by using resource bandwidth mechanisms that integrate application performance and power models to jointly minimize performance degradation and meet datacenter, and workload SLOs.

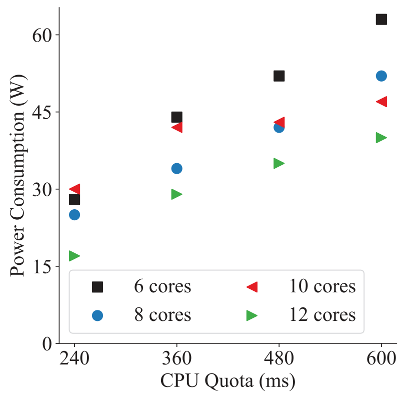


Fig. 1: Power consumption of a compute intensive workload across equivalent nominal resource bandwidths (CPU quota) and different resources (cores). Increasing CPU bandwidth does not necessarily lead to higher power consumption.

Figure 1 illustrates the power consumption of a compute-intensive workload [25] running in a 16 cores server. For each core assignment (in number of cores), we assign a Linux cgroup's CPU quota (in ms), reporting the average power consumption (W) for each combination. We can make two key observations. First, for each CPU quota, the power consumption ranks across the different core assignments are different. This means that power consumption not only depends on the number of cores it uses, but also on the CPU time each core has. For example, while the 12-core assignment (right-arrow) consistently has the lowest power consumption across all CPU quotas, the 6, 8, and 10-core assignments repeatedly change ranks, with up to 5× power consumption variations between the lowest (18W) and largest (65W) CPU-quotas. Second, increases in core allocation do not necessarily result in increased in power consumption. For instance, an increase in 30% in CPU quota bandwidth results in 20% increase in power consumption for the 8-core assignments (circles) while having negligible effects on the 10-core assignment (left-arrows). This means that improvements in application performance may be achieved while keeping the resulting power consumption constant. These observations show that power budget strategies that employ auto-scaling techniques should not only consider simple core-scaling techniques, but also the processing time since it directly affects application end performance.

Key Insight. Applications' power consumption does not necessarily align with resource bandwidth allocation. Unlike simple auto-scaling techniques, power budgeting mechanisms that integrate application power and performance models through diagonal scaling can enhance power consumption optimization tradeoffs by up to 5× while maintaining equivalent levels of SLOs.

III. System Design

A. System Overview

`PADS` is a power budgeting system for data centers that implements power capping employing diagonal scaling alongside application profiles to devise power-performance models and enabling it to explore a wider range of resource provisioning options that can simultaneously maintain the data center's total power consumption within budget and meet application SLOs.

PADS includes three phases. First, the performance and power models are devised based on application profile data, similar to how it is presented in §II. We use first-order regression [26] to build accurate models. Latency sensitive services are profiled according to the impact their different workload levels have on resource utilization, which are then mapped to power consumption. Conversely, batch applications are profiled by submitting them to various resource quotas, mapping them to both workload throughput and power consumption[2]. Second, once the analytical power-performance models are devised, PADS searches sets of CPU bandwidth and resource assignment combinations that reduce power consumption below the power budget. In this operation, PADS ensures that the picked combination satisfies power and performance constraints for all applications. Although multiple policies could be evaluated when choosing which combination to apply, here, we greedily sort the final set by combinations that minimize the performance degradation of batch workloads. Finally, once the final power-performance combination is devised, PADS applies diagonal scaling by dynamically adjusting number of cores along with their CPU bandwidth times. To minimize uncertainty in CPU performance and fluctuations in response times due to reduced power levels, the PADS utilizes a configurable buffer to adjust the total data center power budget. This enables a safety net for priority workloads that are sensitive to changes in resource and power allocations.

B. Algorithm Design

Algorithm 1 details how PADS decides how resource scaling happens for each applications while keeping the total power consumption under an enforced power cap. The algorithm accepts two inputs and uses two system-wide available data. The first input is the power change. It denotes the difference between the system's power cap and active power consumption. The second input is application classes. Cloud datacenters host multiple applications and pack them into the same server as much as possible for efficiency. However, each application has different performance requirements. Thus, certain applications are prioritized in using resources. For example, latency-sensitive web applications might be given a priority in utilizing more resources than batch applications due to their strict responsiveness requirements to users. Therefore, it is essential to put applications into different priority classes so that when PADS takes an action, it does not hurt the performance of high-priority class applications. Rather, PADS revokes resources from the less priority applications. The first system-wide available data is the information on CPU utilization of applications. PADS uses this information to meet SLO requirements of applications, if any. Here, we note that we put applications with SLO requirements into

[2]Our power regression models achieve accuracy over 90%.

Algorithm 1: Find Resource Change

```
Input: $power\_change$, $app\_classes$
Data: $app\_cpu\_utils, workload\_level$
Output: $resource\_change$
1  $resource\_change \leftarrow \emptyset$
2  $residual\_power \leftarrow power\_change$
3  foreach $class \in app\_classes$ do
4      foreach $app \in class$ do
5          if $residual\_power > 0$ then
6              $candidates, nominees \leftarrow \emptyset, \emptyset$
7              $candidates \leftarrow candidates \cup$
                 power_to_cpu_resource($\frac{power\_change}{|class|}$)
8              foreach $cand \in candidates$ do
9                  $new\_cpu\_util \leftarrow$
                     compute_cpu_util( $app_{cpu\_usage}$, $cand$)
10                 $eval\_result \leftarrow$
                     eval_cpu_utils($new\_cpu\_util$, $app\_cpu\_utils$, $workload\_level$)
11                 if ($eval\_result$) then
12                     $residual\_power \leftarrow$
                         $residual\_power - (\frac{power\_change}{|class|})$
13                     $nominees \leftarrow nominees \cup$
                         app_perf_model($cand$)
14             $resource\_change \leftarrow$
                 $resource\_change \cup$ min($nominees$)
15     return $resource\_change$
```

the highest application classes to exclude these applications from resource deflation. Moreover, PADS uses workload level change information that is specific to latency-sensitive web applications. Making PADS proactive by utilizing workload change information improves its decision-making process.

Our algorithm follows classical MAPE loop structure [27]. In the **M**onitor and **A**nalysis phase, we collect system-wide power consumption information and compare it with the system-wide applied power cap. After that, we move to **P**lanning phase to decide what actions should be taken to keep the power consumption under the cap continuously (Algorithm 1). Here, we have two cases: we reduce the given CPU cores and CPU quota from the applications starting from the lowest-priority application classes if the system power consumption exceeds the cap. Otherwise, we add resources to the applications starting from the lowest-priority application classes. In both cases, our algorithm behaves applications placed in the same class uniformly distributed. For example, suppose the $power\_change$ is X Watts, and there is n number of applications in the application class. In this case, we change CPU cores along with their CPU bandwidth such that the selected resource combination accounts for $\frac{X}{n}$ Watts per application according to the application power model. Here,

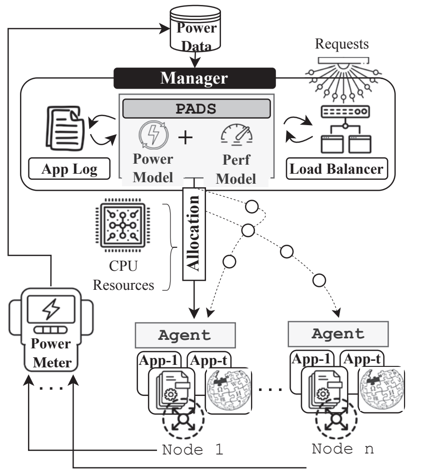


Fig. 2: Architecture of PADS. Arrows going outwards from the circle on nodes show resource changes of applications.

PADS might have multiple CPU core and quota combinations. Thus, we keep all these combinations in *candidates* set (line 7). Among these candidates, we filter the ones whose effect on overall CPU utilization does not make the applications with SLO violate their performance requirements (lines 8-11). After that, PADS computes and compares the performance effects of chosen candidates on the application and picks the one with the least performance impact (lines 13-14). Here, PADS also ensures that the selected candidate does not lower the application's resource under its minimum limit to guarantee that application stays responsive. Our system switches to **E**xecute step as a final step. In this step, planned resource changes are applied to each application accordingly.

IV. Architecture and Implementation

In this section, we describe the architecture and implementation of PADS. The prototype of PADS is developed with approximately 1.3KLOC of Python.

Figure 2 shows how PADS's major components interact with each other. When it makes a decision and subsequently takes action to keep the total power consumption below the power limit, it considers the outcomes of its decisions on the performance of latency-sensitive applications. Thus, PADS is an application performance-aware power capping system. Next, we describe each of the major components of PADS.

Manager. The manager runs on a group of dedicated nodes, monitors data from the agents under their control, and makes power-capping decisions.

Agent. The agent is a lightweight program running on every node in the cluster. It executes CPU resource decisions made by the manager.

PADS manager reads power measurements from the ePDU using a Simple Network Management Protocol (SNMP) every second. Similarly, it reads the performance metric of the latency-sensitive web application, i.e., 95th response time, every second. We were down to a 1-second resolution due to the criticality of frequently monitoring power consumption and application performance. This small monitoring interval

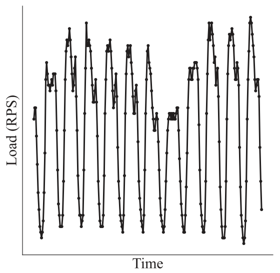


Fig. 3: Wikipedia workload trace for evaluation.

allows PADS to make more stabilized and quicker power-capping decisions. When the manager decides on a new CPU core and quota of applications, the power consumption limits are also considered. After the decision is made, it sends this information to agents running on each node. The *agent* is responsible for setting the given resource limits by the manager. It applies the resource changes by using the Linux cgroup feature. This feature limits the application from using all resources except whatever is given to itself. The agent also provides the manager crucial telemetry data about the applications, such as CPU utilization. For communicating, the manager and agent communicate via gRPC calls [28].

V. Experimental Setup and Evaluation

In this section, we first describe the setup for our experiments, including the real-world applications and workload traces. Next, we evaluate our PADS in respecting the given power cap limit while maintaining the 95th response time performance metric of latency-sensitive application. Then, we compare PADS against Thunderbolt, one of the state-of-the-art power capping systems. In all results, we normalize the power consumption to the highest power consumed when power is not restricted.

A. Experimental Setup

Testbed Setup. Our testbed includes Dell PowerEdge R440 server running Ubuntu 20.04 LTS. The server has a two-socket Intel Xeon Silver CPU, 8 cores each, 2.10GHz, and 64GB of memory. We disable hyperthreading and turbo boost because they affect the comparisons, regardless of the approach used. Moreover, we enable the performance governor of CPUFreq driver [29]. We deploy our applications inside of the LXC container. As a power meter, we use CyberLink Switched Metered-by-Outlet PDU [30]. This power meter provides outlet-level power monitoring in real time.

Applications. We use two application classes: high-priority and low-priority. We put a latency-sensitive cloud application: MediaWiki [31] into the high-priority class, while batch-application: BLAST [25] into the other one. We co-locate them when we deploy. MediaWiki is an open-source wiki software platform that hosts a replica of Wikipedia. It is a traditional LAMP stack software. We use the pre-built version of the German Wikipedia, which comprises 10 GB of content. We use the HAProxy load balancer for the latency-sensitive application to collect application arrival rate and response time data. We use BLAST as a batch application. It is a

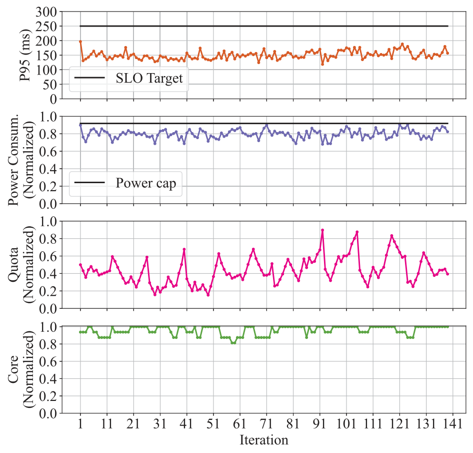


Fig. 4: PADS mechanism. Power consumption is limited while considering p95 latency.

bioinformatics program for finding regions of local similarity between nucleotides of DNA and/or RNA sequences or amino-acid sequences of proteins.

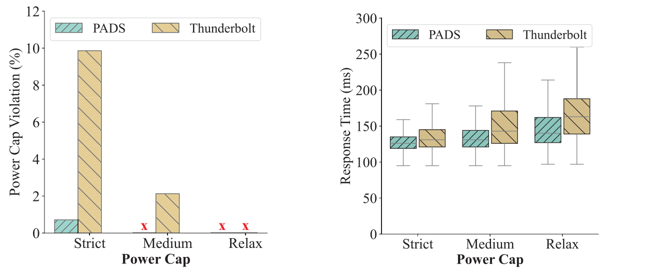
Workloads. In our experimental setup, we use Mediawiki real-world workload trace [32]. shown in Figure 3. We scaled traces by considering the serving capacity of our setup. When submitting these workloads, we issued requests using an open-loop system model [33], which creates a new request whether a response to previous requests is received. We use the httpmon workload generator [34] to emulate users accessing applications.

Metric Collection. We characterize the response time of a Mediawiki request as the time length from the moment the request is dispatched to the moment the request output response is received. Here, we specifically focus on the CPU request processing time. In other words, we do not consider the requests' network latency. We collect Mediawiki's P95 response times at 1-second granularity.

B. Validation

Our first set of experiments aims to validate and show how PADS can maintain the P95 response time below the target while keeping the power consumption under the power cap.

Results. In this experiment, we sampled and used the workload shown in Figure 3. Figure 4 depicts results for PADS under this workload. We empirically set the target SLO response time as 250 ms, the power cap as 92% of the maximum power consumption under an unrestricted environment, and measured the 95th response time of the MediaWiki application. Moreover, we show how PADS changes CPU cores and quota resources allocated to the BLAST application. According to results, PADS achieves zero power cap violation and an average of 150ms P95 latency, well below the SLO target due to accurate Mediawiki workload predictions and CPU core and quota allocations. Overall, PADS effectively applies a power-capping mechanism jointly with performance awareness to respect power cap and performance targets.



(a) Power Cap Violation (b) Mediawiki Response Time

Fig. 5: PADS versus Thunderbolt: x on (a) shows no violation.

C. Comparison with State-of-the-Art

We implement and evaluate Thunderbolt [6] for comparison. It is a reactive power capping system that employs an off-the-shelf Linux CPU bandwidth mechanism to throttle batch applications by changing their quota limits when total power consumption exceeds a given power limit.

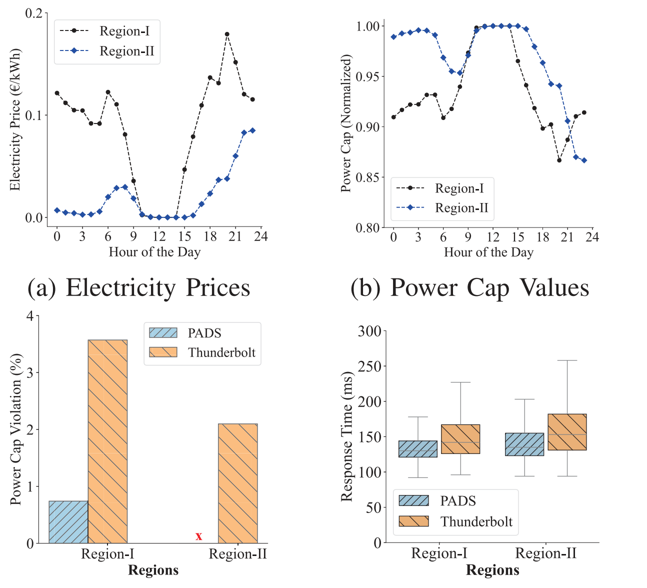
Thunderbolt maintains two capping thresholds, each with a multiplier. In our experiment, we used the same parameters presented in the Thunderbolt paper. In addition, we set the complete unthrottling duration to 10 seconds. This means Thunderbolt gives 10% of resources back to the BLAST application every second if the system is in the unthrottling stage.

Results. Figure 5 presents the power cap violation and latency results for PADS and Thunderbolt. As shown in Figure 5(a), the higher power availability across all policies enables lower power cap violations. As the power cap targets reduce, the impacts of using application model information to make decisions are more noticeable, negatively affecting Thunderbolt but PADS. Since Thunderbolt does not use any application information, such as workload changes of the Mediawiki application, its reaction features do not take the required actions promptly to avoid power cap violations. Noticeably, PADS avoided more than 90% power cap violations for the string power capping setting. Figure 5(b) shows the response time distribution across different power cap settings. Overall, as the power cap gets relaxed, response time distribution increases across both systems because they give more resources to the BLAST application, and this puts pressure on the Mediawiki, but PADS still performs better up to 20% and keeping response time under 250ms. This is because of PADS' better control over response time variability, resulting in more predictable performance.

Key Takeaway. Despite excluding latency-sensitive applications from throttling for power reduction reasons, power-capping mechanisms still tend to degrade their performance while enforcing power limits. Thus, application information should be integrated into the power-capping system to have a more control on performance.

D. Use-case: Demand-Response (DR)

Demand-response (DR) programs aim to address stability challenges by incentivizing power consumers to regulate their



(a) Electricity Prices (b) Power Cap Values

(c) Power Cap Violation (d) Mediawiki Response Time

Fig. 6: Power cap violation and Mediawiki performance under demand-response use case.

consumption within predetermined timescales. By participating in these programs, consumers, such as datacenters, can lower their energy expenses and contribute to grid stability and sustainability. However, significant challenges, such as hefty economic penalties due to power management errors, can arise when operating under limited and fluctuating power availability. [35].

In this experiment, we evaluate how PADS and Thunderbolt work under dynamic power cap changes. We use electricity price data for two different regions located in Europe, shown in Figure 6(a) [36]. We generated power cap data shown in Figure 6(b) from the electricity price graph by taking an inverse proportion of the prices and normalizing between the strict and relaxed power cap settings. In this use-case scenario, the datacenter adjusts its maximum power usage in real time. It wants to use less electricity when the price is high and vice versa. For this reason, we see from Figure 6(b) that when the electricity price is high, the power cap value is lower, and when the price is low, the power cap value is higher. We expect Region-I to be more challenging than Region-II because of the variability in changes.

Results. Figures 6(c) and (d) depicts the results of power cap violation and response time of Mediawiki application for PADS and Thunderbolt. For the Region-I, PADS achieves 75% less power violation than Thunderbolt, while it doesn't have any violation for the Region-II. Across all regions, PADS achieves 10% to 15% better response time than Thunderbolt.

Key Takeaway. Power capping systems should be resistant to changes in power cap over time. PADS shows robustness to the changes while it respects the performance requirement of the latency-sensitive applications.

VI. Related Work

Power capping is a technique to guarantee power consumption does not exceed the user-defined bound [37]. [38] compared the effectiveness of various power capping mechanisms, including DVFS and RAPL. Dynamo [39] is a power management system that uses a three-band algorithm for power capping and uncapping decisions and Intel RAPL to enforce the given power limit. CapMaestro [40] manages power oversubscription by proposing a global priority-aware algorithm to protect high-priority workloads from power throttling while maintaining a minimum performance guarantee for low-priority workloads. Thunderbolt [6] is the other power capping system that throttles the CPU shares of throughput-oriented workloads using Linux cgroup features to stay within the specified power budget while ensuring that latency-sensitive tasks remain unaffected. Microsoft also designs a workload-aware power capping system [10] for the Azure platform employing per-core DVFS and RAPL mechanisms. PARM [41] is an adaptive resource allocation framework under a power capping system. Its difference from PADS is that it re-allocates resources to preserve SLO under the cap. That is, a separate system already does power capping on top of PARM. [42] uses the DVFS boosting technique at scale to quickly and safely expand the computational capacity by providing additional power capacity in an oversubscribed environment.

VII. Conclusion

This paper proposes an application performance-aware power capping system that integrates horizontal and vertical scaling ideas, PADS. We call it diagonal scaling. We designed our system to keep the power consumption of the system under power limit while adhering to the application performance – e.g., 95th percentile latency (P95). To achieve this goal, PADS incorporates the application power-performance models into power cap decisions to dynamically allocate CPU core and quotas in response to power consumption and workload fluctuations of the latency-sensitive web application. In particular, we combine the analytical power-performance model of batch applications and the workload rates of latency-sensitive web applications to make power capping decisions. Finally, we prototype and evaluate our system against a state-of-the-art power capping system under real-world workloads and applications. Our findings indicate that PADS can achieve power to optimal levels, reaching up to 89%, besides avoiding power limit exceedings. Moreover, compared to state-of-the-art solutions, PADS attains lower, controllable P95 latency without power cap violations. Finally, PADS is robust in avoiding power cap violations and controlling p95 latency under scenarios like power cap dynamic change.

Acknowledgment

We thank the reviewers for their valuable comments, which improved the quality of this paper. This research has been supported by the Republic of Türkiye Ministry of National Education YLSY program, NSF grants 23091241, 2213636, 2211302, 2211888, 2105494, and DoE grant DE-EE0010143.

References

[1] V. Rozite, E. Bertoli, and B. Reidenbach, "Data centres and data transmission networks," *IEA*, 2023. [Online]. Available: https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks

[2] S. Bangalore, A. Bhan, A. D. Miglio, P. Sachdeva, V. Sarma, R. Sharma, and B. Srivathsan, "Investing in the rising data center economy," https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/investing-in-the-rising-data-center-economy, 2023.

[3] J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén, "Ict sector electricity consumption and greenhouse gas emissions–2020 outcome," *Telecommunications Policy*, vol. 48, no. 3, p. 102701, 2024.

[4] T. Sukprasert, A. Souza, N. Bashir, D. Irwin, and P. Shenoy, "On the limitations of carbon-aware temporal and spatial workload shifting in the cloud," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 924–941.

[5] J. Krzywda, A. Ali-Eldin, E. Wadbro, P.-O. Östberg, and E. Elmroth, "Alpaca: Application performance aware server power capping," in *2018 IEEE International conference on autonomic computing (ICAC)*. IEEE, 2018, pp. 41–50.

[6] S. Li, X. Wang, F. Kalim, X. Zhang, S. A. Jyothi, K. Grover, V. Kontorinis, N. Narodytska, O. Legunsen, S. Kodakara *et al.*, "Thunderbolt:{Throughput-Optimized},{Quality-of-Service-Aware} power capping at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1241–1255.

[7] M. Savasci, A. Souza, L. Wu, D. Irwin, A. Ali-Eldin, and P. Shenoy, "Slo-power: Slo and power-aware elastic scaling for web services," in *24th IEEE/ACM international Symposium on Cluster, Cloud and Internet Computing (CCGrid 24)*, 2024.

[8] M. Savasci, A. Ali-Eldin, J. Eker, A. Robertsson, and P. Shenoy, "Ddpc: Automated data-driven power-performance controller design on-the-fly for latency-sensitive web services," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 3067–3076.

[9] L. Zhou, L. N. Bhuyan, and K. Ramakrishnan, "Gemini: Learning to manage cpu power for latency-critical search engines," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 637–349.

[10] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura *et al.*, "{Prediction-Based} power oversubscription in cloud platforms," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 473–487.

[11] P. Patel, E. Choukse, C. Zhang, Í. Goiri, B. Warrier, N. Mahalingam, and R. Bianchini, "Polca: Power oversubscription in llm cloud providers," *arXiv preprint arXiv:2308.12908*, 2023.

[12] M. Eriksen, K. Veeraraghavan, Y. Abdulghani, A. Birchall, P.-Y. Chou, R. Cornew, A. Kabiljo, M. Lieuw, J. Meza, S. Michelson *et al.*, "Global capacity management with flux," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 589–606.

[13] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand *et al.*, "Autopilot: workload autoscaling at google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[14] H. Qian, Q. Wen, L. Sun, J. Gu, Q. Niu, and Z. Tang, "Robustscaler: Qos-aware autoscaling for complex workloads," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2762–2775.

[15] Z. Wang, S. Zhu, J. Li, W. Jiang, K. Ramakrishnan, Y. Zheng, M. Yan, X. Zhang, and A. X. Liu, "Deepscaling: microservices autoscaling for stable cpu utilization in large scale cloud systems," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 16–30.

[16] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[17] J. Dean and L. A. Barroso, "The Tail at Scale," *Commun. ACM*, vol. 56, no. 2, Feb. 2013.

[18] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann, "Online Controlled Experiments at Large Scale," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1168–1176.

[19] M. Belshe, "More Bandwidth Doesn't Matter (Much)," 2010, https://bit.ly/3RUykxs.

[20] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble, "Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency," in *Symposium on Cloud Computing (SoCC)*. ACM, 2014.

[21] Q. Wang, Y. Kanemasa, J. Li, C.-A. Lai, C.-A. Cho, Y. Nomura, and C. Pu, "Lightning in the Cloud: A Study of Very Short Bottlenecks on n-Tier Web Application Performance," in *Proceedings of USENIX Conference on Timely Results in Operating Systems*, 2014.

[22] B. Vamanan, H. B. Sohail, J. Hasan, and T. Vijaykumar, "Timetrader: Exploiting Latency Tail to Save Datacenter Energy for Online Search," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 585–597.

[23] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, 2010, pp. 189–194.

[24] T. L. K. documentation, "Control groups," https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html, accessed: 2024-9-29.

[25] "Basic Local Alignment Search Tool," https://blast.ncbi.nlm.nih.gov/Blast.cgi, Accessed August 2024.

[26] A. Karalič and I. Bratko, "First order regression," *Machine learning*, vol. 26, pp. 147–176, 1997.

[27] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin, *Practical guide to the IBM autonomic computing toolkit*. IBM Redbooks, 2004.

[28] "grpc," https://grpc.io/, accessed: 2024-9-29.

[29] "Cpu frequency and voltage scaling code in the linux(tm) kernel," https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt, accessed: 2024-9-30.

[30] "Cyberpower," https://www.cyberpowersystems.com/product/pdu/switched-mbo/pdu81102/, accessed: 2024-9-29.

[31] "Mediawiki," https://www.mediawiki.org/wiki/MediaWiki, accessed: 2024-9-29.

[32] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[33] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *3rd Symposium on Networked Systems Design & Implementation (NSDI 06)*. USENIX Association, 2006.

[34] "httpmon," https://github.com/cloud-control/httpmon, accessed: 2024-9-29.

[35] A. K. Coskun, F. Acun, Q. Clark, C. Hankendi, and D. C. Wilson, "Data center demand response for sustainable computing: Myth or opportunity?" in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–2.

[36] "Nord pool — data portal," https://data.nordpoolgroup.com/auction/day-ahead/prices, accessed: 2024-9-29.

[37] B. Rountree, D. H. Ahn, B. R. De Supinski, D. K. Lowenthal, and M. Schulz, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 947–953.

[38] P. Petoumenos, L. Mukhanov, Z. Wang, H. Leather, and D. S. Nikolopoulos, "Power capping: What works, what does not," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2015, pp. 525–534.

[39] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, "Dynamo: facebook's data center-wide power management system," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 469–480. [Online]. Available: https://doi.org/10.1109/ISCA.2016.48

[40] Y. Li, C. R. Lefurgy, K. Rajamani, M. S. Allen-Ware, G. J. Silva, D. D. Heimsoth, S. Ghose, and O. Mutlu, "A scalable priority-aware approach to managing data center server power," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 701–714.

[41] H. Qiu, L. Zhang, C. W. H. Franke, Z. T. Kalbarczyk, and R. K. Iyer, "Parm: Adaptive resource allocation for datacenter power capping," in *Annual Conference on Neural Information Processing Systems*, 2023.

[42] L. Piga, I. Narayanan, A. Sundarrajan, M. Skach, Q. Deng, B. Maity, M. Chakkaravarthy, A. Huang, A. Dhanotia, and P. Malani, "Expanding datacenter capacity with dvfs boosting: A safe and scalable deployment experience," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2024, pp. 150–165.