January 2019

Toward Energy Efficient Systems Design For Data Centers

Bing Luo
Wayne State University

TOWARD ENERGY EFFICIENT SYSTEMS DESIGN FOR DATA CENTERS

by

BING LUO

DISSERTATION

Submitted to the Graduate School,

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2019

MAJOR: COMPUTER SCIENCE

Approved By:

_____

Advisor Date

_____

_____

_____

DEDICATION

To my beloved family.

ACKNOWLEDGEMENTS

I would like to express my deep and sincere thanks to those who supported and encouraged me throughout my doctoral studies.

First and foremost, I am delighted to express my respectful gratitude to my advisor, Prof. Weisong Shi for sharing his words of wisdom. His humor and insight encouraged me to keep working on valuable research problems. Not only did he inspired me to think big in academic research, but also he helped me in life.

I am also grateful to Dr. Nathan Fisher, Dr. Daniel Grosu, And Dr. Shaolei Ren for serving as my committee members. Their professional suggestions on my prospectus and dissertation familiarize myself with a broader range view of how to analyze and solve an academic problem. This also helped to have a more comprehensive consideration and review of problems.

I want to extend my thanks to Dr. Wensong Zhang, Dr. Ming Dian, Dr. Xiaozhong Li, and their team, who provided me opportunities to cooperate with their team and solve real data center challenges. Without their precious support, it would not be possible to conduct these research. My special thanks also goes for former and present MIST and LAST group members that I have had the pleasure to work with or alongside of.

Last but not least, I deeply appreciate the support and love from my parents and my wife. Studying abroad is always tough but they provide me a warm harbor to eliminate fatigue and continue to forge ahead.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

The continuous growth of numerous cloud services, along with the explosive emergence of Internet of Things(IoT) and edge computing, has significantly fueled a surging demand for data centers worldwide. Modem large-scale data centers(e.g Google and Facebook) can consume megawatts of electricity and cause a growing concern over carbon emission [62]. And 30% of the enterprise data center are expected to run out of power capacity within 12 months according to an industry survey from the Uptime Institute [45]. As a result, the power consumption of data centers over the world is predicted to approach 1,000TWh within a decade(2013-2025), which is more than the combined total power consumed by Japan and Germany for all purposes [47]. The huge power demands not only imply significant electricity cost but also lead to tremendous carbon emission.

Power consumption of data centers has tremendous implications on both operating and capital expense. The power infrastructure, along with the cooling system, incurs a capital expense of US$10-25 per watt of IT critical power, which cost a multi-million or even billion dollar project to add new data center capacities [6,21,67,95,101]. The power infrastructure capEx even exceeds 1.5 times of the energy cost over a 15-year lifespan [26,30]. Further, compounded by the rising electricity price, energy cost has been quickly escalating and become a major fraction of data center's operating expense. Cloud service providers such as Microsoft and Google have been integrating green energy sources (e.g., wind or solar power) into data centers' power supply to cut their energy cost and emission [39]. Given the high cost, it is important to fully utilize the capacity of data centers to reduce the Total Cost of Ownership(TCO).

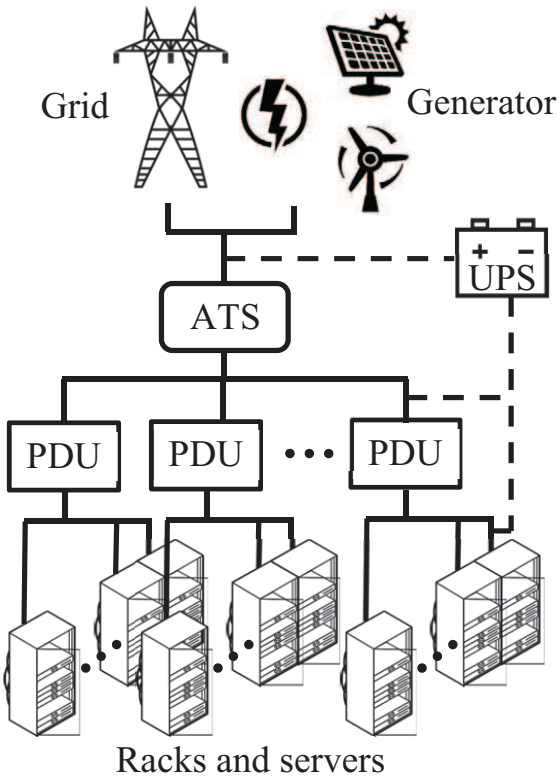


Figure 1: Power hierarchy in data center.

Understanding the power hierarchy and power behavior in data center is the foundation for managing the power infrastructure for maximum utilization and minimum Total Cost of Ownership(TCO). Most data centers, including new constructions, rely on diesel generators, uninterrupted power supplies (UPS), and power distribution unit (PDU). The left side of Figure 1 illustrate the power architecture of a typical data center. A tree-type power hierarchy is commonly found in today's data centers. High-voltage grid power first enters the data center through an automatic transfer switch (ATS), which selects between grid power (during normal operation) and standby generation (during utility failures). Then, power is stepped down and fed into the UPS, which outputs "protected" power to multiple power distribution units (PDUs) each supporting a few tens of racks. These racks

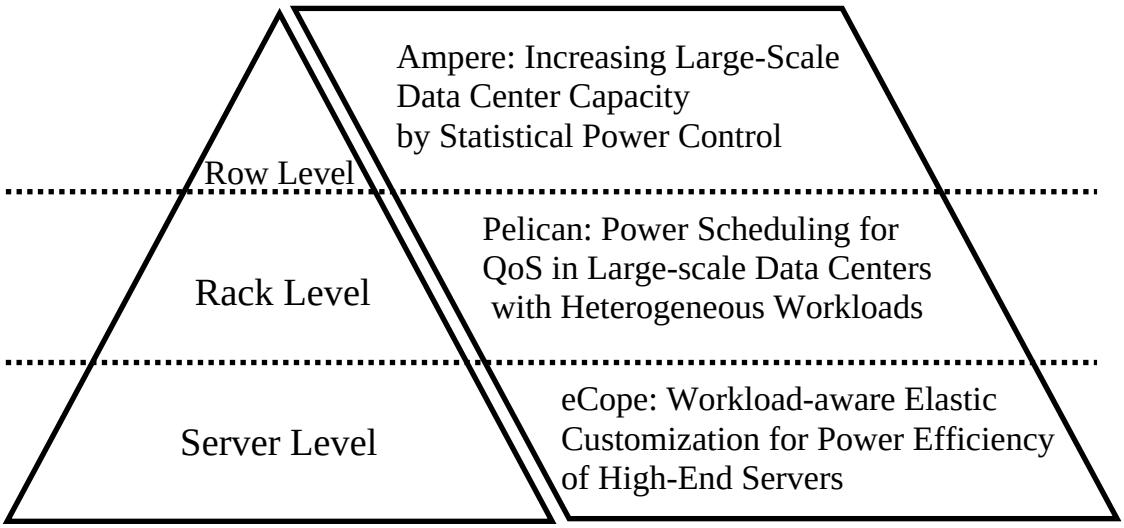


Figure 2: A high level overview of proposed research topics.

are some times called a row in data center. At the rack level, there is a power strip (also called rack PDU) that directly connects to servers. Then servers are provisioned according to the power budget at each level. The provisioning is often based on the maximum power of a server. In some data centers, extra UPS may be deployed before row/rack PDU to protect a row/rack, complementing the centralized UPS [95]. The UPS before ATS and cluster-level PDUs handle power at a high or medium voltage and hence are expensive, while the rack-level PDU and other UPSs have a lower voltage and is relatively cheap (e.g., 20-50 US cents per watt) [4,67]. Note that redundancy (e.g., duplicating each non-IT unit or "2N") is also common in data centers to achieve a high availability.

1.1 Problem Statement and Overview

One of the most important challenges in our data centers is insufficient capacity given the ever increasing demand on computation. There are multiple factors constraining the capacity of a data center. The data center is designed with a space budget and power budget. With the adoption of high-density rack designs such as Scorpio [79] and Open

Compute Project [65], the capacity of a modern data center is usually limited by the power budget. So the core of the challenge is scaling up power infrastructure capacity. Upgrading power systems is a radical solution that allows one to add more servers, racks, and other supporting devices. However, resizing data center initial power capacity is similar to build a new data center, which can be a great undertaking since conventional centralized power provisioning scheme does not scale well [45]. On the other hand, increasing power utilization means that more (powerful) servers can be deployed to deliver better performances and higher user satisfaction without additional capital expense for capacity expansion. So how to maximize the power utilization and optimize the performance per power budget is critical for data centers to deliver enough computation ability.

There are several main problems in current data centers that preventing maximize the performance per power budget. First of all, each server in data center is usually build for general purpose computing and its configurations are not optimized for running workloads. While it is necessary to allow different services running on the same server, it is also common that some server holds specific service for a long time. Further more, for the same service, the load changes over time. A static configuration is usually not suitable for changing load. Second, power utilization of data centers is often low and also limits the performance of the data center. In other words, the performance per power budget is low. Third, the already over-complicated scheduler in real production environment prevents power control from integrating with job scheduler.

To explore and attack the challenges of improving the power utilization, we work on different levels of data center, including server level, row level. Figure 1 shows an overview of our system. For server level, we take advantage of modern hardware to maximize power

efficiency of each server. More specifically, given a specific application and a workload range, we proposed a framework that can find a way to achieve energy proportionality through hardware customization for a server. With eCope, we can find an optimized dynamic workload-power function and customize hardware according to both workload and the related optimized configuration. For rack level, we investigate the behavior of power and task in data centers and present Pelican, a new power scheduling system for large-scale data centers with heterogeneous workloads. Instead of moving tasks on spatial dimension, we tried to move tasks on temporal dimension. Based on the current power of a rack, we will find an optimized power budget for each server and by limiting resources for tasks, we can directly control the performance and power of a server. For row level, we present Ampere, a new approach to improve throughput per watt by provisioning extra servers. Ampere keeps the total power under the budget and brings zero performance disturbance to the existing jobs. Different from existing approaches that react to an over-committing event by capping the power draw, Ampere proactively reduces power violations by driving the workload to other less utilized rows.

In the following several subsections, we will introduce briefly why and how we address these research topics on each level to provide comprehensive system support for energy efficiency in data centers.

1.2 Server Customization for Power Efficiency of High-End Servers

Hardware components, especially CPU and Memory, have made a lot of progress in terms of energy efficiency in the last decade. The state-of-the-art servers in datacenters are still far from being energy proportional [7]. Barraso and Hale report the CPU utilization

of more than 5,000 servers during a six-month period, and they propose an energy proportional design for datacenter servers [7]. It means that “performance per watt“ should be considered as the most important metric, particularly when the server is at the normal utilization level. After that, many approaches [24,63,81,82,85,87] have been proposed to improve energy-proportionality in datacenters, using both software and hardware. However, they do not consider the different workloads of a server. Reiss *et al.* notice that workload characteristics are heterogeneous in resource types and their usage according to their analysis of the first publicly available trace data from a sizable multi-purpose cluster [76]. Furthermore, Voigt *et al.* find that workload characteristics are less steady and less predictable because applications are more agile and flexible [93]. This makes energy proportional design more difficult. Metri *et al.* try to understand how exactly the application type and the heterogeneity of servers and their configurations impact the energy efficiency of datacenters [61]. And they observe that each server has a different application specific energy efficiency values based on the type of application running, the size of the virtual machine, the application load, and the scalability factor.

Furthermore, Even for the same server and same application running on it, Dean and Barroso notice that the latency variability is common, and the variability can be amplified by the scale [18]. In fact, variability is not only limited to the latency, it exists in all components of a server. Such dynamics and heterogeneity reduce the effectiveness of traditional energy proportional schema because traditional energy proportional schemas are usually optimized for a certain type of hardware or operating system or workload. So, it is better to design an elastic customization schema for servers.

Motivated by the recent observations that the energy efficiency of hardware compo-

nents varies to a great extent depending on the workload characteristics, we propose eCope, workload-aware elastic customization for power efficiency of high-end servers, to reduce power consumption by workload aware and hardware customization for servers in datacenters. Our unique contribution is that eCope platform can take advantage of any configurable hardware that fits our assumption to improve the energy proportionality for various kinds of services without knowing the details of the target service. We illustrate three case studies to show how can we apply our idea to typical real-world back-end services(file system, database services and web-based services).

1.3 Increasing Large-Scale Data Center Row Level Capacity by Statistical Power Control

The power budget in a data center is, unfortunately, often not fully utilized. As studies [22, 40, 57] point out, the typical power utilization is around only 60% of the provisioned power, which effectively doubles the cost of data centers. We found similar utilization numbers in our own data centers.

The main reason for the under-utilization is conservative server provisioning. People commonly ensure that the sum of the *rated power*[1] of all equipment does not exceed the power budget. However, with modern power saving mechanisms, actual power draw from a server depends on its utilization, which seldom reaches the peak level [8, 27, 53]. The low utilization is due to the variations in workload, as well as trying to guarantee the SLAs for latency-critical services. In other words, servers are provisioned according to the worst case power draw, while they operate at the average case.

[1]Following the definition in [22], we use the rated power, or the measured maximum power draw from equipment, instead of the name plate power that is often higher.

To fully utilize power budget in data center, we proposed Ampere, a novel power management system for data centers to increase the computing capacity by over-provisioning the number of servers. Instead of doing power capping that degrades the performance of running jobs, we use a statistical control approach to implement dynamic power management by indirectly affecting the workload scheduling, which can enormously reduce the risk of power violations. Instead of being a part of the already over-complicated scheduler, Ampere only interacts with the scheduler with two basic APIs. Instead of power control on the rack level, we impose power constraint on the row level, which leads to more room for over provisioning.

The approach is conceptually simple: by scheduling fewer jobs to rows with less unused power or letting them wait in the scheduler queue, we can reduce the amount of power increase and prevent power violations. However, we face several challenges implementing this scheme into a production data center.

First, the data center job scheduler is a very complicated system. Different data centers use different scheduling policies to achieve diverse goals. It is hard to have a general solution that directly adds power scheduling into these already-over-complicated policies. To solve this problem, we limit our interface with the job scheduler to two simple operations: freeze a server and unfreeze a server. Freezing a server advises the scheduler not to assign new jobs to the server (the existing jobs are unaffected), while unfreezing does the opposite. By controlling the number of frozen servers on a row, we can *statistically* affect the number of jobs scheduled there without changing the scheduling policy.

The second challenge is how to estimate the number of servers we need to freeze. The goal is to freeze as few machines as possible to minimize the negative impact on

scheduling and overall computation capacity, while keeping the row-level power below the budget. We solve this challenge with a *data-driven* approach. We collect data from production data centers, and derive a statistical model on the effects of freezing servers on the power consumption. We compute the number of servers to freeze at each time interval based on the model. Obviously the model is not perfect, and we use Receding Horizon Control (RHC) techniques [43] to continuously adjust the number of frozen servers to compensate for the inaccuracy of the model.

1.4 Pelican: Power Scheduling for QoS in Large-scale Data Centers with Heterogeneous Workloads

Conservative server provisioning along with the diurnal pattern [59,68], unfortunately, lead to typical low power utilization in modern data centers [22, 32, 40, 57]. The Ampere solution ake this opportunity to improve QoS. However, it is mainly suitable for offline workloads. Furthermore, scheduling a task to a different server do not affect its performance only when the task is location independent. Unfortunately, many offline workloads require to run on the server where data is located because of the huge data size. This also limits the type of workloads.

Combining online and offline services on the same set of servers can obviously increase the power utilization but also introduce more challenges. **1)** The number of online services tasks is purely depending on user input, which data center manager cannot control. Large power fluctuations are common and the duration for the increase in power is much shorter than offline workloads. **2)** To make heterogeneous workloads work harmoniously, the complexity of job scheduler is ever increasing. Furthermore, we may rethink the rela-

tionship between job scheduler and power management.

To address the above limitations and challenges, we investigate the behavior of power and task in data centers and present Pelican, a new power scheduling system for large-scale data centers with heterogeneous workloads. Instead of moving tasks on spatial dimension, we tried to move tasks on temporal dimension. Based on the current power of a rack, we will find an optimized power budget for each server and by limiting resources for tasks, we can directly control the performance and power of a server. We also find that task priority is a good indicator when selecting target tasks to improve overall QoS. In this way, our system can control power resource without modifying the task assignment. This work is evaluated in a production data centers in Baidu, the largest search engine and one of largest cloud service providers in China with millions of servers running billions of tasks per day.

1.5 Summary of Contributions

By combining all these studies, we will provide:

1. A general workload-aware framework, eCope [55], is proposed to achieve energy proportionality for various kinds of services in data centers. Energy proportionality is able to be achieved by eCope without knowing the details about the service by taking advantage of any configurable hardware that fits our assumption. The advantage of eCope has been demonstrated in three typical back-end services: file system, database service and web-based service.

2. We proposed a new effective method to indirectly control the data center power by statistically influencing the amount of jobs scheduled to a row. We implemented

a new system, Ampere [96], that increases the computation capacity in large-scale data centers with fixed power budget by cultivating the otherwise unused power at a large scale; Ampere suggests a simple yet powerful interface to connect the power controller with the job scheduler, without modifying the job scheduler logic. And we conduct a large-scale empirical evaluation in a real data center with production workload, and detailed performance measurement using controlled experiments.

3. A new power scheduling system, Pelican [54], is proposed for QoS in large-scale data centers with fixed power budget and heterogeneous workloads by improving power utilization on a large scale; A two-level design is adopted to separate overall power scheduling and local power controlling for each server, which makes our power scheduling system more flexible and efficient for heterogeneous workloads; A simple and effective method is introduced that leverage task priority and heterogeneous workload property to improve QoS without modifying the task assignment; A large-scale empirical evaluation of Pelican is conducted against production workloads in a real data center using controlled experiments to show the effectiveness and performance of our system.

1.6 Outline

The rest of this document is organized as follows: Chapter 3 introduces an energy efficient framework, eCope, for server level. A row level power controlling system, Ampere, will be described in Chapter 4. A rack level power scheduling system for heterogeneous workloads, Pelican, is proposed in Chapter 5. Finally, Chapter 6 concludes the dissertation and describes future work.

CHAPTER 2 BACKGROUND AND RELATED WORK

As we discussed in Chapter 1, the key of delivering more computation ability given existing power infrastructure is how to practically maximize the power utilization and optimize the performance per power budget.

The problem can be divided to different levels because of the nature of the power hierarchy in data center. In this chapter, we describe the background and state of the art energy efficient system designs for data center.

2.1 Server Level Energy Efficient System Designs

On server level, researchers usually try to improve energy efficiency from two different aspects of view: the service itself or the available hardware. So previous studies in this area generally fall into two ways: One way is to understand how a specific service is running and the using these information to do optimization. The other way is to utilize new features of hardware or design new hardware or device.

Characteristics of a service or application is helpful to do fine-grained optimization. Xu *et al.* [103] propose an energy-aware query optimization framework, PET, enables the database system to run under a DBA-specified energy/performance tradeoff level via its power cost estimation module and plan evaluation model. They also introduce a power-aware online feedback control framework for energy conservation at the DBMS level based on rigorous control-theoretic analysis for guaranteed control accuracy and system stability [104]. Both work are built as a part of the PostgreSQL kernel.

Zheng *et al.* [108] notice that storage servers consume significant amounts of energy and are highly non-energy-proportional. So they propose a storage system, called Log-

Store, that enables two-speed disks to achieve substantially increased energy proportionality and, consequently, lower energy consumption. Psaroudakis *et al.* [71] argue that databases should employ a fine-grained approach by dynamically scheduling tasks using precise hardware models and so they propose a dynamic fine-grained scheduling for DBMS memory accessing. Lang *et al.* [44] focus on designing an energy-efficient clusters for database analytic query processing. They explore the cluster design space using empirical results and propose a model that considers the key bottlenecks to energy efficiency in a parallel DBMS. Amur *et al.* [2] focuses on large-scale cluster-based storage and data-intensive computing platforms that are increasingly built on and co-mingled with such storage. They propose Rabbit, which is a distributed file system that arranges its data-layout to provide ideal power-proportionality down to very low minimum number of powered-up nodes.

On the other hand, using new hardware design can also reduce the power directly for different kinds of services. Malladi *et al.* [56] observed that currently DDR3 memory in servers is designed for high bandwidth but not for energy proportionality. Mobile-class memory, however, addresses the energy efficiency challenges of server-class memory by forgoing more expensive interface circuitry. Therefore they take advantage of mobile DRAM devices, trading peak bandwidth for lower energy consumption per bit and more efficient idle modes. Zhang *et al.* [107] believes that current fine-grained DRAM architecture incurs significant performance degradation or introduces large area overhead. So they propose a novel memory architecture called Half-DRAM. In this architecture, the DRAM array is reorganized that only half of a row can be activated. Hu *et al.* [33] focused on how energy-saving mechanisms through the design of Internet transmission equipment e.g. routers, and green reconfigurable router (GRecRouter). they mainly contribute to the

design and manufacture of some core components of a green Internet like energy-efficient routers. Lo *et al.* [51] present PEGASUS, a feedback-based controller that using new feature of current available CPU, called Running Average Power Limit (RAPL) to improves the energy proportionality of WSC systems.

Yong *et al.* [23] present a practical and scalable solution, Cloud- PowerCap, for power cap management in a virtualized cluster. It is closely integrated with a cloud resource management system, and dynamically adjusts the per-host power caps for hosts in the cluster. Chen *et al.* [12] try to address challenges of reliability and energy efficiency of resource-intensive applications in an integrated manner for both data storage and processing in mobile cloud using k-out-of-n computing. Kazandjieva *et al.* [38] take the advantages of different classes of devices and put the application running on the best location. Their implementation , called Anyware, provides desktop-class performance while reducing energy consumption through a combination of lightweight clients and a small number of servers. Recently, it is suggested that we can halt the system when the it is idle, and using a static rate when it is busy. This strategy performs almost as good as an optimal speed scaling mechanism [99]. Wong *et al.* [100] present Knight Shift that presents an active low power mode. By the addition of a tightly-coupled compute node, their system enables two energy-efficient operating regions. Liu *et al.* [49] present a runtime power management tool called SleepScale, which is designed to efficiently exploit existing power control mechanisms. Pillai and Shin [69] present real-time DVS algorithms that modify the OS's real-time scheduler to provide energy savings while maintaining real-time deadline guarantees.

Compared with previous work, our work is looking for a general workload-aware

framework that can improve energy proportionality without knowing the details about the target service. So that we do not need to modify the current service, and thus can support various kinds of services. On the other hand, although we use DVFS as an example of the configurable hardware in our case study, our framework can take advantage of any configurable hardware (even for future hardware) that fits our assumption. For example, in MySQL and PHP/Apache case studies, the optimized configuration includes the NIC configuration.

2.2 Higher Level Energy Efficient System Designs

On a higher level, Fan *et al.* did the first quantitative study on large-scale data center power consumption [22]. They showed that there were wide gaps between the average power utilization on rack, PDU, and cluster levels. They pointed out the potential opportunities of using power over-provisioning to increase data center capacity, and proposed a theoretical approach to implement over-provisioning with power capping mechanisms. Wang *et al.* [94] further characterized the power utilization. In particular, they focused on the power peaks and valleys. Many projects on power optimization, including ours, confirmed the observations in these work, and designed control mechanisms based on these observations.

There are two major approaches to manage power: using hardware features like DVFS, and using power-aware workload scheduling [64, 66]. We introduce related work in both categories and describe the uniqueness of our approach.

First main approach is controlling power by hardware power capping. Simple mechanisms directly control the hardware power states (sleep, off, on). PowerNap [58] and

Anagnostopoulou *et al.* [3] target to minimize idle power and transition time within different power states. PowerNap turns several components into power saving states when the server is idle, and wakes them up upon a user request. Given the time it takes to switch between states, people have proposed different ways to minimize the impact to SLAs during transitions [60]. Liu *et al.* showed that it is possible to exploit the best power policy for a given SLA constraint on a single node, and proposed SleepScale, a runtime power management tool to efficiently apply power control [50].

More advanced mechanisms use hardware features like power scaling. Power capping by DVFS imposes an effective power control over CPU and DRAM [22]. The challenge is to lower the system speed while keeping the job SLA violations as few as possible. Sharma *et al.* implemented a feedback-based power management protocol that can provide some SLA guarantees when DVFS is enabled [80]. Lo *et al.* [52] proposed a more general version called PEGASUS, which works on a data center level. Raghavendra *et al.* proposed a theoretical power management proposal [73] with provable correctness, stability, and efficiency. This proposal is hard to implement in real data centers, as it requires highly coordinated control on both hardware and software layers, which is costly to achieve in realtime.

On the other hand, workload scheduling and power management Many approaches use server consolidation. They transition idle servers into low-power or power-off states when the utilization is low [9, 15, 36, 42, 46, 48, 70, 77, 83, 98]. IBM proposed a real-time power management algorithm for parallel computers, which uses workload history to predict short-term workload fluctuations and then decides which servers to turn on and off [10]. Xu *et al.* [102] proposed a technique by adjusting the number of active nodes

using workload information under certain time intervals. However, turning off servers is a complex process that requires process migration or restarts, and thus it is very hard to guarantee the SLA requirements [52]. Freezing servers in Ampere is different, as it just rejects new jobs and does not affect existing ones.

Researchers have also proposed ideas on how to integrate power management into the job scheduler to achieve better power usage pattern. Chase *et al.* used a dynamically reconfigurable switch to control the routing of requests, so they would use a server that could optimize energy consumption while satisfying the SLA [11]. Verma *et al.* built a power-aware application placement controller for high performance computing clusters [91]. Facebook built *Autoscale* to keep all active servers above a moderate CPU utilization, in order to achieve better power efficiency [5]. It was tightly coupled with the job scheduler, who chose a subset of servers as the *active pool* and automatically adjusted the pool size. On the other hand, Power aware scheduling policies can also be integrated into other QoS-aware cluster management systems [19, 20, 53, 90, 92, 105], which will further complicate the design and implementation of the cluster schedulers. Instead of a tight coupling with a scheduler designed for a simple workload pattern or a scheduler which has been over-complicated, our loose coupling with scheduler allows Ampere to integrate with complex data-center-level job schedulers with unknown custom policies and job patterns. Govindan *et al.* used workload power profiles and implemented dynamical power provisioning on the PDU-level using DVFS [25]. They implemented their technique in a data center prototype and showed the improvement of the Computation per Provisioned Watt (CPW) for a few types of applications (e.g. TPC-W). For multi-tenant data centers, Niangjun Chen *et al.* analyze the supply function bidding in the context of demand response [13], which find

the difference between the tenants' performance cost minimization problem and another problem that characterizes market outcome. Qihang Sun *et al.*, on the other hand, study the fair demand response in multi-tenant data centers based on max-min fairness [86].

Commercial data center power management solutions focus on power monitoring, visualization and reporting. Some tools provide an interface to implement power capping. At low level, most tools, including ours, use the IPMI specifications to communicate with the Baseboard Management Controller (BMC), to monitor power draw, among other information, from individual servers. The software tools integrate the power at rack, row or data center level.

Common tools include Intel DCM Energy Director [35], IBM PowerExecutive [34], HP Thermal Logic [31], and Cisco Unified Computing System [16]. Most of the tools are vendor specific, and do not handle data centers of our scale, and thus we build our own power monitoring solution.

2.3 Energy Efficient System Designs for Heterogeneous System

The major risk of using over-provisioning is power outage so power scheduling becomes the core part of using it. It can be classified into two directions: scheduling on the spatial dimension like power-aware workload scheduling and scheduling on the temporal dimension like task resource management [64, 66].

Ideal power scheduling on temporal dimension reduces power draw of a task when the power budget is not enough and assigning more resources after there are available power resources in order do not affect the performance of the task. Sharma *et al.* proposed a real-time power management protocol in the Linux kernel mainly designed for

web service [80]. Lo *et al.* proposed a feedback system that learns from request latency statistic and adjusts hardware limits to provide just fast enough server power for data center [52]. Luo *et al.* designed a general platform for back-end workloads to achieve energy proportionality by finding the best hardware configuration for given workload and service properties [55]. Zheng *et al.* explored the combination of power capping using use four kinds of DVFS algorithm and power shaving by UPS batteries [109]. On the other hand, utilizing other configurable hardware or software resource can be found in recent works. Sun *et al.* proposed performance-equivalent resource configuration(PERC) to reduce power usage while keeping the same performance [84]. Kontorinis *et al.* introduced an architecture that equips each server with a UPS so that the server can discharge the battery hen power budget is low and charge the battery otherwise [41].

Wrong task placement is one of the reasons why the power budget of a subsystem is not enough while global power budget is still enough. Scheduling task for energy or power has been widely studied. One of the simple ideas is place tasks together and turn off unused servers, which is called server consolidation. By reducing the idle power, researches tried to improve the overall energy efficiency of a data center [9,15,36,42,46,48,70,77,83,98]. PowerNap [58] and Anagnostopoulou *et al.* [3] leveraged hardware power states to change the power usage of a server according to the current workload. One key problem to do so is the long transition time [60] so SLAs or QoS is usually taken into consideration.

The core reason for wrong task placement is usually considered as the fault of Job scheduler. Nevertheless, many researchers tried to integrate power management into the job scheduler or combine power scheduler with QoS-aware cluster management systems [11, 19, 91]. Yao *et al.* proposed a framework called TS-BatPro to rearrange batching jobs

to save energy for multi-core servers in data center [106]. It studies the performance and power characteristics of a server and schedules global batching jobs based on the model. Tesfatsion *et al.* introduced a dynamic resource management and scheduling system to improve energy efficiency for cloud services [88]. Cheng *et al.* proposed heterogeneity-oblivious task assignment method, E-Ant, which can improve overall energy efficiency without hurting the performance of a heterogeneous Hadoop cluster [14]. Petrucci *et al.* proposed a QoS-aware task management solution that learns from existing tasks map efficient device to meet the QoS requirements [68]. These systems work well on its own but it is hard to integrate into real data center job scheduler because there have been tremendous factors for it to decide where to place task.

CHAPTER 3 ECOPE: WORKLOAD-AWARE ELASTIC CUSTOMIZATION FOR POWER EFFICIENCY OF HIGH-END SERVERS

In this Chapter, we try to find a general workload-aware approach to achieve energy proportionality for servers in a datacenter. What we propose is called eCope, Workload-aware **E**lastic **C**ustomization f**o**r **P**ower **E**fficiency of High-End Servers, aiming to improve energy-proportionality by workload-aware hardware customization for servers in data centers. More specifically, given a specific application and a workload range, we want to provide a framework that can find a way to achieve energy proportionality through hardware customization for a server. With eCope, we can find an optimized dynamic workload-power function and customize hardware according to both workload and the related optimized configuration.

3.1 eCope Design

Although Barraso and Hale propose energy proportional design for datacenter servers [7], there is no precise definition of how we can describe energy proportionality. The workload-power relation functions for current servers are still much higher than linear relation function [7], especially in the regular workload interval. To improve the energy proportionality, we want to reduce the power for the same workload. So, we use $\frac{workload}{power}$ to describe the energy proportionality. If the power is reduced for the same workload, this metric becomes larger. The aim of eCope is to find a general method and framework to improve energy proportionality for servers within a datacenter. The servers that we focus on should satisfy the following assumptions:

- It is dedicated to run a particular application.

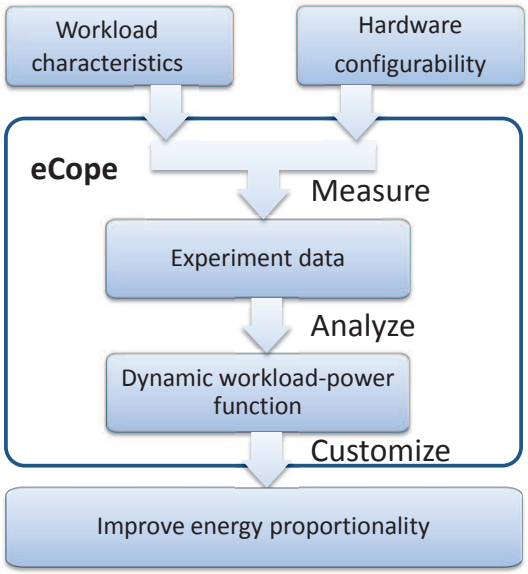


Figure 3: The outline of eCope.

- Components of the server should be configurable to different states.
- For each configuration, the workload-power relation does not change over time.

There are two key observations behind our methodology. First, if we can fix the workload, different hardware configurations may result in different power behavior. There must be an optimal hardware configuration for this particular workload, so that we can do customization to improve energy proportionality. Second, for different workloads, the optimal hardware configuration may be different. Thus we need to have an elastic customization. In short, our goal is to identify the best hardware configuration under different workloads.

Figure 3 shows the outline of eCope. The main input of eCope is workload characteristics and hardware configurability. Workload characteristics refer to the metric of instant performance such as network throughput, request per second, CPU utilization and so on. It can be measured by monitoring the NIC, CPU or the service. User needs to choose a suitable metric to describe the workload for their service. In our case studies, we choose

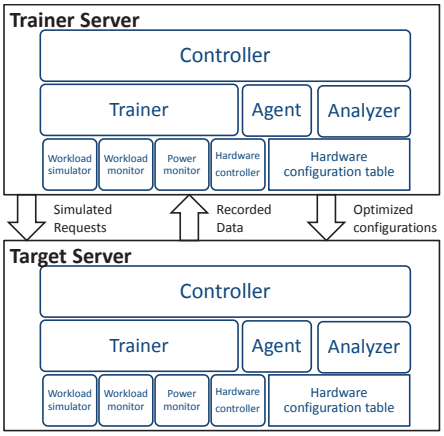


Figure 4: The pair training process of eCope.

the network throughput. Hardware configurability means what and how components can be configured (i.e. CPU can be switched into different frequencies, the hard disk can be set to different modes and so on). Although every configurable hardware can be included, it's better to choose the ones that can affect power effectively.

The basic eCope process consists of three phrases:

(1)Pair training. We do training to get the relationship between the workload and the power for a given environment.

(2)Analyzing. We then fit measuring data to get an optimized dynamic workload-power function.

(3)Application. We apply the customization to improve energy proportionality.

3.1.1 Pair training

The first phase is to do the training. Figure 4 shows the process of training as well as the structure of the eCope framework. Two servers are paired to train each other sequentially. The server that we want to optimize energy proportionality is the target server, and the other one that simulate requests and do analysis is the trainer server. eCope is deployed on both trainer server and target server.

The trainer component on both trainer sever and target server are active in this phase. They cooperate to call the hardware controller on target server so as to set the hardware to every possible configuration. And for each configuration eCope will do the following: 1)The power monitor on the target server measures the idle power. 2)The workload simulator on trainer server trigger requests to target server at different levels to generate necessary workload. Meanwhile, the power monitor and the workload monitor on target server record the real workload and related power dissipation on target server.

The workload simulator, power monitor, and workload monitor and hardware controller involved in these processes are the basic components of eCope and may varies for different hardware or services. For example, we can use NodeManager to monitor system power, or we can use wattsup to monitor system power. It depends on what kind of devices are available. So, in our design, we use plugin mechanism to makes it flexible. Each of these components has a selector to determine which one to use at run time. So that we can implement both NodeManager based monitor and wattsup based monitor as plug-ins and the power monitor selector will choose the right one to use according to the user input.

3.1.2 Analyzing

After all the data described are collected, the target server sends them to trainer server. In the second phase, the Analyzer on the trainer server is responsible to analyze the measured data and sends the optimized configuration table to target server. To do so, we first need to find a proper function to do curve fitting for workload-power relation. After the curve fitting for each configuration, we can get a set of static workload-power functions $\{f_1,f_2,...,f_n\}$.

The static workload-power function is the workload-power relation function associated with just one configuration. If a workload-power relation is achieved by using more than one configuration (which means in different workload intervals the power may be related to different configurations), then we refer it as the dynamic workload-power relation. Its function is called the dynamic workload-power function. We denote dynamic workload-power function as $(\{f_1,f_2,...,f_n\}, \{x_1,x_2,...,x_n\})$, where f_1 to f_n are static workload-power functions and x_1 to x_n are intervals that f_1 to f_n are effective on respectively. The union of all x_i should be the whole possible workload interval, and x_i should be pair-wise disjoint. Formally, $(\{f_1,f_2,...,f_n\}, \{x_1,x_2,...,x_n\})$ means:

$$f(x) = \begin{cases} f_1(x), & if \quad x \in x_1 \\ f_2(x), & if \quad x \in x_2 \\ & \quad ... \\ f_n(x), & if \quad x \in x_n \end{cases} \tag{3.1}$$

In this way, we can mix different configurations on one graph. For simplicity, we can treat static workload-power function as a special dynamic workload-power function that

has only one function and one interval. Among all possible dynamic workload-power functions that can be constructed by a certain set of static workload-power functions, there must exist an optimal dynamic workload-power function that achieves the best energy proportionality under every possible configuration and also meets the performance requirement and energy condition(which we will discuss in detail in Section 2.4). We refer it as the optimized dynamic workload-power function. So the aim of this phase is to find the optimized dynamic workload-power function.

Generally, we can obtain all the intersection points to separate the workload interval and find the functions that have the lowest power in each interval and meet the performance limitation. Then combine these functions together with the interval that is between two neighboring intersection points. In this way, the complexity is $O(n^3)$.

Since obtaining optimized dynamic workload-power function only need to be done once, and there are not too much hardware configurations on current servers, such complexity is acceptable. In fact, in our case study, the calculation spends less than 1 second. Even though, For most particular fitting functions, we may have better ways to get the optimized dynamic workload-power function. These methods are not mainly for improving performance, but for easier programming. We will see an instance of how to do so in the case study part.

Here, the analyzing process is done on the trainer server. However, since both trainer server and target server have an analyzer component, the analyzing process can be done on target server as well. User can choose which one to use for their convenience.

3.1.3 Application

After the optimized dynamic workload-power function for both servers are obtained, these servers can just work on its own(not paired), and customization can be achieved according to this function. In other words, when the service is running, the agent component monitors the workload and applies the configuration related to the interval where the current workload is. For example, if the optimized dynamic workload-power function is ({f_1,f_2,f_3}, {(10,30],[0,10),[30,50]}) and the current workload is 20, then configuration 1 will be applied.

3.1.4 Discussing

Our methodology can be applied to any applications running on the server that meet our assumptions. We do not limit the type of hardware or application in our method. Currently, we can modify CPU frequency, network speed, hard disk mode. In the future, we may be able to change the memory frequency. User can choose any hardware that satisfy our assumption.

According to our definition, we try to increase $energy \quad proportionality = \frac{workload}{power}$. And

$$\begin{aligned} Energy &= \int_0^t \text{power}\,\mathrm{d}t \\ &= \int_0^t \frac{\text{workload}}{\frac{\text{workload}}{\text{power}}}\,\mathrm{d}t \\ &= \int_0^t \frac{workload}{Energy \quad proportionality}\,\mathrm{d}t. \end{aligned}$$

We can see that if the workload can be fixed or the workload does not change too much, then only when energy proportionality increases, the energy decreases. And the workload is determined by the user, which means it is independent to the configuration. Thus to

improve energy proportionality is equivalent to reducing the energy. On the other hand, if the workload is allowed to change with in a performance requirement, we need to know how much energy proportionality improvement is required to ensure energy saving.

To do so, we need to have a performance limitation to prevent too much performance loss. Assume the maximum performance loss could be α(percentage), after we change the configuration, denote the new workload as $workload'$, and denote the new execution time as t'. Then

$$\overline{workload'} \geq (1-\alpha) \cdot \overline{workload}$$

$So \quad that$

$$t' = \frac{1}{\overline{workload'}} \leq \frac{1}{(1-\alpha)\cdot\overline{workload}} = \frac{1}{1-\alpha} \cdot t$$

$Then,$

$$\begin{aligned} Energy &= \int_0^{t'} \frac{workload'}{Energy \quad proportionality} \, \mathrm{d}t \\ &\leq \int_0^{\frac{1}{1-\alpha}\cdot} \frac{workload}{Energy \quad proportionality} \, \mathrm{d}t \\ &\approx \frac{1}{1-\alpha} \cdot \int_0^{t} \frac{workload}{Energy \quad proportionality} \, \mathrm{d}t. \end{aligned}$$

This means that if energy proportionality increase more than ($\frac{1}{1-\alpha}$) times, it can ensure energy saving, otherwise, otherwise, the configuration should not be considered. This is called energy condition in our paper. Since it is deduced by performance requirement, when we say performance requirement in this paper, it also includes energy condition.

We find that the dynamic workload-power relation is a good tool to show the effect of hardware configuration. Not only because we can easily compare energy proportionality under different workload-power relations on the graph, but also because it provides a uniform method to calculate the optimized configuration. We can also use it to do customization. Therefore, the dynamic workload-power relation is the core data structure of

eCope.

3.2 Case Study

As we described in Section 2, we are interested in certain applications running on the dedicated servers in datacenters. There are three particular services: file system, database services and web-based services.

We take TFS, MySQL, and PHP/Apache as our case studies since they fit our assumption in Section 2 very well, and they are also typical types of back-end services running in real-world. TFS is a Linux-based distributed file system which provides high reliability and concurrent access by redundancy, backup, and load balance technology. TFS is mainly designed for small files less than 1MB in size and adopts a flat structure instead of the traditional directory structure. The open source TFS project is developed and maintained by Taobao, a part of Alibaba Group.

In our case studies, the throughput of the network transfer rate is a good metric for workload. It is obvious for TFS and MySQL. For PHP/Apache case study, although request per second is also a good metric, network throughput can equivalently describe the workload since the page sizes are the same in our experiment. In addition, network throughput is service independent, which means network throughput monitor can be also used for a wide range of services. We would like to emphasis that user can choose any other metrics that are able to describe workload. To avoid confusion, however, all 'workload' in the case study section refer to the network throughput (measured by Mbps).

We conduct TFS case studies in the same environment as TFS production environment in Alibaba. MySQL and PHP/Apache case studies are conducted on our lab servers.

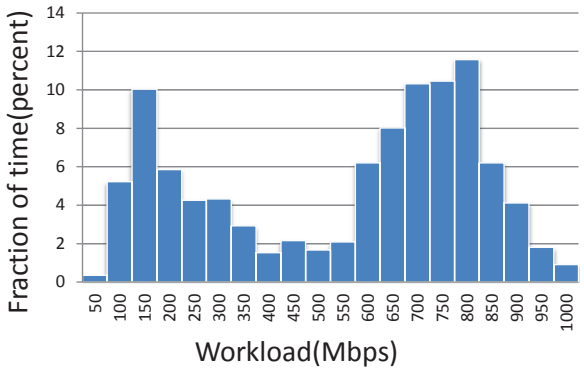


Figure 5: The workload distribution of TFS.

The performance limitation for three case studies is maximum 5% of performance loss. The hardware configurations include the combination of 16 CPU frequencies(from 1.2GHz to 2.5GHz, 2.7GHz, and Turbo boost mode), 3 kinds of Network interface controller(NIC) speeds(10Mbps, 100Mbps, and 1000Mbps), and 3 kinds of disk modes (Normal, Standby, and Sleep). So, each configuration should include these 3 components, e.g. (1.2GHz, 100Mbps, Normal).

The network switches we used in all our case studies are 1G network switches, which means that the maximum network throughput is 1000Mbps, so the range of wokload is 0 to 1000 Mbps. TFS and MySQl are IO intensive. Figure 5 shows a typical normal workday workload distribution of TFS on one server provided by Taobao Crop. We can see that the server is rarely idle, and in most time, the workload is around 100 to 20 Mbps and 600 to 800 Mbps, while the CPU utilization is always lower than 20%. For PHP/Apache case study, we implement a simple service that dynamically computes π and return it through the web interface. Our experiment data shows that when the CPU utilization is 100%, the network throughput is under 60Mbps, which is far below the maximum network throughput, so PHP/Apache case study is CPU-bounded.

3.2.1 Basic components implementation

For TFS, it has its own interfaces to access the files, so the workload simulator was implemented by using a TFS client API. Before the experiment starts, we store amount of files to TFS, and save the filenames of all these files to a filename list. When we launch the workload simulator, we pass a desired number of files and number of processes to it. Each workload simulator process first reads all filenames from the list. Then, randomly picks a desired number of files to read from TFS. To read a file, the workload simulator connects with the nameserver first, and then it sends the block id to main nameserver to get the address of desired dataserver. Later, the workload simulator uses that address to connect with the dataserver, send both the block ID and file ID to it, and get the file data from it directly. Thus, the entire workload can be controlled by passing different numbers of files and processes. Most of the energy is consumed by the dataservers throughout the entire process. In our experiment, we only optimize the dataserver. The nameservers and heart agents run on separate servers.

We use SQL workbench as the MySQL workload simulator. Before the experiment starts, we store a dataset on a MySQL server. When launching the workload simulator, we pass the number of records and the number of processes to it. Each workload simulator process queries the same number of records by generating a SQL statement.

We use Apache Bench as the PHP/Apache workload simulator. We implement Chudnovsky algorithm in a php page that dynamically computes π using BCMath arbitrary precision mathematics functions in PHP. Then, we invoke Apache bench with a desired number of requests and number of concurrency to access the page.

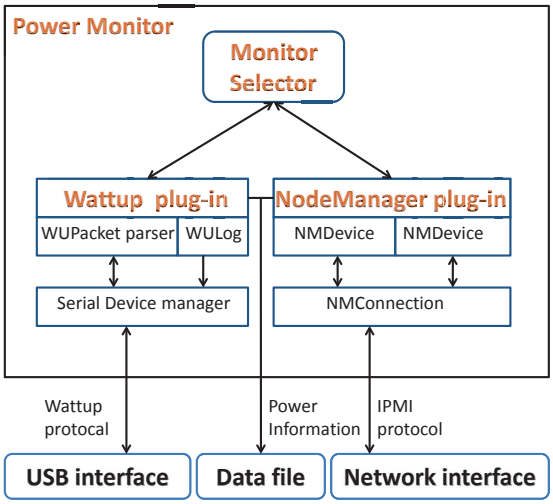


Figure 6: The structure of power monitor.

We use the same power monitor and workload monitor for our case studies. The workload monitor records NIC throughput and the power monitor reads power by using IntelÂő Node Manager and Watts up. Node Manager is a set of hardware and software to optimize and manage power and cooling resources in the data center. This server power management technology extends component instrumentation to the framework level and can be used to get power information from sensors integrated on motherboard chips.

Our power monitor can read power information from both Wattsup and Node Manager. We connected the Wattsup device to our dataserver, but the power data can be read from the USB interface connected to the Wattsup device. Node Manager is supported on our dataserver, we can read information locally. We can also read power information from Node Manager through the network interface by using IPMI protocal. In order to limit the overhead on the dataserver, we read all power data from another server. Since the data transferred by Node Manager is quite small (less than 1kbps each time) compared to the workload of TFS or MySQL (measured by Mbps), we can neglect it and consider all

network data to be generated by TFS or MySQL.

Figure 6 shows the structure of our power monitor. When it is launched, we must first specify which drive to use. Both drivers implement the same interfaces. For Wattsup, it can only monitor the whole system power. We construct a serial device manager to communicate with the USB interface. It sends the command packets and receives data packets(called WUPacket) by using Wattsup protocol, but it does not know the meaning of these packets. The WUPacket parser mainly processes the data packet, and WUlog component is used to construct a command packet to control the Wattsup device. Wattsup drive asks WUlog to initialize the Wattsup device on start. When receiving the packet from the serial device manager, WUPacket parser extracts each field in the packet and returns it to the Wattsup drive. However, in some cases, the Wattsup device does not send out data for a long time.

By default, we ask Wattsup device to collect data for every one second. If Wattsup drive finds that the serial device manger has failed to read data from USB for a certain time, then it will ask WUlog to reset the device. For Node Manager, we have two ways to connect to the network. We choose the Intelligent Platform Management Interface(IPMI) approach because it can be used on other servers that do not support Node Manager but still support IPMI. Node Manager is different from Wattsup. It can monitor not only total system power, but also component level power, such as CPU and memory. So there are several Node Manager device classes that are responsible for each component. All Node Manager device classes use the same Node Manager connection class to communicate with Node Manager on the target server. Both the Wattsup driver and NodeManager driver can save the power information to a specific data file.

3.2.2 Process

In the measuring phase, the trainer component collects the power and workload information under various situations, including the following: (1)when the system is idle. (2)when turning on the TFS or MySQL but not putting any workload on it. (3)when there are workload on TFS or MySQL server, but no hardware control. (4)when there are workload on TFS or MySQL server, and TFS or MySQL is running under a certain hardware configuration. We can analyze these data to identify the optimal configuration for a particular workload.

Before we start analyzing, we need to determine a fitting function for workload-power relation. This function is related to the environment. User can choose the best one fits their training data. We tried different types of functions like linear, polynomial, power, exponential function and so on. We decide to use power function because its coefficient of determination(or R-square) shows the best fitting result among all these functions, which means the power function is the best one to describe the relation between workload and power for our experiment platform. The power function has the form:

$$DynamicPower = a * workload^{b}. \tag{3.2}$$

Although the optimized dynamic workload-power function can be calculated by the method described in the Section 2, we find a better way to do this for our case studies. Suppose two configurations(denoted as A and B) have the workload-power relation $f = a_1 * x^{b_1}$ and $g = a_2 * x^{b_2}$, where $a_1, c_2 > 0$ and $b_1, b_2 > 0,$ there is only one positive intersection point:

$$x = (\frac{a_2}{a_1})^{\frac{1}{b_1 - b_2}} \tag{3.3}$$

procedure OPTIMIZED DYNAMIC WORKLOAD-POWER FUNCTION
$i = 0; f_1 = g_1; w_1 = Maxworkload;$
  $y = g_1(Maxworkload);$
  $Candidate = \{g_v | v = 1 \quad to \quad n\};$
  for $k \leftarrow 1, N$ **do**
    if $g_i \quad violate \quad the \quad perf. \quad limitation$ **then**
      $Candidate = Candidate - g_i;$
    else
      if $g_i(Maxworkload) < y$ **then**
        $f_1 = g_i;$
        $y = g_i(Maxworkload);$
  $Candidate = Candidate - \{f_1\};$
  $i = 2; tmp = 0; Over = \emptyset; f_{i+1} = NULL;$
  while $f_{i-1} \neq NULL$ **do**
    $w = 0;$
    for $g \in Candidate$ **do**
      $tmp = (\frac{aValue(g)}{aValue(f_{i-1})})^{\frac{1}{bValue(f_{i-1}) - bValue(g)}};$
      if $tmp >= w_{i-1}$ **then**
        $Over = Over \cup \{g\};$
      else
        if $tmp > w$ **then**
          $w = tmp;$
          $w_i = w;$
          $f_i = g$
    $Candidate = Candidate - Over - f_i;$
    $i = i + 1;$
  return $(\{f_1, f_2, ..., f_{i-1}\}, \{[0, w_{i-1}), ..., [w_3, w_2), [w_2, w_1]\});$

Algorithm 1: Obtain Optimized dynamic workload-power function

It means that this point is a turning point. If configuration A consumes less power when the workload is lower than this point, then configuration B consumes less power when the workload is higher than this point. Of course, mathematically, this point can be any value even higher than the maximum possible workload, so we need to check whether the point is in the range(in our case 0 to 1000).

Assuming that the set of static workload-power functions is $G = \{g_v | v = 1 \quad to \quad n\}$.

Figure 7: The system idle power over 3 periods.

Using equation (3), algorithm 1 gives a better way to obtain the optimized dynamic workload-power function.

In the worst case, we can find one function during each iteration and the set $Over$ is always empty. This results in a running time complexity of $O(n^2)$. Given the fact that the configurations on current servers are not too much, and the training just need to be done once, the performance is not an issue. We finish the calculation less than one second for both TFS and SQL case studies. The benefit of this algorithm is to make programming easier.

At last, eCope applies customization. The agent component lookup the optimized dynamic workload-power function periodically and change the hardware configuration if needed.

3.2.3 Evaluation

To evaluate eCope, we first measure system idle power, TFS idle power, MySQL idle power, and PHP/Apache idle power under all possible hardware configurations. We also

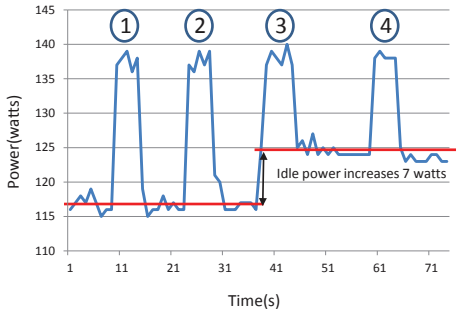


Figure 8: Total system power when four jobs are executed sequentially with same workload but different idle power.

Table 1: The average system idle power and the average TFS idle power

Freq. (GHz)	Sys idle power (Watts)	TFS idle power (Watts)	Freq. (GHz)	Sys idle power (Watts)	TFS idle power (Watts)
1.2	164.53	173.91	1.8	164.54	173.91
1.3	164.51	173.93	1.9	164.51	173.87
1.4	163.49	173.88	2.0	164.50	173.93
1.5	164.52	173.97	2.1	164.54	173.90
1.6	164.54	173.89	2.2	164.55	173.96
1.7	164.50	173.96	Turbo Boost	117.27	121.19

measure the static workload-power relation without any optimization to get a baseline. Then we launch the simulator to supply workload on TFS, MySQL and PHP/Apache, and proceed to measure the system power under different hardware configurations. Next, these data are fit into the power function in order to get the static workload-power function, and an optimized dynamic workload-power function is calculated by using algorithm 1. Lastly, we apply the customization and compared the power saving.

When we do the baseline measurement, the DVFS funtion is turned off in BIOS setting so that no governor is activated, the NIC speed is 1000Mbps, and the disk mode is normal. Otherwise, userspace governor is used, so that the CPU frequencies are controlled by eCope completely.

Table 2: The average system idle power and the average MySQL idle power

Freq. (GHz)	Sys idle power (Watts)	MySQL idle power (Watts)	Freq. (GHz)	Sys idle power (Watts)	MySQL idle power (Watts)
1.2	117.46	117.51	2.0	117.71	117.96
1.3	117.50	117.73	2.1	117.66	117.79
1.4	117.40	117.47	2.2	117.51	117.63
1.5	117.40	117.54	2.3	117.59	117.90
1.6	117.34	117.63	2.4	117.45	117.65
1.7	117.41	117.74	2.5	117.35	117.69
1.8	117.31	117.88	2.7	117.33	117.35
1.9	117.23	117.79	Turbo Boost	117.27	117.46

Experimental environment MySQL 5.1.52, PHP 5.5.22, Apache 2.4.12 are set up on an Intel R2000GZ family server in our lab as target server with Intel Xeon CPU E5-2680 0 @ 2.70GHz and DDR3 1333MHz 8*8GB Memory. Our workload simulators are deployed on a Dell 01V648 server. The Intel server is the target server and the Dell server is the trainer server. The operating system of the Intel server and the Dell server are Red-Hat 6 x86_64 and CentOS 4 x86_64 respectively. The Intel server and the Dell server are connected by a 1G network switch. The Intel server support Node Manager that enables reading the system power, CPU power, and memory power information. In addition, Wattsup is set up to compare system power to the data collected from Node Manager. TFS 2.1.13 is set up on the same type of server as the production TFS server in Alibaba Group for our experiment. The TFS server is equipped with Xeon CPU E5-2400 0 @ 2.20GHz and 10*10TB disks. Also servers are connected by a 1G network switch, so that the range of workload is 0 to 1000Mbps.

Base line Figure 7 shows some results of system idle power under 3 kinds of hardware configurations. We observe that in this period, the difference between the average power of the highest CPU frequency (2.71GHz) and the average power of the lowest CPU

frequency(1.2GHz) is still less than 1 Watt. Therefore, the average idle power under different configurations is almost the same. In addition, we notice that the power changes periodically. Thus, we define idle power as the average power over integral times of periods. As a result, when we calculate the idle power, we always take the same number of periods of data to avoid errors caused by such periodic phenomena. For each configuration, we collect the system idle power for one day. Table 1 shows the average system idle powers(si power), which indicates that the system idle power is almost the same under different hardware configurations although the power vibrates over time. Thus, we can treat the system power the same under different configurations. Table 1, 2, and 3 also show the average idle power when the service is on but no workload. When TFS is running but has no workload on it, the power increases when the CPU frequency increases. The difference between maximum and minimum power is about 1.4%. On the other hand, When the idle power of MySQL and PHP/Apache does not change so much.

Next, we measure the power under different workloads without any optimization. Since the idle power changes periodically, the dynamic power should be calculated carefully. Figure 8 shows the total system power when the simulator launched four jobs sequentially with the same workload. Number 1 to 4 in the figure shows when these four jobs are launched, and the red line roughly shows the idle power. We can see that when the first two jobs are working, the idle power is about 117 Watts. When the last two jobs are executing, the idle power is about 125 Watts. The total system powers for all those four jobs are, however, almost the same. It is surprising that when the idle power increased about 7 watts, the execution time and total system power are almost the same.

We also check it for those low workloads that consume 130 Watts total system power,

Table 3: The average system idle power and the average PHP/Apache idle power

Freq. (GHz)	Sys idle power (Watts)	PHP/Apache idle power (Watts)	Freq. (GHz)	Sys idle power (Watts)	PHP/Apache idle power (Watts)
1.2	117.33	117.43	2.0	117.41	117.47
1.3	117.43	117.47	2.1	117.54	117.61
1.4	117.38	117.43	2.2	117.63	117.81
1.5	117.40	117.51	2.3	117.39	117.55
1.6	117.51	117.59	2.4	117.67	117.79
1.7	117.42	117.53	2.5	117.42	117.61
1.8	117.57	117.63	2.7	117.52	117.55
1.9	117.39	117.54	Turbo Boost	117.73	117.81

and observed the total system powers are almost the same while the idle power changes periodically. So, it is not capped at a single server level. We repeated the experiment under different configurations and different workloads, and found that this phenomenon is common in our experimental environment. This means that the total power might not always equal to the idle power plus dynamic power. This may be caused by uncore power, but we haven't yet identified why this happens. We will continue to explore the reasons. Since it is not related to this paper, in our evaluation, we define the dynamic power as the average total power minus the average service idle power so that on average, the total power is still equal to the idle power plus the dynamic power. We used the power function, shown in equation (2), to fit the workload-power relation. For TFS, the function is:

$$DynamicPower = 0.1140 \cdot workload^{0.8449} \tag{3.4}$$

For MySQL, the function is:

$$DynamicPower = 0.1310 \cdot workload^{0.8313} \tag{3.5}$$

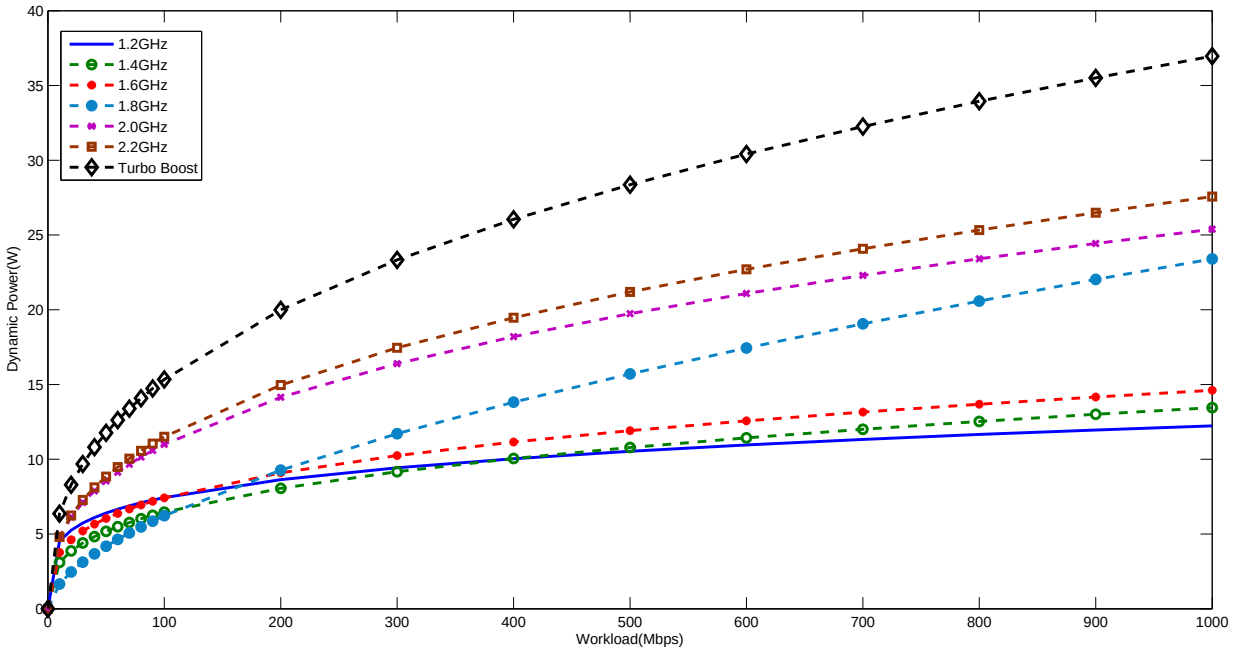


Figure 9: TFS Static workload-power functions(when NIC is 1000Mbps and disk is in normal mode).

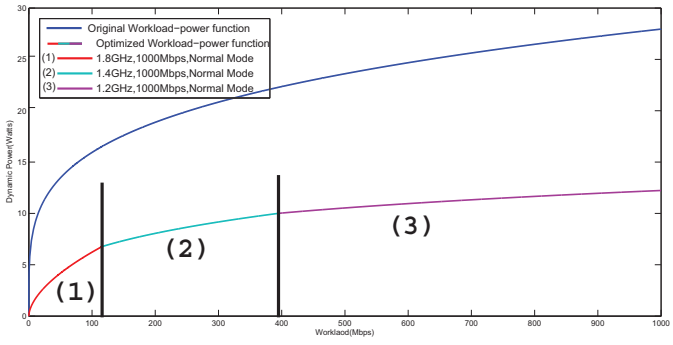


Figure 10: The original workload-power relation and The optimized workload-power relation for TFS.

Table 4: Fitting results for TFS

frequency(GHz)	a	b	R-square	frequency(GHz)	a	b	R-square
1.2	2.7495	0.2161	0.9341	1.8	0.4405	0.5751	0.9445
1.3	2.5566	0.2336	0.9362	1.9	1.5794	0.3956	0.9376
1.4	1.4905	0.3185	0.9276	2.0	2.0666	0.3631	0.9575
1.5	2.2639	0.2610	0.9207	2.1	2.2627	0.3504	0.9554
1.6	1.9094	0.2946	0.8932	2.2	2.0027	0.3796	0.9569
1.7	0.7858	0.4613	0.9311	Turbo Boost	2.6411	0.3820	0.9123

For PHP/Apache, the function is:

$$DynamicPower = 24.8839 \cdot workload^{0.5087} \tag{3.6}$$

These functions are used as the base line to make comparisons with our optimization.

Analyzing In this part, we obtain the optimized workload-power functions. For TFS, Different workloads are achieved by using different numbers of workload simulator processes. All the sizes of test files were 100KB, and each thread operates on 1000 files. Next, we calculate the dynamic power and used power function to do least squares fitting on these data. Table 4 shows the fitting result when the NIC speed is 1000Mbps and hard disk mode is normal. Most b values in the table are smaller than 0.7, which means that the power function fits better than a linear function because the set of linear functions is a subset of power functions. Figure 9 shows part of the related figure of static workload-power functions. Using the algorithm 1, we obtain the optimized workload-power function for TFS as:

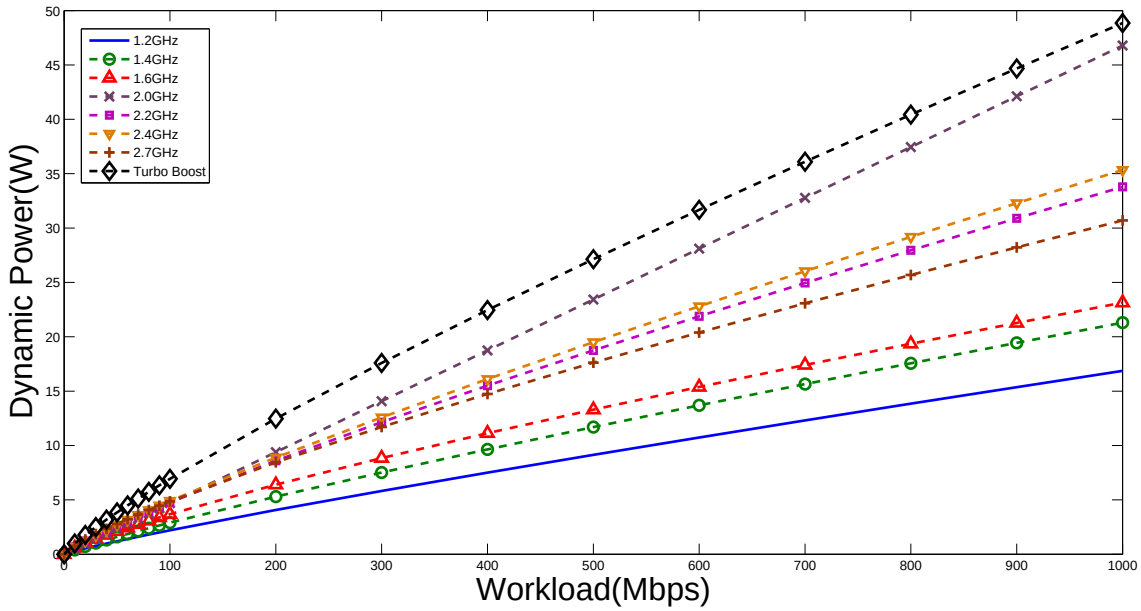


Figure 11: MySQL Static workload-power functions(when NIC is 1000Mbps and disk is in normal mode).

$$f(x) = \begin{cases} f_{1.8}(x), & if \quad x \in [0 \quad , 115.6) \\ f_{1.4}(x), & if \quad x \in [115.6, 395.3) \\ f_{1.2}(x), & if \quad x \in [395.3, 1000] \end{cases} \tag{3.7}$$

The curve with four colors in Figure 10 shows the graph of the function. It contains three configurations that related to different CPU frequencies.

For MySQL, Different workloads are achieved by using different numbers of workload simulator processes. Each process selects half of the data in the database. Table 5 shows the fitting result when the NIC speed is 1000 Mbps and hard disk mode is normal. We can see that b values for MySQL are larger than those for TFS. Some b value even reach 0.9994, which means that it is almost linear. Figure 11 shows part of the related figure of static workload-power functions. Using the algorithm 1, we obtain the optimized workload-

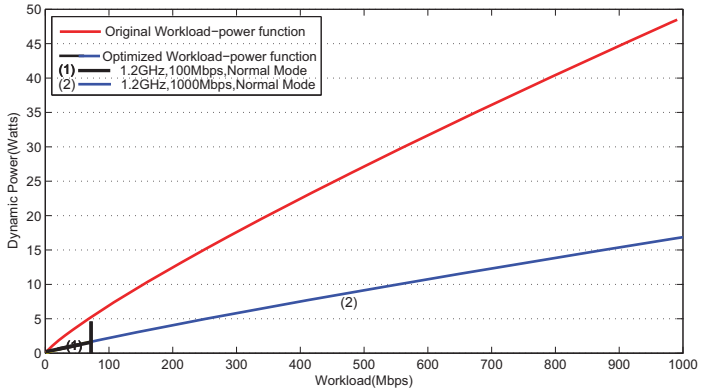


Figure 12: The original workload-power relation and the optimized Workload-power relation for MySQL.

Table 5: Fitting results for MySQL

frequency(GHz)	a	b	R-square	frequency(GHz)	a	b	R-square
1.2	0.03771	0.8835	0.9926	2.0	0.0475	0.9978	0.9901
1.3	0.04482	0.9636	0.9997	2.1	0.1932	0.7234	0.9560
1.4	0.05423	0.8647	0.9937	2.2	0.0951	0.8502	0.9881
1.5	0.16080	0.7411	0.9606	2.3	0.1921	0.7534	0.9842
1.6	0.09302	0.7986	0.9923	2.4	0.0947	0.8573	0.9996
1.7	0.17740	0.7287	0.9806	2.5	0.0832	0.9330	0.9873
1.8	0.18410	0.7534	0.9591	2.7	0.1214	0.8010	0.9376
1.9	0.03406	0.9994	0.9620	Turbo Boost	0.1394	0.8482	0.9220

power function for MySQL as:

$$f(x) = \begin{cases} f_{1.2GHz/100Mbps}(x), if & x \in [0 \quad , 70.9) \\ f_{1.2GHz/1000Mbps}(x), if & x \in [70.9, 1000] \end{cases} \tag{3.8}$$

The curve with two colors in Figure 12 shows the graph of the function. It contains two configurations that have the same CPU frequencies, but different network speed. Figure 12 also shows the base line of MySQL that we obtained in Section 3.3.2.

For PHP/Apache, Different workloads are achieved by using different concurrency level. Table 6 shows the fitting result when NIC speed is 100 Mbps and hard disk mode is normal. Figure 13 shows all related figures of static workload-power functions that meets

the performance requirement and energy condition, when NIC speed is 100 Mbps and hard disk mode is normal. Notice that configurations with lower frequency are not shown in the figure 13 because they either causes more performance loss than performance requirement(which is maximum 5% performance loss) or they violate the energy condition. Using the algorithm 1, we obtain the optimized workload-power function for PHP/Apache as:

$$f(x) = f_{2.4GHz/100Mbps}(x), if \quad x \in [0 \quad , 60] \tag{3.9}$$

The blue curve in Figure 14 shows the graph of the function. It contains only one static workload-power relation function which is easier to apply.

Customization results After we apply the customization to TFS, MySQL, and PHP/Apache, we measure the power under our control and find that compared to the original behavior, TFS can save up to 51.1% of dynamic power and 41.5% dynamic power on average (up to 12.0% total system power, and 7.0% total system power on average) with average 0.57% performance loss. For MySQL, it can save up to 65.5% of dynamic power and 65.3% dynamic power on average (up to 19.3% total system power, and 12.2% system power on average) with average 0.98% performance loss. For PHP/Apache, it can save up to 19.6% of dynamic power and 18.37% dynamic power on average (up to 11.6% total system power, and 9.8% system power on average) with average 4.7% performance loss.

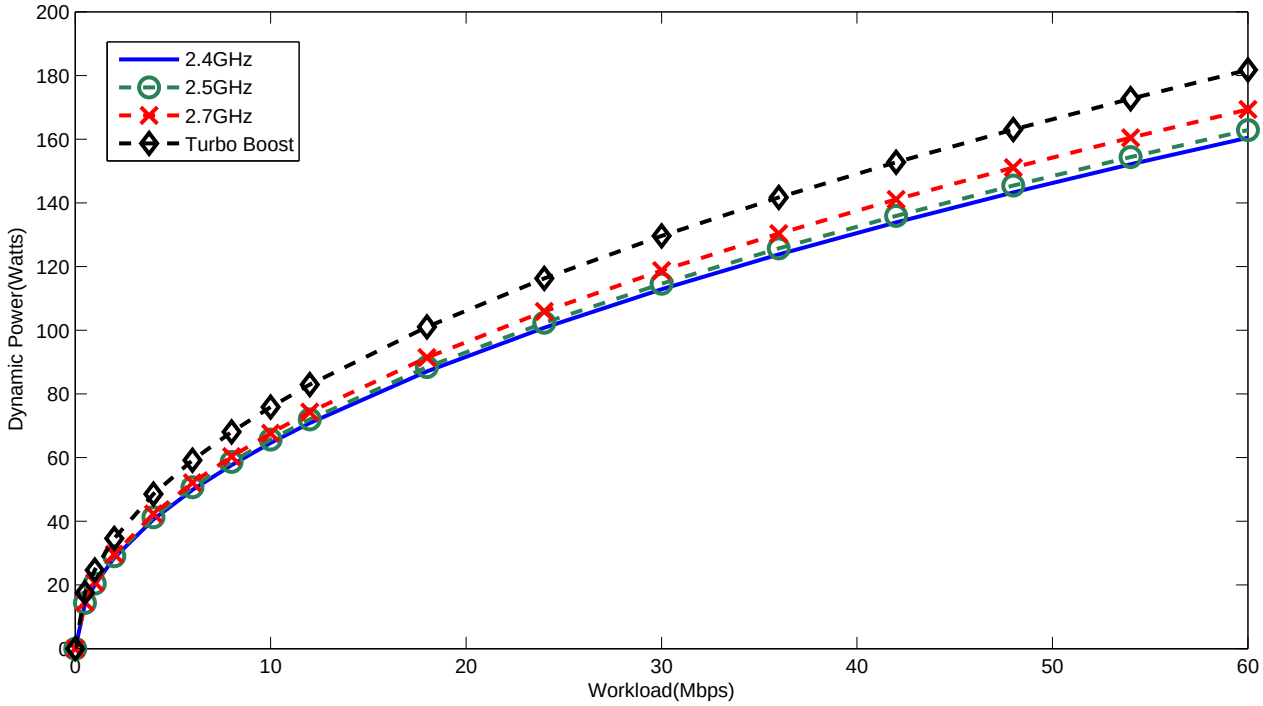


Figure 13: PHP/Apache Static workload-power functions(when NIC is 100Mbps and disk is in normal mode).

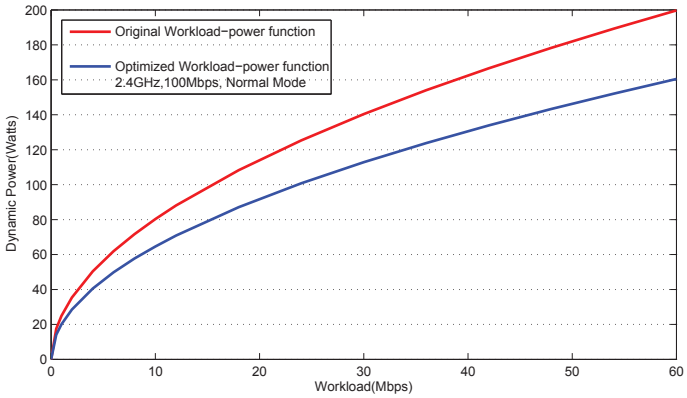


Figure 14: The original workload-power relation and the optimized Workload-power relation for PHP/Apache.

Table 6: Fitting results for PHP/Apache

frequency(GHz)	a	b	R-square	frequency(GHz)	a	b	R-square
1.2	12.8805	0.5651	0.9991	2.0	17.3629	0.5242	0.9990
1.3	14.7596	0.5255	0.9978	2.1	19.1891	0.5043	0.9997
1.4	14.0645	0.5499	0.9993	2.2	20.0848	0.5001	0.9988
1.5	14.7331	0.5431	0.9986	2.3	20.0652	0.5078	0.9979
1.6	15.8533	0.5277	0.9989	2.4	20.4294	0.5070	0.9983
1.7	16.3604	0.5226	0.9976	2.5	20.8062	0.5121	0.9997
1.8	17.2294	0.5140	0.9983	2.7	24.6902	0.4876	0.9689
1.9	17.8611	0.5094	0.9988	Turbo Boost	30.7797	0.5095	0.9983

3.3 Summary

In this chapter, we introduced eCope to improve energy-proportionality by workload-aware hardware customization for servers in data centers. The solution is optimized for a server. To achieve global energy efficiency in large scale data center requires higher level scheduling because of conservative server provisioning and diurnal pattern, which we will see in the next chapter.

CHAPTER 4 INCREASING LARGE-SCALE DATA CENTER ROW LEVEL CAPACITY BY STATISTICAL POWER CONTROL

In this chapter, we present Ampere, a new approach to improve TPW by provisioning extra servers. Ampere keeps the total power under the budget and brings zero performance disturbance to the existing jobs. Different from existing approaches that react to an over-committing event by capping the power draw, Ampere proactively reduces power violations by driving the workload to other less utilized rows and consolidating the unused power scattered among them.

4.1 Background on Data Center Power Provisioning

In this section, we provide some background information on the data center power supply architecture, job management and some important observations of data center power utilization, which lead to the Ampere design.

4.1.1 Data center power provisioning and job scheduling

Row-level power provisioning. The power budget of a data center is normally partitioned into a number of PDUs, each of which serves about 20 racks. Each rack has a power budget of 8-10KW. As the typical rated peak power of a server is about 250W, we can have 40 servers per 10KW rack. This translates to 800 servers per PDU, which we call a *row* of servers. The partitioning is due to the physical limits of commercial power equipment, such as Uninterruptible Power Supply (UPS) and PDUs.

Servers are provisioned according to the power budget at each level. The provisioning is often based on the rated power and we will show that it leads to significant under-utilization of the provisioned power budget.

Power capping. The row-level power budget is enforced by physical circuit breakers (fuses) in each PDU, in order to protect the PDU from overloading. Since it would cause catastrophic service disruptions to cut down the power of hundreds of servers at the same time, power capping is used when the total power utilization of servers in a row is over the row budget. Power capping uses Dynamic Voltage and Frequent Scaling (DVFS) features provided by modern server hardware, and slows down the servers to reduce the power draw [28]. The recently proposed DVFS technique, running average power limit (RAPL) [17], reacts in a very short period of time ($<$ 1ms) to avoid triggering the circuit breaker at the higher level. The downside of DVFS is that it slows down a server without informing applications or the scheduler, which may cause unpredictable performance disturbances and SLA violations (details in Section 4.3.3). We have DVFS enabled in our data centers, but using Ampere, we dramatically reduce the cases where DVFS is triggered.

Job scheduling. Independent of the statically-partitioned power budget, jobs in a data center are scheduled by a centralized scheduler using the entire data center as a single resource pool. Data center job schedulers, such as Borg [92], Omega [78], Mesos [29] and YARN [89] track the utilization of various resources including CPU, memory and storage, and allocate them to different applications. Modern job schedulers allow complex and application-specific scheduling policies, and apply advanced optimization algorithms to achieve a variety of scheduling objectives.

The scheduler in our data center is a custom system similar to Omega [78]. It is a two-level scheduler. The low level tracks the status of resources, bundles them into abstract resource containers and provides the containers to the upper level. The upper level is

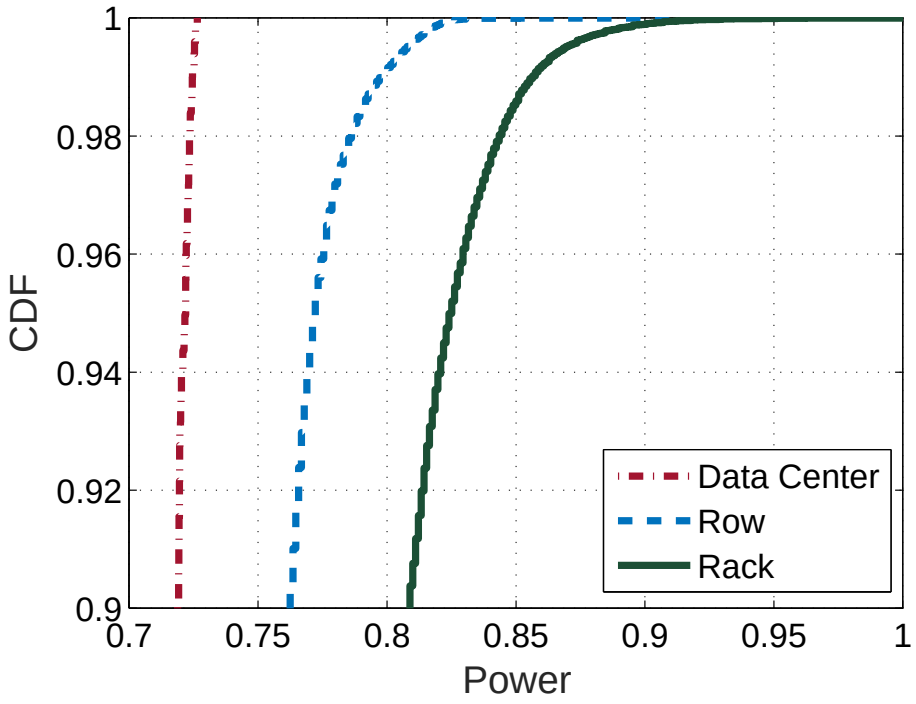


Figure 15: The CDF of the power utilization normalized to the provisioned power budget on rack, row and data center levels.

application-specific and decides how to efficiently allocate containers to jobs.

`Freeze` and `unfreeze` are two APIs provided by the lower level of the job scheduler. `Freeze` makes a server unavailable (frozen), so that the lower level can no longer add it to the candidate list. On the contrary, `unfreeze` makes a frozen server available again. In Ampere, these APIs enable us to control power indirectly by workload scheduling. We believe both APIs are simple enough to implement in any scheduler, making our approach generally applicable.

4.1.2 Characteristics of data center power utilization

We have the following important observations of data center power utilization, which directly lead to our design.

First, the average power utilization is low in the data center. Specifically, the utilization

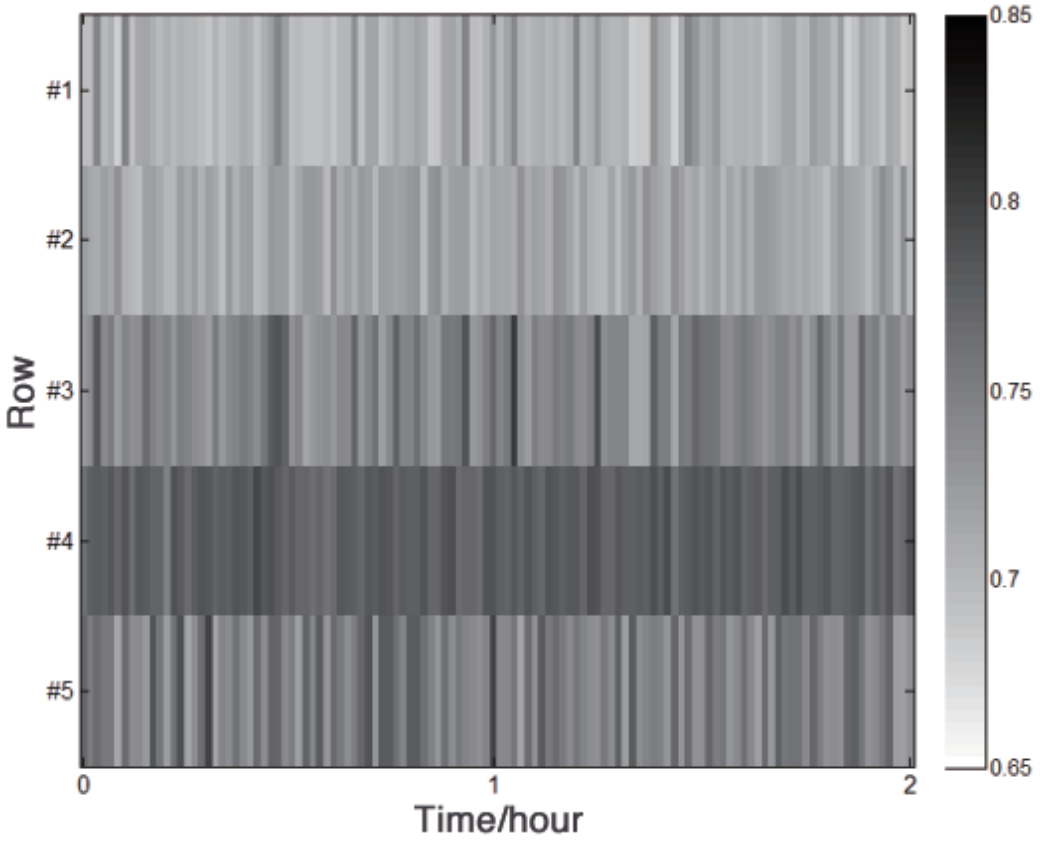


Figure 16: Row power of five randomly chosen rows during a two-hour period. The grayscale represents the power utilization. We can see both temporal and spatial variations.

is lower at a larger scale. Figure 15 shows the cumulative distribution function (CDF) of the observed power utilization in one of our production data centers for a week at rack, row and data center levels. The servers are provisioned based on the rated power. We can see that the average power utilization at the data center level is only 70%, wasting almost one third of the available power budget. We emphasize that the under-utilization is not due to lack of workload demand, as there are often jobs waiting in the scheduler queue and the company is still building out new data center facilities to meet the increasing demand.

Intuitively, like in many systems with statistical multiplexing, it would be desirable to consolidate, rather than statically partition the power at the data center level as a single

pool [22]. Our system helps to indirectly achieve the consolidation.

To show the source of the unused power, we provide a formal notation of the power at level $X \in$ {row, rack}. Assume that there are n homogenous servers at X level and the rated power of each server is P_m, and P_M is the provisioned power budget of X, then we have $P_M = nP_m$. At runtime, it is unlikely that all servers are at their rated power simultaneously, and the total runtime power of the level will generally be lower than nP_m. Thus we define the unused power $\overline{P}_t^X$ of level X at time t as

$$\overline{P}_t^X = P_M - \sum_i^n P_{it} \tag{4.1}$$

where P_{it} is the realtime power of the i-th server in level X at time t. Obviously the unused power at row level is always no lower than that at rack level, as $\overline{P}_t^{row} = \sum_{rack \in row} \overline{P}_t^{rack}$.

The second observation is that there are large variations on power utilization at the row level. The variations are both temporal (over time) and spatial (across different rows), and Figure 16 shows the variations. We see that the power draw across different rows is highly unbalanced. The reason for this imbalance is that different rows mainly focus on running different sets of products. Also the power across these rows shows weak correlations over time (80% of the correlation coefficients are under 0.33). The variations and imbalance in workload provide us with opportunities to dynamically schedule power to where it is required.

4.2 Ampere Design and Implementation

In this section, we first discuss the important design choices and an overview of the Ampere architecture. Then, we focus on the controller, introducing the variable under

control and its effects on the power, as well as the control algorithm. Finally we provide details about our controller model.

4.2.1 Design choices and rationales

We have the following four important design choices.

Managing power at the row level. We decide to manage power at the row level because (1) it matches the row-PDU physical fuse configuration in our hardware; (2) there is a larger amount of unused power at the row level than at the rack level, as discussed in Section 4.1.2; (3) there are abundant servers and tasks at row level, enabling our probabilistic control mechanism; (4) we can leverage the unbalanced power draw across different rows, and statistically direct jobs to different rows with optimal power conditions.

Minimal interface with the scheduler. To implement power-aware scheduling, one straightforward design would be making the scheduler power distribution aware. However it is not practical mainly due to the complexity of incorporating the information into different scheduling policies, especially those application-specific schedulers.

Thus, Ampere does not read any data from the scheduler and only requires the `freeze/unfreeze` interface. This enables Ampere to easily integrate with different schedulers and scheduling policies.

Power control with statistical influence on new job placement. Using the `freeze/unfreeze` API, we can affect the probability of placing new jobs to specific rows, and thus control the power usage. This has no impact on the performance of existing jobs. Furthermore, by driving away job assignments from a row, we leave the choice of where to put such jobs to

the scheduler, allowing it to take advantage of the existing policies. This is equivalent to creating a virtual pool of unused power for the scheduler.

Using simple system model and tolerating inaccuracy with control. We use data-driven predictive models to characterize the potential impact on realtime power of our control activities. Given the statistical nature of our control input, we observe high variations on the effects of the control input. Instead of demanding a very precise model as most power capping approaches do, we use RHC to periodically obtain optimized control decisions and correct the random errors in our system model.

4.2.2 Ampere architecture

Figure 17 shows the architecture of Ampere. An in-house developed power monitor collects and aggregates the power utilization at the server, rack and row level. The centralized controller implements most functionality of Ampere. For each minute, the controller reads the data from the database, computes the number of servers to freeze in the next time period, and uses the `freeze/unfreeze` interface to advise the scheduler to freeze them. The data center operator can set a control target for the maximum allowed power budget, which can be lower than the physical limit, to provide an extra safety margin. Note that the controller is stateless, and thus if the controller fails, we can easily switch to a replacement.

4.2.3 Power monitoring

We implement our own power monitor, which collects server-level power utilization, among other metrics through the intelligent platform management interface (IPMI). We leverage our in-house streaming computation framework to aggregate the data to provide

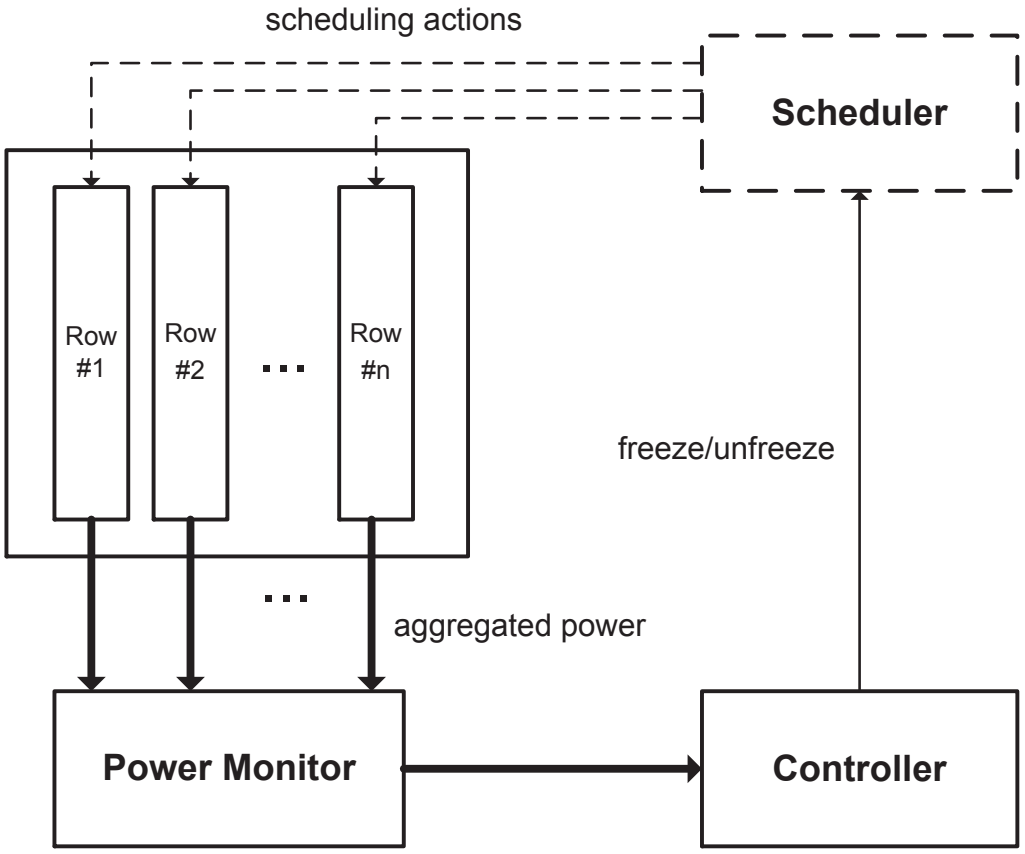


Figure 17: System architecture of Ampere.

the row-level power. We store the history data in a MySQL database and export a RESTful API for efficient query against these data. The power monitor samples the power from every server at every minute, and stores the numbers in a time series database. We rely on the time series database to provide data persistence and failover. Our power monitoring service remains stateless for easy recovery. We believe one minute is a good tradeoff between measurement accuracy and monitoring overhead.

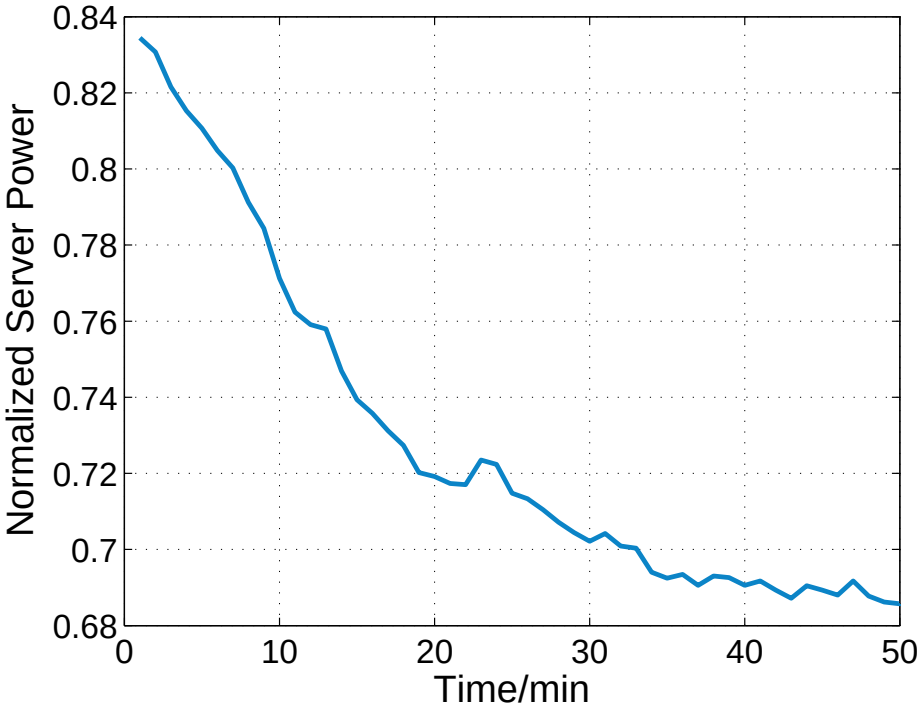


Figure 18: Power drops over time when a server is frozen. The figure shows the average power normalized to the rated power of about 80 servers at different time points after being frozen. The noises on the curve are due to the randomness of the workload on the servers.

4.2.4 Interface to the job scheduler

We use the `freeze/unfreeze` API to indirectly control job scheduling. When we freeze a server, it has two effects: (1) the power of frozen servers will go down over time, because the existing jobs will finish; (2) there will be statistically fewer jobs scheduled to the row with frozen servers, so the power increase will slow down.

To examine the first effect, we randomly select a group of about 80 servers with relatively high power utilization, freeze them for a period of time, and observe their power drop. Figure 18 displays the average power change over time. We can see the power gradually drops to the minimum (close to the idle power) after about 35 minutes.

As for the second effect, the number of jobs scheduled to a row is roughly proportional to the number of available servers of the row, assuming that there are multiple rows with different workloads. As we have discussed in Section 4.1.2, the assumption is generally true for our case. Thus, freezing a percentage of servers will likely reduce the number of *new* jobs assigned to the row, lowering the power increase during the next time period.

These two effects impact the row-level power jointly. We define the *freezing ratio* u_t as the percentage of frozen servers in the total number of servers in a row at a given time t. We want to identify a function $f(u_t)$ given a specific over-provisioning ratio r_O to quantify the effect of freezing u_t servers.

We empirically identify the impact of u_t on the row power using a controlled experiment. We will describe the detailed setup in Section 4.3.1. We denote the power of the control group and the experiment group at time t as P_t^C and P_t^E respectively. We set up the experiment so that $P_t^C = P_t^E$ without power control. In other words, the only cause of the difference between P_t^C and P_t^E is the control input u_t. With this setup, we can express $f(u_t)$ as $f(u_t) = P_{t+1}^C - P_{t+1}^E$.

In order to collect data to evaluate $f(u_t)$ using a regression model, we set u_t to a variety of different values over a period of 24 hours, and measure the power of the experiment group and the control group in the controlled experiment on a cluster with about 400 servers. Figure 19 shows the measurement result. We approximate $f(u_t)$ using a linear function $y = k_r x$ where k_r is a parameter dependent on r_O. We can use this simple model because our RHC mechanisms can help correct errors in the model over time.

We also want to point out that the linearity assumption of $f(u_t)$ helps us simplify our controller model greatly, as we will discuss in Section 4.2.6.

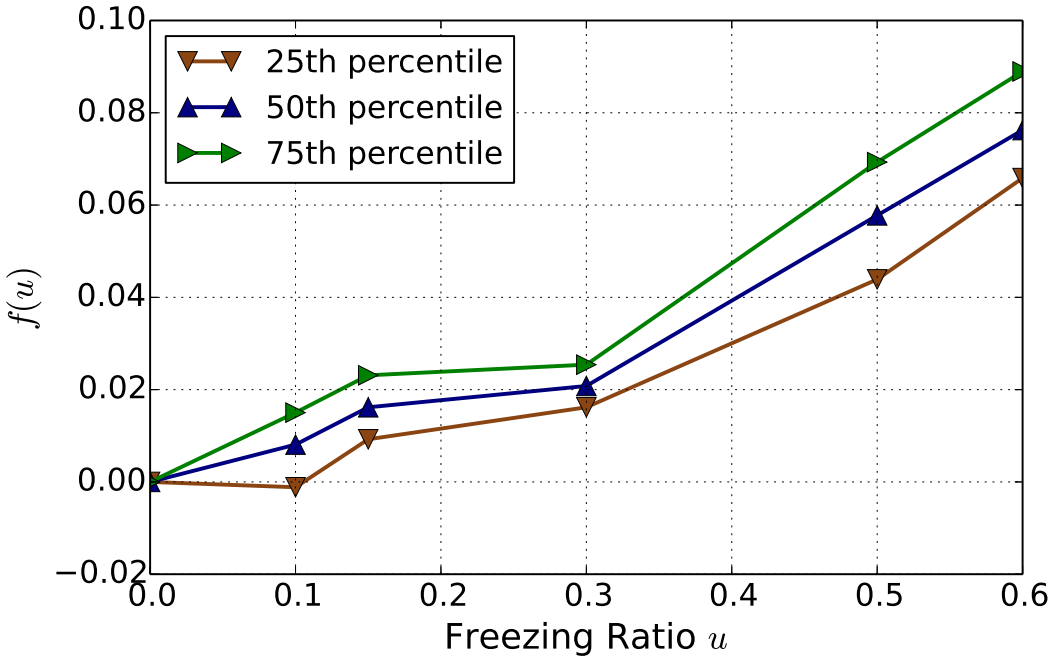


Figure 19: The effects of freezing ratio u on the power change $f(u)$. We plot the 25th, 50th and 75th percentile for $f(u)$ under different u. $f(u)$ is r_O-dependent, so we plot the normalized values.

4.2.5 Controller

With the `freeze/unfreeze` API, we implement a controller that periodically adjusts the power draw of a single row so that it stays under the power budget.

Algorithm 2 shows the control logic. At each interval (one minute in our implementation), we obtain the power utilization from the power monitor, and compute the unused power (as defined in Section 4.1.2). Then we compute how many servers of each row to `freeze`, using the control model we will discuss in Section 4.2.6. Finally, we select a set of servers to `freeze` or `unfreeze`.

We prefer to freeze servers with highest power draw mainly because servers with lower power utilization may have more computation capacity left and thus freezing them may result in a higher cost. To avoid freezing and unfreezing a server too frequently, we introduce the r_{stable} parameter to the algorithm. The algorithm will only `unfreeze` a server and `freeze` another one if the server's power drops by at least $(1 - r_{stable})$. We find that the value of r_{stable} does not affect the performance much, and we choose $r_{stable} = 0.8$ in all of our experiments.

We take a control action every minute, which is an interval matching our power monitoring frequency. Note that the controller cannot monitor or control any power fluctuation within a minute, imposing a risk of short-term power violations. This is why we still have DVFS-based hardware power capping on as a safety-net against these rare cases.

4.2.6 Computing the percentage of frozen servers

The most important decision of the controller is the number of servers to freeze. We want to freeze enough servers to avoid power violations, and in the mean time, freeze as

Input:

- P_M: Power limit
- $r_{threshold}$: Threshold ratio
- r_{stable}: Stable ratio
- P_k: Current power of row k
- p_s: Current power of server s
- $S_f[k]$: The set of frozen servers at row k
- n_k: The number of servers in row k
- $F(\cdot)$: The function from row power to freezing ratio

Output: Updated $S_f[k]$ for each row k

procedure POWER CONTROLLING
  for $k \leftarrow 1, N$ **do** ▷ N is the number of rows
    create set S ▷ candidate servers for freezing
    if $P_k/P_M > r_{threshold}$ **then**
      $n_{freeze} \leftarrow \lfloor F(P_k/P_M) \cdot n_k \rfloor$
      $S \leftarrow n_{freeze}$ servers with highest power
      $p_{threshold} \leftarrow r_{stable} \cdot \min_{s \in S} p_s$
      for all s s.t. s is in row k, $s \notin S$ **do**
        if $p(s) > p_{threshold}$ **then**
          $S.add(s)$ ▷ for stability
      for all $s \in S_f[k] - S$ **do**
        unfreeze s, update $S_f[k]$
      if $|S_f[k]| > n_{freeze}$ **then**
        unfreeze arbitrary $|S_f[k]| - n_{freeze}$ servers, update $S_f[k]$
      else if $|S_f[k]| < n_{freeze}$ **then**
        freeze $n_{freeze} - |S_f[k]|$ servers with highest power in $S - S_f[k]$, update $S_f[k]$
    else
      unfreeze all servers, $S_f[k] \leftarrow \emptyset$
  return S_f

Algorithm 2: Power controlling algorithm

Symbol	Description
U_t	The control actions obtained at time t.
u_t	The percentage of servers to be frozen at time t.
P_t	The normalized row power draw at time t.
P_M	The normalized provisioned row power budget ($= 1.0$).
E_t	The normalized power increase at time t.
$f(u_t)$	The relative reduction of power by the control u_t.
$C(U_t)$	The cost function of U_t.
k_r	The gradient of the function that fits $f(u_t)$.

Table 7: Key notations in problem formulation. All power metrics used in the problem formulation is normalized to P_M.

few as possible to minimize the negative impact on computation capacity. We discuss how we obtain the number of servers to freeze in this section. We first formulate the problem of computing the optimal number of servers to freeze in a general form of a receding horizon control (RHC) problem [43], and then use heuristics based on data-driven observations to reduce the RHC problem to a simplified version. Table 7 summarizes the key notations we use in the problem formulation.

The general model. The idea of controlling power using RHC is as follows. At each time t, we calculate an optimal control $U_t = \{u_t, u_{t+1}, \cdots, u_{N-1}\}$ on a finite fixed horizon, starting at time t, say $[t, t+N-1]$ (N is a parameter representing how much time ahead we want to predict). We only carry out the first control u_t, that is, freezing u_t servers. We repeat this computation at each time t, and the "horizon" of the control recedes as the time proceeds.

Suppose P_t is the row-level power at time t. We introduce E_t to denote the power demand increase, which is a predicted value that indicates the first order difference of row-level power. Say the predicted power for time $t+1$ is $P_{t+1}^{predict}$, then $E_t = P_{t+1}^{predict} - P_t$. The change of power is basically affected by the temporal variation of workload. Instead of implementing a predictor, we show how we use heuristic method to estimate E_t in the later part of the section.

We use the function $f(u_t)$, as we have described in Section 4.2.4, to model the effect of frozen servers on row power. Thus we have $P_{t+1} = P_t + E_t - f(u_t)$.

We use $C(U_t)$ to denote the cost function, which indicates the degradation of computing capacity or other performance metrics due to freezing servers. We use a simple linear

combination of u_t and model the cost function as

$$C(U_t) = \sum_{t \le k \le t+N-1} u_k. \tag{4.2}$$

Therefore, we formulate the Power Control Problem (**PCP**) as

$$\text{min} \qquad C(U_t) = \textstyle\sum_k u_k \tag{4.3}$$

$$\text{s.t.} \qquad P_{k+1} \le P_M \tag{4.4}$$

$$P_{k+1} = P_k + E_k - f(u_k), \tag{4.5}$$

$$0 \le u_k \le 1 \tag{4.6}$$

$$k = t, \cdots, t+N-1 \text{ for (4.3)-(4.6)}.$$

PCP is a typical RHC problem. Note that we do not need to assume $f(u_t)$ linear. Section 4.2.4 provides the method to empirically evaluate $f(u_t)$. Given a series of predicted E_k, there are many methods and tools to compute the solution of this RHC problem if a solution exists [1, 37, 43, 72].

Control model simplification. Instead of solving the general problem directly, as in our case, the function $f(u_t)$ is close to linear, we can reduce **PCP** to a much simpler problem. Using the empirically obtained $f(u_t)$ that can be approximated by a linear function $y = k_r x$, we get

$$f(u_t) = k_r u_t. \tag{4.7}$$

In this way, we replace Eq. (4.5) by

$$P_{k+1} = P_k + E_k - k_r u_t. \tag{4.8}$$

With the linear function $f(u_t)$ we define a simplified **PCP** (**SPCP**) as follows:

$$\text{min} \qquad C(u_t) = u_t \tag{4.9}$$

$$\text{s.t.} \qquad P_{t+1} \leq P_M \tag{4.10}$$

$$P_{t+1} = P_t + E_t - k_r u_t \tag{4.11}$$

$$0 \leq u_t \leq 1. \tag{4.12}$$

This is a special case of RHC problem in that the distance to horizon is 1. Assuming there is a feasible solution, the constraints and the object function are all linear so it is very easy to obtain the optimal solution, which is

$$u_t = \max\{\min\{(P_t + E_t - P_M)/k_r, 1.0\}, 0\}. \tag{4.13}$$

Empirically, $E_t - 1.0 \cdot k_r \leq 0$, which means that if all servers are frozen, the row-level power will not rise. Assuming **PCP** has feasible solutions (and thus **SPCP** also has feasible solutions), and denoting the optimal solution of **SPCP** as u'_{t_0} (Eq. (4.13)), we prove the following lemma:

Lemma 4.1. *$U'_{t_0} = \{u'_{t_0}, u'_{t_0+1}, \cdots, u'_{t_0+N-1}\}$ is the optimal solution of **PCP**, where u'_i ($i = t_0, \cdots, t_0 + N - 1$) is the optimal solution of **SPCP** in which $t = i$, $E_t = E_i$, and $P_t =$*

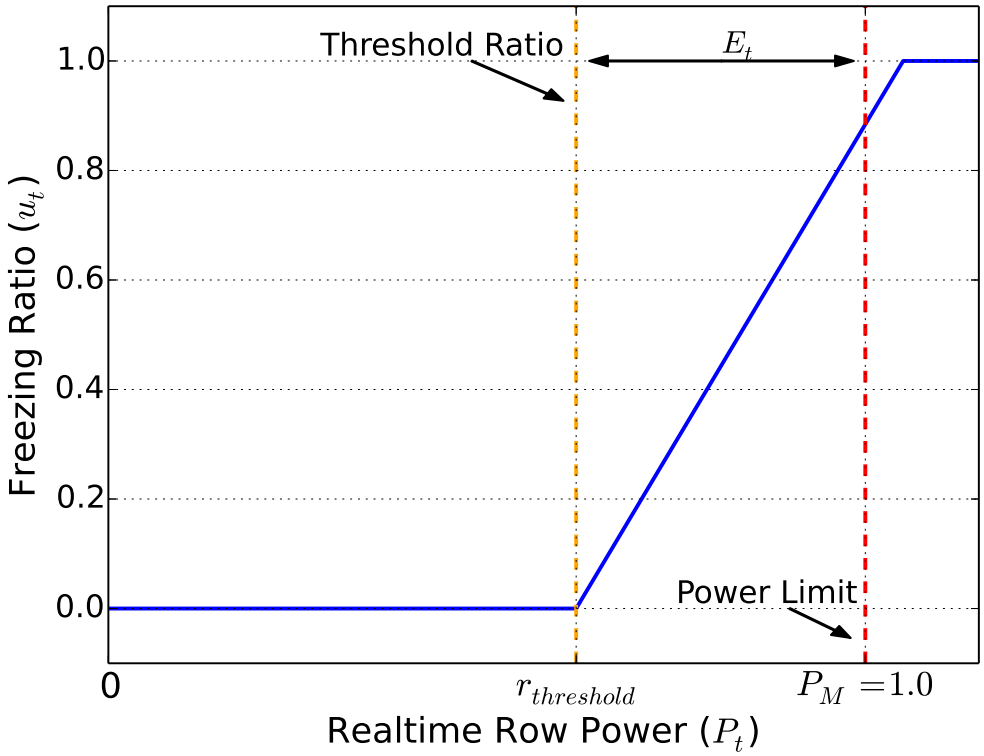


Figure 20: The control function F from P_t to u_t. The threshold $r_{threshold}$ is defined by $P_M - E_t$. Note the curve varies for different E_t and k_r.

$P_{t_0} + \sum_{k<i}(E_k - k_r u'_i)$.

Proof. Without the loss of generality, we assume that $E_k \geq 0$ ($k = t, \cdots, t+N-1$) and $P_t + \sum_k E_k > P_M$. We can easily verify U'_{t_0} is a feasible solution to **PCP**. Assuming that $U^*_{t_0} = \{u^*_{t_0}, \cdots, u^*_{t_0+N-1}\}$ is an optimal solution to **PCP**, by applying Eq. (4.8) iteratively for all k, we get

$$P_{t_0} + \sum_k E_k - k_r \sum_k u^*_k = P_{t_0+N} \leq P_M. \tag{4.14}$$

Therefore

$$C(U^*_{t_0}) = \sum_k u^*_k \geq (P_{t_0} + \sum_k E_k - P_M)/k_r \geq C(U'_{t_0}). \tag{4.15}$$

Thus, U'_{t_0} is the optimal solution of **PCP**. □

This reduction greatly simplifies the problem: we only need to optimize the freezing ratio for a horizon at a distance of 1 at each time t.

In the optimal control strategy, the predicted power demand E_t defines a safety margin $[1.0 - E_t, 1.0]$. That is, if the power P_t is below a threshold $r_{threshold} = 1.0 - E_t$, we do not need any control as there is unlikely a imminent power violation. However, if $P_t > r_{threshold}$, the closer the current power is to the power limit, the more servers we will freeze. We call $r_{threshold}$ the *threshold ratio*. Figure 20 shows the intuition and relationship among $r_{threshold}$, P_M and E_t.

Estimate the power change E_t. In order to avoid power violation due to a sudden power surge, we need to leave a safety margin. The estimated E_t determines the margin, as it

indicates the expected power increment during the next minute. We use a data-driven approach to estimate E_t. We would like to keep E_t small to improve the power utilization.

We monitor the power of all rows in our data center for a long time, and collect the power increase for every minute. We discover that the distribution of power increase varies for different hours in a day, so we calculate the 99.5-percentile power increase for each hour and use the one matching the hour of t to estimate E_t. By experiments we find that Ampere's performance is not sensitive to E_t. Nevertheless, our E_t estimation is conservative as we are preparing for almost the largest change in observed history (99.5th percentile). We can use a better online power prediction model to get a better estimation, which we leave for future work.

4.3 Evaluation

In this section, we present the evaluation results from a production data center. We first introduce the experiment setup and characterize the workload. Then we show the effectiveness of Ampere and its advantage over existing power capping methods. Finally we evaluate the effects of various parameter choices in Ampere.

4.3.1 Experiment setup

In this section, we briefly introduce the cluster setup, the production workload properties and our controlled experiment setup that allows us to perform experiments on a production cluster.

Cluster setup and workload We have implemented Ampere in one of our production data centers. Here we present results on a single row with 400+ homogeneous servers. All servers in this row are part of a datacenter-wide resource pool managed by a single job

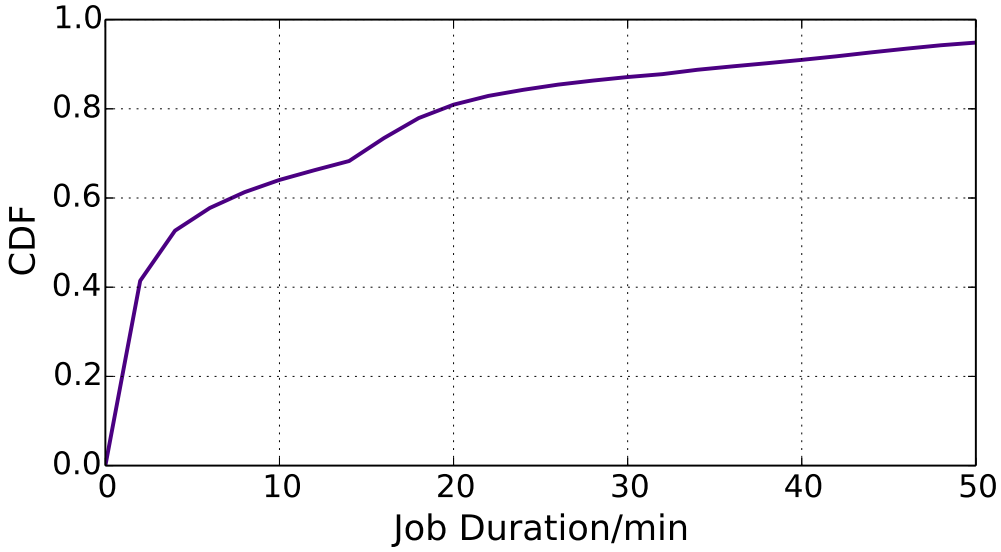


Figure 21: The CDF of batch job durations in the production cluster.

scheduler as described in Section 4.1.1. We only focus on the power draw from servers, as other devices like network switches consume only a negligible fraction of the total power.

All servers run production workload comprised of mainly batch jobs (e.g., Map-reduce tasks). We add interactive jobs to the cluster to evaluate the effectiveness of Ampere, and we show the results in Section 4.3.3. The durations of these batch jobs vary and Figure 21 shows the CDF of the job durations. The average job duration is about 9 minutes, and about 40% jobs finish in 2 minutes. The arrival rate of jobs in the cluster also varies a lot over time, and usually the rate is 400-600 jobs per minute. The variations of job durations and arrival rate make our probabilistic control more effective, as there is a good chance that some job will finish on some frozen machine, reducing the power utilization.

As we describe in Section 4.1.2, the row level power that is directly affected by the workload on the row, also varies significantly over time. Figure 22 shows the power utilization in a twenty-four hour period with a reading every minute. There are two observa-

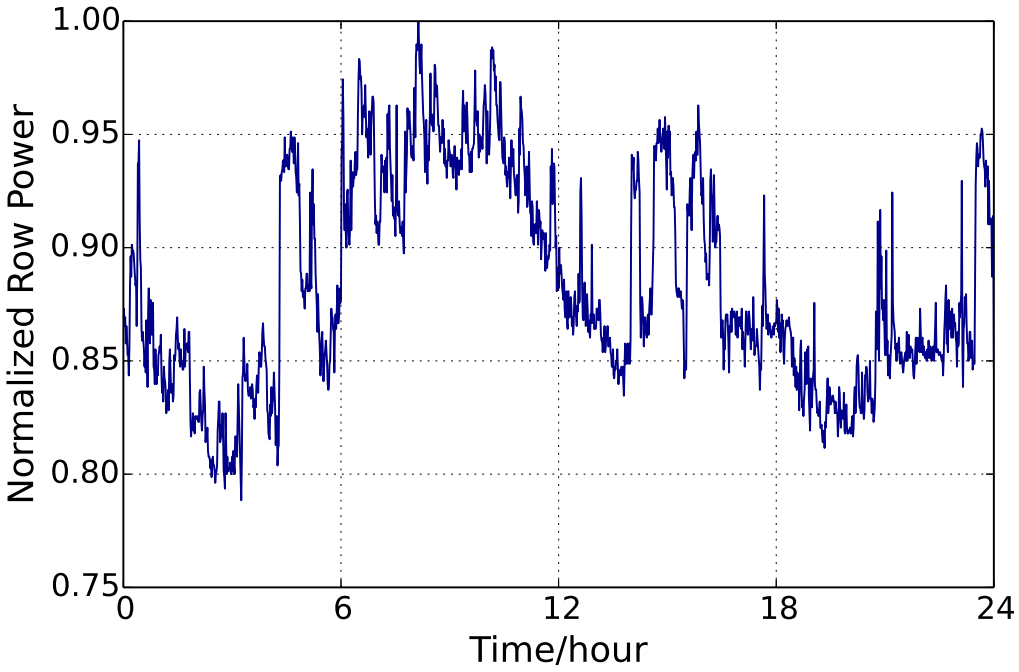


Figure 22: The power of a row in 24 hours, normalized to the maximum power. The values on the X-axis do not correspond to actual wall clock.

tions:

1) At a larger time scale (hours), we observer high variations. This larger-scale variations leave us with room at many time periods to over-provision servers and keep the power below the daily peak.

2) At a smaller time scale (a few minutes), we can see many spikes and valleys on power utilization. It is hard to predict these spikes. To better characterize these spikes, we plot the CDF of the first order differences of the power (the power changes at 1-minute scale) in Figure 23. We can see that within a single minute, the power change is generally small (smaller than +-2.5% for 99% of the time), but it can be a change as large as 10%. We design the approach described in Section 4.2.6 to handle these relatively large spikes. Section 4.3.2 shows the results.

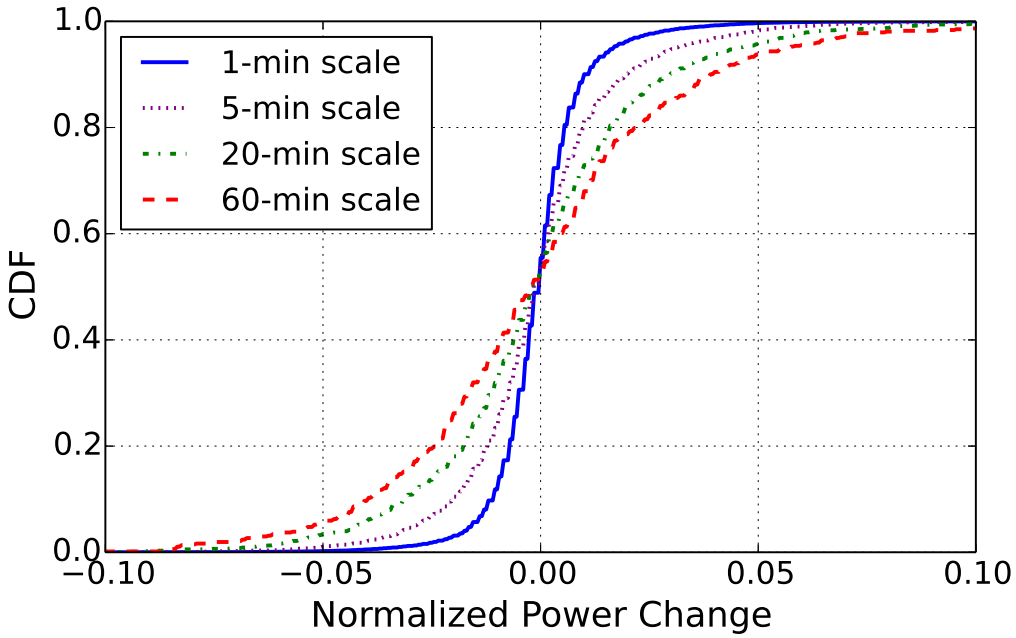


Figure 23: The CDF of the power changes of the control group on various time scales. For the k-minute scale, we compute a sequence of the maximum power for every k minutes, and then plot the CDF of the first order differences of the power sequence. The power changes are normalized to the provisioned power budget.

Considering some operational maintenance issues of the scheduler, we limit the maximum ratio of freezing servers to 50%. This limitation causes a few power violations, as we will show later in the section.

Controlled experiment design As we cannot isolate a large number of servers to conduct trace-based experiments to compare the system performance with or without Ampere, we perform a controlled experiment. We partition the servers into two virtual groups, the *experiment group* and the *control group*. Specifically, we partition them based on the parity of the server IDs and thus a server is assigned to a group in a uniformly random way. Both groups accept jobs from the same scheduler and thus statistically they should have similar workload. We verify the fact by calculating the average power and the correlation coefficients of power of the experiment and control groups (with Ampere turned off) over five days. The difference between the average power is less than 0.46%, and the correlation coefficient of the power is 0.946. Thus, we can safely assume that any differences between these two groups are results of the control actions from Ampere.

We turn off the power capping so we can observe the real power demand. In order not to cause real-world power violation, we emulate the power violation events by scaling down the power budget of these servers. Consider we set the power budget to P'_M instead of the actual P_M, with N_r servers per rack, we can emulate the case where in each rack, $\lfloor P'_M/P_m \rfloor$ of the servers are designed to be provisioned, and the other $N_r - \lfloor P'_M/P_m \rfloor$ servers are over-provisioned. Thus we can calculate the over-provisioning ratio r_O as

$$r_O = N_r/\lfloor P'_M/P_m \rfloor - 1 = P_M/P'_M - 1. \tag{4.16}$$

We use the scaling-based emulation differently in our evaluations. In Section 4.3.2, we scale down the power budgets of both groups to compare the power of two over-provisioned groups and show the effectiveness of our control. In Section 4.3.4, we only scale down the power budget of the experiment group so that we can observe the impact on throughput. We emphasize that we only use the scale-based emulation to provide more insights into how Ampere works, and it is not part of the production system.

We set the over-provision ratio to 0.25, a very high value, in most of our experiments to demonstrate the effectiveness of Ampere under extreme conditions. As we will show later in Section 4.3.4, we choose $r = 0.17$ as the optimal value for real production.

Key performance metrics We focus on the following three key performance metrics:

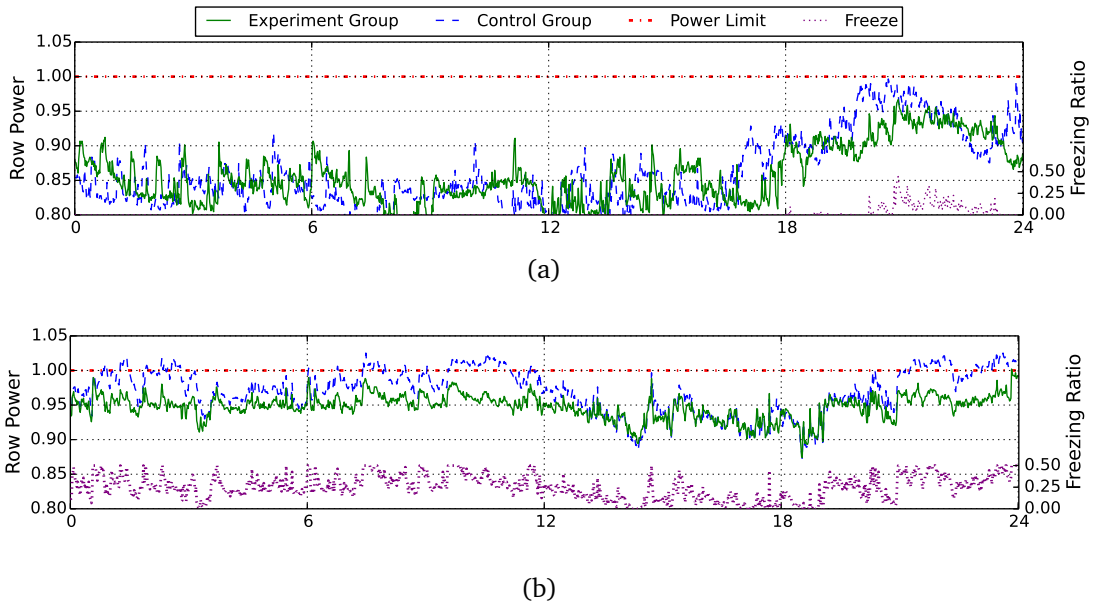
1) The number of power violations. Given the power capping mechanism , the user may choose to allow a few power violations, and small violation number shows the effectiveness of Ampere;

2) The ratio of frozen servers (u_t). Obviously, a smaller frozen ratio can help minimize the impact on the overall performance of the cluster;

3) The gain in throughput per provisioned watt (TPW). We define TPW as:

$$\text{TPW} = \frac{\text{Total throughput during a time interval } T}{P_M \cdot T} \tag{4.17}$$

where P_M is the total provisioned power budget, and the throughput is the number of jobs accepted during the time period T. We simply choose the job count as the throughput indicator because with large number of jobs, each job has similar average resource



(a)

(b)

Figure 24: The freezing ratio u_t and its effects on the power utilization, on both light (a) and heavy (b) workload over 24 hours. The control group does not have power control and thus the power differences between the control group and experiment group are approximately the effects of the power control.

requirements.

The *Gain in TPW*, G_{TPW}, is the increase of TPW by over-provisioning. We use this metric to evaluate the balance between the computation capacity gain from adding more servers, and the capacity loss from freezing some servers. We discuss the evaluation of G_{TPW} in Section 4.3.4.

4.3.2 The effectiveness of Ampere's control

We first provide evaluation results using two extreme types of workload in our cluster - heavy and light. We fix the over-provision ratio at 0.25 for both experiments in this section.

Table 8 shows a few performance metrics of Ampere. We observe a smaller maximum power draw from the experiment group. Meanwhile, under the heavy workload, in the control group without any power control, we observe 321 power violations, while we

Workload	Light		Heavy	
Group	Exp	Ctr	Exp	Ctr
u_{mean}	1.5%	0%	24.7%	0%
u_{max}	44.1%	0%	50.0%	0%
P_{mean}	0.857	0.860	0.948	0.970
P_{max}	0.967	0.997	1.002	1.025
$Violations$	0	0	1	321

Table 8: Controller effectiveness under light / heavy workload. The experiment runs for 24 hours and the measurements are taken every minute. u_{mean} and u_{max} are the mean/max freezing ratio. P_{mean} and P_{max} are the mean/max power draw. $Violations$ is the total number of power violations.

observe only one violation using Ampere's control, and this violation is due to the 50% freezing ratio limitation. These observations have proved the effectiveness of Ampere's power control ability.

Taking a closer look at the control actions with different workload, Figure 24 plots the power draw and control actions over a period of 24 hours under heavy and light workload. Figure 24(a) shows the light workload case, i.e., the power draw mostly under the power limit. In this case, we only take control actions occasionally, and thus cause little impact on the overall power or throughput. In contrast, Figure 24(b) shows the heavy workload situation, during which the power draw would exceed the power budget quite often without control. We can see from the figure (purple/dotted lines) that Ampere freezes a significant number of servers at many time periods and successfully avoids power violations. Table 8 shows the statistics for a 24 hour period for light workload and heavy workload.

Note that because we limit the maximum freezing ratio of servers to 50%, we get many saturation on the control input in Figure 24(b). This limitation effectively reduces how much we can react to a power surge and thus makes it more vulnerable to power violations. We will try to remove this scheduler limitation in our future work.

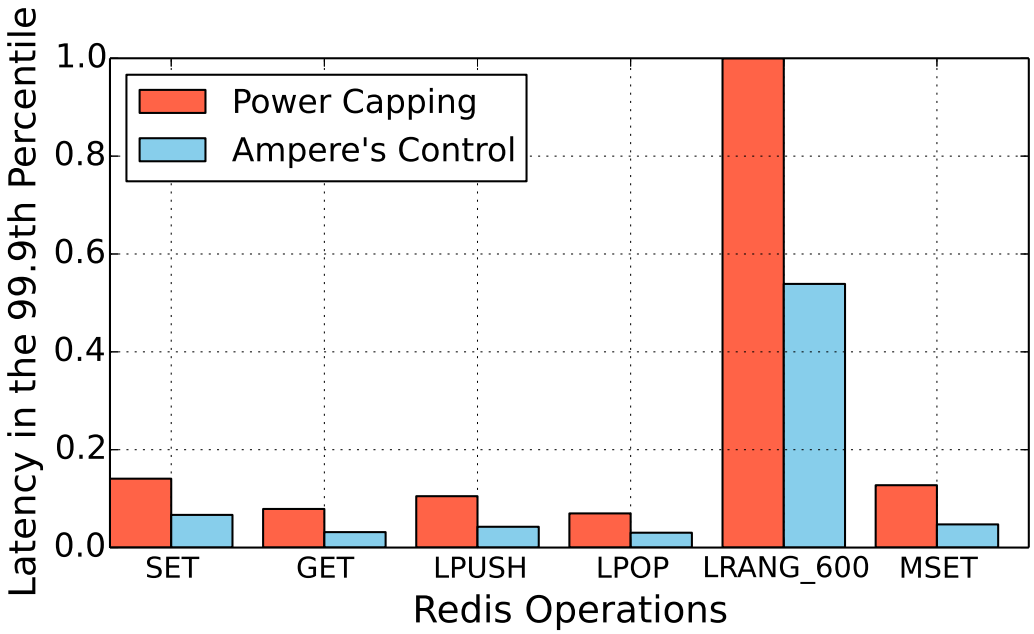


Figure 25: The 99.9th percentile normalized latency of various operations in the Redis-benchmark, using either power capping or Ampere as power controller.

4.3.3 Advantage over power capping approach

Power capping, as we have already mentioned, may cause performance disturbances. In this section, we compare the performance disturbances, and demonstrate that Ampere has big advantage in this aspect.

We deploy a Redis [74] cluster on a row with over-provision ratio r_O set to 25%. We repeatedly run a Redis-benchmark [75] on a number of clients located in another cluster that does not have any power control. We compare the performance of the Redis cluster under power capping and under Ampere, respectively. We report the 99.9th percentile latency observed on the client side. Figure 25 shows the results.

We can see power capping reduces the performance of the Redis, almost doubling the 99.9th percentile latency in almost all benchmarks. This is because Redis servers are CPU-

bound, and reducing CPU frequency on a busy server slows down request processing, causing significant queuing effects that lead to longer latency. In comparison, with the control of Ampere, we rarely trigger power capping, and `freeze/unfreeze` operations do not affect existing jobs. Thus Ampere results in a much smaller latency for interactive applications like Redis.

Without the control of Ampere, power capping is very common in the cluster. To quantify this fact, we collect 8640 power utilization records (one per minute) on all servers over several days. Among these records, 1306 are over power budget. For each of these 1306 minutes, we check each individual server to see if it is power capped. Our data shows that on average, 54.34% servers are power capped for roughly 15% of the total time, which is quite unacceptable for latency-sensitive jobs. In comparison, Ampere has no impact on running jobs, and thus is applicable to both interactive and batch jobs.

4.3.4 Factors that affect the TPW

The goal of our work is to increase the computation capacity given a fixed power budget, and thus TPW is an important metric. Two parameters affect TPW, the over-provision ratio r_O and the throughput ratio r_T. The throughput ratio r_T is in turn affected by the workload. In this section, we provide some quantitative analysis on the r_O choice and its effect on TPW under different workload.

Over provision, workload and the gain in TPW. The increase on TPW is obvious from the number of extra servers provisioned. To evaluate the throughput loss due to control, we compare the number of jobs accepted in the experiment group $thru^E$ and the control group (with the same number of servers, but with Ampere turned off) $thru^C$ during the

same time period. We define the throughput ratio r_T as $thru^E/thru^C$. Generally $r_T \leq 1.0$ as freezing servers reduces the throughput.

Given r_T and the over-provision ratio r_O, we estimate G_{TPW} by

$$G_{TPW} = \frac{TPW^E}{TPW^C} - 1 = \frac{thru^E/(P'_M * t)}{thru^C/(P_M * t)} - 1$$
$$= r_T \cdot (1 + r_O) - 1. \tag{4.18}$$

For example, if the over-provision ratio is 0.25, i.e. we add 25% more power budget (and thus 25% more servers) to the row. With sufficient power budget, we should get an increase of capacity by 25% and the throughput should increase by 25% with enough workload. The power control of Ampere reduces the throughput. We measure the loss by comparing the experiment group throughput with the control group. For example, if we observe a 10% decrease in throughput in the experiment group, then the overall TPW gain is $G_{TPW} = (1 - 0.1) * (1 + 0.25) - 1 = 0.125$.

Note that G_{TPW} is workload dependent. Under a light workload, adding servers just cause more servers to stay idle without any positive effects. With a fully utilized cluster already taking the entire power budget, we cannot run new jobs even with more servers. However, as we show in Section 4.1.2, when the workload shows high variation, we have plenty of opportunities to get a good G_{TPW}.

An intuitive example. We illustrate that TPW does not increase monotonically with the over-provisioning ratio r_O.

Figure 26 shows an example of how r_O affects TPW. During this experiment, we set

$r_O = 0.25$ and run the experiment for four hours. The boxed area on the left shows the time period when power utilization is high. Comparing to the control group, we observe a throughput decrease by about 20% in the experiment group, as expected. Thus we have $r_T = 0.8$ and $G_{TPW} = (1 + 0.25) \times 0.8 - 1$, which is close to zero. Intuitively, as we are already using all the power without over-provisioning, adding more servers do not bring in extra capacity for jobs due to the power limit. It is even worse that the extra servers consume idle power, eventually hurting the overall throughput. Thus, we need to avoid this situation in production.

If we choose a smaller $r_O = 0.17$, under the same heavy workload in Figure 26, we are under the power budget most of the time, and thus r_T is close to 1.0, making TPW gain close to the over-provision ratio $G_{TPW} = (1 + 0.17) \times 1.0 - 1 = 0.17$, indicating that we can fully utilize the over-provisioned resource even at a high workload.

Of course, workload varies over time. During the four hour period shown in Figure 26, the average r_T is 0.95, a higher number than 0.8. Therefore, we can estimate that when $r_O = 0.25$, we can get a gain in TPW, $G_{TPW} = (1 + 0.25) * 0.95/1.0 - 1 = 0.19$, a better number than the case with high workload.

From the example, we see that both r_O and the average workload have a high impact on the gain in TPW.

Evaluation on different over-provision ratio and workload. Over an experiment period of 20 days, we run Ampere using different over-provisioning ratio under varying production workload. Table 9 captures representative results from 13 days. Note that the workload is from real production that we have no control over, and it is the reason why

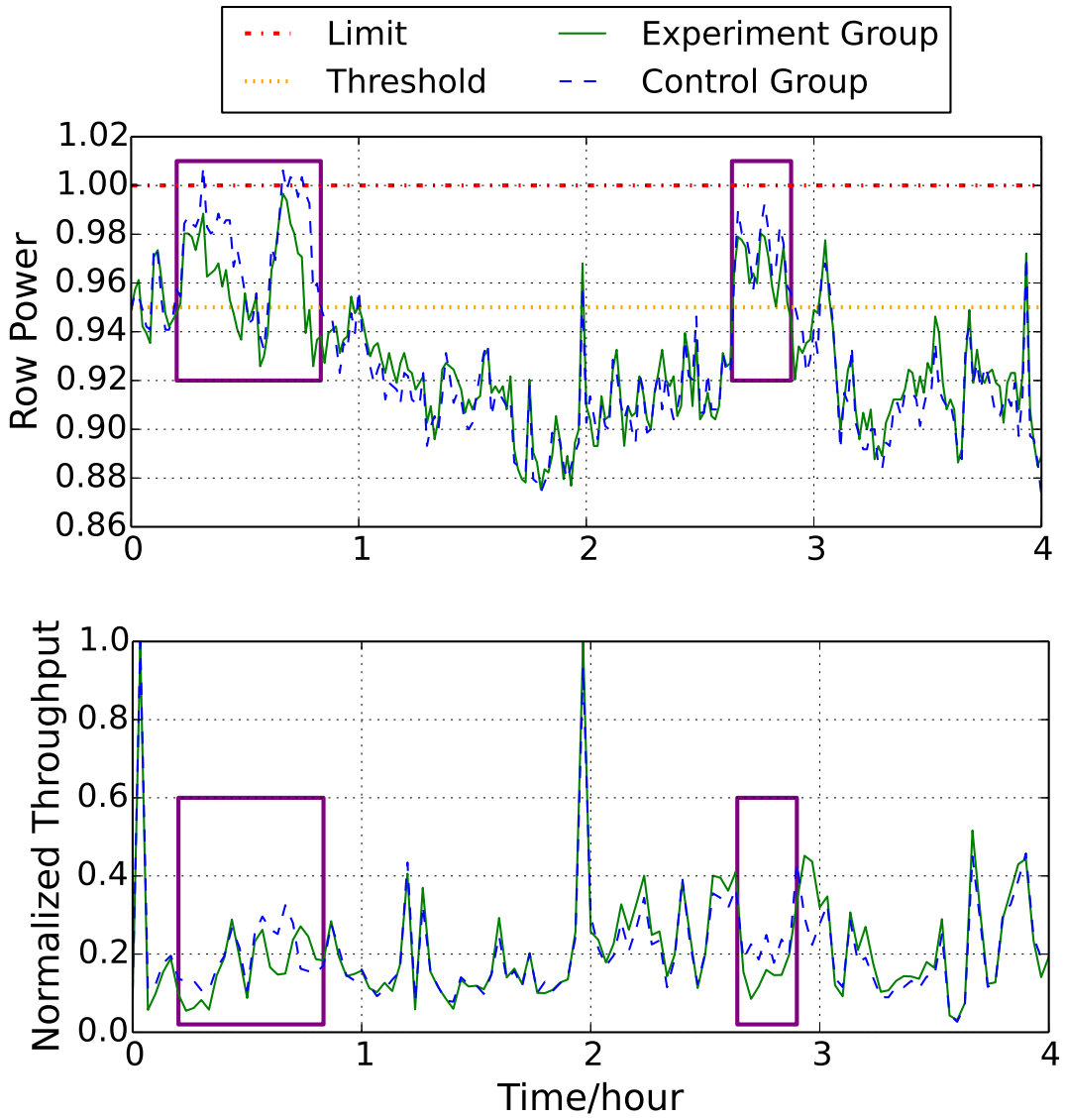


Figure 26: The effect of Ampere on power and throughput. The box highlights the effect: the control actions effectively reduces both the power and the throughput of the experiment row. Ampere only applies the control action when the power is above the threshold, leaving other regions unaffected.

we show different workload in different cases. We have the following two observations.

First, with a given r_O, G_{TPW} is directly affected by the control input u_{mean}. It is intuitive that the more servers we freeze on average, the smaller the capacity is, and so is the G_{TPW}. A deeper analysis of Table 9 shows that u_{mean} is largely affected by the average power demand P_{mean} [2]. With similar P_{mean}, occasional spikes on P trigger large u_t at certain times, also affecting G_{TPW}. For example, $\#4$ shows worse G_{TPW} mainly due to the high maximum workload.

Second, the over-provisioning ratio r_O also has a large impact on G_{TPW}. For example, $\#4$ and $\#7$ have very close P_{mean} (scaled by r_O), but r_T and G_{TPW} are both better with a smaller r_O. It is due to the same reason as we have discussed earlier: $r_O = 0.25$ is too high an over-provision ratio, causing u_t to be high quite often. As another extreme, $r_O = 0.13$ is too low, because G_{TPW} is upper bounded by r_O, the gain is only 0.13, making it a less attractive choice.

Choosing the optimal r_O. Given the observation above, we want to choose a moderate r_O. There are two metrics to consider: (1) We want to choose a r_O so that we can maximize G_{TPW} under the *typical* workload; (2) Choosing r_O is also a tradeoff between safety (fewer power violations) and performance (higher TPW).

In our experiments in Table 9, we find that 0.17 is a safe and effective choice. From our observation over a month, the 85th and the 95th percentile power is 0.909 and 0.924 (scaled to match r_O), which means most of the time G_{TPW} will be at least 15%. Thus both

[2] We estimate the power demand using the control group power, which is unaffected by the control. To show the effect of over-provisioning, we normalize the power of the control group to the scaled power budget of the experiment group. That's why P_{max} may exceed 1.0 in some cases.

#	r_O	P_{mean}	P_{max}	u_{mean}	r_{thru}	G_{TPW}
1	0.25	0.903	1.028	0.019	0.953	19.70%
2		0.931	1.062	0.134	0.941	17.60%
3		**0.936**	**1.062**	**0.152**	**0.885**	**10.60%**
4		0.927	1.061	0.196	0.835	4.30%
5	0.21	0.786	0.913	0	1.0	20.70%
6		0.835	0.982	0.0016	1.0	20.70%
7		0.894	1.000	0.009	0.979	18.20%
8		**0.903**	**1.036**	**0.11**	**0.88**	**6.20%**
9	0.17	0.836	0.931	0	1.0	17%
10		0.839	0.926	0	1.0	17%
11		**0.908**	**0.992**	**0.07**	**0.984**	**14.90%**
12		0.938	1.004	0.12	0.904	5.50%
13	0.13	0.847	0.969	0	1.0	13%

Table 9: G_{TPW} under different over-provision ratio r_O and workload condition. P_{mean} and P_{max} are the mean and max power of the control group, respectively, which are good indicators of the power demand. u_{mean} is the average freezing ratio. Bold rows represent results under typical workload.

G_{TPW} and over-provisioning efficiency are relatively higher compared to other r_O choices. In conclusion, we choose 0.17 as our over-provisioning ratio considering safety, G_{TPW} and efficiency.

4.4 Summary

In this chapter, we introduced Ampere, to to improve TPW of data center by provisioning extra servers and statistical power control. It is mainly designed for offline workloads. Combining online and offline services on the same set of servers can further increase the power utilization but supporting such heterogeneous workloads also introduce more challenges, which we will discuss in the next chapter.

CHAPTER 5 PELICAN: POWER SCHEDULING FOR QOS IN LARGE-SCALE DATA CENTERS WITH HETEROGENEOUS WORKLOADS

To address the above limitations and challenges in the previous chapter, we investigate the behavior of power and task in data centers and present Pelican, a new power scheduling system for large-scale data centers with heterogeneous workloads. Instead of moving tasks on spatial dimension, we tried to move tasks on temporal dimension. Based on the current power of a rack, we will find an optimized power budget for each server and by limiting resources for tasks.

5.1 Review and Observation

As shown in [97], low average power utilization, especially at a larger scale, in the data center along with conservative server provisioning is one of the main opportunities for over-provisioning. Here, we are going to provide background information as well as review our data center power architecture, provisioning, which lead to our design.

5.1.1 Rack power

A server does not have hard power budget as long as rack power distribution unit(PDU) can supply enough power but maximum power of each model of server is measured for provisioning. Following the definition in [97], we called it the *rated power*, or measured maximum power draw from equipment. Rack level-power, however, is limited by both physical limits and limits from row level supply. If the power of a rack exceeds its power budget, we call it a *power violation*.

The overall power utilization of data centers in Baidu, Inc. is about 70% to 80%. In order to improve power utilization, some data centers have provisioned more server. To ensure safety, the data center operator chooses a power limit that is lower than physical

power limit. And a rack level DVFS is deployed to those data centers to enforce the power limit. We monitored the power of a rack for 31 days and found out that the rack power exceeded rated power for 2% of the time. The maximum normalized power can reach 1.06. As a result, we need a better power management solution to improve QoS for over-provisioned racks with heterogeneous workloads.

5.1.2 Power controlling

Without modifying existing software and hardware on a server, there are two major ways to control power. One way is cooperating with job scheduler to change the placement of jobs so as to move the power in the spatial dimension. The other way is to control the computing resources assigned to tasks so as to move the power in the temporal dimension. The power variations in both temporal (over time) and spatial (across different racks) makes it possible to do either choice. In this paper, we choose to use the second way. One of the reason is that our services are location sensitive so that power controlling should not influence job placement. The most difficult part for the second way is how we can reduce performance impact. If we can somehow "move" part of power usage of long run and resource hungry tasks during busy time to idle time, then we can make less performance impact on short and latency sensitive tasks. To do so, there are two questions. The first question is how to select proper tasks. In a heterogeneous workloads environment, offline workloads are usually much longer than online workloads and also consumes more power. Short tasks finished quickly so that we have no chance to do such operation. On the other hand, online workloads are more latency sensitive while offline workloads are more focus on throughput. As a result, offline workloads are preferred. Within offline tasks, we can choose lower priority instead of estimating the length of execution, which is much easier

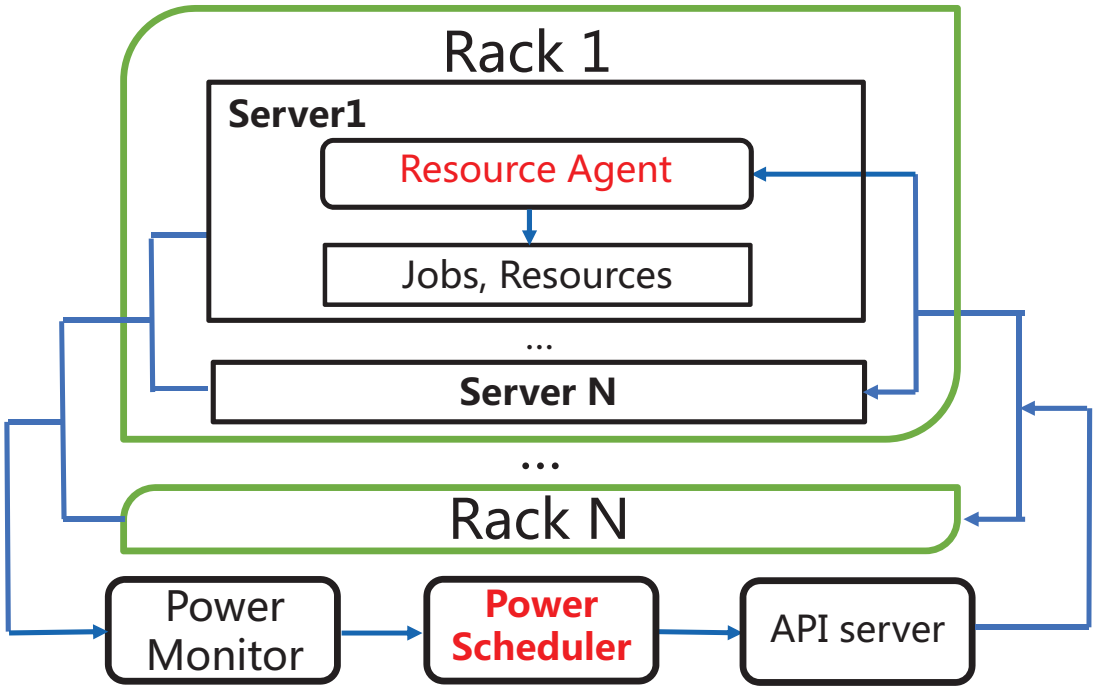


Figure 27: System Architecture of Pelican.

to obtain. In addition, the priority of offline workload is lower than online workload so that priority could be a good indicator to choose tasks. The second question is how can we "move" the power. Generally, we can achieve it by controlling computing resources assigned to a task. To make the controller applicable to all servers in our data center, as well as keep our controller simple, we decide to use core binding to control the power of a specific task. Because core management is a common part when handling heterogeneous workloads, for example, some cores are reserved only for online workloads. Core binding can also react in a short time period and it is widely supported both by OS and hardware in all servers in our data centers.

5.2 Pelican Design and Implementation

5.2.1 Architecture

The overall architecture of Pelican is shown in Figure 27. A centralized power monitor collects and aggregates the power utilization at the server, rack level. The power scheduler implements most functionality of Pelican, which is responsible to manage one or multiple racks. At the beginning of each time interval(represented by T), the scheduler reads power data from our centralized power monitor, computes the power budget for each server if necessary, and uses the budget API to deliver power budget to resource agent on each server. The resource agent then limit power usage by process management and resource management. The data center operator may choose a control target for the maximum allowed power budget, which can be lower than the physical limit, to provide an extra safety margin.

5.2.2 Power monitoring

We implement our own power monitor, which collects server-level power utilization, among other metrics through the intelligent platform management interface (IPMI). The power monitor samples the power from each server every time period. Our power monitoring service remains stateless for easy recovery. In our experiment, the monitoring cycle is ten seconds, which we believe is a good trade-off between measurement accuracy and monitoring overhead.

5.2.3 Resource agent and budget API

We use the budget API to indirectly control the resource assigned to tasks running on servers and leave power controlling policy to resource agents on servers so that different

cluster could have various policies depending on workload and type of service. Since local server knows exact types and priorities of tasks running on it, it is appropriate to do local resource management instead of managing by power scheduler.

A centralized API server is implemented to act as an adapter to deliver power budget commands to a specific server. In this way, the scheduler does not need to locate and communicate with individual servers in the data center.

5.2.4 Scheduler

With API server, we implement a scheduler that periodically adjusts the power budget of servers so that the total power of a rack stays under the target power specified by data center operators. A scheduler can manage several racks at the same time. The decision process is independent and identical based on different inputs from racks. So here we discuss the scheduling method for one rack.

Algorithm 3 shows the overall schedule logic. Data center operator first decides a fixed power budget B_r for a rack so that our goal is to limit overall rack power less than it. Then a power threshold P_t is determined based on the change of power of a rack within a time period (See details in Section 5.2.5). At each interval (ten seconds in our implementation), the scheduler obtains power utilization information from the power monitor. If current rack power exceeds power threshold, we will compute the power budget for each server in a rack using the schedule model, which we will discuss in Section 5.2.5. The goal here is to keep rack power under power threshold. However, even if the rack power is lower than the power threshold, we should not stop controlling immediately because the observed rack power is a result of working resource agent. On the other hand, rack power in the next interval may exceed the power threshold again causing resource agent to be on and

off frequently. That's why a safe power threshold P_s is introduced to determine if the rack power is safe enough. P_s should be less than P_t and is also determined based on the change of power but we will tune this value in Section 5.3.5. Only if current rack power is less than safe power threshold, the scheduler will notify resource agent to stop resource limit. Finally, we send power budget command to API server to apply on each server. To maximize the power utilization, P_t and P_s should be as close to B_r as possible and they are both related to future rack power. Instead of implementing a predictor, we show how we use a heuristic method to estimate P_t and P_s in the later part of this section.

The scheduling cycle matches the power monitoring cycle. Note that the scheduler cannot monitor or control any power fluctuation within a time interval, imposing a risk of short-term power violations. This is why we still have DVFS-based hardware power capping on as a safety-net against these rare cases.

The scheduler is designed to be stateless, as a result, we can easily switch to a replica if any scheduler fails.

5.2.5 Computing the power budget for servers

In order to avoid power violation due to a sudden power surge, we need to leave a safety margin. P_t and P_s describe the margin, as it determines when to turn resource agent on and off. The change of power is basically affected by the temporal variation of workload. We use a data-driven approach to estimate them. We would like to keep P_t and P_s close to rack power budget.

We monitor the power of all racks in our data center for a long time and collect the power increase for every minute. We discover that the distribution of power increase varies for different hours in a day, so we calculate the 99.5-percentile power increase.

Input:

- P_r^k: Current power of rack k
- P_t^k: Threshold of rack k
- P_s^k: Safe rack power of rack k
- N: The number of racks
- P_i^k: Current power of server i in rack k
- D_i^k: Dynamic power of server i in rack k
- n_k: The number of servers in rack k

procedure POWER SCHEDULING
  for $k \leftarrow 1, N$ **do**
    if $P_r^k > P_t^k$ **then**
      $P_{rest} \leftarrow P_r^k - P_t^k$
      Sort servers by D_i^k in decreasing order
      for $i \leftarrow 1, n_k$ **do**
        if $P_{rest} \geq D_i^k \cdot R_{max}$ **then**
          $B_i^k = P_i^k - D_i^k \cdot R_{max}$
        else if $P_{rest} > 0$ **then**
          $B_i^k = P_i^k - P_{rest}$
        else
          $B_i^k = P_i^k$
        Send B_i^k to budget API
    else if $P_r^k < P_s^k$ **then**
      Send stop command to budget API
  return

Algorithm 3: Power scheduling algorithm

Our estimation is conservative as we are preparing for almost the largest change in observed history. We can use a better online power prediction model to get a better estimation, which we leave for future work.

To determine the budget for each server at the begin of a time interval is the most important task for the scheduler. We want to limit enough power usage to avoid power violations, and in the meantime, reduce overall negative performance impact on a rack.

Symbol	Description
P_r	The current overall rack power.
P_t	The rack power threshold.
P_i	The power of i-th server in the rack.
I_i	The idle power of i-th server.
D_i	The dynamic power of i-th server.
Δ_i	The expected reduced power by resource limitation.
R_i	The expected power reduce ratio.
R_{max}	The maximum allowed power reduce ratio.
B_i	The result power budget for i-th server.
P'_r	The expected rack power in next time interval.

Table 10: Key notations in problem formulation. All power metrics used in the problem formulation is normalized to P_M.

Suppose P_i is the current power of i-th server in the rack and I_i is the idle power of the server. Then an effective power budget B_i should belong to $[P_i, I_i]$, where the maximum power of a server can possibly reduce is the dynamic power of the server $D_i = P_i - I_i$. So if we decide the value of B_i, the expected reduced power is $\Delta_i = P_i - B_i$. Thus, the expected rack power $P'_r = P_r - \sum_i \Delta_i$ Now we introduce expected power reduce ratio as $R_i = \Delta_i / D_i$. This ratio basically compares the expected reduced power to its current dynamic power, which quantifies the impact for current control. Obviously, the range of R_i is $[0, 1]$. To limit performance impact of control, our goal is to minimize average of R_i, which we call it impact ratio $R = \sum_i R_i / n$. Furthermore, we have an upper bound on ratio R_i, denoted as R_{max}.

Therefore, we formulate the Power Scheduling Problem (**PSP**) as:

$$\text{min} \qquad R = \textstyle\sum_i R_i/n \tag{5.1}$$

$$\text{s.t.} \qquad 0 \leq R_i \leq R_{max} \tag{5.2}$$

$$P'_r \leq P_t, \tag{5.3}$$

$$P'_r = P_r - \textstyle\sum_i \Delta_i \tag{5.4}$$

$$B_i = P_i - R_i \cdot D_i \tag{5.5}$$

where B_i is the result power budget for each server.

Table 10 summarizes key notations we used in the problem formulation.

PSP problem is a typical linear programming (LP) problem. There are many methods and tools to compute the solution of this LP problem if a solution exists. The solution of PSP problem, however, is special so that we can obtain the solution much easier instead of solving the general problem directly. Without the loss of generality, we assume that servers are sorted by D_i in decreasing order. Then the solution will be:

$$B_i = \begin{cases} P_i - R_{max} \cdot D_i, & \text{if } i < k \\ P_i - (P_r - P_t - \sum_{i=0}^{k} R_i \cdot D_i), & \text{if } i = k \\ P_i, & \text{otherwise} \end{cases} \tag{5.6}$$

where

$$k = \sup_{x \in \mathbb{Z}} (\sum_{i=0}^{x} R_i \cdot D_i \leq P_r - P_t) \tag{5.7}$$

So basically the scheduler will limit power budget for servers with higher dynamic power.

5.2.6 Resource agent

As we discussed in Chapter 5.1, the local resource agent limit power usage by leveraging core binding to control resources of tasks in our implementation. When a resource agent received a power budget command, it will control the number of binding cores in order to control the server power under a specified power budget until it receives a stop command. The resource agent reads power information directly from the server and the controlling cycle in our implementation is $1s$. Suppose the resource agent on server i receive a power budget B_i, then we can calculate the expected power reduce ratio $R_i = \frac{P_i - B_i}{P_i - I_i}$ where P_i, I_i are defined in Section 5.2.4. Note that resource agent works on a higher frequency than power scheduler so that B_i is fixed within a power scheduler cycle but P_i may change over time. As a result, we need to calculate R_i every Resource agent control cycle. Intuitively, the number of cores to unbind is $R_i \cdot N_c$ where N_c is the number of running cores. However, the number of cores does not always linear related to power consumption. They are just positive correlated. So we need to gradually control the number of cores binding to tasks to both increase control accuracy and reduce chattering. On the other hand, we also need to ensure that the resource agent can control the power within the given time. So we adopt a simple linear closed loop control system to unbind $R_i \cdot N_c / C$ cores, where C is convergence coefficient. We need to trade off between the impact of

changing binding cores and convergence time by tuning C and the results are shown in Section5.3.2.

After we calculate the number of cores to operate, resource agent will choose task with lower priority to unbind. In addition, R_i here is different from power scheduler that is not always positive because server power may be reduced below power budget. Negative R_i means release unbinded cores or binding more cores to tasks and the order to choose task is reversed.

The resource agent may stop partial of low priority tasks depends on the power budget required but it will never stop all the task on a server as we have a limit on expected power reduce ratio introduced in Section 5.2.4.

5.3 Evaluation

5.3.1 Experimental setup

Cluster setup and workloads The experiment platform is several racks which contain more than 200 homogeneous servers(with 56 cores) in a real over-provisioning production cluster. By real over-provisioning, we mean that the measured maximum power can be higher than the designed power budget. The over-provisioning ratio is 12.7% (this is measured by turning off DVFS and providing extra power source) and the rack is protected by DVFS and UPS. Given the hardware configuration, our goal is to utilize the existing hardware to improve QoS and significantly reduce power violation.

All servers in these racks are part of a datacenter-wide resource pool that is managed by a single job scheduler. Resource agent and power monitor are deployed on every server and a central power controller is responsible for manage all racks with independent decisions.

Three workloads involved in our evaluation: web service, MPI service, and Map-reduce. The first one is online service and the last one is offline service. MPI service can be either online or offline service depends on individual task but MPI task requires more on stability. Thus it always have a higher priority than Map-reduce. And Web service always have a higher priority than MPI service. These are all representative workloads in our data center.

Key performance metrics First key performance metric is the number of power violations. Safety is the first priority when considering over-provisioning. Users may choose to allow a few power violations, and small violation number shows the effectiveness of Pelican; Since our monitoring cycle is 10 seconds, the total time of violations is estimated as the production of violation number and monitoring cycle. Second is QoS. For online services, we focus on latency. And for offline services, we measure throughput. Our goal is to improve the throughput of offline workloads while not affecting the latency of online services. Third is the impact ratio (R). Besides the QoS, we hope to reduce the impact of operations on the number of servers. Obviously, a smaller average impact ratio is better;

5.3.2 The effectiveness of resource agent

Before we evaluate Pelican, We need to verify if the resource agent can work as expected given power budget from API server. To determine the convergence coefficient C described in Section 5.2.6, let's first consider an extreme situation that all 56 cores are running but only one core with the highest priority consumes 100% of power and the power budget is 0% of dynamic power. Since scheduler cycle is 10s and resource agent cycle is 1s, resource agent has 10 times to control the resource and reduce power. In this case, C should be equal or less than 5.6 in order to unbind the last target core. Then, let's

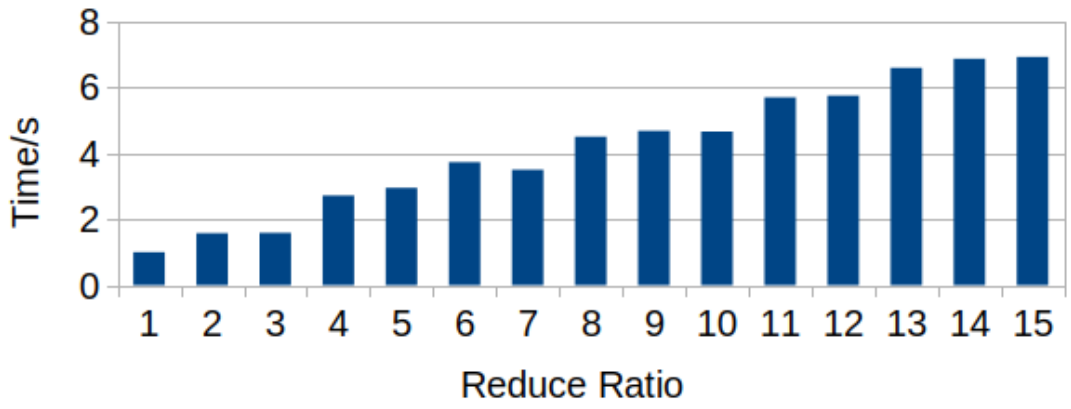


Figure 28: Average time for resource agent to reduce power under various workloads and given different reduce ratio.

consider the other extreme situation that the server is running at maximum power and all cores consume the same amount of power. In this case, since the maximum allowed power reduce ratio in our experiment is set to over-provisioning ratio, 12.7%, we need to unbind 12.7% of 56 cores, which is 7.11 cores. Even we unbind 1 core every second, we are able to unbind 8 cores within 10 seconds. So we tested the resource agent with various workload when C is 5 6. We found that the effectiveness of the resource agent is not very sensitive to C so we set C to 5.6. Figure 28 shows the average time for resource agent to reduce power under various workloads and given different reduce ratio. We can see that the resource agent can always control the power under target value within a scheduling cycle(< 10s).

5.3.3 The effectiveness of Pelican's control

To evaluate the effectiveness of our system, we first mirror production MPI and MapReduce workloads to two groups of racks: experiment and control groups (with Pelican turned off). We use the scale-down method in [97] to observe power violation while

Workload	Light		Heavy	
Group	Exp	Ctr	Exp	Ctr
R_{mean}	0.01%	0%	1.7%	0%
R_{max}	6.31%	0%	7.12%	0%
P_{mean}	0.921	0.921	0.972	0.975
P_{max}	0.992	1.036	0.998	1.048
$Violations$	0	21	0	759

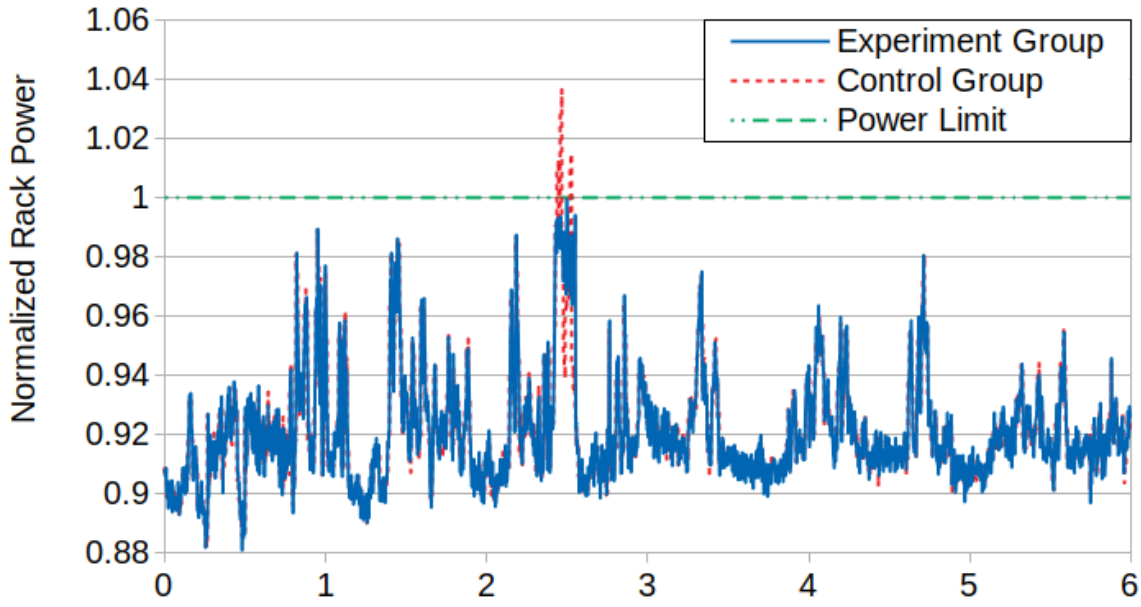
Table 11: Controller effectiveness under light / heavy workload. The experiment runs for 24 hours and the measurements are taken every 10s. R_{mean} and R_{max} are the mean/max impact ratio. P_{mean} and P_{max} are the mean/max power draw. $Violations$ is the total number of power violations.

keeping the physical devices safe. Also, we turn DVFS off so that our results can reflect real power change. We conduct our experiment for 24 hours using two extreme types of workloads in our cluster: heavy and light.

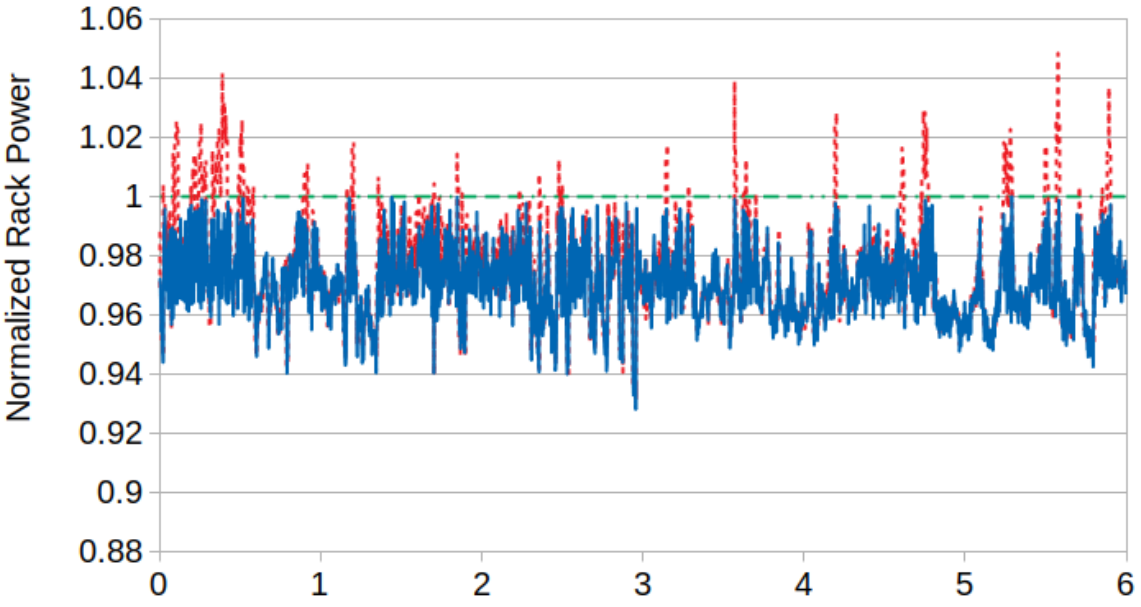
Table 11 shows a few performance metrics of Pelican. Under the heavy workload, there are 759 power violations in the control group without any power control, while there is no violation using Pelican's control. This proved the effectiveness of Pelican's power control ability.

The vibration of our workload is very high especially under heavy workloads. if we plot 24-hour power change, we can not see any detail so we plot 6 hours power change for both situations to show typical rack power utilization.

Figure 29(a) shows the light workload situation where the power draws mostly under the power limit. Even in this situation, we can see sharp power increases from time to time and several power violations, but overall, very few control actions are triggered. We can see a very high power increase happened at about 2.5h. In contrast, the heavy workload case Pelican control triggered quite often because the average power draw is very close to the power limit as shown in Figure 29(b). Besides the effectiveness test, one of our main goals is to figure out the appropriate rack threshold that can ensure safety. And then we



(a)



(b)

Figure 29: Power utilization under light (a) and heavy (b) workload. Pelican is deployed on Experiment group. And the control group is the base line for comparison.

apply the threshold to the same test on the production environment without scaling down for 220 hours with no power violence.

5.3.4 DVFS approach comparison

As we mentioned in Chapter 5.1, DVFS is another widely used way to control power resource for tasks but it can cause overall performance disturbance, which is not suitable for our situation. In this section, we compare the QoS and demonstrate our advantage under same over-provisioning ratio. The DVFS approach limit the power of each server to provisioned power when the overall power of a rack is close to the power limit. It is also threshold based approach and we choose the lowest threshold that can achieve same safety requirements as Pelican can.

We deploy Web service along with MPI and MapReduce on two racks with the same over-provision ratio configuration running the same workload trace. As we mentioned in workload description, Web service tasks are always on online services and MapReduce tasks are offline workloads but MPI can be both depending on the applications. We compare QoS of such cluster under DVFS and under Pelican respectively, i.e. compare latency on different levels for online tasks and throughput distribution for offline tasks.

Figure 30 shows the result of the median, 99.5th percentile and 99.9th percentile latency. Note that the y-axis is logarithmic. DVFS reduces the performance of online tasks almost doubling in terms of 99.5th percentile and 99.9th percentile latency. And DFVS also increase the median latency by about half compared with our system.

Figure 31 shows the results for throughput. We can see that the distribution of resulted throughput is similar. This is because DVFS can response faster but our method may provide more cores after power budget control. However, we still improve 1.43%

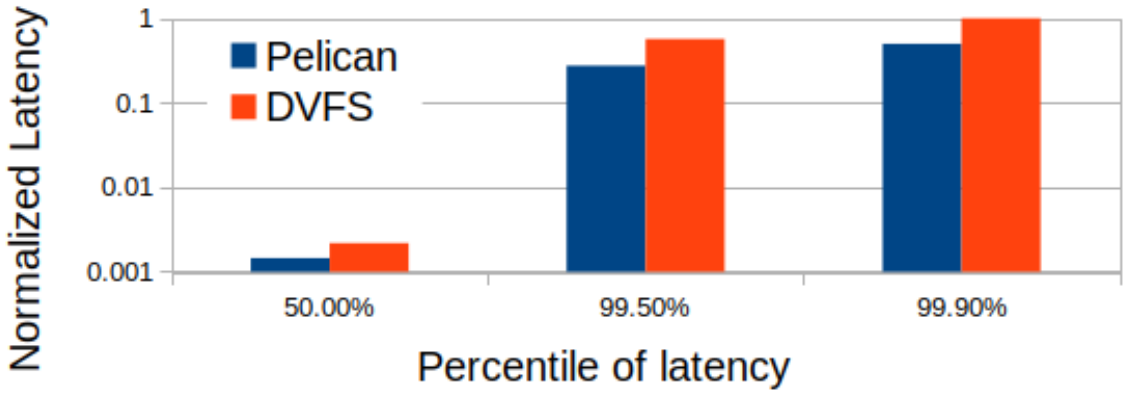


Figure 30: Latency comparison using either DVFS or Pelican as power controller. Latency normalized to the Latency throughput in both case.

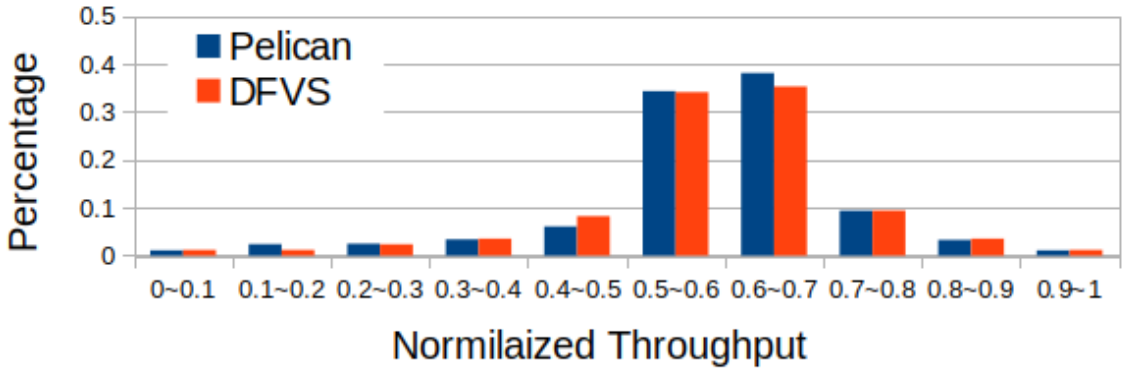


Figure 31: Throughput distribution using either DVFS or Pelican as power controller. Throughput normalized to the maximum throughput in both case.

overall throughput compared to DVFS method while achieving much better latency for online workloads. This is mainly because we limit power usage on low priority and longer running offline workloads and not affecting online services while DVFS slows down overall performance of server causing longer task queue that significantly increase the latency of online tasks. This also proves that our method successfully schedules power on the temporal dimension.

5.3.5 Tuning for QoS

Our goal in this work is to improve overall QoS by utilizing provisioning more server given a fixed power budget. Specifically, we would like to keep safety first, then ensure that the latency of online services is not affected and last improve throughput for offline services.

We have performed a heavy workload evaluation and production test to determine rack threshold P_t mainly for safety. Here we present our evaluation on safe rack power P_s which determines when to stop resource agent control. Low P_s can ensure that it is safe to release more computing resources but it may also cause low performance. On the other hand, high P_s results in more computing power to be used as soon as possible but may also lead to surge power increase so that online services are also involved in resource control.

Thus, we run different P_s from 0.80 to 0.97(the safe threshold is 0.98) under varying production workload. The workload input is from production input that we can not control so we show different workload in different cases. Table 12 shows the representative results. Bold rows represent results under typical workloads. G_t is the throughput gain. R_{mean} is the average impact ratio. P_{mean} and P_{max} are the mean and maximum power. P_{max} can exceed, for example in $\#12$, because it is the power of the control group.

First, we notice that the throughput gain G_t is positive related to P_s, especially in typical cases $\#4$, $\#6$, $\#9$, and $\#12$. But it is also related to high power demand. For example, in $\#11$, the overall power demand is low, which cause lower throughput gain. Then we found that R_{mean} is sensitive to P_s when P_{mean} is high, shown in $\#1$, $\#4$, $\#7$, and $\#10$. Also, the latency is always not affected when P_s is low.

#	P_s	P_{mean}	P_{max}	R_{mean}	$Latency$	G_t
1	0.8	0.936	1.031	0.0069%	1.0	9.7%
2		0.901	1.062	0.032%	1.0	9.30%
3		**0.925**	**1.011**	**0.067%**	**1.0**	**9.7%**
4	0.90	0.930	0.994	0%	1.0	10.50%
5		0.864	0.998	0%	1.0	10.10%
6		**0.923**	**1.006**	**0.06%**	**1.0**	**10.31%**
7	0.95	0.939	1.018	0.07%	1.0	11.2%
8		0.894	1.003	0.011%	0.995	13.5%
9		**0.929**	**1.013**	**0.074%**	**1.0**	**11.4%**
10	0.96	0.930	1.049	0.81%	1.0	11.6%
11		0.868	0.926	0%	1.0	10.8%
12		**0.925**	**1.021**	**0.078%**	**0.973**	**11.90%**

Table 12: Qos under different safe rack power P_s and workload condition. Bold rows represent results under typical workloads. G_t is the throughput gain. P_{mean} and P_{max} are the mean and maximum power for the control group, which are good indicators of the power demand. R_{mean} is the average impact ratio.

Overall, we find that 0.95 is an effective choice. The latency in the typical case is not affected and the throughput increases by 11.4%. The increment is higher than other choices except when P_s is 0.96. But the latency is reduced in the typical case when P_s is 0.96. As a result, we choose 0.95 for P_s considering safety and QoS.

5.4 Summary

In this chapter, we investigate opportunities and challenges of improving QoS for large-scale data center with heterogeneous workloads by over-provisioning. While power controlling is still the key to this problem, heterogeneous workloads requires faster response time and the ability to deal with the temporal power management. We design and implement Pelican in our real over-provisioned production cluster then empirically demonstrate the feasibility of scheduling power budget within a rack without affecting task placement to utilize computing resources but prevent power outage.

CHAPTER 6 CONCLUSION

The power budget has become one of the most contentious resources in data center management. Given the tremendous expense, it is crucial to fully utilize the power capacity of data centers to reduce the Total Cost of Ownership(TCO) and improve Quality of Services(QoS). One of the biggest problems in data centers we observed is insufficient power budget given the ever increasing demand on computation. While it is economically attractive to provision more servers into an existing data center with a fixed budget, it is a hard tradeoff between the cost saving and other considerations such as power system safety, performance stability especially for interactive jobs, as well as the complexity it brings to integrate with existing system. To provide comprehensive energy efficient system designs for data center, we conduct studies on different levels of power hierarchy.

On rack level, we investigate opportunities and challenges of improving QoS for large-scale data center with heterogeneous workloads by over-provisioning. While power controlling is still the key to this problem, heterogeneous workloads requires faster response time and the ability to deal with the temporal power management. We design and implement Pelican in our real over-provisioned production cluster then empirically demonstrate the feasibility of scheduling power budget within a rack without affecting task placement to utilize computing resources but prevent power outage. We adopt a two-level design that separate overall power scheduling and local power controlling for each server, which allows resource agent to maximize its ability to access local task information and react to large variation quickly while cooperate with other server with simple interface provided by power scheduler. This makes our power scheduling system more flexible and efficient

for heterogeneous workloads.

On row level, we design and implement Ampere, and empirically demonstrate the feasibility of using statistical control to indirectly manage the power utilization across a cluster. Specifically, as we use receding horizon control (RHC) to correct errors over time, we can achieve effective power control using statistical and inaccurate system models that are inexpensive to maintain in production. Also, using the simple freeze/unfreeze API, Ampere can be loosely coupled with complex job schedulers, which greatly simplifies the system implementation. For evaluation, we conduct controlled experiments with real production workload and provide detailed insights into the performance of the system. In production, we deploy Ampere to a data center, allowing us to provision 17% more servers, leading to a throughput gain of 15%.

CHAPTER 7 FUTURE WORK

The idea of eCope can be also extened to rack or cluster level. Since we know the workload-power relation for each server, we can do workload schedule according to these workload-power relations to make the entire rack or cluster energy proportional. We notice, that currently there is not much configurable hardware available on the market. This may be a limitation when applying the approach. However, our eCope methodology is easy to extend to software customization because in fact, we only assume that there are different configurations that can affect system power. So we will also explore software customization. On a higher level, there are two kinds of future work we are pursuing. First, on the power management side, we are exploring ways to schedule the jobs to different rows so that there can be a larger variance in power utilization across different rows, leading to more unused power to cultivate. Note that even with the improvement, we can still use the simple interface of Ampere. Second, we believe the simple statistical interface is a promising design to connect the low-level data center infrastructure to the higher-level software components such as the job scheduler and even applications, and allows cross-layer optimization. We are building a workload-sensitive cooling control system based on a similar interface. For heterogeneous workloads, it is likely to achieve better QoS if higher level power scheduling is also integrated with our Pelican although our power scheduler focuses on rack level power scheduling because this is the real problem that we need to solve first. The same power budget interface can be used to send through our API server. On the other hand, we are exploring other ways of controlling task resources so as to improve the stability and efficiency on power control.

REFERENCES

[1] ACADO Toolkit. http://acado.github.io/.

[2] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 217–228, 1807164, 2010. ACM.

[3] V. Anagnostopoulou, S. Biswas, H. Saadeldeen, A. Savage, R. Bianchini, T. Yang, D. Franklin, and F. T. Chong. Barely alive memory servers: Keeping data active in a low-power state. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 8(4):31, 2012.

[4] APC. Metered-by-outlet with switching rack PDU, http://goo.gl/qvE8NV.

[5] Facebook Autoscale. https://code.facebook.com/posts/816473015039157/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale/.

[6] L. A. Barroso, J. Clidaras, and U. Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2013.

[7] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[8] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, (12):33–37, 2007.

[9] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient idle desktop consolidation with partial VM migration. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 211–224. ACM, 2012.

[10] D. J. Bradley, R. E. Harper, and S. W. Hunter. Workload-based power management for parallel computer systems. *IBM Journal of Research and Development*, 47(5.6):703–718, 2003.

[11] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 103–116. ACM, 2001.

[12] C. Chen, M. Won, R. Stoleru, and G. Xie. Energy-efficient fault-tolerant data storage and processing in mobile cloud. *Cloud Computing, IEEE Transactions on*, 3(1):28–41, Jan 2015.

[13] N. Chen, X. Ren, S. Ren, and A. Wierman. Greening multi-tenant data center demand response. In *IFIP Performance*, 2015.

[14] D. Cheng, X. Zhou, P. Lama, M. Ji, and C. Jiang. Energy efficiency aware task assignment with dvfs in heterogeneous hadoop clusters. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):70–82, Jan 2018.

[15] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *ACM SIGOPS Operating Systems Review*, 44(1):76–80, 2010.

[16] Cisco Unified Computing System. `http://www.cisco.com/c/en/us/products/servers-unified-computing`.

[17] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 189–194. IEEE, 2010.

[18] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, Feb. 2013.

[19] C. Delimitrou and C. Kozyrakis. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, 41(1):77–88, 2013.

[20] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[21] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, 2007.

[22] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[23] Y. Fu, A. Holler, and C. Lu. Cloudpowercap: Integrating power budget and resource management across a virtualized server cluster. In *11th International Conference on Autonomic Computing (ICAC 14)*, pages 221–231, Philadelphia, PA, June 2014. USENIX Association.

[24] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. *SIGMETRICS Perform. Eval. Rev.*, 37(1):157–168, June 2009.

[25] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 317–330. ACM, 2009.

[26] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1), Dec.

2008.

[27] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[28] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17, 2012.

[29] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[30] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[31] HP Thermal Logic. `http://h20621.www2.hp.com/video-gallery/us/en/4b547a2b6afff5fc7fc512069ad54d3928a124c9/r/video`.

[32] C.-H. Hsu, Q. Deng, J. Mars, and L. Tang. Smoothoperator: Reducing power fragmentation and improving power utilization in large-scale datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 535–548, New York, NY, USA, 2018. ACM.

[33] C. Hu, C. Wu, W. Xiong, B. Wang, J. Wu, and M. Jiang. On the design of green reconfigurable router toward energy efficient internet. *Communications Magazine, IEEE*, 49(6):83–87, June 2011.

[34] IBM PowerExecutive. https://www-01.ibm.com/marketing/iwm/tnd/demo.jsp?id=IBM+PowerExecutive+Power+Capping+Mar07.

[35] Intel DCM Energy Director. http://www.intel.com/content/dam/www/public/us/en/documents/articles/intel-dcm-energy-director-overview.pdf.

[36] C. Isci, S. McIntosh, J. Kephart, R. Das, J. Hanson, S. Piper, R. Wolford, T. Brey, R. Kantner, A. Ng, et al. Agile, efficient virtualization power management with low-latency server power states. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 96–107. ACM, 2013.

[37] jMPC Toolbox. http://www.i2c2.aut.ac.nz/Resources/Software/jMPCToolbox.html.

[38] M. Kazandjieva, C. Shah, E. Cheslack-Postava, B. Mistree, and P. Levis. System Architecture Support for Green Enterprise Computing. In *Proceedings 5th International Green Computing Conference (IGCC)*, November 2014.

[39] F. Kong and X. Liu. Greenplanning: Optimal energy source selection and capacity planning for green datacenters. In *Proceedings of the 7th International Conference on Cyber-Physical Systems*, ICCPS '16, pages 5:1–5:10, Piscataway, NJ, USA, 2016. IEEE Press.

[40] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing distributed UPS energy for effective power capping in data centers. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 488–499. IEEE, 2012.

[41] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing distributed ups energy for effective power

capping in data centers. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 488–499, June 2012.

[42] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

[43] W. H. Kwon and S. H. Han. *Receding Horizon Control: Model Predictive Control for State Models*. Springer Science & Business Media, 2006.

[44] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis. Towards energy-efficient database cluster design. *Proc. VLDB Endow.*, 5(11):1684–1695, July 2012.

[45] C. Li, Y. Hu, R. Zhou, M. Liu, L. Liu, J. Yuan, and T. Li. Enabling datacenter servers to scale out economically and sustainably. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 322–333, New York, NY, USA, 2013. ACM.

[46] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, 21(5):1378–1391, 2013.

[47] L. Liu, C. Li, H. Sun, Y. Hu, J. Gu, T. Li, J. Xin, and N. Zheng. Heb: Deploying and managing hybrid energy buffers for improving datacenter efficiency and economy. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 463–475, New York, NY, USA, 2015.

[48] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen. GreenCloud: A new architecture for green data center. In *Proceedings of the 6th international*

conference industry session on Autonomic computing and communications industry session, pages 29–38. ACM, 2009.

[49] Y. Liu, S. C. Draper, and N. S. Kim. Sleepscale: Runtime joint speed scaling and sleep states management for power efficient data centers. *SIGARCH Comput. Archit. News*, 42(3):313–324, June 2014.

[50] Y. Liu, S. C. Draper, and N. S. Kim. SleepScale: Runtime joint speed scaling and sleep states management for power efficient data centers. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 313–324. IEEE, 2014.

[51] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 301–312, Piscataway, NJ, USA, 2014. IEEE Press.

[52] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st annual international symposium on Computer architecuture*, pages 301–312. IEEE Press, 2014.

[53] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 450–462. ACM, 2015.

[54] B. Luo, W. Chen, X. Liu, X. Li, L. Zhang, and W. Shi. Pelican: Power scheduling for qos in large-scale data centers with heterogeneous workloads. In *The tenth*

international green and sustainable computing conference, IGSC '19, Alexandria, VA, U.S.A., 2019.

[55] B. Luo, S. Wang, W. Shi, and Y. He. ecope: Workload-aware elastic customization for power efficiency of high-end servers. *IEEE Transactions on Cloud Computing*, 4(2):237–249, April 2016.

[56] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz. Towards energy-proportional datacenter memory with mobile dram. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 37–48, Washington, DC, USA, 2012. IEEE Computer Society.

[57] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos. Cloud computing: Survey on energy efficiency. *ACM Computing Surveys (CSUR)*, 47(2):33, 2014.

[58] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating server idle power. In *ACM Sigplan Notices*, volume 44, pages 205–216. ACM, 2009.

[59] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 319–330, New York, NY, USA, 2011. ACM.

[60] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[61] G. Metri, S. Srinivasaraghavan, W. Shi, and M. Brockmeyer. Experimental analysis of application specific energy efficiency of data centers with heterogeneous servers.

In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 786–793, Washington, DC, USA, 2012. IEEE Computer Society.

[62] M. P. Mills. The cloud begins with coal: Big data, big networks, big infrastructure, and big power. In *National Mining Association and American Coalition for clean Coal Electricity*, 2013.

[63] T. Minartz, T. Ludwig, M. Knobloch, and B. Mohr. Managing hardware power saving modes for high performance computing. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, pages 1–8, 2193377, 2011. IEEE Computer Society.

[64] S. Mittal. Power management techniques for data centers: A survey. *arXiv preprint arXiv:1404.6681*, 2014.

[65] Open Compute Project. `http://www.opencompute.org/`.

[66] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys (CSUR)*, 46(4):47, 2014.

[67] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood. Power routing: Dynamic power provisioning in the data center. In *ASPLOS*, 2010.

[68] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. MossÃľ, J. Mars, and L. Tang. 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)*, pages 246–258, Feb 2015.

[69] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35(5):89–102, Oct. 2001.

[70] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on compilers and operating systems for low power*, volume 180, pages 182–195. Barcelona, Spain, 2001.

[71] I. Psaroudakis, T. Kissinger, D. Porobic, T. Ilsche, E. Liarou, P. Tozun, A. Ailamaki, and W. Lehner. Dynamic fine-grained scheduling for energy-efficient main-memory queries. In *Proceedings of the Tenth International Workshop on Data Management on New Hardware*, DaMoN '14, pages 1:1–1:7, New York, NY, USA, 2014. ACM.

[72] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[73] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 48–59. ACM, 2008.

[74] Redis. `http://redis.io/`.

[75] Redis-benchmark. `http://redis.io/topics/benchmarks`.

[76] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012.

[77] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 418–428. IEEE, 2006.

[78] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 351–364. ACM, 2013.

[79] Open Data Center Committy. `http://www.opendatacenter.cn/`.

[80] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 63–72. IEEE, 2003.

[81] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. Power containers: An os facility for fine-grained power and energy management on multicore servers. *SIGPLAN Not.*, 48(4):65–76, Mar. 2013.

[82] D. Shin, J. Kim, N. Chang, J. Choi, S. W. Chung, and E.-Y. Chung. Energy-optimal dynamic thermal management for green computing. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, pages 652–657, New York, NY, USA, 2009. ACM.

[83] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, volume 10, pages 1–5. San Diego, California, 2008.

[84] F. Sun, H. Li, Y. Han, G. Yan, and J. Ma. Powercap: Leverage performance-equivalent resource configurations for power capping. In *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, Nov 2016.

[85] J. Sun, R. Zheng, J. Velamala, Y. Cao, R. Lysecky, K. Shankar, and J. Roveda. A self-tuning design methodology for power-efficient multi-core systems. *ACM Trans. Des. Autom. Electron. Syst.*, 18(1):4:1–4:24, Jan. 2013.

[86] Q. Sun, C. Wu, S. Ren, and Z. Li. Fair rewarding in colocation data center: Truthful mechanism for emergency demand response. In *IWQoS*, 2015.

[87] I. Takouna, W. Dawoud, and C. Meinel. Dynamic configuration of virtual machine for power-proportional resource provisioning. In *Green Computing Middleware on Proceedings of the 2Nd International Workshop*, GCM '11, pages 4:1–4:6, New York, NY, USA, 2011. ACM.

[88] S. K. Tesfatsion, E. Wadbro, and J. Tordsson. Perfgreen: Performance and energy aware resource provisioning for heterogeneous clouds. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*, pages 81–90, Sep. 2018.

[89] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[90] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica. The power of choice in data-aware cluster scheduling. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 301–316, 2014.

[91] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 28–28. USENIX Association, 2009.

[92] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, page 18. ACM, 2015.

[93] H. Voigt, T. Kissinger, and W. Lehner. Smix: Self-managing indexes for dynamic workloads. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, SSDBM, pages 24:1–24:12, New York, NY, USA, 2013. ACM.

[94] D. Wang, C. Ren, S. Govindan, A. Sivasubramaniam, B. Urgaonkar, A. Kansal, and K. Vaid. ACE: Abstracting, characterizing and exploiting peaks and valleys in datacenter power consumption. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):333–334, 2013.

[95] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy. Energy storage in datacenters: what, where, and how much? In *SIGMETRICS*, 2012.

[96] G. Wang, S. Wang, B. Luo, W. Shi, Y. Zhu, W. Yang, D. Hu, L. Huang, X. Jin, and W. Xu. Increasing large-scale data center capacity by statistical power control. In *EuroSys*, 2016.

[97] G. Wang, S. Wang, B. Luo, W. Shi, Y. Zhu, W. Yang, D. Hu, L. Huang, X. Jin, and W. Xu. Increasing large-scale data center capacity by statistical power control. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 8:1–8:15, New York, NY, USA, 2016. ACM.

[98] Z. Wang, N. Tolia, and C. Bash. Opportunities and challenges to unify workload, power, and cooling management in data centers. In *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, pages 1–6. ACM, 2010.

[99] A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *Perform. Eval.*, 69(12):601–622, Dec.

2012.

[100] D. Wong and M. Annavaram. Knightshift: Scaling the energy proportionality wall through server-level heterogeneity. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 119–130, Washington, DC, USA, 2012. IEEE Computer Society.

[101] Q. Wu, Q. Deng, L. Ganesh, C.-H. R. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song. Dynamo: Facebook's data center-wide power management system. In *ISCA*, 2016.

[102] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé. Energy-efficient policies for embedded clusters. In *ACM SIGPLAN Notices*, volume 40, pages 1–10. ACM, 2005.

[103] Z. Xu, Y.-C. Tu, and X. Wang. Pet: Reducing database energy cost via query optimization. *Proc. VLDB Endow.*, 5(12):1954–1957, Aug. 2012.

[104] Z. Xu, X. Wang, and Y. cheng Tu. Power-aware throughput control for database management systems. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 315–324, San Jose, CA, 2013. USENIX.

[105] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers. *ACM SIGARCH Computer Architecture News*, 41(3):607–618, 2013.

[106] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam. Ts-batpro: Improving energy efficiency in data centers by leveraging temporal-spatial batching. *IEEE Transactions on Green Communications and Networking*, pages 1–1, 2018.

[107] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie. Half-dram: A high-bandwidth and low-power dram architecture from the rethinking of fine-grained activation. In

Proceeding of the 41st Annual International Symposium on Computer Architecuture, ISCA '14, pages 349–360, Piscataway, NJ, USA, 2014. IEEE Press.

[108] W. Zheng, A. P. Centeno, F. Chong, and R. Bianchini. Logstore: toward energy-proportional storage servers. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 273–278, 2333723, 2012.

[109] W. Zheng, K. Ma, and X. Wang. Hybrid energy storage with supercapacitor for cost-efficient data center power shaving and capping. *IEEE Transactions on Parallel and Distributed Systems*, 28(4):1105–1118, April 2017.

ABSTRACT

TOWARD ENERGY EFFICIENT SYSTEMS DESIGN FOR DATA CENTERS

by

BING LUO

December 2019

Advisor: Dr. Weisong Shi

Major: Computer Science

Degree: Doctor of Philosophy

Surge growth of numerous cloud services, Internet of Things, and edge computing promotes continuous increasing demand for data centers worldwide. Significant electricity consumption of data centers has tremendous implications on both operating and capital expense. The power infrastructure, along with the cooling system cost a multi-million or even billion dollar project to add new data center capacities. Given the high cost of large-scale data centers, it is important to fully utilize the capacity of data centers to reduce the Total Cost of Ownership. The data center is designed with a space budget and power budget. With the adoption of high-density rack designs, the capacity of a modern data center is usually limited by the power budget. So the core of the challenge is scaling up power infrastructure capacity. However, resizing the initial power capacity for an exist data center can be a task as difficult as building a new data center because of non-scalable centralized power provisioning scheme. Thus, how to maximize the power utilization and optimize the performance per power budget is critical for data centers to deliver enough computation ability. To explore and attack the challenges of improving the power utilization, we have planned to work on different levels of data center, including server level,

row level, and data center level. For server level, we take advantage of modern hardware to maximize power efficiency of each server. For rack level, we propose Pelican, a new power scheduling system for large-scale data centers with heterogeneous workloads. For row level, we present Ampere, a new approach to improve throughput per watt by provisioning extra servers. By combining these studies on different levels, we will provide comprehensive energy efficient system designs for data center.

AUTOBIOGRAPHICAL STATEMENT

Bing Luo is a Ph.D. candidate in the Department of Computer Science at Wayne State University. He joined the Ph.D. program in 2012. He received his Bachelor of mathematics degree in Information and Computing Science and Bachelor of Engineering in Computer Science as second major at East China University of Technology and Science(ECUST) in Aug 2008. His research interests include Energy efficiency in data center, Energy-aware system design, Edge Computing, and he has published several papers in workshops, conferences and journal, such as EuroSysIGCC, IGSC, ICDCS, SEC. He has also served as a peer reviewer for many conferences and journals.