DDPC: Automated Data-Driven Power-Performance Controller Design on-the-fly for Latency-sensitive Web Services

Mehmet Savasci
University of Massachusetts Amherst
USA

Ahmed Ali-Eldin
Chalmers University of Technology
Sweden

Johan Eker
Ericsson Research & Lund University
Sweden

Anders Robertsson
Lund University
Sweden

Prashant Shenoy
University of Massachusetts Amherst
USA

Abstract

Traditional power reduction techniques such as DVFS or RAPL are challenging to use with web services because they significantly affect the services' latency and throughput. Previous work suggested the use of controllers based on control theory or machine learning to reduce performance degradation under constrained power. However, generating these controllers is challenging as every web service applications running in a data center requires a power-performance model and a fine-tuned controller. In this paper, we present DDPC, a system for *autonomic* data-driven controller generation for power-latency management. DDPC automates the process of designing and deploying controllers for dynamic power allocation to manage the power-performance trade-offs for latency-sensitive web applications such as a social network. For each application, DDPC uses system identification techniques to learn an *adaptive* power-performance model that captures the application's power-latency trade-offs which is then used to generate and deploy a *Proportional-Integral (PI) power controller with gain-scheduling* to dynamically manage the power allocation to the server running application using RAPL. We evaluate DDPC with two realistic latency-sensitive web applications under varying load scenarios. Our results show that DDPC is capable of *autonomically generating and deploying controllers* within a few minutes reducing the active power allocation of a web-server by more than 50% compared to state-of-the-art techniques while maintaining the latency well below the target of the application.

Keywords

Web service performance, power-management, datacenter

ACM Reference Format:
Mehmet Savasci, Ahmed Ali-Eldin, Johan Eker, Anders Robertsson, and Prashant Shenoy. 2023. DDPC: Automated Data-Driven Power-Performance Controller Design on-the-fly for Latency-sensitive Web Services. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543507.3583437

1 Introduction

Today, our world relies on web services running on massive cloud data centers. In the US alone, it is estimated that cloud data centers consume around 70 Billion kWh per year [50], the equivalent of 7 million households. It is expected that this number will grow significantly in the coming decade to support the computing demands. Even small inefficiencies in the power consumption of these data centers can translate to large costly resource, environmental, and financial waste. Hence, there has been significant work directed towards enhancing the energy efficiency of cloud data centers, with techniques ranging from resource management [29, 45], to better chip architectures [37] and power capping.

Power capping is a technique used by many data centers to limit their total power consumption from going above a pre-defined power threshold. Capping is typically achieved using CPU power management features such as Dynamic Voltage and Frequency Scaling (DVFS) [20] and Running Average Power Limit (RAPL) [54]. There are two traditionally used power reduction techniques: DVFS or RAPL. DVFS is a technique to control power consumption of processors by scaling up and down the voltage (and frequency) of the processor. RAPL is a technology on Intel processors that allows monitoring and controlling the average power that a processor consumes [39]. Capping is enforced when there is either an overload on the (expensive) data center power equipment, or to reduce the electricity consumption to avoid violations of the power agreements the data center operator has with the power provider [16, 57]. Much prior work has focused on power-capping for throughput-oriented or batch workloads [33, 41, 54], with far fewer efforts focusing on latency-sensitive workloads. One major issue with latency sensitive workloads is the non-linear performance degradation in latency with reduced power allocation [27] which can lead to severe performance degradation. Hence, many researchers have suggested the use of power-performance controllers for latency-sensitive workloads using, e.g., control theory [20, 28, 36] or machine-learning [59]. However, one of the main shortcomings of these approaches is that they require a different fine-tuned control model and large amounts of training data for each web application, each workload mix [51], and each possible server configuration [21]. In addition, the power-performance model may need to be updated or re-trained every time the application is updated, e.g., when any components of the application are upgraded or modified. A further complication is that training and optimizing a controller requires substantial manual effort in collecting training data and tuning the controller. Ideally, the process of controller generation should be automated, allowing for full autonomy for the management of the power-performance controller generation and management.

In this paper, we look for an answer to the question of *can we automate the process of power-performance controller generation for web-services such that the generated controllers reduce power-allocation while meeting SLOs and provide equal/better performance compared to state-of-the-art techniques?* As an answer, we present DDPC, a Data-Driven Power-performance management system that autonomically manages all steps of power-performance management of data centers, from controller generation to power allocation for latency-sensitive workloads. Our system enables data center operators to decrease the total data center power allocation while maintaining the required Service-Level-Objectives (SLOs) for these workloads. To do so, DDPC autonomically benchmarks each application and models its power-performance tradeoffs using an ARX model–a popular model from control theory used in system identification [31]. The model is then used to automatically build Proportional-Integral (PI) controllers with gain scheduling [31] to control power allocation under a wide set of load conditions for the application using RAPL. Our controllers maintain the average response time well below the required SLO under power caps. When the system operates under no power-cap, the controllers minimize the tail response time, while reducing the overall power allocation in comparison to state-of-the-art solutions. DDPC deploys the controllers automatically in the cluster, and in case of deviations from the SLO, the controllers are automatically retrained and redeployed online. Our main contributions can be summarized as follows.

- We design and implement DDPC, an autonomic power management system that manages the power-performance tradeoffs of web applications.
- DDPC generates ARX models for the power-latency tradeoffs. The models are used to generate PI controllers with gain scheduling for power allocation. The controllers reduce the overall active power allocation with up to 50% while maintaining the SLO.
- We implement a prototype of DDPC in a Linux cluster, testing the framework with different workloads and applications. For reproducibility, we open source our framework.

2 Background

2.1 Cloud Data Centers

Today, many internet services run in large-scale data centers. Operating these data center is expensive with a large part of the operational cost coming from their massive power-consumption. In Europe, a recent study estimates that data centers power consumption will double in five European countries, with an estimate that it will consume up to 25% of the total power consumption in Ireland by 2030 [8]. In the US, it is estimated that using better power and energy management techniques could yield saving of over 25 billion kWh [50]. This massive power consumption (and wastage) increases the carbon footprint of these systems.

Since most web-clusters run at low utilization [26], there is an ever increasing interest to reduce their power consumption to reduce the energy waste of data centers. Techniques based on control theory, statistical learning, and deep-learning [34, 40, 42, 57, 59] for managing the power-allocation have been suggested. Since CPU power is the largest contributor to power consumption in a server [1], most work on power management focus on reducing the consumption of CPU power using techniques such as Dynamic voltage and frequency scaling (DVFS), and Running Average Power Limit (RAPL). DVFS enables system administrators to set the voltage used by a CPU, reducing the power consumption of the CPU, and hence the server. RAPL on the other hand enables a system administrator to set the average power consumption of a CPU to a certain wattage. While these two techniques reduce the power usage of servers considerably, they affect the performance of latency-sensitive web workloads—potentially reducing performance by up to 60% when the application is running independently, and by up to 80% when co-located [17, 30].

2.2 Web service workloads

Web services such as social networking, web search, email service, and online shopping are latency-sensitive with any latency increases leading to user dissatisfaction. Hence, many research efforts focus on reducing the response times (both average and tail) of web applications [11, 25, 48, 53]. However, many of these latency optimization solutions involve replication [11, 46], which increases the overall power consumption of running the same workload.

Web QoS measures. There are multiple measures of QoS for web services. Latencies of Web pages have three main components; the server response time, the network latencies, and the client load/rendering times. Server response times are calculated from when a request is received at the front-end of the web service to the time it leaves the front-end. Any delays in server response times affect the user-perceived QoS. There are many user-centric client QoS measures such as Time to First Byte (TTFB) and First Input Delay (FID) [56].

These user-centric QoS measures usually exhibit high variability across different web services, with some services taking a few milliseconds, while others taking up to 4 seconds [43]. Some of this variability is due to the large dynamics that web-service servers experience, e.g., the number of requests, or the request mix. However, a much more contributing factor is the difference in how these services are designed and built, yielding very different (achievable) server response times for different web services. This variability between the different applications results in increasing complexity for management systems, with respect to knowing the achievable latency for each application, and how to guarantee this latency for each web service given the variability.

Tail versus Average Latency SLOs. In order to fulfill the QoS requirements of users, service providers and data center operators need to maintain a pre-defined SLO. The SLO can be in terms of, e.g., system availability, the average response time, or the tail response time. Using tail-latency to define server-level SLOs has gained wide interest from the academic community [12, 27, 32, 36] but this usually comes at the cost of using extra resources to improve tail performance. In the context of power management, it usually entails being conservative and allocating more power than what would be needed if only the average latency is to be guaranteed [36]. Hence, when power is limited, it can be more beneficial to focus on average latency when the main optimization required is to reduce the power consumption of the system. In other cases, power over-provisioning the power can help reduce tail latency.

2.3 Feedback Controller Design

Generally, to build a robust feedback controller, there are three steps. The first step in building a controller is to build a mathematical model of the controlled system behaviour. One technique to build and adapt these models is to use *System identification* [35]. System identification utilizes statistical methods to model the input-output relation of the system. These models can then be used to design a feedback controller using one of many techniques such as: Proportional-Integral-Differential (PID) control and its variations—P, PI and PD control; stochastic control; and optimal control. The controllers use the mathematical models to enforce the system to operate around some preset control target. Since many systems exhibit non-linear and dynamic behaviors, deviations from the developed models in most practical applications are inevitable. To deal with deviations, control engineers typically use adaptive control theory technique to adapt the controller to the system dynamics. One popular technique for adapting PID controllers is *gain scheduling* [2]. Gain scheduling is used when a single controller does not provide the desired performance for all system dynamics. In essence, the engineer designs several controllers capable of managing the performance under all possible operating ranges points of the system, and switch between these controllers as the system dynamics change.

2.4 The case for DDPC

There are three main challenges when designing feedback controllers for web server power management. First, web-services get updated frequently changing their power-performance trade-off curves. Second, within a data center, they can run on different machines with different power consumption footprints. Finally, web-services have non-linear relationship between the power consumption and their performance that changes with the workload dynamics and the workload-mix. To use feedback controllers for managing the power-performance trade-offs of web-servers, a controller needs to be generated for every single possible VM/container configuration since the power-performance trade offs change with the size of the VM/container. The modelling needs to be repeated for every possible application. In addition, any change in the web service, e.g, by updating some of its components, would require the entire process to be repeated. DDPC aims to solve this problem by automating the entire process, from profiling all the way to controller generation, and deployment, by utilizing three techniques from control theory, namely, system identification using ARX models, PI control, and gain-scheduling.

Besides using traditional feedback control, there are two other main approaches that have been suggested in the literature to control power-performance of web services. The first approach is to use an open-loop controller based on heuristics that does not require mathematical models and hence use more simple control techniques such as bang-bang controller similar to the one used by Pegasus [36]. The second approach uses machine learning based models of the performance, such as the one used in Rubik [28].

Heuristic based open-loop control approaches can provide stable and easy-to-understand controllers for power-performance, alas at the cost of reduced performance.For example, Pegasus uses a multi-step bang-bang controller, a controller that switches between a number of pre-defined heuristics, e.g., if response time is greater than x ms, set the power to the maximum. Since there is no accurate model as the ones used by gain-scheduling, heuristics can waste power as they are empirically chosen. In addition, these heuristics can not be easily inferred from the monitoring data and can not adapt to different applications easily.

While machine learning can be used to build a controller that is model free, a major shortcoming of using machine learning for managing power-performance is the lack of explainability which makes debugging the output from these algorithms hard [6, 7]. Control theory on the other hand relies on explainable mathematical models that can be debugged and understood by all stakeholders managing a cloud infrastructure. Prior work, e.g., Rubik [28], suggested the use of statistical learning approaches as a possible way to remedy the explainability issue. Rubik is a fine-grain DVFS scheme for latency-critical workloads that uses light-weight online profiling to update a statistical model of the completion cycle of each request, building target tail tables based on performance counters to estimate per-request compute and memory-bound cycles. The authors note that this approach works well at lower utilization levels of around 30%. However, at higher utilizations above 50%, the efficacy of Rubik decreases considerably.

3 DDPC Architecture

In this paper, We introduce DDPC, a framework for the automatic generation, and deployment of power-performance controllers for web services. We argue that for a power management approach to be useful, it needs to be data-driven with the ability to adapt and scale to the workload dynamics seen in a data center. DDPC tackles this problem by using a data-driven approach to automatically generate power controllers. Our approach can be compared to recent advances in continual learning but for—the by-design explainable—feedback controller generation.

DDPC generates controllers for managing the power-performance tradeoffs of applications running either on a single server or on a cluster of servers. We assume that a cluster runs multiple instances of a web service. The web service is behind a load-balancer. The system operates with the target of reducing the power allocation while maintaining an SLO defined in terms of the average response time. Our system supports runs in one of three modes; a guided optimization goal mode, i.e., operating the system below a certain power budget; an unguided operation where the goal is to reduce the power allocation generally but with no target power budget; and finally a tail-latency reduction mode were the target is to reduce the power-consumption while maintaining a smaller tail. In all of these modes, DDPC adapts to the any controller updates required due to, e.g., a software update, a change of the workload mix, or a new application being deployed.

DDPC has four main components as shown in Figure 1 ; the Controller Generator; the Cluster Power Manager; the Application Performance Monitor; and the Local Power Managers. The core of DDPC is the Controller Generator component which is responsible for the data-driven generation of gain-scheduled PI-controllers for the different applications. This is the first step in our framework and the most important one. To design the controllers, the generator

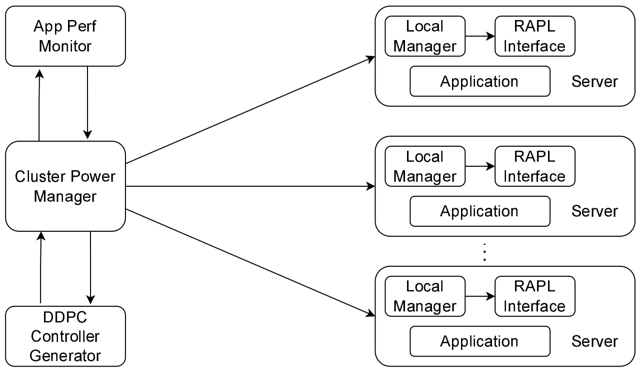


Figure 1: The DDPC architecture

runs a benchmarking phase where the web service behaviour is modeled using a set of ARX models that are then used to design a PI-based controller with gain scheduling. Once a controller is produced, it is deployed into the cluster power manager. We discuss the controller generator in the next Section in details.

The cluster power manager is responsible for the run-time management of the cluster. Once a controller is generated, the controller generator updates the manager, deploying the new controller at runtime. The Cluster Power Manager then controls the per server power allocation based on the measured response time of the application. The cluster power manager uses the monitored data from the Application Performance Monitor, mainly the number of requests per second, and the average response time to decide on the power allocation using the gain-scheduled PI controller for an application. If the cluster power manager observes that the controller is incapable of maintaining the SLO, the manager sends a signal to the DDPC controller generator to regenerate a new controller.

If a power-budget is set, the cluster manager divides the power on the servers such that the total cluster allocation does not exceed the budget. In this paper, we use fair-share division between the servers, with all servers allocated the same power similar to Pegasus [36]. The per server allocated power is forwarded to the DDPC local manager that runs on each server in the cluster. The local manager sets the power limit using the RAPL interface to control the power allocation for the hosted web-service. DDPC supports using c-groups for controlling the per-application power allocation for colocated applications.

Relaxed-DDPC for tail reduction. Control theory provides guarantees on the average system behavior. Hence, DDPC sets the target of the controller based on the average response-time. When operating in tail-reduction mode, relaxed-DDPC calculates the power required to set the average response time to the target, and then allocates 20% more power than suggested by the controller for maintaining the average response time target. We have empirically found that with 20% extra power, we curtail the tail.

4 Data-Driven Controller Design

Figure 2 shows the DDPC pipeline. An application owner only provides the images of the web-service components and the request types that the application supports. DDPC then uses the image and information to develop and deploy accurate power controllers for

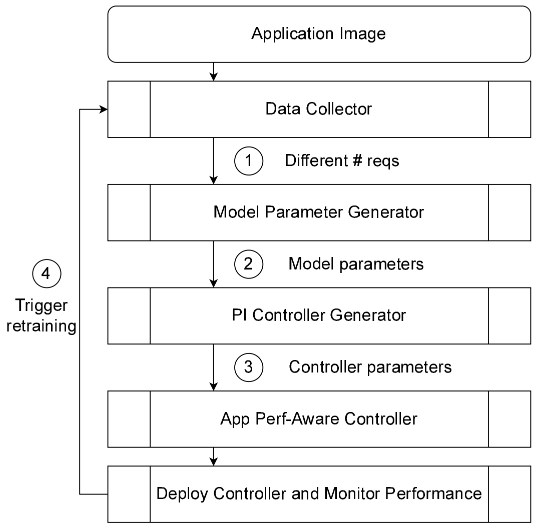


Figure 2: DDPC Pipeline

the application and deploys the application. DDPC starts by running a short benchmarking phase using a custom made workload generator that benchmarks the web-service's image with different request rates and request types under different power allocations. The logged data is then used to build a power-performance model of the application. The model is used to build a PI-controller with gain-scheduling for the power allocation. Both the application and the controller are then deployed in production. While in production, DDPC logs monitoring data for the application performance. If a large deviation in performance is detected, the web-service is re-profiled to update the controllers for the web service. We now discuss each of these steps in more details

4.1 Building data-driven ARX models

In the benchmarking phase, DDPC collects the necessary data required to build the web-service's system model of the power-performance tradeoffs. Our system uses ARX models to model the power-performance tradeoffs in the system [23]. ARX models are black-box system identification techniques that are commonly used to build PI controllers. ARX models find a *linear representation* of a dynamic system in discrete time using a linear representation of the system. These models form the basis for many methods in control methods. Focusing on our application, we would like to build a power-performance latency model which predicts latency for a given set power using RAPL at a given request rate which is a non-linear relation.

Let us denote the power level set to a web application cluster at time t to be $u(t)$, and the output of the model, the average response time to be $y(t)$. At a given workload level $W(t) = w$, an ARX model of this system is:

$$y(t+1) = \alpha y(t) + \beta u(t) \quad (1)$$

where α and β are the model parameters that characterize the relationship between the input and output of the model at a given workload level. The model assumes that the response time of a service for a given load-range is dependant on both the power

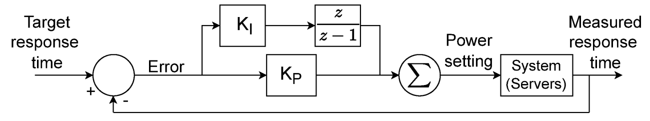


Figure 3: Single Controller Architecture

allocation and the previous response time. To calculate the values of α and β, we use the least-squares method over the logged data [55].

Since the standard ARX model is a linear model assuming a limited range for the workload (average number of requests per second), it is not accurate if the workload changes significantly. Hence, the model is not sufficient to capture the performance under varying workload levels. To solve this problem, we build multiple models for the system at different linearization points which in our case are workload levels.

4.2 Data-Driven gain-scheduled PI control

The profiling stage yields a number of ARX models for the web service at different request rate levels. The next stage in the pipeline is to design a PI controller for *each model of the different request levels generated by the ARX modeling step.*, i.e., creating a number of PI controllers for different operation points. In essence, a PI controller has two components; the Proportional component responsible for correcting any errors between the required value and the current state of the system; and the Integral component responsible for accounting for also the historical trends in the error. To give an example, consider a web-server cluster where we try to control the power allocation to all the servers using a PI controller. The system has an SLO on the average response-time of 500 milliseconds, but due to a sudden spike sees a response time of 800 milliseconds. The proportional term of the controller will only consider the current error in the response time of 300 ms. The integral term of the controller will consider the integral of the error in response time over time, i.e., the historical value of the error. Control engineers use the system model to choose the coefficients of both components of the PI controller, i.e., how much should the current error between the measured response time and the SLO affect the controller corrective action of increasing the power allocation versus how much should the historical error affect the corrective action.

The control loop for a single PI controller is shown in Figure 3. In our design, the set point to the controller is the target average response time from the web-service $r(k)$. Let us denote the measured response time at k^{th} time to be $y(k)$. The error is then the difference between the set point and the measured response time $e(k) = r(k) - y(k)$. A negative error means that the measured response time is higher than the target response time, and hence would need to be corrected by allocating more power to the web service. A positive error however indicates that the current response time is lower than the target average response time, and hence it might be safe to reduce the power allocation to the web service. The PI controller corrective action $u(k)$ is then:

$$u(k) = K_p e(k) + K_i \sum_{i=0}^{k-1} e(i). \qquad (2)$$

To calculate the controller parameters, we use Matlab's *pidtune* library. *pidtune* takes as input a single ARX model, and uses a proprietary frequency-domain algorithm to outputs the gains, K_p and K_i for a robust controller for a given request level. The algorithm is very efficient taking at maximum seconds to output the controller parameters. For each ARX model generated, a single PI-controller is generated for each request level. The generated gains for each request level is used to generate the power controller gain-scheduling based PI-controller that is robust across the entire possible operating capacity of a single server.

Implementation. DDPC is implemented in around 1300 lines-of-code in Python3 with the PI Controller tuning using Matlab's *pidtune* library. DDPC is modular enabling the use of, e.g., different controllers than what we use in the paper. We open-source the code for DDPC for the interested reader. [1]

5 Experimental Evaluation

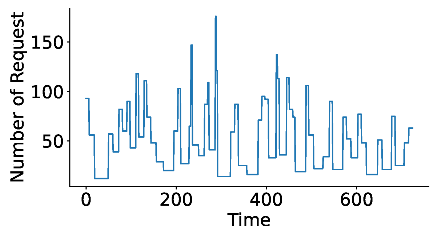
Hardware. Our experiments run on a four-node cluster consisting of Dell PowerEdge R440 servers, running Ubuntu 18.04.3 LTS. The servers have Intel Xeon Silver 4110 CPU with two sockets, 8-cores, 2.10GHz, and 11M cache. Each server has 64GB of Memory and two SSDs. The idle power consumption for each socket is 25 watts. In our comparisons, we use the active power and also provide the total power allocation per socket except in the cluster level experiments where we report the entire cluster's allocation.

Workloads. We use three different workloads in our experiments. The first workload (shown in Figure 4a) is a generated synthetic workload where the number of requests is randomly selected between 10 requests and 180 requests per second. We used this workload to validate our generated controllers. For testing the operation of our controller under different scenarios, we use the Azure Function Trace [52]. The trace is scaled down to suit our cluster size. Since the Azure traces are data center scale, each trace has a length of 30 minutes, with per-second request rate granularity. Figures 4b and 4c show a three minutes zoom-in of the two traces we use. For our experiments, we developed a closed-loop workload generator that replays the requests of the above workloads.

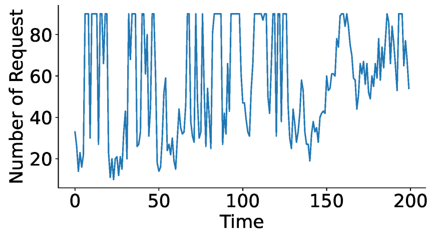
Applications. We use two realistic latency-sensitive applications in our experiments. The first application is a Mediawiki VM hosting a replica of the entire German Wikipedia. MediaWiki is a custom-made, free, and open-source wiki software platform written in PHP and JavaScript using a traditional LAMP stack software, running on Linux, Apache web-server, deploying a MySQL database, and Memcached for caching database objects in memory. Our second application is the microservice-based social networking application from DeathStarBench [14]. The application has over 30 microservices. In all experiments with the two applications, we replicate all application components on all servers. In our experiments, we run the applications behind an HA-Proxy load-balancer that we also use to measure the response times. We set the load-balancer to send requests to the server with the least number of connections.

State-of-the-Art comparisons. In all of our experiments, we compare our results with Pegasus, with respect to the response times, and active and total power savings. In addition to Pegasus, in some of our experiments, we compare the controller generated by DDPC

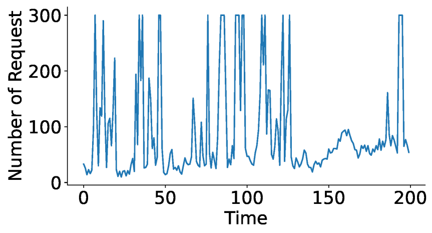
[1]Please see: https://github.com/umassos/ddpc-power-performance-controller



(a) Randomly generated synthetic workload

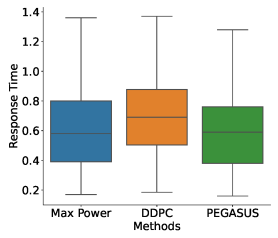


(b) Azure traces for single server experiments.

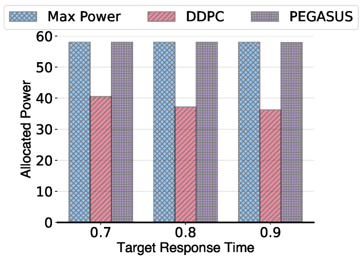


(c) Azure trace for DeathStarBench

Figure 4: Workload Traces for our experiments



(a) Response Time Box-plots (target 0.9s)



(b) Average power allocation per socket

Figure 5: Comparing DDPC, Pegasus, and using maximum power with a synthetic workload and Wikipedia.

to a controller that is hand-crafted using Matlab's PID Tuner Application and Simulink, one of the most commonly used tools by control engineers to tune PID controllers.[2]

5.1 DDPC validation

Our first set of experiments aims to validate and show how DDPC generates and automatically deploys controllers capable of maintaining the average response time below the target while reducing the overall power allocation. In this experiment, we use the German Wikipedia replica and the synthetic workload shown in Figure 4a. We use a single server for this experiment. We pass to DDPC the applications image. DDPC profiles the application, generates, and deploys a controller. We then use our developed workload generator to play the synthetic workload, We set the target response time to 0.9s, 0.8s, and 0.7s, and measure the request response times, and the allocated power. We repeat the experiment with Pegasus.

Figure 5a shows a box-plot of the response times observed with DDPC, Pegasus, and when the system is always given full power (58 watts per socket) when the response time target is 0.9s (we omit the response time graphs for lower targets from this experiment for space constraints but show lower target response time graphs in other experiments). We make the following observations: more than 75% of all requests using DDPC and Pegasus have a response time lower than the target response time. Pegasus has a slight improvement in the tail response time. Figure 5b shows the average power allocation of the three approaches for different response time targets, we see that in the worst case at the target response time of 0.7, DDPC yields a power allocation reduction of more than 32% (with 48% saving with a target response time of 0.9s) in the total power in comparison to Pegasus. If we only consider active power, then DDPC provides a 56% when the target response time is 0.7s (67% with a target response time of 0.9s) decrease in the active power allocation. In addition to the power savings, Figure 5b shows that with a higher response time target, the power savings can increase considerably.

Pegasus fails to provide much power savings because of the nature of the workload and Pegasus's very conservative approach to power allocation, allocating maximum power for five minutes to the application if the average latency over a period of 30 seconds exceeds the target latency.

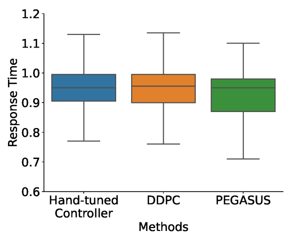
DDPC controller generation overhead. Our final validation point is to validate the practicality of DDPC in terms of overhead. We measure the time DDPC requires to generate and deploy the controllers. At the profiling stage, DDPC runs a workload with step increases of 20 requests running for 20 seconds, starting from 20 requests per second, and log the achieved performance for different power-level, increasing the power allocation to the application by 2 watts–starting from minimum server power—and monitoring the response time. In our experiments, the total profiling time was between 5 and 12 minutes depending on the application. Since the profiling is done mostly when new applications are added or updated, or when there are significant changes in the workload mix, we believe that this overhead is adequate.

5.2 Azure workload evaluation: Wikipedia

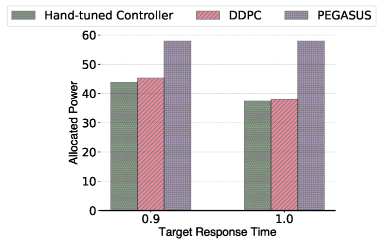
In this experiment, we use the Azure trace shown in Figure 4b with the German Wikipedia replica running on a single server, and repeat the above set of experiments. We set the target response time to 1s. Figure 6a shows the response time box-plots for our experiment using DDPC, a hand-crafted controller using Matlab Simulink, and Pegasus to control the power allocation. We see that Pegasus provides a slight improvement over DDPC (and the hand-crafted controller) in tail-latency. Figure 6b shows the power allocation for the different approaches. DDPC provides a 61% reduction in active power in comparison to Pegasus (and a reduction of 45% in total allocated power). We also show the results for a lower response time target of 900ms where DDPC again provides savings of 43% in active power (25% total power). Compared to the hand-crafted controller DDPC allocates between 1% to 4% extra power.

To better see how Pegasus behaves with this workload, we plot the CDF graph of the power-level allocations with both DDPC, Pegasus, and a hand-crafted controller in Figure 6c. We note that while DDPC uses the full range of RAPL active power available

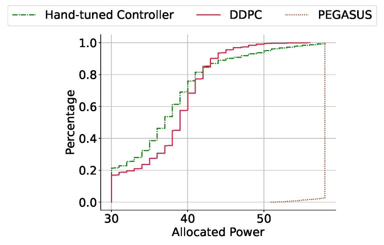
[2]https://www.mathworks.com/help/control/ref/pidtuner-app.html



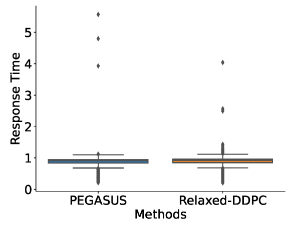
(a) Response Time box-plots.



(b) Average power allocation for different response time targets.



(c) A CDF of power allocation levels using DDPC and Pegasus



(d) Relaxed DDPC versus Pegasus

Figure 6: Comparing DDPC, a hand-crafted controller, and Pegasus using the Azure workload and Wikipedia.

(with 1-watt granularities), Pegasus is almost always allocating the maximum power with very little changes in the allocation. This shows the downside of using a heuristic since the allocation needs to be conservative. We note that our data-driven generated controllers almost overlaps with a hand-crafted controller that is tuned by an engineer showing the merit of our automation.

Relaxed DDPC for Tail Reduction: Our next set of experiments aim to evaluate DDPC when operating in tail-reduction mode. In this experiment, we run a two hour Azure trace, setting the target response time to 1s. Figure 6d shows the box-and-whiskers plot of the response time for both DDPC and Pegasus. Pegasus has a much longer tail compared to relaxed-DDPC. In addition, over the two hour period, Pegasus dropped 10% of the requests while relaxed-DDPC dropped a negligible number of requests. relaxed-DDPC allocated 25% less power compared to Pegasus. Since our results with relaxed-DDPC were similar for other experiments, we omit them, and only include this result.

5.3 Azure workload evaluation: DeathStarBench

To show how DDPC adapts to the addition of new application, we introduce to our cluster a new application, the social networking application from DeathStarBench. DDPC generates and deploys controllers for the new application. We measure the performance of these controllers with both Azure workloads.

Figure 7a and Figure 7b show box-plots of the response times with the workload in Figure 4b and Figure 4c respectively, while Figure 7c shows the average power allocation for both cases. We see a similar pattern with this application where the power allocation savings range between 25% and 50% for the total power (50 to 70% in active power) compared to Pegasus. However, in this experiment, we also observe that the tail response time of DDPC is lower than that of Pegasus by 100 to 200 ms. This result shows that for some applications, even at a fraction of the power, DDPC is able to reduce the average and tail response times compared to Pegasus.

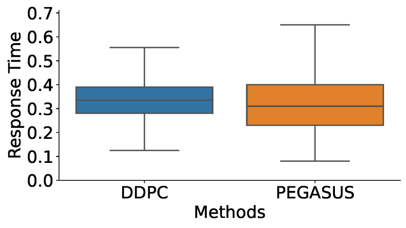
5.4 Non-Constrained clusters

We now turn our focus to experiments running the workloads on a cluster of four servers (with a total power capacity of 464 Watts). In this set of experiments we assume that there is no application co-location on the servers. We start with the case when DDPC is running with no power budget constraints, but with the aim of reducing the total power allocation. In this experiment, we again use the Wikipedia application but use it with the Azure workload in Figure 4c but has 800 requests as maximum number of request rather than 300. On a single server, this workload would not be sustainable due to its high peaks and the high processing demands of Wikipedia. For Pegasus, since we wanted to compare with a less conservative version of Pegasus that we hoped would result in reduced power waste, we modified Pegasus. When faced with an average response time above the target response time of 30 seconds, Pegasus runs at full power for 5 minutes. Since this proved to be very conservative for our workloads, we modified Pegasus to run for 30 seconds instead of five minutes when there is a sustained violation of the target for 30 seconds. We believed that this change would enable Pegasus to handle the workload better.

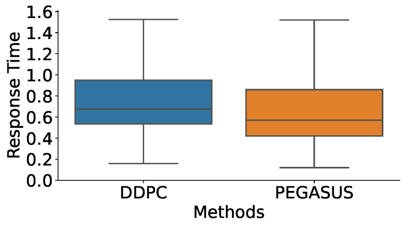
Figure 8a shows the latency CDF when using Pegasus versus when using DDPC to control the power-allocation for the cluster. The Figure suggests that while Pegasus lowers the latency for requests for most of the requests, it has a longer tail compared to DDPC. To better see the tail, we plot the violin plot of the distribution. The violin plot confirms that the tail response time of Pegasus is slightly longer than DDPC for this experiment. However, this can be due to the presence of outliers in the response times of Pegasus. Finally, when considering the cluster average power allocation, we measured that DDPC allocates on average 136.46 watts with over 40% reduction of the total cluster power compared to Pegasus which allocates on average 226.82 Watts.

5.5 Performance under a cap

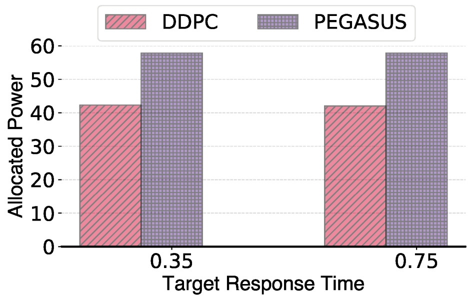
In the final set of experiments, we repeat the above experiments with Wikipedia but in the presence of a maximum power budget. We run several experiments with several budgets and response time targets. We show only results from two such experiments with power budgets of 280 and 320 watts for the entire cluster, which correspond to a budget of roughly 60% and 70% of the total power cluster power, and a target response time of 1s. Figure 9a shows the average power allocated for the two budgets. While Pegasus allocates close to the maximum available power in the budget, specially for the higher budget, DDPC keeps the power allocation to less than the budget by roughly 10% for the lower budget and by around 20% for the higher budget. At the lower budget of 280 watts, DDPC maintains an average response time well below the target response time of 1s. In fact, its 90th percentile response time is below 1s. Pegasus provides a lower latency distribution. When considering the higher power budget of 320 Watts, Pegasus, while having a lower average response time, has a longer tail compared to DDPC. DDPC is able to maintain the average latency well below the target latency. This experiment shows that the tail latency of



(a) Box-plot of the observed response times for the Social Networking application using the trace in Figure 4b with a target response time of 350ms.

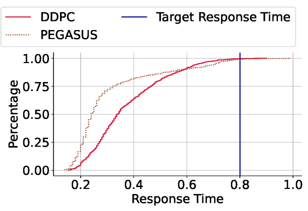


(b) Box-plot of the response times for the Social Networking application using the trace in Figure 4c with a target response time of 750ms.)

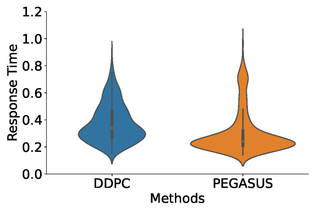


(c) The average power allocation for the two different workloads.

Figure 7: Comparing DDPC to Pegasus used to control the social network.

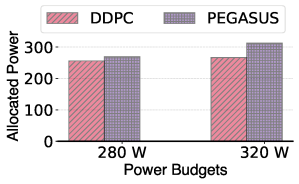


(a) The CDF of response times of Wikipedia running on a cluster.

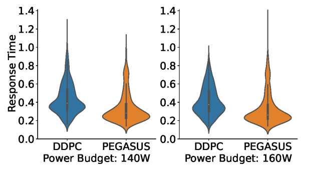


(b) Violin plot of the distribution of response times for Wikipedia.

Figure 8: DDPC versus Pegasus with no power capping.



(a) The power allocation of both Pegasus and DDPC under the two different caps.



(b) Violin plots of the distribution of response times under the two different caps.

Figure 9: DDPC versus Pegasus when using a cluster with power capping.

DDPC operating in the normal mode, is comparable and in some cases better than a system designed mainly for tail latency control because of the superiority of control theory using accurate models over heuristics in adapting to the different workloads and setups.

6 Related Work

Using control theory for systems problems has gained wide interest within the systems community recently [13, 44, 47]. A particular example is on the problem of controlling the power-performance trade offs for web clusters has been studied since the early 2000s [4, 9, 49]. This early work was during the early days of the web, when neither virtualization nor modern power management techniques were prevalent. The advent of hyperscale data centers [5] has sparked new interest in the problem [28, 34, 36, 59]. Our work extends on the previous work, solving the problem of how to automate the process of controller design for power-performance tradeoffs.

In addition, a lot of previous work focused on task-based workloads or on throughput-oriented workloads [18, 22, 24]. Other work focus on how to handle workloads under a power-cap [3, 19, 58]. One interesting approach for HPC jobs is the one used in PShifter [15] where a power budget given to a certain application is distributed by shifting power from sockets that incur extended wait times due to waiting at a blocking call and redirects the dissipated power to where it can best improve the overall performance of the job. Mishra et al. [38] introduce CALOREE, a resource manager that meets application latency requirements with minimal energy by fine-tuning controllers using transfer learning for task-based workloads. ReTail [10] represents request-level latency prediction and applying it to power management.

7 Conclusion

In this work we introduce DDPC, a framework for managing power-performance tradeoffs in hyperscale clusters. DDPC fully automates the process of designing and deploying a power-performance controller for a web service running on a cluster of servers. Our controller generation only requires the image of the application as an input and everything else is fully automated. DDPC generates different number of PI-controller based on gain scheduling by profiling the web-service autonomically to control the power allocation to the service. The controllers uses the average response time of the web-service as the target. Our results show that the generated controllers consistently show significant reductions in power allocation ranging between 30 to 50% in total power and 40 to 75% in active power compared to other state-of-the-art approaches although this significant power improvement sometimes comes with modest increases in the tail response time. We open source DDPC.[3] One limitation of DDPC is that it does not include any autoscaling capabilities. For future work we will integrate both horizontal and vertical autoscaling with DDPC.

Acknowledgments This research is supported by the Republic of Türkiye Ministry of National Education YLSY program, NSF grants 2213636, 2105494, 2021693 and 2020888, Army Research Laboratory IoBT project contract W911NF-17-2-0196, the Swedish Foundation for Strategic Research grant FFL21-0091, Horizon 2020 NGI-Atlantic, and WASP-Sweden.

[3]The code available at: https://doi.org/10.5281/zenodo.7630390

References

[1] Jordi Arjona Aroca, Angelos Chatzipapas, Antonio Fernández Anta, and Vincenzo Mancuso. 2014. A measurement-based analysis of the energy consumption of data center servers. In *Proceedings of the 5th international conference on Future energy systems*. ACM, 63–74.

[2] Karl J Åström and Björn Wittenmark. 2013. *Adaptive control*. Courier Corporation.

[3] Reza Azimi, Masoud Badiei, Xin Zhan, Na Li, and Sherief Reda. 2017. Fast Decentralized Power Capping for Server Clusters.. In *HPCA*. 181–192.

[4] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. 2003. Web search for a planet: The Google cluster architecture. *IEEE micro* 23, 2 (2003), 22–28.

[5] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. 2018. The datacenter as a computer: Designing warehouse-scale machines. *Synthesis Lectures on Computer Architecture* 13, 3 (2018), i–189.

[6] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José MF Moura, and Peter Eckersley. 2020. Explainable machine learning in deployment. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*. 648–657.

[7] Stella Biderman and Walter J Scheirer. 2020. Pitfalls in machine learning research: Reexamining the development cycle. (2020).

[8] BloombergNEF. [n. d.]. Data Centers Set to Double Their Power Demand in Europe, Could Play Critical Role in Enabling More Renewable Energy. https://about.bnef.com/blog/data-centers-set-to-double-their-power-demand-in-europe-could-play-critical-role-in-enabling-more-renewable-energy/. Accessed: 2021-11-17.

[9] Pat Bohrer, Elmootazbellah N Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. 2002. The case for power management in web servers. In *Power aware computing*. Springer, 261–289.

[10] Shuang Chen, Angela Jin, Christina Delimitrou, and José F Martínez. 2022. Retail: Opting for learning simplicity to enable qos-aware power management in the cloud. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 155–168.

[11] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.

[12] Christina Delimitrou and Christos Kozyrakis. 2018. Amdahl's law for tail latency. *Commun. ACM* 61, 8 (2018), 65–72.

[13] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2014. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*. 299–310.

[14] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 3–18.

[15] Neha Gholkar, Frank Mueller, Barry Rountree, and Aniruddha Marathe. 2018. Pshifter: Feedback-based dynamic power shifting within hpc jobs for performance. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. 106–117.

[16] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. 2012. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*. 75–86.

[17] Akhil Guliani and Michael M Swift. 2019. Per-application power delivery. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–16.

[18] Can Hankendi, Ayse Kivilcim Coskun, and Henry Hoffmann. 2015. Adapt&Cap: Coordinating system-and application-level adaptation for power-constrained systems. *IEEE Design & Test* 33, 1 (2015), 68–76.

[19] Can Hankendi, Sherief Reda, and Ayse K Coskun. 2013. vCap: Adaptive power capping for virtualized servers. In *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 415–420.

[20] Md E Haque, Yuxiong He, Sameh Elnikety, Thu D Nguyen, Ricardo Bianchini, and Kathryn S McKinley. 2017. Exploiting heterogeneity for tail latency and energy efficiency. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 625–638.

[21] Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. 2004. *Feedback control of computing systems*. John Wiley & Sons.

[22] Henry Hoffmann. 2014. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 223–232.

[23] Alf J Isaksson. 1993. Identification of ARX-models subject to missing data. *IEEE Trans. Automat. Control* 38, 5 (1993), 813–819.

[24] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 347–358.

[25] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L Cox, and Scott Rixner. 2014. Predictive parallelization: Taming tail latencies in web search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 253–262.

[26] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 158–169.

[27] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. 2014. Tradeoffs between power management and tail latency in warehouse-scale applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 31–40.

[28] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 598–610.

[29] Saehoon Kim, Yuxiong He, Seung-won Hwang, Sameh Elnikety, and Seungjin Choi. 2015. Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining* (Shanghai, China) *(WSDM '15)*. Association for Computing Machinery, New York, NY, USA, 7–16. https://doi.org/10.1145/2684822.2685289

[30] Jakub Krzywda, Ahmed Ali-Eldin, Trevor E Carlson, Per-Olov Östberg, and Erik Elmroth. 2018. Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling. *Future Generation Computer Systems* 81 (2018), 114–128.

[31] William S Levine. 2018. *The Control Handbook (three volume set)*. CRC press.

[32] Jialin Li, Naveen Kr Sharma, Dan RK Ports, and Steven D Gribble. 2014. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–14.

[33] Shaohong Li, Xi Wang, Faria Kalim, Xiao Zhang, Sangeetha Abdu Jyothi, Karan Grover, Vasileios Kontorinis, Nina Narodytska, Owolabi Legunsen, Sreekumar Kodakara, et al. 2020. Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 1241–1255.

[34] Yang Li, Charles R Lefurgy, Karthick Rajamani, Malcolm S Allen-Ware, Guillermo J Silva, Daniel D Heimsoth, Saugata Ghose, and Onur Mutlu. 2019. A scalable priority-aware approach to managing data center server power. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 701–714.

[35] Lennart Ljung. 1998. System identification. In *Signal analysis and prediction*. Springer, 163–173.

[36] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards energy proportionality for large-scale latency-critical workloads. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 301–312.

[37] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. 2016. Asic clouds: Specializing the datacenter. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 178–190.

[38] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 184–198.

[39] Srinivas Pandruvada. [n. d.]. Running Average Power Limit. https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl. Accessed: 2023-2-7.

[40] George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. 2019. Adaptive voltage/frequency scaling and core allocation for balanced energy and performance on multicore CPUs. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 133–146.

[41] Tirthak Patel and Devesh Tiwari. 2019. Perq: Fair and efficient power management of power-constrained large-scale computing systems. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. 171–182.

[42] Vinicius Petrucci, Michael A Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. 2015. Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 246–258.

[43] Philip Walton. [n. d.]. Largest Contentful Paint (LCP). https://web.dev/lcp/. Accessed: 2022-10-11.

[44] Raghavendra Pradyumna Pothukuchi, Joseph L Greathouse, Karthik Rao, Christopher Erb, Leonardo Piga, Petros G Voulgaris, and Josep Torrellas. 2019. Tangram: Integrated control of heterogeneous computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 384–398.

[45] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. 2015. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM symposium on cloud computing*. 342–355.

[46] Zhan Qiu, Juan F Pérez, Robert Birke, Lydia Chen, and Peter G Harrison. 2017. Cutting latency tail: Analyzing and validating replication without canceling. *IEEE Transactions on Parallel and Distributed Systems* 28, 11 (2017), 3128–3141.

[47] Amir M Rahmani, Bryan Donyanavard, Tiago Mück, Kasra Moazzemi, Axel Jantsch, Onur Mutlu, and Nikil Dutt. 2018. Spectr: Formal supervisory control and coordination for many-core systems resource management. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 169–183.

[48] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019. ?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*. 3215–3222.

[49] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian Lu. 2003. Power-aware QoS management in web servers. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*. IEEE, 63–72.

[50] Arman Shehabi, Sarah J Smith, Eric Masanet, and Jonathan Koomey. 2018. Data center growth in the United States: decoupling the demand for services from electricity use. *Environmental Research Letters* 13, 12 (2018), 124030.

[51] Rahul Singh, Upendra Sharma, Emmanuel Cecchet, and Prashant Shenoy. 2010. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceedings of the 7th international conference on Autonomic computing*. 21–30.

[52] Microsoft Open Source. [n. d.]. Azure Functions Trace 2019. https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md. Accessed: 2021-11-13.

[53] Amoghavarsha Suresh and Anshul Gandhi. 2019. Using Variability as a Guiding Principle to Reduce Latency in Web Applications via OS Profiling. In *The World Wide Web Conference*. 1759–1770.

[54] Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and TN Vijaykumar. 2015. Timetrader: Exploiting latency tail to save datacenter energy for online search. In *Proceedings of the 48th international symposium on Microarchitecture*. 585–597.

[55] Michel Verhaegen and Vincent Verdult. 2007. *Filtering and system identification: a least squares approach.* Cambridge university press.

[56] Philip Walton. [n. d.]. User-centric performance metrics. https://web.dev/user-centric-performance-metrics/. Accessed: 2022-10-11.

[57] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. 2016. Dynamo: Facebook's data center-wide power management system. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 469–480.

[58] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *ACM SIGPLAN Notices* 51, 4 (2016), 545–559.

[59] Liang Zhou, Laxmi N Bhuyan, and KK Ramakrishnan. 2020. Gemini: Learning to Manage CPU Power for Latency-Critical Search Engines. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 637–349.