Energy-Efficiency Optimization Techniques for Computing Clusters: Exploiting the Heterogeneities

by
Xin Zhan

M.Sc., Brown University; Providence, RI, 2015
B.Sc., Shandong University of Science and Technology; Shandong, China, 2012

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in School of Engineering at Brown University

PROVIDENCE, RHODE ISLAND

May 2018

This dissertation by Xin Zhan is accepted in its present form
by School of Engineering as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Recommended to the Graduate Council

Date_____ _____

Sherief Reda, Advisor

Date_____ _____

Jacob Rosenstein, Reader

Date_____ _____

Na Li, Reader

Approved by the Graduate Council

Date_____ _____

Andrew G. Campbell, Dean of the Graduate School

Vitae

Xin Zhan was born in Jinan, China. He received his B.Sc. in Computer Engineering from Shandong University of Science and Technology in 2012. He started his Ph.D. project since Fall 2012. His research focus is performance and energy-efficiency optimizations for computing systems. He worked on power management frameworks that aim to deliver better energy-efficiency.

xin_zhan@brown.edu
Brown University, RI, USA

Publications:

1. R. Azimi, M. Badiei, X. Zhan, L. Na and S. Reda, "Fast Decentralized Power Capping for Server Clusters", in *IEEE Symposium on High-Performance Computer Architecture*, pp. 181-192, 2017.

2. K. Dev, X. Zhan and S. Reda, "Power-Aware Characterization and Mapping of Workloads on CPU-GPU Processors", in *IEEE International Symposium on Workload Characterization*, pp. 1-2, 2016.

3. X. Zhan, R. Azimi, S. Kanev, D. Brooks and S. Reda, "CARB: A C-State Power Management Arbiter For Latency-Critical Workloads", in *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 6-9, 2017.

4. X. Zhan, M. Shoaib and S. Reda, "Creating Soft Heterogeneity in Clusters Through

Firmware Re-configuration", in *IEEE Cluster, Cloud and Grid Computing*, pp. 540-549, 2016.

5. M. Badiei, X. Zhan, R. Azimi, S. Reda and N. Li, "DiBA: Distributed Power Budget Allocation for Large-Scale Computing Clusters", in *IEEE Cluster, Cloud and Grid Computing*, pp. 70-79, 2016.

6. R. Azimi, X. Zhan and S. Reda, "How Good Are Low-Power 64-bit SoCs for Server-Class Workloads?," in *IEEE International Symposium on Workload Characterization*, pp. 116-117, 2015.

7. X. Zhan and S. Reda, "Power Budgeting Techniques for Datacenters," in *IEEE Transactions on Computers*, Vol. 64(8), pp. 2267-2278, 2015.

8. R. Azimi, X. Zhan and S. Reda, "Thermal-Aware Layout Planning for Heterogeneous Datacenters," in *IEEE International Symposium on Low-Power Electronics and Design*, pp. 245-250, 2014.

9. X. Zhan and S. Reda, "Techniques for Energy-Efficient Power Budgeting in Data Centers," in *Design Automation Conference*, pp. 176:1-176:7, 2013.

Acknowledgements

First of all, I would like to thank my Ph.D. advisor, Prof. Sherief Reda, for his mentorship, advice and support during my graduate study at Brown University. His vision and insights on computer architecture researches guide us to achieve great progress and this thesis. From him, I learned not only knowledge but also many things that benefit my future life. I also want to thank Prof. Jacob Rosenstein and Prof. Na Li for taking time to review my dissertation and to be on my defense committee.

I would like to thank all my research collaborators: Reza Azimi and Kapil Dev from Brown, Masoud Badiei, Prof. David Brooks, Svilen Kanev and Prof. Na Li from Harvard University and Shuayb Zarar from Microsoft Research. I would also like to thank the fellow graduate students at Prof. Reda's group and colleagues at Brown. They are all brilliant people and made me a great time here. Specially for Reza Azimi, we have collaborated for four years and we delivered five papers together. He is the best partner I have ever been working with. Thank you also to the administrative staffs at School of Engineering for their assistance.

I would like to thank my best friends, Wentao Guan, Xiaoxiao Hou and Shanshan Dai for the fun memories. Finally, I would like to thank my parents, Enyuan Zhan and Xuqin Yang for their love and support while I am thousands of miles away from home.

Abstract of "Energy-Efficiency Optimization Techniques for Computing Clusters: Exploiting the Heterogeneities" by Xin Zhan, Ph.D., Brown University, May 2018

The development of cloud computing and online services result in rapid increases of number and scale of computing clusters. Because of cost and sustainability concerns, energy efficiency has been a major goal for cluster architects. Computing clusters are designed to serve a wide range of workload types. Heterogeneous workloads stress different aspects of the server hardware. The servers in a computing cluster also can be configured with various hardware configurations. These heterogeneities that exist in computing clusters and the hosted applications create potential opportunities of optimizations. Given the complexity and scale of the infrastructure and workloads in clusters, optimizing the performance and energy-efficiency of clusters are the most challenging problems.

Towards energy-efficient computing clusters, we formulate and devise efficient algorithms to optimize the heterogeneous cluster performance. We start with the power budgeting problem: in a power-constrained computing cluster, how to optimally allocate the power across the servers to maximize the performance. The cluster operates computing nodes and cooling infrastructure. We propose optimization techniques that find the optimal partition between cooling power budget and computing power budget, as well as the optimal computing power allocation among individual server nodes. As the size of cluster grows, the centralized optimization control will become a bottleneck of computation and networking. To improve the scalability and robustness of the computing power allocation method, we propose a fully distributed optimization algorithm that achieves optimality and fast convergence.

Computing cluster typically is composed of servers with diverse configurations where each type has distinct performance-power characteristics. The layout of the server (racks) impact significantly the hotspot and peak temperature in clusters, which subsequently determines the minimum cooling power. We propose an optimal optimization technique that

minimizes the cooling power while taking peak inlet temperature as constraint. With the characteristics of the heterogenous servers and the corresponding workloads are realized, our algorithm find the optimal organization of the server racks to reduce the required cooling power consumption.

System heterogeneity provides a great opportunity for energy-efficiency optimizations. However, the performance of the server nodes highly rely on the composition of the hosted workloads. As the workload characteristics changes rapidly in modern clusters, the static heterogeneous server makeup is not always effective. To introduce flexibility and reconfigurability in server heterogeneity, we propose a method to reconfigure a homogeneous server cluster to become heterogeneous cluster through firmware reconfiguration with respect to the workloads characteristics.

This dissertation summarizes our interesting insights in optimization problems for computing clusters and presents the corresponding methods of addressing each of the problems. We demonstrate that our optimization techniques achieve significant improvements over existing solutions. In addition, we present novel approaches to create and to better leverage the server heterogeneity in computing clusters.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computing clusters consist of thousands of servers with diverse functionalities. They enable large web services such as web search, e-commerce and social networks, as well as super computings and cloud computing with huge amount of data. Groups of servers are responsible for computing, storage or networking. Altogether they consume excessive amounts of power, with large facilities consuming up to 50 MW [4, 40]. Computing clusters in world-wide contribute to about 30 BW of power, which roughly equals to the power output of 30 large nuclear power plants [28]. As a result, the total cost of ownership of computing clusters is dominated by power consumption, which constrains total performance and scalability [24, 40, 59]. Improving the performance and energy-efficiency of clusters are quickly becoming urging issues in need of effective solutions.

The power consumption of computing infrastructures and the power of cooling units are two major components of the total power of the cluster, where the power consumption of the Computer Room Air Conditioning (CRAC) units depends on the power consumption of servers and the hot spots in the layout of the center [1]. To increase the efficiency of

computing clusters, many servers are normally hosted by an electric circuit than its rated power permits [8]. This power over-subscription is justified by the fact that the nameplate ratings on servers are higher than the servers' actual power utilization. Moreover, rarely all servers work at their peak powers simultaneously. In the case that the power consumption of subscribed servers peak at the same time and exceed the circuit capacity, power must be capped quickly to avoid tripping the circuit breakers. Power capping can be used as a safety mechanism to reduce power consumption of servers when supporting equipment fails [24, 63, 76]. For instance, the breakdown of a cluster's CRAC system may result in a sudden temperature increase of IT devices. In this scenario, power capping can help maintain the baseline temperature inside the cluster.

On the other hand, cluster workloads exhibit large variation in their characteristics. Different workload can stress different components of the server. For instance, numerically intensive high performance computing (HPC) jobs saturate the CPU usage, big data processing jobs are usually memory-bounded, and most the web service jobs are bounded by disk/network IO performance. Allocating higher power budget benefits differently for workloads due to there characteristics. This heterogeneity provides a potential opportunity for power budgeting and power allocation optimizations.

Chapter 3 and Chapter 4 of this dissertation aim to address the power budgeting optimization problem. In Chapter 3, we describe our centralized solution of total power budgeting, where the total power budget is allocated among the servers and cooling equipment to maximize the system normalized performance (SNP), or equivalently minimize the average runtime. A novel method is devised to split the total power budget between the computing servers and cooling units in a self-consistent way, where the cooling power meets the heat removal requirements for the computing power, which is allocated using an optimal power budgeting technique. We propose an optimal computing power budgeting technique that is inspired by methods for solving the well-known knapsack problem. The

budgeting technique identifies the optimal power caps for the servers, such that the total server power meets the computing budget and the system performance is maximized. We simulate a computing cluster with thousands of servers, where the power estimates for the servers are derived from real measurements on a server executing heterogeneous workload sets. Our experimental results demonstrate the advantages of our power budgeting method and performance improvements, in terms of SNP, slowdown norm and unfairness, over previous approaches.

Incorporating new server nodes is only practical if the computing cluster has a modular architecture. Currently used power management methods are not aligned with such an architecture as they have a centralized design, *i.e.*, there is a central controller aggregating information from all active computing nodes to allocate power efficiently [45]. Specifically, in conventional power management techniques, local data of all computing nodes, such as workloads, application priorities, and thermal distributions, are constantly monitored to efficiently allocate power budget among them in a centralized manner. Therefore, adding new computing nodes to a cluster necessitates reconfiguration of power management hardware and software which drives up the scaling cost. To tackle this challenge, in Chapter 4, we describe DiBA, a distributed power budgeting framework that scales gracefully for large clusters. It is a novel fully-distributed algorithm for power budget allocation to maximize the combined utility of the entire cluster subject to a given total power budget. The utility of each computing node is a metric of its throughput that depends on the benchmark of the workload as well as server specifications. In our distributed framework, each server exchanges its decision to increase/decrease its power usage with neighboring nodes in the cluster through rounds of communications. After each round, the local parameters of each node are updated in the form of a state-space model. In this phase, the local actions of neighbors are implemented as a control input. Further, the actions of servers take into account the global power constraint that must be satisfied. In contrast to centralized meth-

ods that have communication bottlenecks at the central coordinator, DiBA provides a fully decentralized mechanism which eliminates the role of the central coordinator. As a result, we demonstrate that for large scale clusters with thousands of nodes, the communication time of DiBA is substantially less than that of centralized scheme.

Cooling infrastructure is a major source of energy consumption in computing clusters. Recent studies show that cooling power can contribute 42% of the total energy consumption of clusters [4]. Cooling power of the CRACs is used to extract the heat produced from the servers and to supply cold air back to the facility through the perforated tiles. The supply temperatures of the cold air from the CRAC units should be set as high as possible to reduce cooling power while not violating the red temperature specifications of the servers. The planning of homogeneous computing clusters is relatively simple as servers are all identical in their configurations; thus, there is no inherent advantage from changing the locations of the racks as the servers will have the same power consumption behavior irrespective of their locations. In homogeneous clusters, it is the allocation of workloads to the servers that mainly determines the spatial power distribution inside the cluster, which consequently determines the thermal characteristics and the cooling power.

Modern computing clusters are heterogeneous in nature, where they deploy clusters of heterogeneous server platforms that offer the same instruction set architecture, and thus, they are capable of executing the same workloads. However, the hardware makeup of the servers can be completely different. Heterogenous servers can use different processor types, number of cores, and DRAM capacities. The heterogenous makeup helps clusters cater to workloads with different characteristics (e.g., transactional, batch, numerically intensive, etc) by matching workloads with the right platforms to better achieve the cluster's target metrics (e.g., performance and energy consumption). Another reason for the heterogeneity arises from multiple replacement, upgrades and the deployment of more cost-efficient systems that become available over time [4]. We observe that heterogeneous

clusters provide an interesting opportunity to plan their facilities in a way to reduce cooling power. Heterogeneous servers have different power specifications, and thus, the spatial positions at installment will lead to inherent thermal characteristics in the cluster. Thus, we can reduce cooling power by carefully laying out the racks of heterogeneous servers during the planning phase of computing clusters.

In Chapter 5, we formulate the problem of rack layout for planning of heterogeneous clusters, where the goal is to identify the best locations of the racks of servers with different hardware capabilities to improve the supply temperatures of the CRAC units and the total cooling power. Given the nature of modern computing clusters with varying utilization that is a function of time and sophisticated job schedulers, we reformulate the rack layout problem in a probabilistic manner to identify layouts that are likely to provide the best cooling subject to various operating conditions. An optimal solution methodology is presented based on integer linear programming (ILP) and evaluated using realistic cluster configurations. Since the planning of clusters occur only once, the ILP runtime is practically feasible.

However, since servers are meant to support diverse workloads in a cluster, making application-specific hardware component changes is impractical. Furthermore, creating a cluster by purchasing servers with different hardware capabilities can complicate resource management and increase costs. Instead, a more realistic approach is proposed to change the configuration of the available hardware and software components. Among the many potential ways of changing the configuration of the hardware components, we observe that modern servers offer a large number of firmware settings that can be tuned with significant impact on the runtime and power consumption of a server. Some of these settings include hardware-prefetching, adjacent cache-line perfecting, memory turbo boost, CPU turbo boost and hyper threading. Thus, by configuring these settings differently for different servers, we can create a soft heterogeneous mix of servers out of an originally

homogeneous set that delivers improved performance and energy efficiency for targeted workloads. In comparison to purchasing custom servers, we offer a soft approach for creating heterogeneity as it is always possible to change the customizations with a simple reboot to the servers.

The traditional approach of configuring the firmware settings involves a human in the loop. System administrators follow simple ad-hoc rules to identify the appropriate firmware settings [44, 5], which can potentially lead to ineffective use of the hardware components and is naturally prone to human errors. In contrast, we present an automated firmware exploration tool, *FXplore*, in Chapter 6. *FXplore* is effective in finding firmware configurations of servers that can deliver the maximum benefits in performance and energy efficiency. There are several challenges in finding the optimal firmware configurations. First, there are exponential number of configurations as a function of the number of firmware settings, which makes identifying the optimal configuration for a workload a hard problem. Second, creating a dedicated sub-cluster with its own custom firmware settings for each target workload can complicate system management, especially if there are a large number of target workloads. Third, administrators sometimes deploy workload co-runners on the same server. Through *FXplore*, we provide a framework that addresses these challenges.

Chapter 2

Background

In this chapter, we describe the background and related prior works that are related to the optimization problems in this dissertation. As the main goal of our proposed methods is to optimize power and energy-efficiency of computing clusters, we start with introducing the power provisioning mechanism and power capping techniques in clusters. To solve the power budgeting optimization, it is important to understand the performance-power models. Power budgeting techniques allocates power cap based on performance-power model to optimize the system performance. As cooling power in clusters is another major component of the total power, we introduce the thermal approaches to estimate the cooling power. The optimization methods proposed in this dissertation are exploiting the heterogeneities in clusters. We discuss the useful heterogeneities existing in clusters that can be leveraged in optimizations. Finally, we summarize prior computing cluster optimization works.

2.1 Power Provisioning and Capping in Computing Clusters

Cluster power provisioning. Computing clusters power provisioning policies assures sufficient capacity for each server. Given the fact that the typical power usage is much lower than the highest provisioning power, the power infrastructure is usually over-provisioned [24, 29, 74, 65]. Power capping mechanisms is designed to trade acceptable performance degradation for substantial saving of the power provisioning infrastructure. More importantly, it avoids the cascading failures due to overloading.

Power budgeting. Power budgeting control mechanisms are proposed to allocate a power cap for each server. Ghandi *et al.* proposed power budgeting methods for servers that execute the same workload [27]. This situation can be useful for computing clusters that execute transactional workloads of the same nature; however, they are not relevant for computing facilities that execute high-performance computing (HPC) applications. These later facilities typically have high utilizations where most of the servers are fully utilized executing a large range of workloads with potentially heterogeneous characteristics. Nathuji *et al.* consider the case of power budgeting for heterogeneous workloads and servers [58]. The main proposed approach is a greedy method, where the throughput per Watt for the servers is calculated, and then servers with higher throughput per Watt are allocated more power during budgeting.

Power allocation in multi-core processors is a related problem to power budgeting in computing clusters [37, 67]. Power budgeting for computing cluster is different in a number of ways: (1) unlike independent servers, multi-core processors do not offer power cap controllers for the individual cores; (2) workloads on a multi-core processor are likely

to show memory interference issues, whereas workloads servers are relatively independent unless they explicitly communicate using message passing; (3) computing clusters feature air conditioning units that have to be considered during power budgeting; and (4) the interactions between computing and cooling power in cluster are highly complex in nature.

Server power capping. To enforce a power cap on a server, a number of previous approaches have been proposed in the past [38, 63, 13]. One possible approach is to equip each server with a feedback controller that computes the observed difference between the measured power and the power cap, and accordingly adjusts the p-state of the server using dynamic frequency and voltage scaling (DVFS). As is shown in Figure 2.1. if the difference is positive then DVFS is decreased, and if the difference is negative then DVFS is increased.

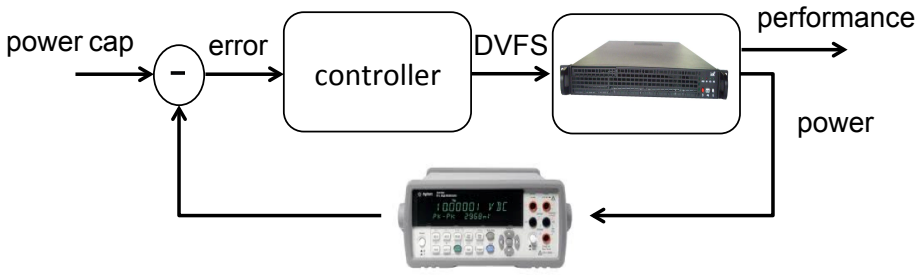

Figure 2.1: power capping feedback controller.

2.2 Cluster Performance-Power Models

Dynamic power budgeting requires the ability to estimate the impact of changing the operational power cap of the server on its performance characterizations. The performance characteristics of the workload can be realized by the performance monitoring counters

available in processors.

Performance monitoring counters. To collect runtime performance information of computers, most processors have hardware performance monitoring counters (PMCs) implemented by the Performance Monitoring Unit (PMU) of the CPU. Various performance events can be selected and counted with PMCs. The generally available performance events include branch prediction, cache/TLB misses, memory accesses and executed/stalled clock cycles. PMCs provide data for identifying program performance bottleneck and optimizing code performance [22, 70, 71]. Also, processor power consumption can be estimated with PMCs [14, 12, 68]. PMC is a powerful tool for modeling the characteristics of system and workloads.

Performance modeling. A number of models have been proposed in the literature to capture the relationship between the performance and power of a single server. In early works, Rajamani *et al.* proposed linear models [64] and Gandhi *et al.* proposed linear and cubic models [27]. The coefficients of these models are functions of the server configuration and the workload characteristics. In these works, fixed values for these coefficients were assumed irrespective of the workload characteristics. These values were obtained through prior characterization of standard benchmarks. As a result, these models are likely to show prediction errors for throughput and power in case heterogeneous applications with wide range of characteristics are executed on a cluster. In some recent works, Rountree *et al.* analyzed existing performance models for heterogeneous applications and proposed a linear model [66] based on IPC (instructions per cycle) and LLC (last level cache misses per cycle).

Performance metrics. Many metrics are used to assess different type of computing clusters. HPC clusters usually focus more on the throughput of the system, which can be

quantified as Billion Instructions Executed Per Second (BIPS) or Floating Point Operations Per Second (FLOPS). In case of certain heavy job dominates the throughput, performance normalization methods are introduced. The following normalized metrics are considered:

1. *System normalized performance* (SNP): SNP is the geometric mean of all the *Application normalized performance* (ANP) for the workloads running in the cluster, where ANP is the ratio of ideal runtime to actual runtime of a workload [77]. It is equivalent to the ratio of actual throughput to ideal throughput. Associated with ANP;

2. *Slowdown norm* (denoted by s): slowdown norm of workload i is calculated as $1/ANP_i$ and the slowdown norm of the computing cluster is computed by $(\sum_i s_i)/N$, where N is the number of workloads executing in the cluster;

3. *Unfairness* (denoted by f): Unfairness is the coefficient of variation of the ANPs for all the workloads in a cluster.

2.3 Thermal Modeling for Clusters

To compute the minimum requirement of cooling power given a distribution of computing power, a model is needed to translate the power distribution to thermal distribution in computing clusters. There exist some works in thermal management in clusters. For example, Tang *et al.* proposed a heat cross-interference coefficient matrix based method to model the thermal distribution of computing cluster in a fast way [73]. To quantify the efficiency of CRAC units (a typical CRAC unit is shown in Figure 2.2), Moore *et al.* proposed a

metric of Coefficient of Performance (CoP) [56], which enable us to build a relationship between cooling power consumption and the supply temperatures of CRAC units.

Modeling Cooling Power. CRAC units consume cooling power to remove the heat generated by the computing servers. The cooling power p_{cool} of a CRAC is a function of the inlet heat flowing into the CRAC, its supply temperature t_{sup}, and its coefficient of performance CoP, such that

$$p_{cool} = \frac{\text{inlet heat from racks}}{CoP(t_{sup})}, \tag{2.1}$$

where the $CoP(\cdot)$ gives the efficiency of the CRAC unit as a function of its supply temperature [56]. The exact function of the CoP is determined empirically by the CRAC manufacturer [56]. Achieving the minimum sufficient cooling power is equivalent to finding the maximum t_{sup} that guarantees that the inlet temperatures of all the servers are below the manufacturer's redline temperature t_{red}.

The inlet temperatures t_i^{in} of rack r_i is impacted by the CRAC supply temperature and the heat recirculation from all the racks in the cluster. Given a floor plan of the cluster, the heat recirculation matrix $\mathbf{D}$ can be derived using CFD simulations [73]. Each element

Figure 2.2: computer room air conditioning (CRAC) unit.

$\mathbf{D}(i,j)$ of the heat circulation matrix defines the contribution of the power consumption of rack j to the temperature increment of rack i. For a cluster that consists of n racks, let $\mathbf{t}^{out} = [t_1^{out}, t_2^{out}, \cdots, t_n^{out}]$ denotes the vector of the outlet temperatures of the racks, while $\mathbf{t}^{in} = [t_1^{in}, t_2^{in}, \cdots, t_n^{in}]$ denotes the vector of inlet temperatures. If $\mathbf{p} = [p_1, p_2, \cdots, p_n]$ denotes the vector of power consumption of racks in cluster, we can compute the inlet temperatures as [73]:

$$\mathbf{t}^{in} = \mathbf{t}^{sup} + \mathbf{Dp} \tag{2.2}$$

where $\mathbf{t}^{sup}$ is a vector of length n that denotes the supply temperatures of the racks, where each element is equal to t_{sup} of the CRAC units.

2.4 Heterogeneity in Computing Clusters

Since applications often have varying hardware-resource requirements, heterogeneity helps improve the performance and energy-efficiency of large-scale computing platforms. In the literature, employing heterogeneous hardware resources has been shown to improve the performance of HPC and cloud-computing clusters [17, 50, 52, 51, 18, 34]. Specifically, in [51] and [52], Mars *et al.* have quantified the potential performance improvements that can be achieved by smartly leveraging the heterogeneity in the instruction-set architecture and hardware resources of servers. Subsequent works such as [50] and [17, 18] have proposed methods that employ in-place continuous workload profiling techniques and scheduling algorithms to better utilize heterogeneous computing fabrics in warehouse scale computers. Despite these benefits, server cluster operators usually tend to purchase servers with ho-

mogeneous hardware configurations in order to minimize cost and resource-management issues. Thus, the existing heterogeneity in server clusters is minimal, which is simply a result of replacing and mixing new server hardware with older ones. Consequently, the attainable performance benefits using the aforementioned approaches are limited. A potential complementary is altering the firmware configurations to increase the level of heterogeneity in servers. Many of the firmware configurations cannot be changed in software, for instance *via* the operating system or virtual-machine managers. There exists a large amount of prior work that employs software techniques such as voltage and frequency scaling of the processor [35, 37, 13] and main memory [15, 20, 47]. Although these are effective methods to control the server behavior, we believe there are many more control knobs to improve or impair server performance and energy efficiency, which can only be controlled through the firmware. CPU hyper-threading is one such example, which is shown to significantly impact application performance [39, 32].

There is also some related work in the compiler community that employs machine learning (ML) to tune memory prefetch settings in software and in BIOS [25, 42, 43]. A few works provide an overview of some of these approaches [49, 36]. Liao *et al.* adjust four memory prefetch settings in the firmware and collect performance counters to build ML models, which optimize any memory performance parameter such as throughput or cache miss rate [42]. For new workloads, they use these models and predict the optimal firmware configurations.

2.5 Related Previous Optimization Works

The problem of efficient power management in clusters has been studied extensively from different perspectives, for a recent survey see [7]. Different practices can be applied to

reduce power consumption. Nevertheless, they can be divided into one of the following categories:

1. Schemes based on dynamic voltage/frequency scaling (DVFS) [46, 6, 21, 33].
2. Load balancing and workload scheduling techniques [11, 60].

For memory-intensive workloads, dynamic voltage and frequency scaling (DVFS) during memory-bound phases of execution have been shown to provide power savings with minimal impact on the quality of service (QoS) [16]. Using the same technique, Beloglazov *et al.* [6] proposed heuristics for energy management under strict service level agreement (SLA) constraint. Specifically, by dynamically migrating virtual machines (VMs) across physical machines, workloads are consolidated and subsequently idle resources are put on a low-power state using DVFS. In conjunction with the DVFS technique, Elnozahy *et al.* [21] evaluated five different policies for cluster-wide power management in server farms. In particular, in the independent voltage scaling policy, the authors consider a mechanism in which each node independently manages its own power consumption using dynamic voltage scaling. Nevertheless, this policy does not consider the hard constraint on the power consumption of the entire cluster.

Previous work proposed a centralized power feedback controller for satisfying end-to-end delay [46]. The central application performance monitor measures the latency of delay sensitive workloads. It then communicates the information to a centralized DVFS controller to adjust allocated power for each server. Clearly, such a centralized scheme does not suite large-scale clusters: it is prone to single point of failure and the centralized controller must solve a complex, high-dimensional optimization to determine the power cap for each computing node, which can introduce delay.

Along the path of leveraging DVFS techniques, distributed power management schemes are proposed[33]. Specifically, the problem of minimizing the power consumption in a three tier computing cluster under the end-to-end SLA constraint has been investigated. To solve this problem, two strategies have been investigated. In the first strategy dubbed as *OptiTuner*, a primal-dual decomposition method has been employed, where each tier of servers updates its power consumption by communicating with a central unit. In the second strategy, a Linear-quadratic regulator (LQR) control method has been employed. Despite having a distributed structure, OptiTuner suffers from the single point of failure. Because, it requires a central node to coordinate the optimization parameters, and therefore, is susceptible to node failure. Moreover, OptiTuner allocates power uniformly at the tier level, *i.e.*, all machines at each tier are assumed to host same amount of workload.

In the context of load balancing techniques for power management, a "bidding" mechanism akin to the economic models for managing shared resources has been studied [11]. In the proposed framework, services bid for resources and negotiate for SLA based on the offered price and their QoS requirement. Therefore, SLA is assumed to be a flexible parameter. While load balancing improves the system performance, load concentration (unbalancing) can save energy by making servers idle and thus consume less energy [60]. To optimize the trade-off between performance versus power saving, a load balancing and unbalancing algorithm is proposed [60] that takes into account both the total load imposed on the cluster and the power and performance implications of turning nodes off.

Chapter 3

Total Power Budgeting in Computing Clusters

3.1 Introduction

We describe our holistic total power budgeting technique in this chapter. The total power consists of computing power and the cooling power. The major challenge is that the cooling power is a function of computing power distribution. Under a total power budget, partitioning the budget between the cooling power and computing power is a hard problem. Another challenge, in server power budgeting, is that different workloads trigger different power consumption patterns, and thus the power management settings that work for one set of workloads do not necessarily work for another set of workloads [37, 62]. As a result, one needs to find settings for each server that lead to a global optimal for the entire computing facility. Our goal is to devise a new power budgeting method, where the total power budget is allocated among the servers and cooling equipment to maximize the

system normalized performance (SNP), or equivalently minimize the average runtime. We summarize our contributions as follows.

1. We propose a novel method to partition the total power budget between the computing servers and cooling units in a *self-consistent* way, where the cooling power meets the heat removal requirements for the computing power, which is allocated using an optimal power budgeting technique.

2. We propose a novel *throughput predictor* for servers with heterogeneous workload sets, where the measurements from the performance counters are used to estimate the change in the throughput as a function of the server power cap, on top of which, we can estimate the Application normalized performance (ANP) beyond current power cap.

3. Leveraging the throughput predictor, we propose an *optimal computing power budgeting* technique that is inspired by methods for solving the well-known knapsack problem. The budgeting technique identifies the optimal power caps for the servers, such that the total server power meets the computing budget and the system normalized performance is maximized.

4. We setup a realistic simulation environment for a computing cluster with thousands of servers, where the power estimates for the servers are derived from real measurements on a server executing heterogeneous workload sets. We use Computational Fluid Dynamics (CFD) simulations to ensure accurate modeling of air flow and heat transfer within the center, and use the CFD results to compute the cooling power requirement for a given power distribution. To speedup CFD simulation, we use an approximate approach based on heat cross-interference coefficient matrix. We experimentally demonstrate the advantages of our power budgeting method and

performance improvements, in terms of SNP, slowdown norm and unfairness, over previous approaches.

The organization of this chapter is as follows. We formulate the power budgeting problem and describe our proposed framework in Section 3.2. Our experimental results are presented in Section 3.3, and Section 3.4 provides the conclusions of this work and directions for future work.

3.2 Proposed Approach

We assume that a computing cluster is composed of n servers with identical hardware configuration and m CRAC units. We make no restriction on the operational workloads, i.e., the workload among the servers and even within the processors of a single server could be different. We also assume that the supply temperature of the CRAC units can be controlled independently. We assume a closed-loop queueing model where all servers are fully utilized. As a result, maximizing the performance of a server is equivalent to minimizing its response time [27].

Problem Statement: Given n fully utilized servers with heterogeneous workloads, m CRAC units, and a total B power budget, the objective is to distribute the total power among the n servers and m CRACs, such that the SNP is maximized or equivalently the average response time is minimized. That is, if p_i denotes the allocated power for server i and t_{sup} denotes the supply temperature of CRAC units, then the goal is to determine the power caps of the servers and the supply temperatures of the CRACs to maximize the SNP such that $B_s + B_{CRAC} \leq B$, where $B_s = \sum_{i=1}^{n} p_i$ is the total server computing power, and

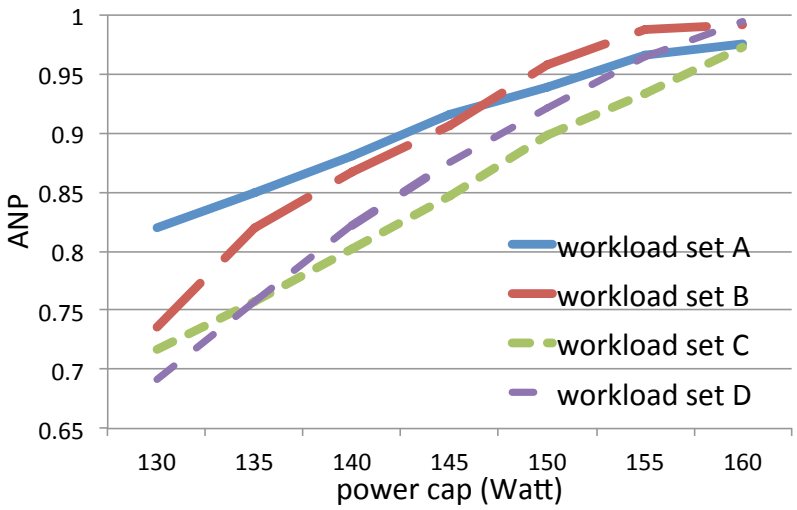

Figure 3.1: Impact of power cap on the SNP of a server.

B_{CRAC} is the total cooling power, such that the budget is allocated in a *self-consistent* way, where cooling power is able to extract the heat generated from the servers while staying below the manufacturer redline temperature.

If the workloads on all servers are identical then the budgeting problem is trivial since the total power can be divided uniformly among all servers. To get a better understanding of the relationship between SNP and the allocated power cap in case of heterogeneous workloads, we equip our experimental server with a power capping controller. The capping controller executes once every 100 millisecond and adjusts the p-state of the server using DVFS based on the difference between the allocated power cap and actual power consumption [38]. We report the application normalized performance (ANP) as a function of the power cap for four identical servers with different workload sets in Figure 3.1, where each server is executing a heterogeneous mix of four workloads from the SPEC CPU06 benchmarks. The plot leads to a number of observations.

1. The observed ANP is highly dependent on the workload characteristics. Workloads

C and D show large improvement in ANP with increased power allocations while workload set A shows modest improvements. ANP of workload set B grows fast under lower power budgets, while it saturates at higher power budgets. Thus, some workload sets will not be able to leverage their allocated power caps to improve ANP.

2. The plot shows that the gradient of an individual workload set plot changes as a function of the operating power cap. For instance, Workload D shows a larger gradient in the range of 130-140 W compared to other regions of operation. Thus, accurate modeling requires considering the impact of the operational power cap of the server on its performance characterizations.

3. The plots of workloads C and D show that general greedy allocation methods (e.g., [58]) will not give optimal results. For example, if the current power allocations for workload sets C and D are at the lowest cap, then Figure 3.1 shows that workload set B has higher ANP than workload set D, which can lead to the wrong conclusion that it is better to allocate more power to workload set C. However, the plots of the two workload sets eventually cross over, where workload set D attains large ANP than workload set C at higher power caps.

While clusters contain other elements (e.g., network switches, UPS, and chillers), we do not focus on these elements in this work. We mainly focus on servers and CRACs because they (1) consume the largest chunk of power (total about 75%) in a computing clusters [4]; (2) represent the most adaptable elements in a clusters; and (3) interact indirectly through heat coupling.

Overall Approach: Our power budgeting approach, shown in Figure 3.2, consists of two components: (i) a total power budgeting method that partitions the total power budget, B,

into the computing budget, B_s, and the cooling budget, B_{CRAC}, in a self-consistent way; and (ii) an *optimal computing power budgeting* method that identifies the power cap for each server, such that the total computing power budget, B_s, is met and the SNP is maximized. SNP is the geometric mean of the ANPs of all the servers. Maximizing the SNP requires the information about the ANP value of each server for different power caps, which is not available during runtime. Thus, an ANP prediction method is desired. As ANP is the ratio of current throughput to ideal throughput which is the throughput under highest power cap, the problem could be translated to predict the throughput over power caps. Our optimal computing power budgeter makes use of a novel *throughput predictor* that takes as inputs the measurements, e.g., power and performance counters, of servers at the current power cap, and uses them to predict the change in throughput of each server for every possible power cap. Note that we claim our computing power budgeting algorithm is optimal. The self-consistent partitioning algorithm guarantees a partition between computing power and cooling power where the cooling power meets the minimum requirement for the computing power. However, the combination of these two algorithms do not necessarily yield the optimal total power budgeting solution. Each of these components is described in the next subsections. We summarize the symbols and notations involved in this Chapter in Table 3.1.

3.2.1 Total power budgeting

Our goal is to apply a total power budget for both computing power and cooling power in a self-consistent way, where the cooling power, B_{CRAC}, extracts the heat generated from the computing power. The cooling power is a function of many factors, including the layout of the computing cluster, the spatial allocation of the computing power, the air flow rate,

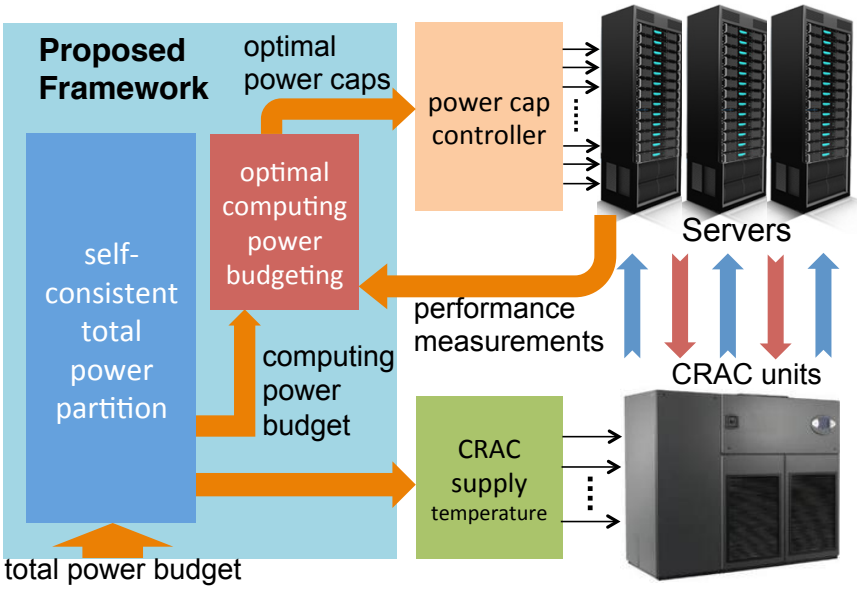

Figure 3.2: An overview of our proposed method.

and the efficiency of the CRAC units. The power consumption p_{crac} of a CRAC unit is equal to

$$p_{crac} = \frac{\sum_i p_i}{CoP(t_{sup})}, \tag{3.1}$$

where $\sum_i p_i$ is the power consumption of servers with their heat flow directed towards the CRAC unit, and CoP is the *coefficient of performance* that gives the performance of the CRAC units [56]. For example, based on physical measurements, an empirical model for CoP of the chilled-water CRAC units at the HP Labs Utility cluster is equal to:

$$CoP(t_{sup}) = 0.0068t_{sup}^2 + 0.0008t_{sup} + 0.458, \tag{3.2}$$

where t_{sup} is the supply air temperature of the CRAC unit in degrees Celsius [56]. To find the minimum sufficient cooling power for an allocation of a certain computing power,

it is necessary to maximize the supply air temperature t_{sup}, while ensuring that the inlet temperatures of all the servers will not exceed the manufacturer's redline temperature t_{red}. Identifying the inlet temperature for the servers requires accurate CFD models for the air flow and the heat transfer dynamics inside the computing cluster.

Figure 3.3 shows a typical pattern of the air flow and heat flow racks for a computing cluster, which is plotted from simulating 3200 servers using 6SigmaRoom Lite [26]. The

Symbol	Description
m	the number of CRAC units in the clusters
n	the number of servers in the clusters
r	the number of power caps on the server
B_s	total computing power
B_{CRAC}	total cooling power
P_{CRAC}	the power consumption of CRAC units
p_i	power cap of the i-th server
CoP	coefficient of performance
$\vec{P}$	power cap settings of all servers
t_{red}	the manufacturer's redline temperature
$\vec{T}_{red}$	the manufacturer's redline temperature vector
t_{sup}	the supply temperature of CRAC units
$\vec{T}_{sup}$	the supply temperature vector
t_{in}^i	the inlet temperature of the i-th server
$\vec{T}_{in}$	the inlet temperature vector represents all servers
t_{out}^i	the outlet temperature of the i-th server
$\vec{T}_{out}$	the outlet temperature vector represents all servers
K	a diagonal matrix where K_{ii} representing the coefficient to translate the energy consumption to temperature rise of server i
A	heat cross-interference coefficient matrix
X	a matrix representing the power cap of each server
w	the power increment over the least power cap
$\tau_i(p_i)$	the throughput of server i
a, β	coefficients of the quadratic model of throughput

Table 3.1: Symbols and definitions

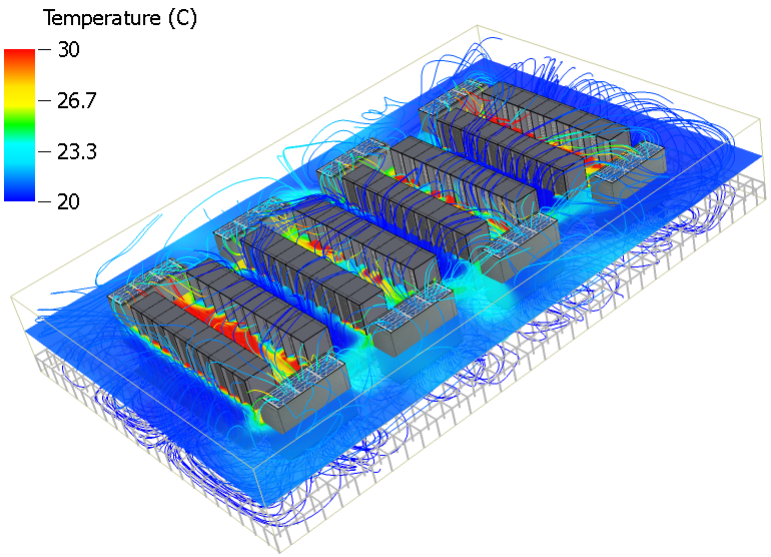

Figure 3.3: The air flow in computing cluster and the spatial temperature maps.

streamlines indicate air flow and air temperature from floor grilles to top side of CRAC units via server racks. By simulation the thermal map inside the cluster, we can evaluate the inlet temperatures of servers from the report of the tool. If the results of CFD simulation show that the inlet temperature of any server violates t_{red}, then t_{sup} should be lowered to bring the inlet temperature back under t_{red}, and if the inlet temperature has not reached t_{red}, then t_{sup} should be increased without causing an inlet temperature of racks increase beyond t_{red}. Given the result from CFD simulation and the total computing power budget, Equation (3.1) and Equation (3.2) enable us to compute the required cooling power B_{CRAC}.

To ensure that the sum of the computing power, B_s, and the cooling power, B_{CRAC}, meets the total power budget B, we propose an algorithm, given in Algorithm 1, to identify a self-consistent partitioning of the total power budget. The main loop of the iterative algorithm first calculates the computing power budget B_s in Step 3, and then in Step 4,

the computing budget, B_s is allocated by power budgeting algorithms. We use the dynamic programming based algorithm described in the next subsection to guarantee B_s is allocated optimally among the servers. Given power allocation and the cluster configuration, Step 5 estimates the minimum required cooling power, B_{CRAC}, as described in the previous paragraph. If it happens that $B_{CRAC} + B_s = B$ (step 6), then the algorithm has converged to a solution; otherwise, it continues iterating.

Proof of Convergence: Let (B_s^*, B_{CRAC}^*) denote the self-consistent solution of a total power budget $B_s^* + B_{CRAC}^* = B$ at a maximum CRAC supply temperature of t_{sup}^*. Let the computing power at iteration k of the algorithm is denoted by $B_s(k)$, and the minimum cooling power required for heat extraction is $B_{CRAC}(k)$ at a maximum CRAC supply temperature of t_{sup}^k. To prove it by induction, the hypothesis is, at any iteration k, $|B_s(k) - B_s^*| > |B_s(k+1) - B_s^*|$. For iteration $k+1$, our method will update the computing power as:

$$\begin{aligned} B_s(k+1) &= B - B_{CRAC}(k) \\ &= B_s^* + B_{CRAC}^* - B_{CRAC}(k), \end{aligned}$$

input : Total power budget B; cluster configuration; T_{red}.
output: Computing power B_s and cooling power B_{CRAC}.
1. initialize B_{CRAC} based on initial CFD simulation;
2. **while** B_{CRAC} *is not equal to* $B - B_s$ **do**
   3. let $B_s = B - B_{CRAC}$;
   4. budget B_s using the multi-choice knapsack algorithm;
   5. run CFD simulations to get minimum B_{CRAC} given T_{red};
end

Algorithm 1: Algorithm for self-consistent total power budgeting.

which can be re-arranged to:

$$\begin{aligned} B_s(k+1) - B_s^* &= B_{CRAC}^* - B_{CRAC}(k) \\ |B_s(k+1) - B_s^*| &= |B_{CRAC}^* - B_{CRAC}(k)| \end{aligned}$$

Given Equation 3.1, we can further formulate $B_{CRAC}(k) = \frac{B_s(k)}{CoP(t_{sup}^k)}$ and, similarly, $B_{CRAC}^* = \frac{B_s^*}{CoP(t_{sup}^*)}$. Thus,

$$|B_s(k+1) - B_s^*| = |\frac{B_s^*}{CoP(t_{sup}^*)} - \frac{B_s(k)}{CoP(t_{sup}^k)}|$$

As dynamic programing algorithm and sophisticated CFD simulations are included in our proposed algorithm, it could be intractable to formulate the relationship between the required supply temperature and a power distribution in cluster in a closed mathematical form. Without this relationship, we are not able to describe how the changing in power budget will impact the cooling power. Thus, in this section, we incorporate empirical data into the proof. Define $\delta_p(k) = |B_s(k) - B_s^*|$ and $\delta_c(k) = |\frac{B_s(k)}{CoP(t_{sup}^k)} - \frac{B_s^*}{CoP(t_{sup}^*)}|$. Further, define Ratio of Distance $R(k) = \delta_c(k)/\delta_p(k)$. We plot the value of R over iterations in Figure 3.4. As k increases, it shows a clear trend that $R(k)$ tend to stabilize at a value less than 1, namely, $\delta_p(k) > \delta_c(k)$ for every iteration. As $\delta_p(k+1) = \delta_c(k)$, $\delta_p(k+1) < \delta_p(k)$.

Thus, the distance, $\delta_p(k+1)$, between $B_s(k+1)$ and B_s^* is less than the distance,

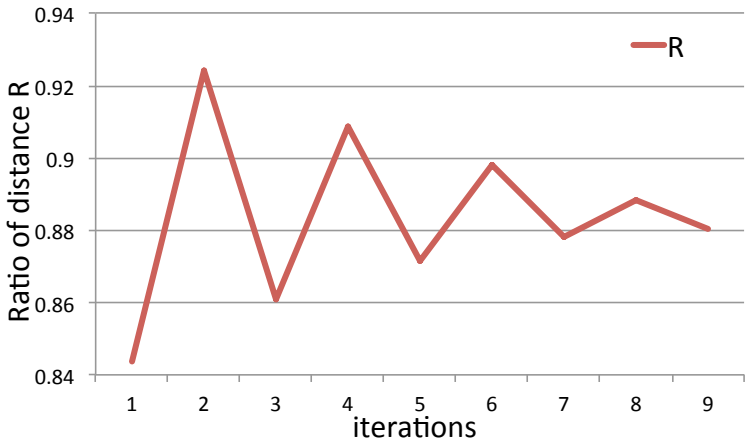

Figure 3.4: The Ratio of Distance over iterations.

$\delta_p(k)$, between $B_s(k)$ and B_s^*. Therefore, the computing power approaches B_s^* with every iteration and finally converges to B_s^*. ■

Simulation Speedup. Accurate CFD simulation usually takes relatively long time and the workload phase might change during this period of time. To speedup this procedure, in our work, we use an abstract heat cross-interference coefficient matrix, denoted by D. It is based on thermal model that can be used to compute the thermal map for a give spatial distribution of computing power in computing cluster [73]. Each element $D(i,j)$ of the heat cross-interference coefficient matrix defines the impact of server j to the temperature increment of server i. $\vec{T}_{out} = [t_{out}^1, t_{out}^2, \cdots, t_{out}^n]$ denotes the outlet temperatures for servers, $\vec{T}_{in} = [t_{in}^1, t_{in}^2, \cdots, t_{in}^n]$ denotes the inlet temperatures and $\vec{P} = [p_1, p_2, \cdots, p_n]$ denotes the power distribution of servers in cluster. Let t_{sup} be the CRAC supply temperature and $\vec{T}_{sup} = [t_{sup}, t_{sup}, \cdots, t_{sup}]$, we can compute the inlet temperatures as [73]:

$$\vec{T}_{out} = \vec{T}_{sup} + (K - D^T K)^{-1} \vec{P} \tag{3.3}$$

$$\vec{T}_{in} = \vec{T}_{out} - K^{-1}\vec{P} \tag{3.4}$$

where K is a diagonal matrix where K_{ii} representing the coefficient to translate the energy consumption to temperature rise of server i. D is the heat cross-interference coefficient matrix.

$$\vec{T}_{in} = \vec{T}_{sup} + [(K - D^T K)^{-1} - K^{-1}]\vec{P} \tag{3.5}$$

where D^T is the transpose of matrix D. If the results of the $\vec{T}_{in}$ prediction show that the inlet temperature of any server violates t_{red}, then $\vec{T}_{sup}$ should be lowered to bring the inlet temperature back under $\vec{T}_{red}$, and if the inlet temperature has not reached $\vec{T}_{red}$, then $\vec{T}_{sup}$ should be increased without causing an inlet temperature of racks increase beyond $\vec{T}_{red}$.

3.2.2 Computing Power Budgeting

Our goal is to maximize the normalized performance under a total computing power budget B_s. In our work, we consider the system normalized performance (SNP) from several metrics as the objective to maximize. We consider a discrete set of individual server power caps with a fixed increment (e.g., 130 W, 135 W, 140W, . . .). The choice of a discrete number of power caps is natural given that p-states are discrete and changing them does not lead to a continuous power range. Thus, the power cap of a server can be described as

$$p_i = p_0 + \sum_{j=1}^{r} w_j X_{ij}, \tag{3.6}$$

where p_0 is the least possible power cap, r is the number of individual server power caps, w_j is the increment power for each cap over the least possible cap, and $X_{ij} \in \{0, 1\}$, where X_{ij} is only equal to 1 when server i is assigned a power cap equal to $p_0 + w_j$. Based on our experimental server, we pick power caps as: 130 W, 135 W, ... In formal notation, $r = 8$, $p_0 = 130W$ and $w_1 = 0$, $w_2 = 5$, $w_3 = 10$, ..., $w_8 = 35$.

The ANP of a server can be calculated from throughput under current power cap and ideal throughput. In this work, we assume the ideal throughput is the throughput under highest power cap, 165 Watt, of a server. A challenging aspect is that we need to estimate the impact of a change in power cap on the throughput of a server. We propose the following *throughput predictor*. Suppose that $\hat{p}_i$ denotes current allocated power cap to server i, and that the attained throughput for the server from using the power cap controller is equal to $\tau_i(\hat{p}_i)$. Given the measurements at the current power cap, the *objective* of the *throughput predictor* is to estimate the throughput of the server resulting from allocating a new power cap p_i to the server. In Figure 3.5, we plot the relationship between power cap and throughput of servers for 10 heterogeneous combinations, where each heterogeneous combination consists of four applications that were selected from SPEC CPU2006 benchmarks at random. And in Figure 3.6, 10 homogenous combinations of four PARSEC and SPEC benchmarks are plotted. We observe that homogeneous data is more quadratic while heterogeneous data is more linear. As quadratic model could reduce to linear model, we propose a quadratic model, where the predicted throughput is equal to,

$$\tau_i(p_i) = a_{1,i} + a_{2,i}p_i + a_{3,i}p_i^2, \tag{3.7}$$

where the parameters $a_{1,i}, a_{2,i}$ and $a_{3,i}$ are functions of the workload characterizations of

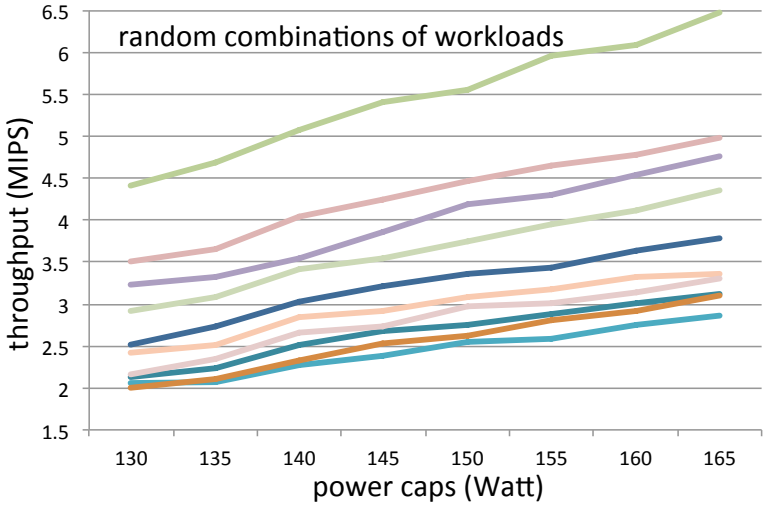

Figure 3.5: Throughput vs. power cap of 10 heterogeneous combinations of SPEC benchmarks.

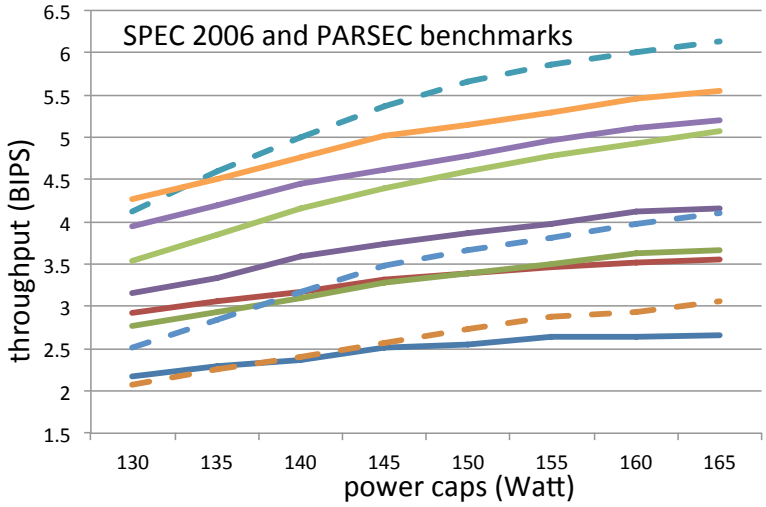

Figure 3.6: Throughput vs. power cap of 10 SPEC (solid line) and PARSEC (dashed line) benchmarks.

server i at the server's current power cap. To predict the throughput, we need to identify these parameters of the model from the observations at the current operating point of server i. To get an insight into the factors that determine the parameters of the throughput-power characteristics, we analyzed a large number of performance counters from off-line characterization data. We have found that the Last Level Cache (LLC) misses is one of the most reliable predictor of the parameters. Figure 3.7 illustrates the relationship between LLC and parameter a_1 from our proposed quadratic model using the characterization data

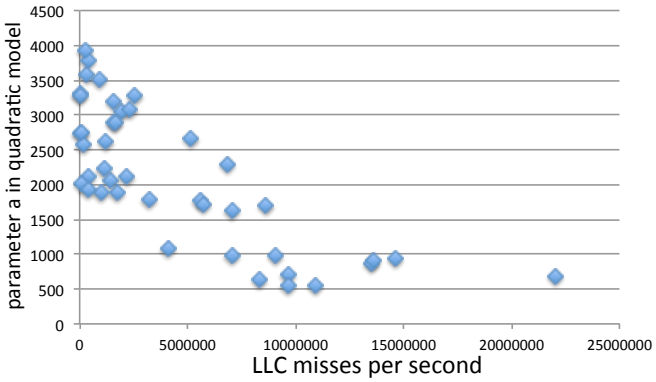

Figure 3.7: Relationship between LLC and the value of parameter ”a” for PARSEC and SPEC 2006 benchmarks.

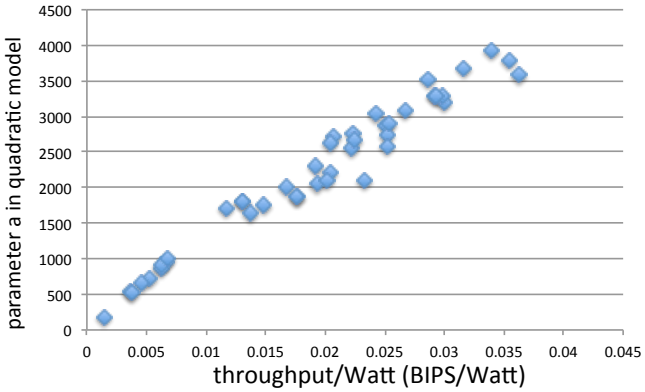

Figure 3.8: Relationship between current throughput/Watt and the value of parameter ”a” for PARSEC and SPEC 2006 benchmarks.

collected from the SPEC CPU 2006 benchmarks [69] and PARSEC benchmarks [9]. The results show a roughly exponential relationship between them. This trend is plausible as LLC misses show memory boundedness [2, 23], and as a result allocating more power caps to memory bound workloads give little improvements to throughput. In addition to the LLC misses, we have found that the current throughput/power ratio, i.e., $\tau_i(\hat{p}_i)/\hat{p}_i$, is a good predictor of the parameters at the setting. Figure 3.8 illustrates the clear linear relationship between the $\tau_i(\hat{p}_i)/\hat{p}_i$ and parameter a_1 using our off-line characterization data. The results show that servers with higher throughput per Watt usually have higher value of parameters. In addition to parameter a_1, we observed that parameter a_2 and a_3 have similar relationships with these two predictors as well.

Our parameter estimator makes use of both $\tau_i(\hat{p}_i)/\hat{p}_i$ and $L\hat{L}C_i$. We experimented with a number of models for the parameters, and we found that the following model gives the best results. The quadratic model parameters a_1, a_2 and a_3 can be modeled as:

$$a_{j,i} = \beta_{j,1} + \beta_{j,2}\frac{\tau_i(\hat{p}_i)}{\hat{p}_i} + \beta_{j,3}e^{\beta_{j,4}\cdot L\hat{L}C_i}, j \in \{1,2,3\} \tag{3.8}$$

where $\beta_{j,1}, \beta_{j,2}$ and $\beta_{j,3}$ are the model coefficients for the current power cap. The coefficients can be easily found through off-line training on subset of workload characterization data.

To evaluate the overall accuracy of throughput predictors, we consider two types of workload combinations on servers: heterogeneous workloads within a server and homogeneous workloads within a server. We simulated 3200 servers which consisted of 1600 servers with heterogeneous workload and 1600 servers with homogeneous workload and we run our proposed optimal power budgeting algorithm with collected data to obtain the optimal performance value as the upper bound of throughput prediction. We compare our predictor in three versions: (i) `quadratic-LLC+TP` which uses quadratic model together with the measurements of throughput, power and LLC as described in Subsection 3.2.2; (ii) `linear-LLC+TP` that uses linear model proposed in previous work [66]; (iii) `linear-TP` just uses the throughput and power; and (iv) `exponential-LLC` that just uses LLC measurements. We also compare against the linear (`previous-linear`) model [64, 27] and cubic (`previous-cubic`) model proposed in previous works [27]. The average absolute error of the predictors are reported in Table 3.2. The results show

prediction method	throughput prediction error
`quadratic-LLC+TP`	1.37%
`linear-LLC+TP` [66]	2.13%
`linear-TP`	2.45%
`exponential-LLC`	2.73%
`previous-cubic` [27]	4.29%
`previous-linear` [64, 27]	6.11%

Table 3.2: Error in throughput prediction for various models.

that our predictor leads to better throughput prediction, and that combining LLC measurements together with throughput and power leads to more accurate results. Both linear and cubic model previously proposed in the literature trail our models in accuracy.

Using Equations (3.6) and (3.7), and given the current $\tau_i(\hat{p}_i)$ and $\hat{p}_i$, it can be shown that the SNP objective can be recast as follows:

$$\max SNP \Leftrightarrow \max \sqrt[n]{\prod_{i=1}^{n} ANP_i(p_i)} \tag{3.9}$$

$$\Leftrightarrow \max \prod_{i=1}^{n} ANP_i(p_i), \tag{3.10}$$

$$\Leftrightarrow \max \prod_{i=1}^{n} \tau_i(p_i)/\tau_i(p_r), \tag{3.11}$$

$$\Leftrightarrow \max \prod_{i=1}^{n} \tau_i(p_i) \tag{3.12}$$

where p_r is the highest power cap. The optimization problem can be formulate as:

$$\begin{aligned}
\text{maximize} \quad & \textstyle\prod_{i=1}^{n} \tau_i(p_i) \\
\text{subject to} \quad & \textstyle\sum_{i=1}^{n} \sum_{j=1}^{r} w_j X_{ij} \leq B_s - np_0, \\
& \textstyle\sum_{j=1}^{r} X_{ij} = 1 \qquad \forall i = \{1 \ldots n\}, \\
& X_{ij} \in \{0, 1\}.
\end{aligned}$$

We observe the similarity between power budgeting formulation and the *multiple-choice knapsack* problem [61]. In the multiple-choice knapsack problem, there are a number of classes, where each class has a few items, each with its own value and weight, and we have to select one item from each class to maximize the total value for the given total weight of the knapsack. Our problem naturally leads to a multiple-choice formulation, where the each server corresponds to a class, and the items within the class correspond to the power cap settings that can be applied to the server, each with its own ANP value ($\tau_i(p_i)/\tau_i(p_r)$) and weight (power cap p_i). For our problem, we slightly modified the algorithm: instead of maximizing the sum of "value", we maximize the product. The multiple-

input : ANP(v) and power for the servers, n, and B_s.
output: Power allocated for every server.
Let V be a vector that holds the total knapsack's value for each possible budget. V is initialized to all zero.
for $i := 1 : n$ **do**
  for $k := B_s : -1 : 1$ **do**
    for $j := 1 : r$ **do**
      $p_i := p_0 + w_j$;
      if $k \geq p_i$ *and* $V(k) \leq v_{ij} \times V(k - p_i)$ **then**
        let $V(k) = v_{ij} \times V(k - p_i)$;
        let $X_{ij} = 1$ and let $X_{il} = 0$ for all $l \neq j$;
      end
    end
  end
end

Algorithm 2: Algorithm for optimal power budgeting.

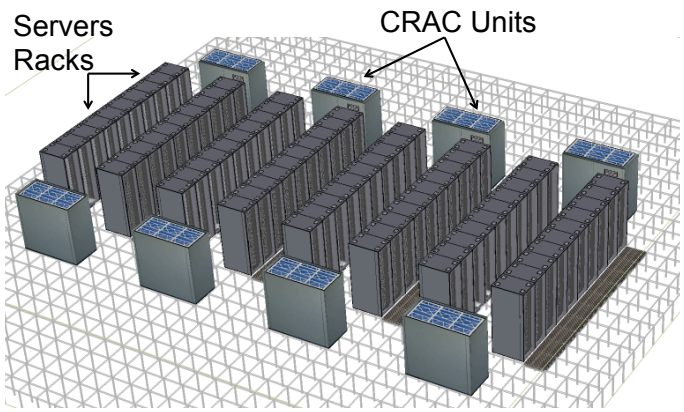

Figure 3.9: The layout of our experimental cluster.

choice knapsack problem is readily solved using dynamic programming. Algorithm 2 provides the details of the dynamic programming algorithm, which has a complexity of $O(nrB_s)$. In a computing cluster with hundreds or thousands of servers, it is easy to envision a server dedicated to carrying out the computations necessary for power budgeting.

3.3 Experimental Results

In our computing cluster configuration, we assume 3200 servers forming 80 U40 racks with 40 servers per rack. To obtain the heat cross-interference coefficient matrix given our experimental cluster layout, we use 6SigmaRoom Lite [26], which is a CFD software tool for simulating cooling characteristics of computing clusters. As illustrated in Figure 3.9, the dimensions of the cluster is $20m \times 13m \times 3m$. The 80 racks are arranged into 8 symmetric rows at the center of the room. In addition, 8 down flow CRAC units are located at two sides of the center. Cold air comes from under the floor through floor grilles between the two front side rack rows. The fans integrated with the racks draw the cold air through servers, which removes the heat generated by the operation of servers. The air heated by servers leaves the racks from the back side and is sucked into the CRAC units at the sides.

The CRAC units extract the heat from the hot air and push cold air back into cluster from perforated tiles on the floor at user specified temperature. We assume a redline inlet temperature of racks is 24 $^{\circ}C$.

To evaluate the trade-off of using matrix-based thermal modeling method, we compare the runtime and accuracy of matrix-based method to CFD simulation. By simulating the heat flow and air flow in the cluster using 6SigmaRoom Lite, we evaluate the inlet temperatures of all the servers to generate the heat cross-interference coefficient matrix. The results show that matrix-based method achieves 94% of accuracy compared to CFD. In terms of runtime, CFD simulation on our experimental computing cluster requires around 10-15 minutes to converge while the matrix-based simulation could finish in few microseconds on the same computer, which is a significant speedup.

The throughput and power estimates for the servers are derived from measurements on a real server executing heterogeneous workload sets. The Linux-based server has a quad-core Intel Core i7 processor and 8 GB of memory. To measure power consumption, the 120 V AC power lines to the server are intercepted and the electric current is measuring using an Agilent 34410A digital multimeter. The total power measurements are read back to the server over USB using the SCPI interface and provided as inputs to the power cap controller. The engagement period of the feedback power cap controller is 1 second. We use the experimental server to construct a database of execution traces of workload sets selected from the SPEC CPU06 [69] and PARSEC benchmarks [9]. For the SPEC CPU06 benchmarks, each workload set consists of four randomly chosen benchmarks, so that all the cores of our server are fully utilized. For the PARSEC benchmarks, all workloads are executed with four-thread configuration. We measured the number of retired instructions per second and LLC misses using the `pfmon` tool library interface. To train our

predictor, we collected a large volume of characterization, where the throughput and LLC are measured for different workloads under different power caps. The database enables us to simulate the impact of different power budgets on a large number of servers in an extremely fast way that preserves the accuracy of results. In particular, each time a new power budget is applied, the power and performance outcomes are computed by reassembling the proper sections of the workload set traces of different servers from the database.

Exp 1. Total Power (Computing+Cooling) Budgeting. In the first experiment, we evaluate our proposed method to calculate the optimal partition between cooling power and computing power of a given total power budget. We consider five total power budgets 0.60 MW, 0.63 MW, 0.66 MW, 0.69 MW and 0.72MW. We execute the self-consistent budgeting algorithm of Algorithm 1 to find the optimal partition of total power budget between computing power and cooling power under several total power budgets. The partitioning of the total power into its computing and cooling components is given in Figure 3.10. From Figure 3.10, we can observe that the cooling power consumption typically takes $30\% - 38\%$ of total power consumption. Another interesting observation from the results in Figure 3.10 is that the proportion of cooling power increases with the increase in total power budget, and that the rate of this increment also increases. Figure 3.11 illustrates the application of the self-consistent budgeting algorithm of Algorithm 1 to the case of 0.72 MW total power budget. The dashed black line gives the power partitions that sum to 0.72 MW. The red points give the intermediate partitions before convergence and the red star shows the self-consistent solution. At the beginning, we initialize the algorithm by computing the minimum requirement of the current computing power distribution using the heat cross-interference coefficient matrix based method. Then we take step 3 to update computing power budget and run our proposed optimal power budgeting algorithm using the new computing power budget and recalculate cooling power. After several iterations

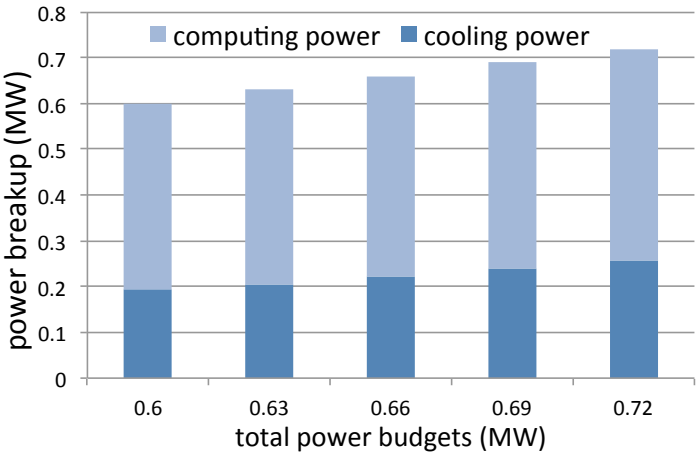

Figure 3.10: The breakup of cooling power and computing power under different total power budgets.

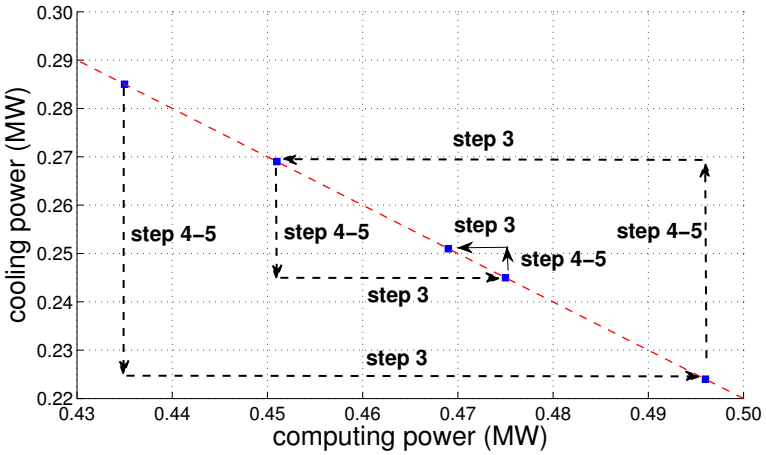

Figure 3.11: Illustration of the self-consistent power budgeting of the algorithm in Algorithm 1 for the case of 0.72 MW.

of step 3-5, we finally reach at the self-consistent solution.

Exp 2. Evaluation of Our Computing Power Budgeting Method for Normalized Performance Improvement. In the second experiment, we evaluate the effectiveness of the proposed knapsack-based optimal power budgeting method given a total computing power budget. We take the self-consistent partitioning solution of computing power

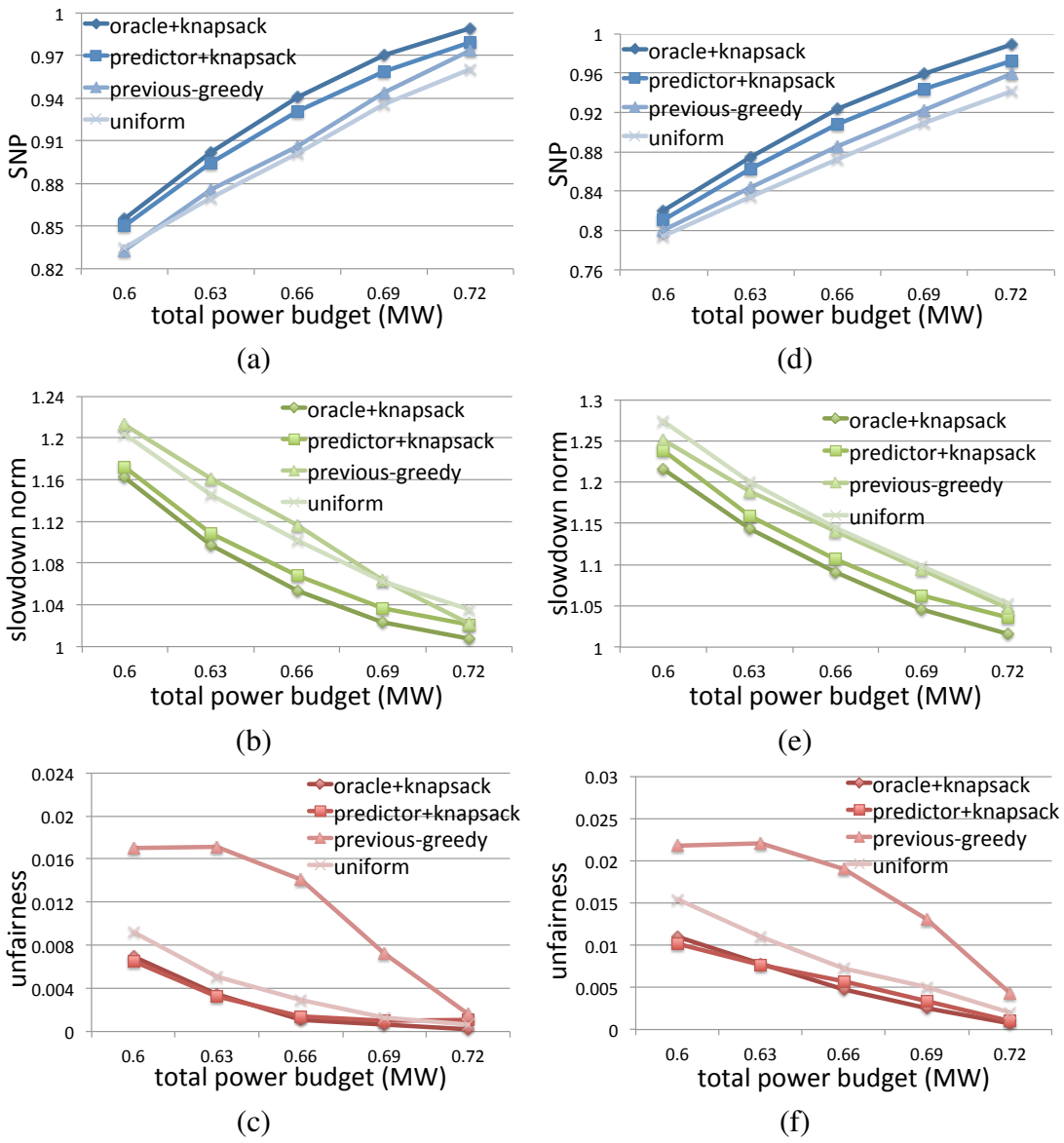

Figure 3.12: (a), (b) and (c) show the SNP, slowdown norm and unfairness respectively of four power budgeting method for heterogeneous workloads across servers, homogenous within server over multiple power budgets, while (d), (e) and (f) show these metrics of the power budgeting method for heterogeneous workloads across servers, heterogeneous within server

from the first experiment as the input of this experiment. We refer to our technique by `predictor+knapsack`. We select (`uniform`) power allocation method as baseline method, where the budget is allocated uniformly among the servers. We also compare against a previously proposed approach (`previous-greedy`) [64], which utilizes a greedy approach for power budgeting, where servers with higher throughput per Watt at the moment of re-calculating the power budget are allocated more power. Finally, we compute an upper bound on the attainable throughput by using the optimal knapsack algorithm on the true throughput and power for each server at the power cap, which are not known during runtime, but can be computed in our simulation environment. We refer to this method by `oracle+knapsack`. We consider two cases and report the performance of each of them:

a) Heterogeneous across servers, homogenous within server: In the first case, the servers execute different workload sets, but the workload set assigned for each server is homogenous, e.g., a PARSEC workload with four threads or four instances of the same SPEC CPU06 benchmark. This is the most common case in modern clusters as administrators prefer to eliminate the interference between workloads arising from execution on the same server. The SNPs of the four power budgeting methods are given in Figure 3.12 (a) for a number of total power budgets. The results demonstrate that our `predictor+knapsack` method consistently outperforms other methods. The performance of slowdown norm and unfairness are given respectively in Figure 3.12 (b) and (c). The results from our predictive method are close to the results from the oracle case. Our method achieves $1.8\% - 3.4\%$ improvement in SNP and slowdown norm over `uniform` and achieves $0.6\% - 4.8\%$ over `previous-greedy`. In terms of unfairness, our proposed method has at best 51.7% improvement over `uniform` and 90.0% improvement over `previous-greedy`. It is even better than `oracle+knapsack` under lower

power budgets.

b) Heterogeneous across servers, heterogeneous within server: In the second case, the servers execute different workload sets, and each workload set on a server consists of different benchmarks (e.g., four instances of different SPEC CPU06 applications). The SNPs of four power budgeting methods are given in Figure 3.12 (d) for five total power budgets. The results show that our proposed method consistently outperforms other methods. The slowdown norm and unfairness are shown in Figure 3.12 (e) and (f). The heterogeneity of workloads within the server causes averaging in characteristics, which leads to less differentiation among the ensemble of servers. Furthermore, the interactions between the workloads within the servers reduce the accuracy of the throughput predictor. This is the reason why the gap between `predictor+knapsack` and `oracle+knapsack` in this case is larger than what is in the first case. Therefore, there might be further room for improvement through better throughput predictors. However, the relative improvements of our proposed method over `uniform` in this case are even better than the first case: SNP and slowdown norm have $2.0\% - 4.1\%$ improvements. Our method has $1.3\% - 2.8\%$ improvements of SNP over `previous-greedy`. Under lower power budgets, the fairness achieved by our proposed method outperforms `uniform` by 33.7%, outperforms `previous-greedy` by 76.7% and even outperforms `oracle+knapsack` by 7.2%.

Both results show that `previous-greedy` has a worse performance than other methods when power budget is low. The reason is that greedy algorithm focuses on throughput, which will undermine its performance in terms of normalized performance. This characterization is clearly shown in the result of unfairness. From the results, our proposed method achieves especially high improvements when power budget is low. Namely, our proposed method will guarantee system performance in under-provision scenario. In

both cases, it is natural to expect that the relative advantages among the methods would disappear when the total power budget is too high or too low. If the total budget is too high, then all servers can afford to run at the highest power cap and throughput irrespective of the method, and similarly when the total budget is too low, then all servers will be forced to the lowest power state.

Exp 3. Evaluation of Our Computing Power Budgeting Method for Power Saving. Another metric to evaluate a power management technique is the power consumption for a target system performance constraint. We take uniform power budgeting method as the baseline algorithm and compare it with our proposed method, previous greedy algorithm and the optimal power budgeting algorithm refered by `oracle+knapsack`, which shows the upper bound of how much power could be saved. The results are given in Figure 3.13. From the results, we can observe that our proposed method consistently outperforms greedy method and save total computing power from 1.3% to 2.5%, while the greedy algorithm shows little improvement or even consume more power than uniform method at lower and intermediate SNP requirements.

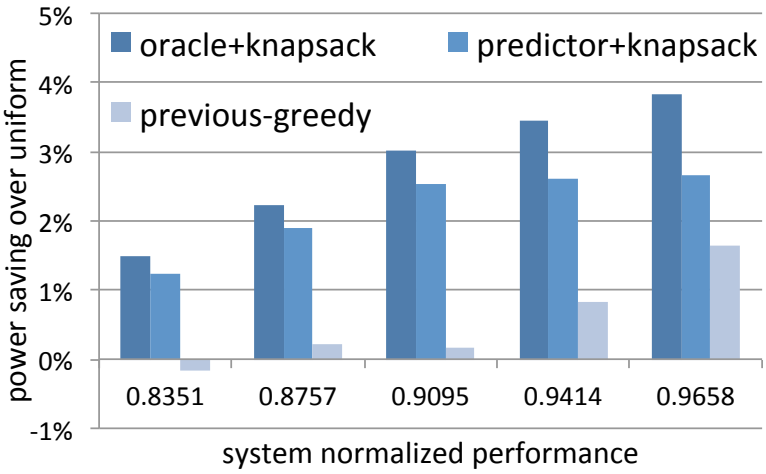

Figure 3.13: power saving percentage over baseline uniform method.

Exp 4. Dynamic Performance Evaluation of Total Power Budgeting Technique. In the fifth experiment, we evaluate our technique's ability to adapt the budget periodically for 3200 servers. In dynamic scenario, when total power budget changes, the cooling power will change and the supply temperature of CRAC units will be set to a new value. Some previous works [56, 72] assume an instant delivery of cooling air from CRAC units to inlet of servers. However, in reality, it takes time for the cooling air to reach servers. In this case, simply assuming a instant arrival of cooling air will lead to a short period of time of undercooling in the cluster. For example, when total power budget increases, both computing and cooling power budget will be increased. After the new budgets applied to cluster, the computing power will increase immediately and the corresponding heat load of servers will increase at the same time. However, the lowered supply temperature has not arrive to the front side of all the servers, which will cause some of the inlet temperature of servers exceeding the red line temperature.

As this latency is highly depended on the model of CRAC units and the layout of computing cluster, we use CFD simulation tool to simulate the air velocity and compute the worst case air travel time for our experimental computing cluster. As the heat loads vary in servers, the air flow velocity at different racks are different. Our simulations consider the lowest cool air velocity for all the server. The result shows that it takes 6.4 seconds for the cooling air travel from CRAC units to the highest server in the rack. When the total power budget is raised, our proposed method apply the new cooling budget first to make sure the temperature decreases to a desired degree. Then, after 6.4 seconds, computing power budget is increased. When the total power budget decreases, the computing power will decrease immediately while the supply temperature of CRAC units will stay low for few seconds. There will be a short period of overcooling, which does not affect the system performance. Re-computing the power caps in a finer granularity does improve the per-

formance of our proposed techniques. However, as the adjusted the CRAC units supply temperature takes 6.4 seconds (in our experimental cluster layout) to reach the inflow side of servers, resolving power caps of servers with a very short interval will potentially lead to the red line temperature being violated. The power cap setting resolve interval should be determined by the actual server room conditions and workload characteristics.

To demonstrate the dynamic performance of our method, we simulate our experimental cluster for 75 seconds, where the total power budget is assumed to be adjusted at the 15th second and the 45th second. Figure 3.14 gives the SNP as a function of time for our proposed method and uniform power budget. And Figure 16 demonstrates the power cap distribution on the servers in the cluster and how does the power caps change with the total power budget. It visualizes the power distributions over different stages. From the figure, we can observe that the servers running diverse workloads are classified based on their workload characteristics and assigned to the different power caps accordingly. We resolve the power budgeting and determine the power caps of each server every 15 seconds. The re-solving is necessary because workloads change their characteristics during runtime, and because the total power budget can also change. In the plot, in the first 15 seconds, servers are initialized at random power caps. At time 15s, a new total power budget of 0.66 MW is imposed and new power caps are applied. At time 30s, we re-calculate the power budgets of all servers using the same total budget of 0.66 MW. This calculation is needed to account for the changes in application characteristics over time. At time 45s, a new budget of 0.62 MW is applied and the new SNP and power caps are calculated, and finally at time 60s, the power budgets are re-calculated. The results show consistent higher SNP for our methods, and its ability to adjust the caps of every server dynamically. The power caps clearly show the reduction in power caps when the total power budget is reduces from 0.66 MW to 0.62 MW.

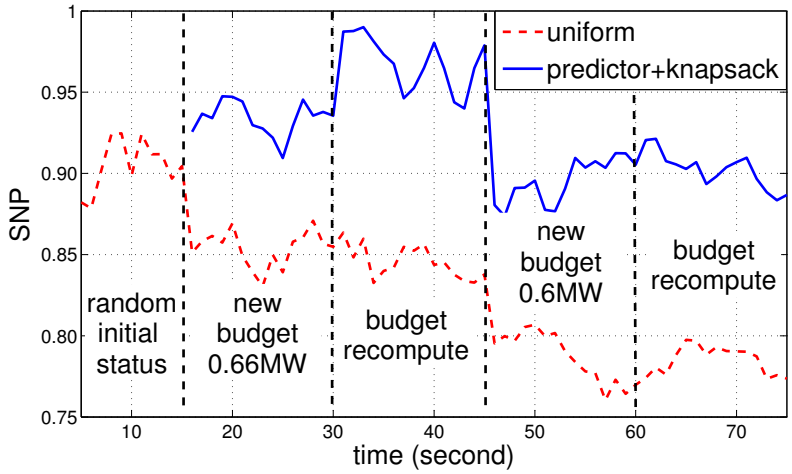

Figure 3.14: SNP over time for proposed method and uniform power budgeting method.

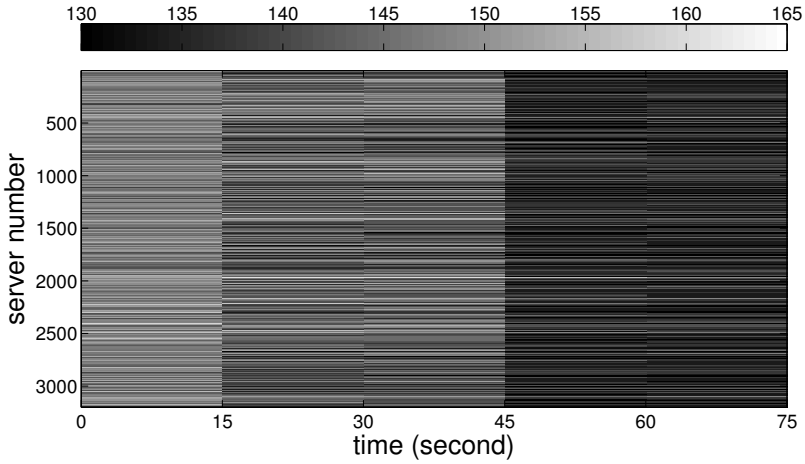

Figure 3.15: Power allocation for each server over time for the SNP schedule plot given in Figure 3.14.

3.4 Summary

In this chapter, firstly, we proposed a self-consistent method to partition the total power budget between the computing and the cooling component of the cluster and speed up the procedure using heat cross-interference coefficient matrix. Secondly, we proposed a method for optimal power budgeting for servers considering the heterogeneity of work-

loads both across and within servers. It is well-known that workloads exhibit different power and performance characteristics depending on their memory or processor boundedness. We leveraged this characterization to devise a computing power budgeting method that allocates power to servers that can efficiently translate their power allocation to improvements in many system performance metrics, such as SNP, slowdown norm and unfairness. During runtime, a power budgeting system has no information about the servers' status beyond their current measurements. Thus, we proposed a throughput prediction method that estimates the changes in throughput as functions of potential changes to allocated power caps. We have demonstrated that our throughput predictor is capable of providing accurate predictions under different power cap and workload characteristics. We have devised an optimal computing power budgeting method based on the multiple-choice knapsack formulation to identify the optimal power allocations for each server such that the SNP is maximized. For the simulated experimental computing cluster, results from our proposed method show 4.1% improvement on the average over previous methods in SNP and slowdown norm. Although we consider SNP as the objective to maximize, the fairness achieved by our method outperforms by 51.7% to uniform method and by 90.0% to greedy method. In addition, we evaluate the performance of our proposed method for power saving and dynamical power management.

Chapter 4

Distributed Computing Power Budgeting

4.1 Introduction

We presented our total power budgeting method in Chapter 3. It is a centralized method that can have difficulty scaling with the size of the computing clusters. Within a very large scale computing cluster, solving the optimal computing power allocation is not fast enough to react to the dynamic status changes. Besides, centralized coordinator compromises the reliability of the system since it introduces a single point of failure.

In this chapter, we propose, *DiBA*, a fully distributed method to solve the computing power budgeting problem under a given computing power budget. In particular, our contributions are as follows:

- We propose *DiBA*, a novel algorithm for distributed power budget allocation to max-

imize the combined utility of the entire cluster subject to a given total power budget. The utility of each computing node is a metric for its throughput that depends on the benchmark of the workload as well as server specifications. In our distributed framework, each server exchanges its decision to increase/decrease its power usage with neighboring nodes in the cluster through rounds of communications. After each round, the local parameters of each node are updated in the form of a state-space model. In this phase, the local actions of neighbors are implemented as a control input. Further, the actions of servers take into account the global constraint that must be satisfied.

- We compare the performance of *DiBA* with a centralized power budgeting scheme in terms of total and per iteration communication/computation time. We note that there is a communication bottleneck at the central coordinator in the centralized method for large clusters. The primal-dual (PD) method also suffers from the same issue since it relies on the same architecture to disseminate information among nodes. In contrast, *DiBA* provides a fully decentralized mechanism which eliminates the role of the central coordinator. As a result, we demonstrate that for large scale clusters with thousands of nodes, the communication time of *DiBA* is substantially less than that of centralized scheme and primal-dual method.

- We also examine the impact of a number of issues on the performance of *DiBA*, including dynamic workloads and dynamic power budgets. We also analyze the impact of topology of communication among the distributed computations, and we show that *DiBA* can provide fast convergence with topologies as simple as a ring.

- We also implement a working prototype of *DiBA* on a real experimental cluster and demonstrates its ability in meeting dynamic total power budget in a fully distributed fashion.

The organization of this chapter is as follows. Section 4.2 motivates the need for distributed power management techniques, and Section 4.3 provides the detailed of our proposed *DiBA* algorithm. Our experimental results are given in Section 4.4. Finally, we summarize our conclusions and directions for future work in Section 4.5.

4.2 Motivation

Large clusters are often constrained by power consumption due to the large operational costs for energy consumption. Furthermore, dynamic constraints such as those arising from demand response energy programs or from variations in utilization require cluster operators to find ways to meet changing total power budget throughout daily operation.

Although current employed power management techniques are effective in their objective of reducing power usage, they have a centralized design. Those techniques require a central controller to measure their workload and throughput and determine the optimal DVFS of processors of active servers. Nevertheless, a centralized approach has the following shortcomings:

- *Fault isolation*: Centralized control creates a single point of failure and also a performance bottleneck. In particular, the malfunction in controller or communication breakdown between controller and local servers renders the SLA violations. To avoid such a scenario in centralized methods, the local servers are set to automatically default to maximum power [46]. We present a decentralized method in which each server determines its action locally. Thus, the failure in one or few servers or the communication breakdown can be mitigated as the overall performance of the system does not hinge on a particular unit.

- *Real-time monitoring*: In centralized power budgeting techniques, the state of each server is measured by a centralized monitoring unit. This introduces additional latency and deteriorates the quality of service (QoS) due to data aggregation and communication delay. In comparison, the decentralized technique obviates the centralized monitoring unit as each computing node measures its performance metrics locally. For variable internet traffic such as search and social networking, the localized performance monitoring provided by the decentralized control approach results in a faster response time to workload and power management.

- *Scalability*: A centralized power management is an inflexible architecture for horizontal scaling, in the sense that adding new racks to the cluster must be complemented with the increase in the communication bandwidth and power management reconfiguration of the central power controller. In comparison, a decentralized architecture facilitates a modular design. Particularly, DiBA can scale to large clusters with distributed communication requirements that are as simple as a ring and that are carried over regular cluster network infrastructure.

4.3 Power Management Methodology

We formulate an optimization problem for maximizing the throughput of a cluster with N server nodes, where we assume a power budget of P for the whole cluster. Thus,

$$\max_{\{p_i\}_{i=1}^N} : \sum_{i=1}^{N} r_i(p_i) \tag{4.1}$$

$$\sum_{i=1}^{N} p_i \leq P, \tag{4.2}$$

$$p_i^{\text{idle}} \leq p_i \leq p_i^{\text{max}}, \quad i = 1, 2, \cdots, N, \tag{4.3}$$

where p_i^{idle} and p_i^{max} denote the power that is used by each server in the idle mode and the maximum possible power usage of server i, respectively. In our formulation, the utility metric $r_i(\cdot) : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ corresponds to the throughput of each server node that can be increased/decreased by adjusting its capacity (e.g., increasing/decreasing the frequency of the processors using DVFS). We note that the throughput of server i as a function of its power usage p_i depends on the characteristic of the workload that is being executed. We defer further details about modeling a proper throughput function to Section 4.4.

Solving the optimization problem in a decentralized fashion requires coordination among servers, since the power consumption of servers are coupled via the constraint (4.2). To achieve coordination among nodes, our distributed algorithm creates a consensus framework for each node.

In this section, we consider two distributed schemes that are structurally different. First, we analyze the more conventional primal-dual decomposition approach and discuss its limitations for practical implementation in large clusters. We then propose an alternative algorithm, *DiBA*, that is fully distributed that outperforms the primal-dual method in terms of convergence speed.

4.3.1 Primal-dual decomposition algorithm

The algorithm that we present here is based on the primal-dual decomposition method which is a well-known scheme for distributed optimization and has been extensively studied in the context of TCP/IP and congestion control in networks with elastic traffic [48]. We also remark that the primal-dual decomposition method has been investigated in [33] in the context of power management of computing clusters. However, we use this algorithm as the benchmark for comparison with the performance of *DiBA* we present in

subsequent subsections. Hence, we give a detailed analysis of the primal-dual method in the following.

We describe the simple structure of the asynchronous primal-dual algorithm when it is characterized for the problem. Consider the dual form of the optimization problem that is formulated in (4.1)-(4.3),

$$\min_{\lambda \geq 0} g(\lambda) \tag{4.4}$$

where λ is the dual variable, and $g(\lambda)$ is defined as

$$g(\lambda) \stackrel{\text{def}}{=} \sup_{\{p_i\}_{i=1}^N} \sum_{i=1}^{N} r_i(p_i) + \lambda \left(P - \sum_{i=1}^{N} p_i \right).$$

Let $\lambda^\star$ be the optimal solution of the dual optimization problem (4.4). Given the optimal Lagrangian multiplier $\lambda^\star$, each server can compute its optimal power budget $p_i^\star, i \in \{1, 2, \cdots, N\}$ via solving the following *local* optimization problem

$$p_i^\star = \arg \max_{p_i^{\text{idle}} \leq p_i \leq p_i^{\max}} r_i(p_i) - \lambda^\star p_i.$$

In this formulation, $\lambda^\star$ can be interpreted as the price of using the shared power budget P. In particular, in the case that power demands exceed the amount of shared power budget P, the central coordinator increases the price $\lambda^\star$ to reduce the demand.

A simple scheme to compute the optimal value of $\lambda^\star$ can be achieved by an iterative update rule in which the price λ is computed by a coordinator unit, and then communicated to all local servers. In particular, the coordinator updates the price using the following

update formula; see [48],

$$\lambda^{t+1} = \left[\lambda^t - \epsilon\left(P - \sum_{i=1}^{N} p_i^t\right)\right]^+, \tag{4.5}$$

where $[z]^+ \stackrel{\text{def}}{=} \max\{0, z\}$, $\epsilon > 0$ is a small step size.

By having the Lagrangian update λ^{t+1}, the i-th server can compute its power consumption $p_i^\star$ as follows

$$p_i^{t+1} = \arg \max_{p_i^{\text{idle}} \le p_i \le p_i^{\max}} r_i(p_i) - \lambda^t p_i. \tag{4.6}$$

The structure of the primal-dual decomposition algorithm for the optimization problem is described in Algorithm 3.

We make few remarks about Algorithm 3.

1. Although the primal-dual decomposition method is scalable, it is not efficient in terms of communication requirements. For large clusters, the central coordinator must be equipped with a high communication bandwidth to accommodate communication with all servers. Moreover, since network switches support a limited number of servers, for a large cluster of thousands server nodes, multiple network

1. initialize $p_i^0, i = 1, 2, \cdots, N$, λ^0, and $\epsilon \in \mathbb{R}_{\ge 0}$;
2. **for** $t = 1, 2, \cdots$ **do**
   At the central coordinator:
   3. update λ^t according to (4.5);
   At the i^{th} server:
   4. update p_i^t according to (4.6) ;
   5. transmit p_i^t to the central coordinator ;

end

Algorithm 3: Primal-dual algorithm.

switches must be employed which is costly and adds additional latency to the network.

2. The existence of a central coordinator in the primal-dual approach is also undesirable since it creates the SPF problem. In the case of failure of the central coordinator, the PD power management technique fails. Therefore, PD does not provide the robustness that is necessary to power management in computing clusters.

3. The PD algorithm we considered is synchronized in the sense that servers update their power usage information simultaneously. This level of synchronization between different servers is usually provided through Network Time Protocol (NTP) in computer systems [55].

4.3.2 *DiBA*: Distributed Power Budget Allocation Algorithm

In this section we propose and analyze our proposed distributed algorithm called *DiBA* to control the power consumption of each computing node to maximize the throughput of the cluster subject to total power budgets. We can qualitatively describe *DiBA* from two different perspectives. From a game design perspective, *DiBA* is a gradient algorithm that achieves the pure Nash equilibrium corresponding to the state based game defined by the optimization [41].

From the view point of distributed optimization, DiBA is a consensus based optimization algorithm. In particular, DiBA establishes the consensus on the estimation from the coupling constraint (4.2) by augmenting the local utility function of each server node with a barrier (penalty) function which when minimized/maximized, gradually reduces the difference between local variables of computing nodes.

Naturally, to create such a consensus, the communication graph between servers needs to be connected. The connectivity requirement is naturally facilitated in clusters, where clusters can communicate using the cluster's network infrastructure.

For each i-th server, we introduce the estimation e_i term for the coupled constraint (4.2), i.e.,

$$e_i \sim \sum_{i=1}^{N} p_i - P. \tag{4.7}$$

During each round of ***DiBA*** algorithm, the i-th server transmits its local measurement and also receives the local variables of all its neighbors. Specifically, lets $\hat{e}_{i\to k}$ ($\hat{e}_{i\leftarrow k}$) denote the estimates from (resp. to) the i-th server to (resp. from) the k-th server, where $k \in \mathscr{S}_i$ and $\mathscr{S}_i \subseteq \{1, 2, \cdots, N\}$ denotes the set of servers that are connected to the i-th server.

By aggregating these estimates, the i-th server updates its state variable as well as its estimate from the coupled system constraint. Specifically, based on a state-space control model, we have the following update rule during the t-th iteration of DiBA algorithm,

$$p_i^{t+1} = p_i^t + \hat{p}_i^t \tag{4.8}$$

$$e_i^{t+1} = e_i^t + \hat{p}_i^t + \sum_{k\in\mathscr{S}_i} \hat{e}_{k\leftarrow i}^t - \sum_{k\in\mathscr{S}_i} \hat{e}_{i\to k}^t, \tag{4.9}$$

where p_i^t is the power state of the i-th server, and $\hat{p}_i$ is the power control input that takes

values from the action space

$$\mathscr{A}_i(p_i^t, e_i^t) \stackrel{\text{def}}{=} \Bigg\{ (\hat{p}_i, \hat{e}_i) : p_i^t + \hat{p}_i \in [p_i^{\text{idle}}, p_i^{\text{max}}], \hat{e}_{i\to k} < 0,$$

$$e_i^t + \hat{p}_i^t - \sum_{k\in\mathscr{S}_i} \hat{e}_{i\to k} < 0 \Bigg\}.$$

The action space guarantees that the controller inputs $(\hat{p}_i, \hat{e}_i)$ in the state-space model does not violate the constraint on the maximum power and idle power of each computing node, and also satisfy the coupled constraint (4.2). For each node, we now define a local utility function for node $i = 1, 2, \cdots, N$ as follows

$$R_i(p_i^t, \{e_j^t\}_{j\in\mathscr{N}_i}) = r_i(p_i^t) - \eta \sum_{j\in\mathscr{S}_i} \log(-e_j^t), \tag{4.10}$$

where $\eta \in \mathbb{R}_{\geq 0}$. The second term $\sum_{j\in\mathscr{S}_i} \log(-e_j^t)$ is a penalty factor that plays a role on the cost function only when $e_j^t > 0$, which translates into the constraint violation condition $\sum_{i=1}^N p_i^t > P$.

We note that the speed of reaching the consensus is controlled by the value of η. By choosing a small positive value for η in (4.10), we can assure that the solution of *DiBA* is close to the optimal solution of Problem 1. But, choosing a small η will cause numerical difficulty which may lead to slower convergence.

In addition, the barrier function in (4.10) enforces the constraint to be satisfied. By employing this barrier function, we thus assure that the resultant answer from the above algorithm fulfills the system power constraint. We now have all the machinery to describe *DiBA*. The steps of *DiBA* are outlined in Algorithm 4.

It is imperative to calculate the communication overhead introduced from decentralizing the power management controller. The communication overhead of each control epoch can be approximated by the product of the number of communications required during each round and the number of iterations to converge. In a cluster with N servers,

1. For primal-dual decomposition algorithm, each computing node sends one packet and then receives one packet from the central coordinator during each iteration. Thus, total number of $2N$ packets are communicated per iteration.

2. For *DiBA*, each computing node sends packets only to its local neighbors which in case of a ring communication topology is two and thus results in the total number of $2N$ packets that are communicated in parallel among the nodes. For a general graph, dN communications per iteration is required where d is the average degree of each node.

1. initialize $\eta \in \mathbb{R}_{\geq 0}$ and a non-increasing sequence $\epsilon_i^t \in \mathbb{R}_{\geq 0}$, $i = 1, 2, \cdots, N$. Choose p_i^0, e_i^0 such that $\sum_{i=1}^N p_i^0 \leq P$;
2. **for** $t = 1, 2, \cdots$ **do**
   At the i^{th} **server:**
   3. compute

$$\hat{p}_i^t = \beta^t \left(-\epsilon_i^t \frac{\partial R_i}{\partial \hat{p}_i}\bigg|_{\hat{p}_i=0} \right);$$

$$\hat{e}_{i\to j}^t = \beta^t \min \left(0, -\epsilon_i^t \frac{\partial R_i}{\partial \hat{e}_{i\to j}}\bigg|_{\hat{e}_{i\to j}=0} \right);$$

   where β^t is computed by backtracking to the constraint $(\hat{p}_i^t, \hat{e}_i^t) \in \mathscr{A}_i(p_i^t, e_i^t)$, and ϵ_i^t is the gradient step size.
   4. update p_i^{t+1}, e_i^{t+1} according to (4.8)-(4.9) ;
end

Algorithm 4: Distributed Power Budget Allocation (*DiBA*).

Although it may appear that *DiBA* has the same communication complexity as the primal-dual method, in practice it has a lower communication latency and packet loss. In particular, if we consider the network overhead in terms of the bottleneck of communication topology, the central coordinator in the primal-dual decomposition method must communicate with all N computing servers which has a degree of N. In comparison, the degree of each node in *DiBA* is a small fixed number (e.g., two in case of ring topology), the packets from the nodes to their neighbors proceed in parallel. As a result, packet loss is less likely to happen in *DiBA* architecture. We will show in the next section that *DiBA* also shows better performance compared to the primal-dual decomposition method in terms of communication latency.

4.4 Numerical Evaluation

4.4.1 Setup

Workloads: To evaluate the performance of our distributed power management architecture, we select 10 HPC benchmarks. Eight benchmarks from NAS Parallel Benchmarks (NPB) suite [3] (`BT`, `CG`, `EP`, `FT`, `IS`, `LU`, `MG`, and `SP`) and two benchmarks from High Performance Computing Challenge (HPCC) benchmark (`HPL`, `MPI-RA`) and the description of each benchmark is listed in Table 4.1.

Infrastructure: We evaluate *DiBA* using numerical simulations and using a real server cluster. The numerical simulations enable us to study *DiBA*'s performance for large-scale clusters, while the real cluster implementation demonstrates a proof-of-concept for *DiBA*.

name	description
BT (NPB)	Block Tri-diagonal solver
CG (NPB)	Conjugate Gradient
EP (NPB)	Embarrassingly Parallel
FT (NPB)	discrete 3D fast Fourier Transform
IS (NPB)	Integer Sort
LU (NPB)	Lower-Upper Gauss-Seidel solver
MG (NPB)	Multi-Grid on a sequence of meshes
SP (NPB)	Scalar Penta-diagonal solver
HPL (HPCC)	High performance Linpack benchmark
RA (HPCC)	Integer random access of memory

Table 4.1: List of the selected benchmarks and corresponding description.

1. **Server cluster infrastructure**: Our cluster consists of 12 Dell PowerEdge C1100 servers, where each server has two Xeon L5520 quad-core processors, 40 GB of memory and 10 Gbe network controller. The servers are connected to a Mellanox SX1012 10Gbe switch. The frequency of each processor can scale from 1.60 GHz to 2.27 GHz. To collect performance counters values from all servers, the `perfmon2` [22] tool is used. We created a power measurement appartus to measure the supplied AC current to each server and its power consumption correspondingly. To enforce the local power targets for the individual servers, we implement a software feedback controller on each server. The DVFS-based controller adjusts the DVFS up or down according to the difference between the power target and the current power consumption, with positive difference triggering an increase in DVFS, while negative differences triggering a decrease in DVFS [13]. The controller enables us to apply the local constraints on each server as derived from the *DiBA*.

2. **Simulation infrastructure**: To evaluate the performance and overhead of *DiBA* on large clusters, we have developed a simulation tool as follows. We first run each workload in Table 4.1 on one of our servers and collect the throughput and power consumption p_i of ith workload under various DVFS levels, and then fit a throughput function $r_i(p_i)$ on the attained data. The power consumed by each workload has

a range that is determined by the lowest and highest DVFS levels. The throughput functions are then used to emulate the behavior of the applications running on the server nodes. To simulate a cluster with arbitrary number of servers N, we draw the throughput functions from a uniform distribution such that each server hosts at least one type of workload and such that the entire server cluster is fully utilized. To mitigate the effect of randomness in our simulation results, we average the convergence results we obtain for *DiBA* over 10 trials. The network architecture of our simulation environment is based on a two-tier star topology. Each rack consists of 40 servers which are connected with one top-of-rack 10 Gbe Ethernet switch. Further, all racks are connected via a higher-layer core switch. We model the network traffic using queuing systems in our simulation, where the network service times are derived from measurements on our real cluster.

Algorithm Topology: For the primal-dual method, each computing node is connected with the central coordinator as shown in Fig. 4.1. Furthermore, for *DiBA* algorithm, we adopt a ring communication network as shown in Fig. 4.1. Note that the algorithm communication topology runs on top of our cluster network, which naturally facilitates any two servers to communicate.

Throughput Estimation: The throughput function $r_i(p_i)$ of each server is dependent on the nature of the workload. Moreover, we create a framework in which the local controller learns the throughput function $r_i(\cdot)$ on-the-fly as the workload changes on the server. Specifically, we can learn the throughput function of each workload in our benchmark suite of Table 4.1 by applying different DVFS power levels and subsequently measuring the resultant throughput value. We then interpolate a quadratic throughput function. Fig. 4.2 illustrates a number of throughput functions for our workloads on our servers. This

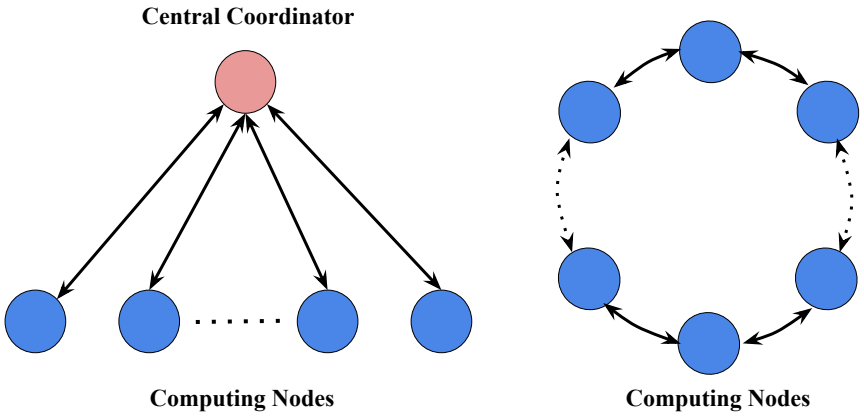

Figure 4.1: The communication topology of the decentralized algorithms. Left: Primal-Dual Decomposition. Right: *DiBA* algorithm.

process is repeated for each workload to derive its throughput function when it is executed on a server. As we observe from Fig. 4.2, the throughput functions for all the benchmarks are concave in power which justifies the use of convex optimization toolbox `CVX` in our simulations.

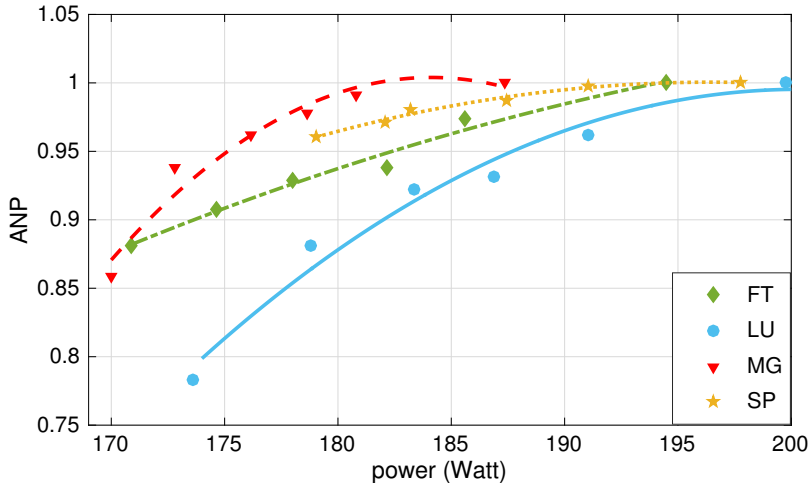

Figure 4.2: The normalized throughput function of 4 workloads.

System Performance metrics: To ensure fairness in our power allocation scheme, we normalize the throughput of each workload by its peak value of r_i^{max}. The application normalized performance (ANP) of each server is computed, where ANP is the ratio of ideal runtime to actual runtime of the workload on the server [77]. In our case, the ANP of application i under power budget p_i is defined as the ratio of actual throughput to its peak throughput:

$$\text{ANP}_i(p_i) = \frac{r_i(p_i)}{r_i^{max}}$$

Then to quantify the performance of the entire cluster, the system normalized performance (SNP) is computed as the arithmetic mean of the ANPs of all M workloads that are running on the cluster. That is,

$$\text{SNP} = \frac{\sum_{i=1}^{M} \text{ANP}_i(p_i)}{M}.$$

We next describe the results from our simulation infrastructure in Subsection 4.4.2.

4.4.2 Simulation results

To evaluate the performance, scalability and resiliency of DiBA, we organize the following experiments.

1. Static experiment of allocating power budget to maximize the system's performance.
2. The scalability of the *DiBA* on clusters with large number of nodes, considering the communication and computation overhead.
3. Dynamic simulations that evaluate how *DiBA* reacts to the power budget reallocation

and workload dynamics.

4. Experiments that evaluate the effect of communication topology of the distributed computations of *DiBA*.

1. Static power budgeting results: In this experiment we demonstrate the value of power budget algorithms that use the utility functions of the servers to maximize performance subject to a total power budget. We simulate 1000 servers that are serving 1000 instances of the benchmarks under total power budgets that range from $P = 166$ kW to $P = 186$ kW with the incremental step size of 4 kW. The results of our decentralized algorithms are compared against the method of allocating power budget uniformly. The SNP results under each total power budget are given in Fig. 4.3. The primal-dual decomposition algorithm improves the SNP by 14.7% over uniform power allocation in average, while *DiBA* achieves 14.5% improvement. The performance gap between uniform and *DiBA* shrinks from 22.6% to 8.2% depending on the amount of total power allocated. Clearly, when the total power budget is sufficiently large, all servers receive enough power for their work-

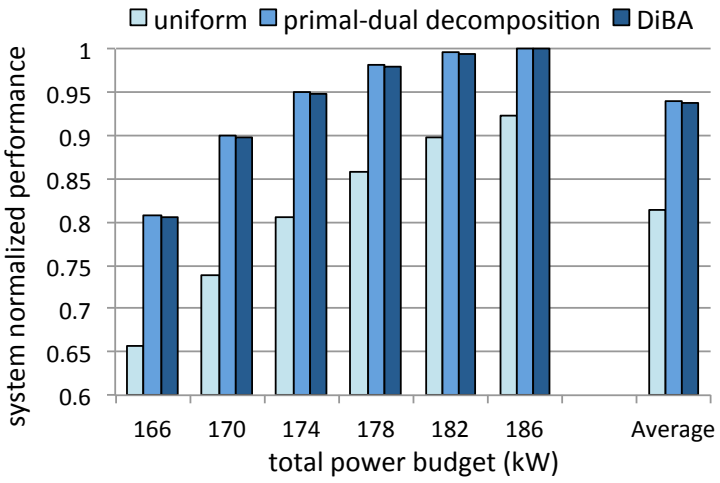

Figure 4.3: The system normalized performance of 1000 servers under different power budgets.

loads, and hence the performance differences between *DiBA* and uniform power allocation diminishes.

2. Scalability: To show the advantage of *DiBA* for large cluster sizes, we compare its performance to that of the centralized approach and primal-dual method. For centralized method, we use `CVX` toolbox [31] to solve the optimization problem formulated in Eqs. (4.1)-(4.3). Specifically, in the centralized method, the computing servers transmit their utility functions to the centralized power management unit to calculate the optimal allocation of the total power budget across all servers. Then the power budget of each server is sent back and applied by the servers. By decentralizing the power budgeting unit, the utility functions are optimized locally and the optimization is accelerated. However, to assess the optimization performance, the communication latency between of each iteration has to be taken into account. In this experiment, we quantify the runtime of each algorithm with different number of server nodes. The average latency of reading and write a packets to TCP sockets between two nodes in our cluster is measured to be approximately 200μs and 10μs respectively. We used these results as the service times in our network queuing model, which is used to model the communication time at the central coordinator in the centralized and primal-dual methods. In the 'uplink' phase, when nodes transmit their packets to the central coordinator, the packets arrival time are drawn from the Poisson distribution with average inter-arrival time of 200μs. In the 'downlink' phase, *i.e.*, transmitting from the central coordinator to the nodes in the centralized and primal-dual methods, the communication time is assumed to be the total time to send all the packets in a serial fashion. In contrast, in *DiBA* architecture, each node only communicates with two neighbor nodes in parallel, thus the communication time for each iteration is time to send and receive a packet.

We compute the runtime of each algorithm with different number of nodes and break

# of nodes	centralized		primal-dual		*DiBA*	
	Comp (ms)	Comm (ms)	Comp (ms)	Comm (ms)	Comp (ms)	Comm (ms)
400	432.76	86.25	1.47	517.52	0.07	28.00
800	477.25	170.37	1.44	1022.22	0.11	26.20
1600	693.40	339.83	1.41	2038.99	0.43	28.00
3200	1041.50	675.95	1.55	4055.70	0.80	27.20
6400	1882.52	1362.50	1.58	8175.00	1.78	28.40

Table 4.2: The breakdown of algorithm runtime with different number of server nodes.

the total runtime between computation time and communication time in Table 4.2. We take the result from `CVX` solver as the baseline and set the convergence condition of the primal-dual decomposition algorithm and *DiBA* as achieving 99% of the base line performance. In particular, the initial power setting for *DiBA* is set as uniform allocation. The convergence time for *DiBA* is the first time that the utility function obtained from *DiBA* satisfies the following inequality

$$\frac{|\text{Optimal Utility} - \text{Utility of } \textit{DiBA}|}{|\text{Optimal Utility}|} < 0.01, \tag{4.11}$$

where the optimal utility is the utility that is computed by solving (4.1) centrally. The convergence of PD algorithm is defined similarly.

The total computation and communication time of the primal-dual decomposition and *DiBA* are computed by taking into account the number of iterations required for each algorithm to converge. The communication time of each iteration is modeled by our simulation network architecture and the computation time of each iteration is the real measurement from our simulation system. *DiBA* is completely distributed and all communications and computations are assumed to be carried out in parallel for each iteration. For primal-dual, the computation of all the nodes are done in parallel but need to be added to the computation time of the central coordinator.

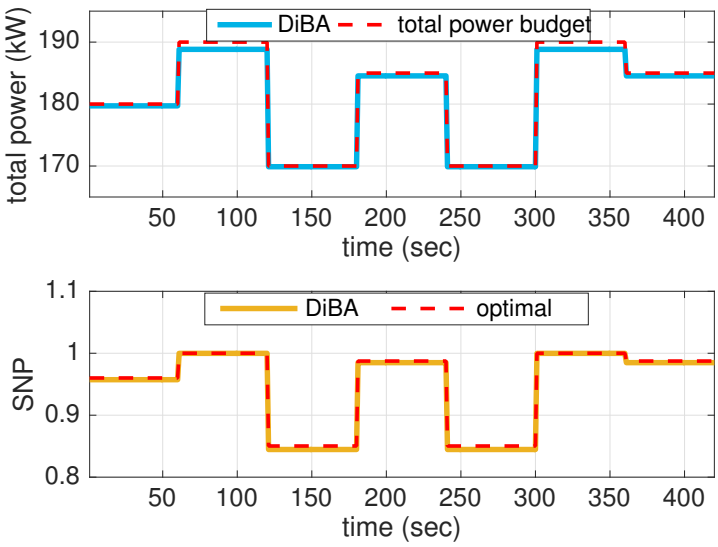

Figure 4.4: Dynamic simulation of total power budget reallocation.

From Table 4.2, we observe that PD method improves upon the centralized power allocation strategy in terms of computation time. This is due to the fact that in the PD method, the optimization problem is solved in a decentralized fashion. Nevertheless, in the overall performance, the PD method performs poorly when the communication time is taken into account. As we mentioned in Section 4.3.1, this increase in latency is due to the communication bottleneck at the central coordinator.

In contrast, we observe that the communication time of *DiBA* on a ring network increases only slightly. Consequently, *DiBA* outperforms both the PD and the centralized method in the overall run-time. *DiBA* also outperforms both the primal-dual and centralized method in the computation time for small to mid range size clusters.

3. Dynamic simulation results: In the previous experiment, we considered a static power budget. Here, we examine a dynamic case where the power budget changes every minute during operation as possibly the case in demand-response programs. We demonstrate how

DiBA re-allocates the power budget. Specifically, we evaluate *DiBA* for cluster of $N = 1000$ servers with dynamic power budget in Fig. 4.4. We observe that *DiBA* provides a near optimal performance without power budget violation. To demonstrate a more detailed response of *DiBA* when there is an adjustment in the total power budget, we consider cases where the power budget decreases/increases between $P = 190$kW and $P = 170$kW power levels as depicted in Fig. 4.5 and Fig. 4.6, respectively. In both cases, *DiBA* immediately adapts to the new power budget.

DiBA is also adaptive to workload variations on servers. *DiBA* dynamically recomputes the power usage of each server as the workloads change on the servers. We simulate a cluster with $N = 1000$ servers at a fixed total power budget of $P = 180$kW. The servers host different instances of the testing workloads that are presented in Table 4.1. After a workload is completed, a new workload is randomly drawn from the workload pool in Table 4.1 and is launched on the free server. We simulated this process for 80 minutes. The resulting power consumption and SNP is given in Fig. 4.7. We observe that despite variations in the server's workload, the SNP metric of *DiBA* is close to optimal. Furthermore, our simulation has shown that under variable workload scenario, the total power consumption is strictly below the power limit.

We now analyze a scenario in which the utility of a single server node undergoes an abrupt change in its utility function. This scenario can occur for example when a node serves a workload and begins processing a new workload from a different benchmark. Under this circumstance, it is interesting to observe the effect of workload changes in a single server on the power states of other servers in the network.

First we examine how the estimation e_i of the affected node propagates through the ring network. More precisely, we consider a ring network of size $N = 100$ nodes. Furthermore, we assume that the coefficients of the quadratic utility function of the node $i = 50$

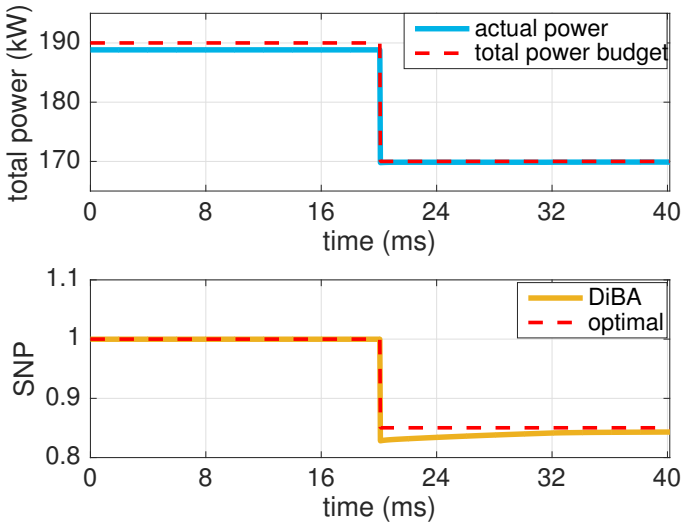

Figure 4.5: Detailed simulation of total power budget reallocation when total power budget drops from high to low.

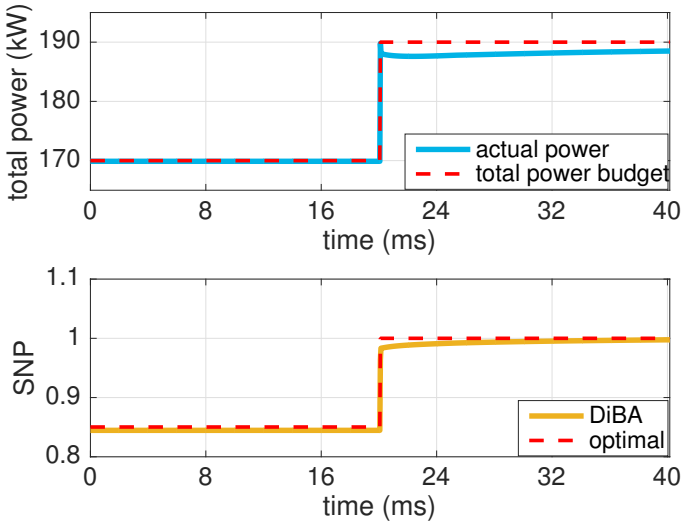

Figure 4.6: Detailed simulation of total power budget reallocation when total power budget jumps from low to high.

undergoes a sudden change. For this scenario, the estimation vector $\mathbf{e} \stackrel{\text{def}}{=} (e_1, \cdots, e_N)$ is shown in Fig. 4.8 at different phases of *DiBA* algorithm. As shown in Fig. 4.8 initially, all nodes in the network, except for the perturbed node, have negligible error estimations $e_i, i \neq 50$. After few iterations, the estimation error propagates through the network while the magnitude of the absolute error decreases. More interestingly, as depicted in Fig. 4.9,

after convergence to the new equilibrium point, only few nodes in the vicinity of the perturbed server node need to adjust their power consumption states. More specifically, Fig. 4.9 shows the difference between power usage before and after utility perturbation at the node $i = 50$. It can bee seen that fluctuation in the utility function has a desirable *local* effect on power consumptions of computing nodes.

4. The effect of algorithm communication topology: The communication topology of the distributed algorithms can be optimized to improve their convergence. A ring topology is particularly ideal for *DiBA* due to its low degree and symmetry. In practice, to guarantee the connectivity of the distributed algorithm in the case of multiple node failures, the ring topology must be equipped with a few chords. To investigate the effect of connectivity of the distributed computation on the convergence speed of *DiBA*, we created 100 instances of connected Erdös-Réyni random graphs [10], where the number of computing nodes is fixed at $N = 100$. In the Erdös-Réyni random graph model, a graph is chosen uniformly at random from the set of all possible graphs with N nodes and a certain number of edges. Note that Erdös-Réyni random graph model is degree homogeneous since its *degree distribution* (i.e., the probability of a node to have a certain degree) decays exponentially fast. In our simulation experiment, the number of edges varies which results in graphs

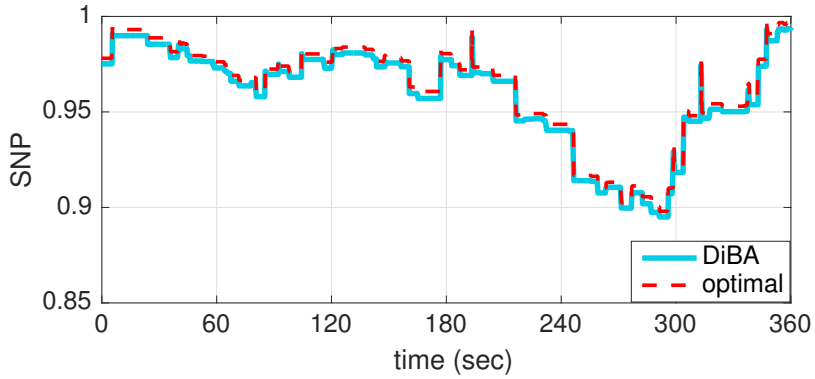

Figure 4.7: The performance of *DiBA* on a cluster with dynamic workloads.

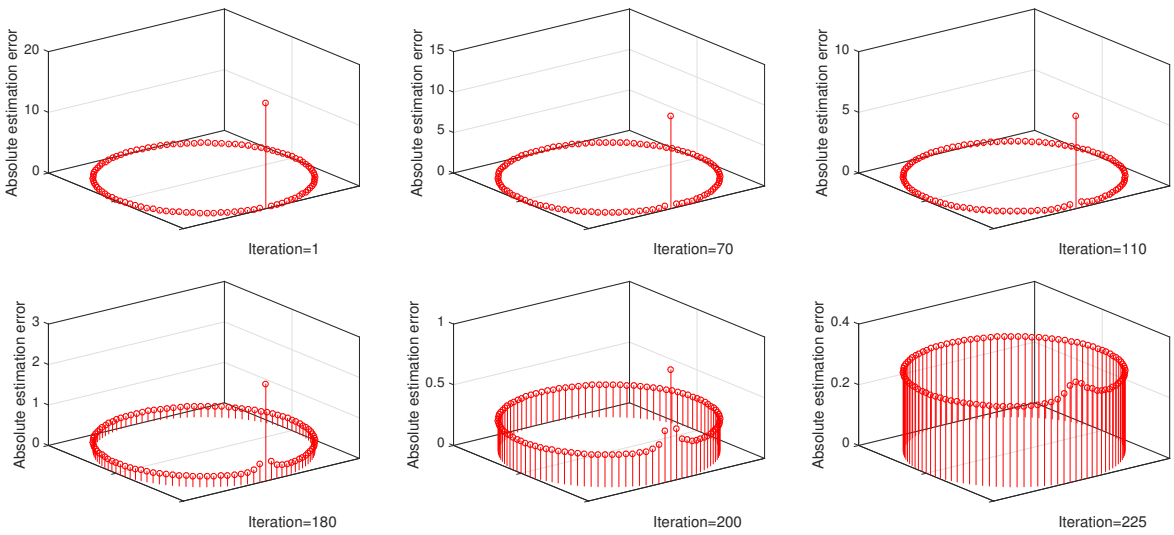

Figure 4.8: The *absolute* estimation error of server nodes in the case of utility change of node $i = 50$.

with different average degrees per computing node.

For each random graph, we have computed the average degree and the number of iterations. The termination criterion for the iterations is defined to be the time at which *DiBA* achieves 99% of the optimal utility value (cf. Eq. (4.11)). The result of this analysis is shown in Fig. 4.10. We observe that *DiBA*'s convergence time has strong correlation with the average degree of connectivity. The appropriate setting for the average communication degree is then a choice of the system administrator, who can assess the required fault tolerance and adjust the topology of the distributed computation accordingly. It is also worth mentioning the convergence time of *DiBA* can be adjusted by tuning the free parameters ϵ_i^t as well as μ in the structure of Algorithm 4.

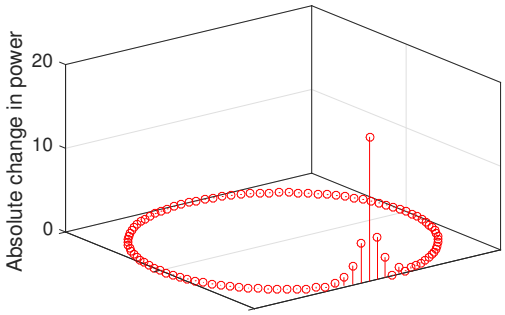

Figure 4.9: The *absolute* change in the power consumption states $(p_1, \cdots, p_N)$ after settling at the new equilibrium.

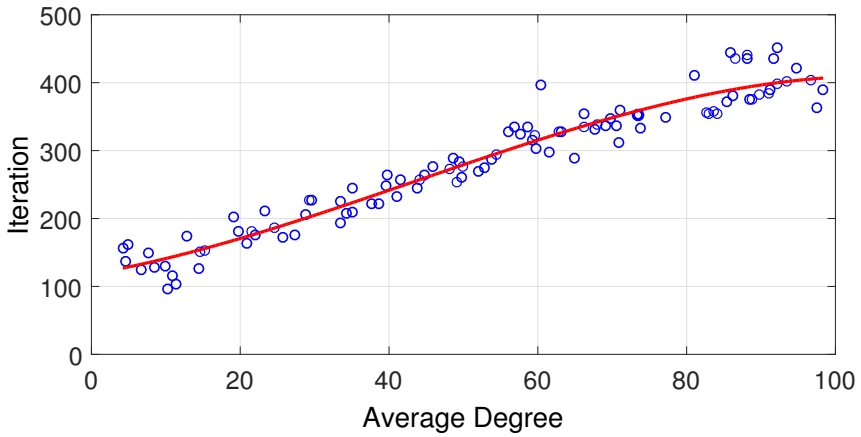

Figure 4.10: The iteration numbers of 100 samples of Erdös-Réyni random graphs with $N = 100$ versus the average degree. The red line shows the 3rd order polynomial regression on the samples.

4.5 Summary

We studied the problem of utility maximization in large-scale server clusters with the power budget constraint. To improve the scalability of power budgeting system, we proposed *DiBA*, a framework to compute the optimal power usage of each node in a fully distributed way. In this framework, each node must exchange its power consumption state to the neighboring nodes in the cluster. We investigated various performance metrics associated with *DiBA*. *DiBA* finds the optimal power allocation as well as the state of art

centralized method, while outperforms the uniform power allocation by 14.5% in average.

In particular, we showed that *DiBA* outperforms the centralized power budgeting scheme as well as the primal-dual algorithm in terms of computation/communication time. We simulated clusters with up to 6400 nodes and demonstrate that *DiBA* always converges in milliseconds which is 272$\times$ faster than centralized method for large scale clusters. The fast convergence of *DiBA* is particularly ideal for dynamic workloads as well as dynamic power budget with fast time scales.

Chapter 5

Thermal-aware Computing Cluster Layout Planning

5.1 introduction

The planning of homogeneous computing cluster is relatively simple as servers are all identical in their configurations; thus, there is no inherent advantage from changing the locations of the racks as the servers will have the same power consumption behavior irrespective of their locations. In homogeneous computing clusters, it is the allocation of workloads to the servers that only determines the spatial power distribution inside the cluster, which consequently determines the thermal characteristics and the cooling power.

However, as is motivated in Chapter 2, modern computing clusters are heterogeneous in nature. The heterogeneous makeup helps clusters cater to workloads with different characteristics (e.g., transactional, batch, numerically intensive, etc) by matching workloads with the right platforms so that the target metrics (e.g., performance and energy

consumption) can be improved. Another reason for heterogeneity arises from multiple replacement, upgrades and the deployment of more cost-efficient systems that become available over time [4].

We observe that heterogeneous computing clusters provide an interesting opportunity to plan their facilities in a way to reduce cooling power. Heterogeneous servers have different power specifications, and thus, the spatial positions at installment will lead to inherent thermal characteristics in the cluster. Thus, we can reduce cooling power by carefully laying out the racks of heterogeneous servers during the planning phase of clusters. We believe the contributions are as follows.

- We formulate the problem of rack layout for planning of heterogeneous computing clusters, where the goal is to identify the best locations of the racks of servers with different hardware capabilities to improve the supply temperatures of the CRAC units and the total cooling power.
- Given the nature of modern clusters with varying utilization that is a function of time and sophisticated job schedulers, we reformulate the rack layout problem in a probabilistic manner to identify layouts that are likely to provide the best cooling subject to various operating conditions.
- We propose a number of of heuristics and an optimal solution methodology based on integer linear programming (ILP) and evaluate them for realistic cluster configurations. Since the planning of clusters occur only once, the ILP runtime is practically feasible.
- Using a state-of-the-art Computational Fluid Dynamics (CFD) thermal modeling tool for computing clusters, we demonstrate the effectiveness of our approach in reducing total cooling power between $15.5\% - 38.5\%$ based on the cluster utilizations

with an average of 23.3%. This improvement is realized by planning the cluster using the rack locations identified from our algorithm and without any side effects.

The organization of this chapter is as follows. In Section 5.2 we formulate the layout planning problems for heterogeneous computing clusters to reduce cooling power and propose methods to solve it. Our experimental setup and results are provided in Section 5.3, where we demonstrate significant improvements in cooling power. Finally, Section 5.4 provides the main conclusions for this chapter.

5.2 Proposed Layout Methodology

In our cluster organization, we assume the existence of multiple racks of heterogeneous servers, where each rack holds a number of servers of the same configuration. The assumption that a rack holds servers of the same type is not restricting, and it is natural given typical cluster purchases and upgrade cycles.

Rack Layout Problem formulation: Given n heterogeneous server racks each consuming power p_i, the objective is to find the optimal layout that maps the n racks to n locations in the cluster such that the sufficient cooling power of the cluster's m CRAC units is minimized.

While the proposed problem formulation bears resemblance to the thermal-ware job scheduling problem, there is an important distinction between the two. The rack layout problem determines the inherent thermal characteristics of the cluster, and a bad rack layout can handicap a thermal job scheduler from achieving its target. By solving the rack

layout problem, we enable the design of computing clusters with inherently less cooling requirements, which can be leveraged for further improvements using thermal-aware job scheduling algorithms.

Given a cluster with a heat recirculation matrix $\mathbf{D}$ that represents the heat recirculation relations among the locations in this cluster, a greedy algorithm that can solve the proposed rack layout problem is given in Algorithm 5:

for *each rack location* i *of the* n *locations* **do**
  Compute $h_i = \sum_{j=1}^{n} \mathbf{D}(i, j)$;
end
Sort h_i in ascend order;
Sort the racks based on p_i in descending order;
Allocate the racks to locations based on their sorting order;

Algorithm 5: Greedy layout planning algorithm.

This algorithm allocates servers racks based on their power consumption rank to the locations based on their heat recirculation rank in the reverse order so the server with the highest power consumption gets in the position where it has the least recirculation effect on other servers and so on for other servers and locations.

Another solution to solve the proposed rack layout problem is through local search techniques as given in Algorithm 6. The local search method changes the allocations at random and will save the one with the lowest heat recirculation effect.

The greedy and local search methods are only heuristics and cannot guarantee finding optimal layouts. Thus we investigate an integer linear programming based algorithm to find the optimal solution for the rack layout problem. We use the greedy and local search for comparison in the experimental results section 5.3.

```
for each rack r_i of the n racks do
    allocate r_i to a free location picked at random
end
Assemble the p vector based on allocated rack locations;
let t_curr = max of Dp;
let t_min = t_curr;
while the maximum iteration is not reached do
    for each rack r_i do
        swap the location of r_i with a random rack;
        recompute t_curr;
        if t_curr ≤ t_min then
            let t_min = t_curr;
        end
        else
            swap racks back to their original locations;
        end
    end
end
```

Algorithm 6: Local search-based layout planning algorithm.

5.2.1 Thermal-aware Layout Optimization

To maximize the $\mathbf{T}^{sup}$ while ensuring the resulted $\mathbf{T}^{in}$ will not exceed the red line temperature t_{red}, we can instead minimize the maximum element in $\mathbf{Dp}$. Let $\mathbf{X}$ be an $n \times n$ permutation matrix, where $x_{ij} \in \{0, 1\}$ is equal to 1 if and only if rack r_j is placed to location i. The rack layout problem can be formulated as follows.

$$\begin{aligned} \text{minimize} \quad & ||\mathbf{DXp}||_\infty \\ \text{subject to} \quad & \forall i : \textstyle\sum_j^n x_{ij} = 1 \quad (1), \\ & \forall j : \textstyle\sum_i^n x_{ij} = 1 \quad (2), \\ & x_{ij} \in \{0, 1\}, \end{aligned}$$

where the infinity norm $||\mathbf{DXp}||_\infty$ gives the maximum element in the vector $\mathbf{DXp}$. Constraint (1) guarantees each location is assigned at most one rack, while constraint (2) guarantees that each rack r_j is assigned to a location. To solve this optimization problem, we can transform the problem into an integer linear program by adding a slack variable s,

where:

$$s = ||\mathbf{DXp}||_\infty \tag{5.1}$$

Then we can establish relationship between s and the temperature increment of each rack location i:

$$\forall i : t_i \leq s, \tag{5.2}$$

where $t_i = \sum_{k=1}^{n} \mathbf{D}_{ik}(\sum_{j=1}^{n} x_{kj}p_j)$. Thus, the formulation can be rewritten as:

$$\begin{aligned} \text{minimize} \quad & s \\ \text{subject to} \quad & \forall i : \textstyle\sum_{k=1}^{n} \mathbf{D}_{ik}(\sum_{j=1}^{n} x_{kj}p_j) \leq s \\ & \forall i : \textstyle\sum_{j=1}^{n} x_{ij} = 1, \\ & \forall j : \textstyle\sum_{i=1}^{n} x_{ij} = 1, \\ & x_{ij} \in \{0, 1\}. \end{aligned}$$

The integer linear program accurately describes the optimization objectives and constraints and by solving it we can guarantee the optimal rack layout during the cluster planning phase for minimizing the cooling power later during operation.

5.2.2 Probabilistic Layout Planning

In the previous subsection, we assumed that the power distribution vector $\mathbf{p}$ is constant over time. However, in reality, the power consumption of each rack, $\mathbf{p}(\lambda)$, varies as a

function of job arrival rate λ and the decisions of the job scheduler. In this case, the optimal rack layout will change with varying server utilization distributions. However, once located, the rack layout cannot be changed dynamically. Thus, it is important to find the optimal layout that minimizes the cooling power on the average over the distribution of utilization cases.

Integrating an accurate server power model that captures the impact of utilization will lead to a better layout that saves cooling power over a larger range of cluster operation. In this work, we consider two power models:

1. **Servers with idle power consumption.** In computing clusters where the number of service requests fluctuates frequently over a large range, the under-utilized servers should remain active in idle mode to enable immediate response. However, when idle, servers still consume $40\% - 50\%$ of their peak power [4]. Let p_{idle} denote the power consumption of a rack in idle mode and p_{dyn} denote the extra power when the rack is fully utilized, then the power p_i of each rack r_i can be modeled as:

$$p_i(\lambda) = p_{idle} + u_i(\lambda)p_{dyn}, \tag{5.3}$$

   where $0 \leq u_i(\lambda) \leq 1$ is the utilization of rack i for a job arrival rate λ to the cluster. Since our clusters are heterogeneous, the racks could have different p_{idle} and p_{dyn} depending on their servers configurations.

2. **Servers with power nap states.** To eliminate the idle power, especially when there is no hard requirement for request response time, some computing clusters put underutilized servers to power nap states to eliminate their idle power consumption. In this case, the power of rack i is given by

$$p_i(\lambda) = \begin{cases} p_{idle} + u_i(\lambda) p_{dyn}, & u_i(\lambda) > 0 \\ 0, & u_i(\lambda) = 0 \end{cases} \tag{5.4}$$

It is important to stress that the utilizations of the racks, which determine their power consumption and consequently the vector $\mathbf{p}$, is a function of both the job arrival rate λ and the policy of the job scheduler of the computing cluster. For each arrival rate λ, the corresponding utilization u_i for each server rack i can be estimated using a discrete-event simulation with the cluster's job scheduler.

To solve the rack layout problem with the consideration of utilization dynamics, we propose a probabilistic formulation for the rack layout problem. For a job arrival rate λ with probability density function pdf(λ), the objective of the optimal rack layout can be formulated as:

$$\mathbf{X}_{OPT} = \underset{\mathbf{X}}{\operatorname{argmin}} \int_0^\infty ||\mathbf{DXp}(\lambda)||_\infty \times \text{pdf}(\lambda) d\lambda \tag{5.5}$$

In reality, computing clusters are provisioned to operate under a maximum job arrival rate $\lambda_{\max}$. Thus we can approximate the integration in Equation (5.5) with a summation over the discrete integer values for λ. Using a slack variable s_λ for every possible λ, the integer linear program can be formulated as:

$$
\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{\lambda=0}^{\lambda_{\max}} s_\lambda \times \text{pdf}(\lambda) \\
\text{subject to} \quad & \forall i, \lambda : \textstyle\sum_{k=1}^{n} \mathbf{D}_{ik} (\sum_{j=1}^{n} x_{kj} p_j(\lambda)) \leq s_\lambda \\
& \forall i : \textstyle\sum_{j=1}^{n} x_{ij} = 1, \\
& \forall j : \textstyle\sum_{i=1}^{n} x_{ij} = 1, \\
& x_{ij} \in \{0, 1\}.
\end{aligned}
$$

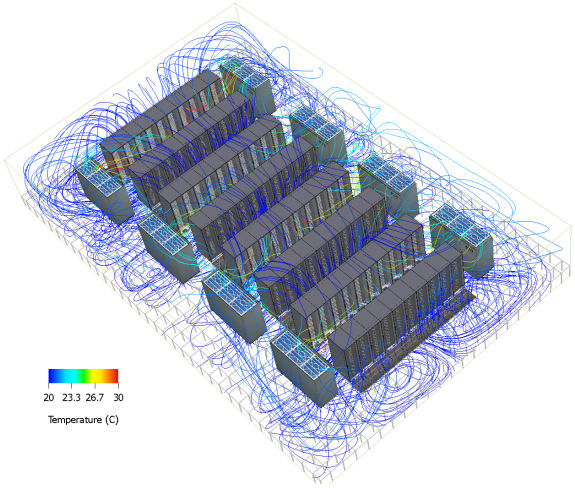

Figure 5.1: the CFD simulation results and configuration of our experimental computing cluster.

5.3 Experimental Results

We setup our cluster space to accommodate 80 U42 racks for a total of 3200 servers organized in the form of 8 aisles with 10 racks per aisle. We used the CFD modeling tool 6SigmaRoom Lite [26] to simulate the thermal and airflow characteristics of the cluster. The tool is used to generate the heat circulation matrix. Our cluster configuration and

Table 5.1: Server configurations.

server	CPU	num CPUs	num cores	freq (GHz)	DRAM (GB)
A	Intel Core i7 920	1	4	2.67	4
B	Intel Core i5 3450S	1	4	2.80	8
C	Intel Xeon E5530	2	8	2.27	12
D	AMD Phenom II X4	1	4	3.40	16

an example of the simulation result are illustrated in Figure 5.1. We consider a baseline of four classes of different servers to emulate the architectural heterogeneity in realistic computing clusters. The configurations for servers of the four classes are given in Table 5.1. We collect execution traces of SPEC CPU2006 benchmarks [69] on each of the four types of servers using the `pfmon` tool, while simultaneously measuring the power consumption trace using an Agilent 34410A digital multimeter. We consider 20 racks for every server type. The cluster has 8 CRAC units located at the sides as illustrated in Figure 5.1. We use the HP CRAC model given in Moore *et al.* with a $CoP(t_{sup}) = 0.0068t_{sup}^2 + 0.0008t_{sup} + 0.458$ [56]. We assume maximum inlet temperature is 25°C. To simulate the dynamic behavior of the cluster under different loading scenarios, we implement a discrete event queuing simulator [54], which uses the execution and power traces and the heat recirculation matrix to estimate the power and thermal characteristics as a function of time. The job arrival rates are drawn from a random exponential distribution with parameter λ, which controls the mean job arrival rate (jobs/second). The simulator uses a greedy job scheduler to assign jobs to servers, where it assigns an incoming job in the queue to the most energy-efficient free server i.e., the server with highest throughput per Watt [56].

Exp 1. Layout Planning with no Utilization Knowledge. In this experiment, we demonstrate our rack layout method under the scenario that the utilization is not *a priori* known. Thus, we assume that all servers are running at their highest power consumption which corresponds to maximum utilization. Because we assume full utilization, this planning experiment does not make value of the queueing simulations, and it corresponds to a case where the cluster is designed based on the plate specifications of the servers regardless of the workloads. We solve the rack layout problem by our proposed ILP algorithm, local search and greedy approaches. The estimated CRAC units supply temperature require-

Table 5.2: Supply tempereture and cooling power for the first experiment

Method	t_{sup} (°C)	cooling power (kW)
ILP	22.1	117
Local Search	20.6	133
Greedy	21.4	124
Heterogeneous oblivious	16.6	191

ment and minimum cooling power computed from each algorithm are given in Table 5.2. We also compare against the case of heterogeneous oblivious planning, where the racks are placed randomly independent of their specifications.

Our results show that the optimal locations identified by our ILP algorithm can achieve 38.5% savings in cooling power from 191 kW to 117 kW. The results also show that the use of the ILP-based method is justified compared to easier and faster alternatives such as greedy and local search, as the ILP method saves 11.8% of cooling power over local search and 5.6% over greedy. The runtime of the ILP took 60 seconds on server configuration A, which is quite reasonable given the large size of our experimental cluster and given that

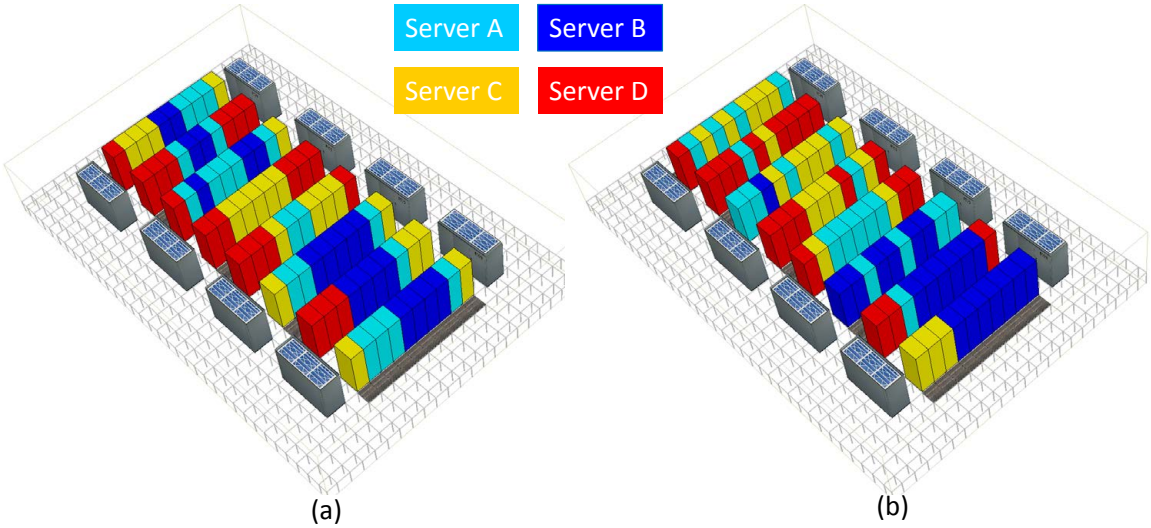

Figure 5.2: The layout planning of the experimental cluster. (a) using greedy on the left and (b) using ILP on the right.

the ILP needs to be solved only once during the planning phase of the cluster.

The cluster layouts using ILP and greedy method are given in Figure 5.2. The slight difference between these two layouts illustrates that placing the high power servers in locations with low heat recirculation effect is not enough. While both techniques have similar general trends in their layouts, ILP is able to find a global optimal solution while the greedy heuristics reaches a local optimal.

Exp 2. Layout Planning with known Utilization. When the utilization is known, our proposed method can find the optimal layout planning for a given utilization. We repeat the queuing simulation five times with five different mean job arrival rates from low mean arrival rate ($\lambda = 8$) to high arrival rate ($\lambda = 24$), which cover all service conditions from undersubscribed to oversubscribed operation. The average server utilizations of the four different server types for different job arrival rates are given in Figure 5.3. Note that for a given λ, the servers show different utilizations, which arise from the role of the job scheduler, which prefers to schedule jobs on the most energy-efficient servers first. Thus, at low job arrival rates, the most energy-efficient servers (e.g., servers of configuration D)

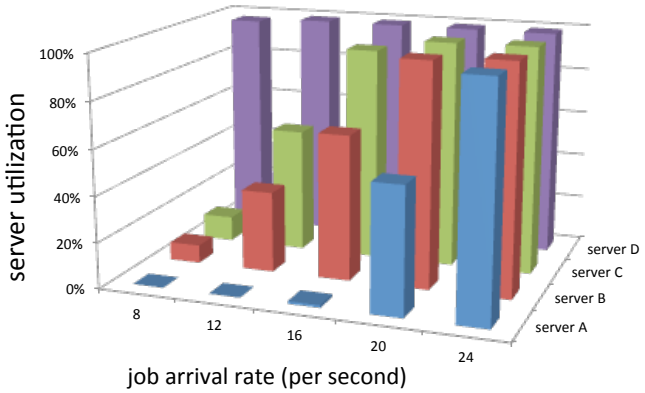

Figure 5.3: The average utilization of each server type.

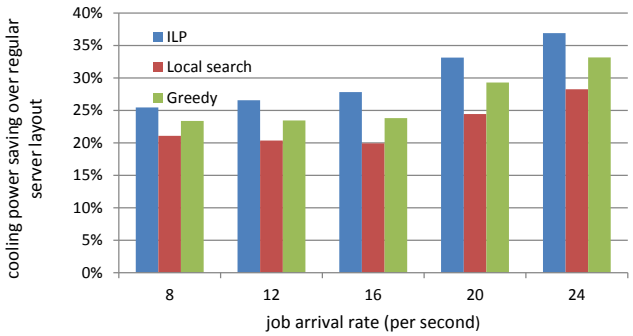

Figure 5.4: Reductions in cooling power over heterogeneous-oblivious planning when non-utilized servers consume idle power.

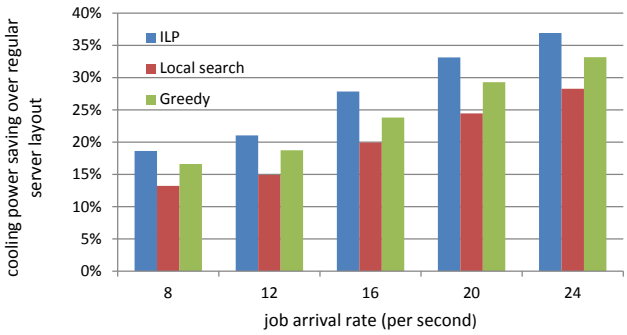

Figure 5.5: Reductions in cooling power over heterogeneous-oblivious planning when non-utilized servers switch to nap states.

will be mostly utilized but other servers will be relatively idle, while all servers tend to become more utilized as the mean job arrival rate increases.

To determine the optimal server rack layout, we consider two power management policies: (a) one where non-utilized servers consume a base amount of idle power depending on their specifications, and (b) one where non-utilized servers are put into near-zero power-saving state (e.g., power nap states [53]).

a. **Servers with idle power consumption.** When service requests transition rapidly, free servers must stay at idle state so that they can respond to new requests immediately. Given the non energy-proportional characteristics of servers [4], an idle server

still consumes $40\% - 50\%$ of its peak power consumption. We evaluate the minimum cooling power achieved by our proposed algorithm and demonstrate its power saving performance. In Figure 5.4, the layouts produced by the ILP reduce the cooling power by $25.5\% - 36.9\%$ compared to heterogeneous-oblivious planning, whereas local search and greedy methods reduce the cooling power requirements by $21.1\% - 33.2\%$.

b. **Servers with power nap states.** In some computing clusters where the service request rate is relatively stable, power management techniques could turn the free servers into near-zero power napping states to improve the energy-efficiency. In this case, the optimal layouts identified by our ILP method algorithm reduce the cooling power by $18.6\% - 36.9\%$ depending on the utilization rate as given in Figure 5.5, where local search and greedy algorithm give only $13.2\% - 33.2\%$ of improvements over heterogeneous oblivious rack layout. Our job scheduler prioritizes energy-efficient servers with low power consumption; thus, at low job arrival rates, the low-power servers have higher utilization while the high-power servers are under utilized and switch to nap mode, which leads to a relatively uniform power consumption by the racks. Thus, the optimization searching space and the reductions in cooling power are limited for small λ.

In both cases, ILP improves the cooling power reduction a maximum of 12.1% over local search and 5.6% compare to greedy method.

Exp 3. Layout Planning with Variable Utilization. In a real computing cluster, the mean job arrival rate and utilization are not static but they rather change over time reflecting user trends. Thus, for this experiment we used real computing cluster utilization traces to drive the distribution of job arrival rate for our cluster simulator. Figure 5.6 shows the proba-

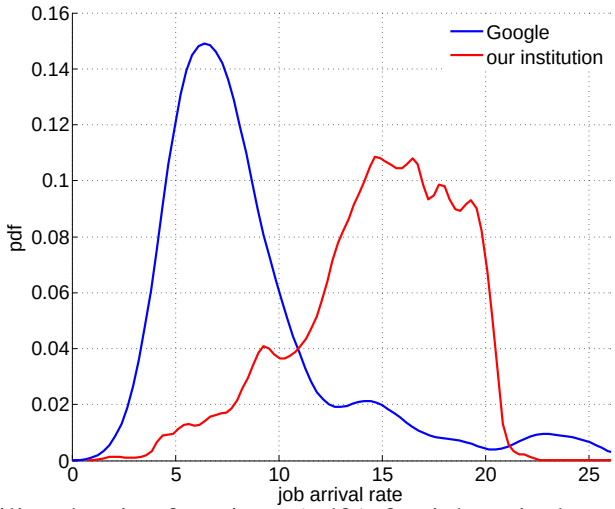

Figure 5.6: Probability density functions (pdfs) for job arrival rate distribution for two cluster: one at our institution and one at Google.

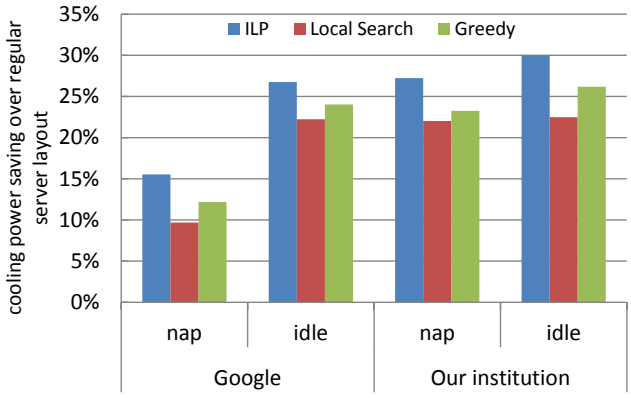

Figure 5.7: Cooling power reductions achieved by layout from our methods compared to heterogeneous-oblivious planning.

bility density function (pdf) of our institution's computing cluster utilization (i.e., Brown University) over a year and the pdf of a Google datacenter during 24 hours over a week [75]. We mapped these distributions to the corresponding job arrival rate for our cluster configuration. Using the probability of each job arrival rate and utilization distribution of each job arrival rate, we use our probabilistic ILP formulation to identify the optimal layout planning.

Figure 5.7 gives the result for our cluster using the PDFs from Google and our institu-

tion for the two power management cases: (a) free servers remain idle and (b) free servers are switched to nap modes off. Result shows that our methods are able to consistently deliver layouts that lead to reduced cooling power for various utilization trends. The magnitude for improvement is higher for our institution's cluster because it has a higher mean job arrival rate.

5.4 Summary

We have formulated the problem of rack layout for heterogeneous computing clusters, where the goal is to identify the locations of the server racks during the planning phase of the cluster to reduce cooling power. Whereas homogeneous racks offer no incentive to lay them out in a particular way, heterogeneous racks offer different power specifications that can be exploited to produce layouts with improved thermal characteristics and cooling. We also investigated the impact of varying cluster utilization and the impact of the job scheduler on cooling power and reformulated the rack layout problem to identify the locations that are likely to reduce cooling power under different operating conditions. We devised optimal solution methods to the rack layout problem using integer linear programming. For the experimental results, we devised a realistic model for computing clusters using state-of-the-art CFD modeling tools and demonstrated that our methods lead to significant improvements in the thermal characteristics and cooling power. We demonstrate the effectiveness of our approach in reducing total cooling power between $15.5\% - 38.5\%$ based on the cluster utilizations with an average of 23.3%

Chapter 6

Creating Soft Heterogeneity Through Firmware Reconfiguration

6.1 Introduction

Leveraging the server architecture heterogeneity delivers a significant improvements in computing cluster performance and energy-efficient. However, the workloads in a computing cluster changes in fast iterations, dedicated hardware configurations do not always guarantee optimal output. Furthermore, creating a cluster by purchasing servers with different hardware capabilities can complicate resource management and increase costs. Instead, a more realistic approach is to change the configuration of the available hardware and software components.

The traditional approach of configuring the firmware involves a human in the loop. System administrators follow simple ad-hoc rules to identify the appropriate firmware settings [44, 5], which can potentially lead to ineffective use of the hardware components

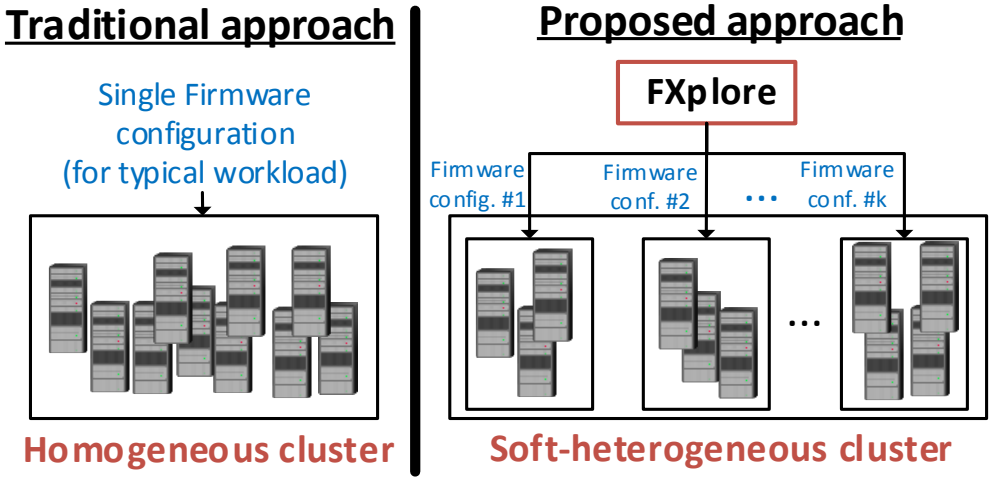

Figure 6.1: *FXplore* enables soft heterogeneity in a cluster by customizing firmware for target workloads to improve performance and energy efficiency.

and is naturally prone to human errors. In contrast, we propose an automated firmware option exploration tool called *FXplore* that is far more effective in finding firmware configurations of servers that can deliver the maximum benefits in performance and energy efficiency. Figure 6.1 contrasts the traditional approach with our approach.

There are several challenges in finding the optimal configurations. First, there are an exponential number of configurations as a function of the number of firmware settings, which makes identifying the optimal configuration for a workload a hard problem. Second, creating a dedicated sub-cluster with its own custom firmware configuration for each target workload can complicate system management, especially if there are a large number of target workloads. Third, administrators sometimes deploy co-runners on the same server. Through *FXplore*, we provide a framework that addresses these challenges. We make the following key contributions.

- We quantify the impact of firmware configurations on the runtime and power consumption of a diverse range of workloads. We demonstrate that the optimal configurations for these workloads can be very different. Furthermore, we show that

the optimal configurations cannot be derived by analyzing the impact of each of the individual firmware settings in isolation.

- We propose an automated firmware configuration exploration methodology called *FXplore* that employs a sequential-search heuristic algorithm to identify the optimal configuration for any given workload with substantial speedups compared to brute-force search. In particular, for every n firmware settings, we show that we can reduce the exploration time from $O(c^n)$ to $O(n^2)$.

- To simplify system management, *FXplore* uses machine-learning (ML) techniques to trade-off the degree of heterogeneity in the cluster with the optimality of the performance and energy. In particular, *FXplore* uses k-means to partition a homogeneous cluster of servers into sub-clusters, each with its unique firmware configuration and suitability for sets of targeted workloads rather than a single workload. Thus, *FXplore* simplifies system management in the presence of heterogeneity.

- We extend *FXplore* to handle co-running workloads, such that it identifies the firmware configurations for cases when multiple workloads are run on the same server. We also evaluate a number of techniques from the literature geared for on-line operation to enable administrators to map new workloads to existing sub-clusters depending on the similarity to workloads in the training set.

- We validate our methodology on a fully-instrumented cluster with eight server nodes using a diverse set of parallel workloads that span HPC applications and cluster workloads. We demonstrate that *FXplore* can improve runtimes by 11% and energy efficiencies by 15% in average, compared to baseline firmware configurations. It can also accelerate firmware configuration exploration by $2.2\times$ compared to brute-force exploration.

The rest of the paper is organized as follows. In Section 6.2, we motivate the need for *FX-plore*. In Section 6.3, we describe our firmware configuration methodology including the sequential exploration, sub-clustering and ML-based mapping approaches. In Section 6.4, we present the experimental results. We summarize the work in Section 6.5.

6.2 Motivation: Impact of firmware configurations

In this study, we aim to improve the performance and energy efficiency of a server by customizing its hardware to the software characteristics of the workloads using the options in the firmware. Modern servers offer many firmware configurations with each offering a different impact on the power-performance levels of different workloads.

Table 6.1 lists five important firmware settings that are available in our servers through

BIOS Setting	Description
Hardware prefetcher (HP)	Enabling HP fetches the data and instructions from memory into cache before the processor loads them.
Adjacent cache-line prefetcher (CP)	Enabling CP will make the processor always fetch two adjacent cache lines.
CPU turbo boost (CTB)	Allows CPU cores to scale up their clock frequency on-demand depending on thermal or voltage slack.
Memory turbo boost (MTB)	Allows adjustment of the memory frequency to a higher or lower value.
Hyper threading (HT)	Enables simultaneous multithreading, which allows threads to share the processor resources where each physical core is regarded as two virtual cores.

Table 6.1: Firmware settings that we explore in our study.

the BIOS. There are two settings related to the cache performance: the hardware prefetcher (HP) and adjacent cache-line prefetcher (CP). HP enables prefetching between the cache and main memory, while CP enables prefetching between cache and CPU cores. Enabling HP and CP will benefit workloads with predictable memory-access patterns and good data locality. CPU turbo boost (CT) mode is another option that is also widely available in server-class processors. CT enables processors to operate at a higher frequency than the nominal especially when thermal and voltage slacks are available. The speed of memory is another configuration option that can be tuned from firmware. Our servers provide control over the frequency through an option called the memory turbo boost (MT). Enabling MT allows the memory to run at a higher frequency (1066 MHz), while disabling it lowers the speed (to 800 MHz). Thus, by enabling MT, the system can potentially lower cost for memory accesses, which will provide a sizable benefit to memory-bound applications. Hyper-threading (HT) allows throttling the server performance through simultaneous multi-threading. By sharing the hardware resources on a single physical processor core, two different threads can be executed simultaneously. Ideally, multi-threaded workloads may benefit from enabling HT at the cost of some additional power. However, for workloads with compute-intensive threads, enabling HT can, in fact, create contention for the CPU core and degrade performance compared to no HT. Thus, optimally configuring these five firmware options is a highly nuanced and workload-driven task.

Among the limited set of options provided by the version of the firmware on our servers, we found that the above five settings are the ones that impact power and performance levels the most. Thus, we focus on these five parameters to demonstrate the benefits of *FXplore*. However, with newer firmware versions and server models, there are many more configuration options that are available [5]. Such options provide an opportunity for much finer grain firmware tuning by *FXplore*. The basic premise of our work lies in the fact that enabling or disabling firmware options can have a large impact on

the performance and power consumption of servers as a function of the workloads. To illustrate this point, we profile a number of parallel workloads on our networked cluster that is composed of eight fully instrumented Xeon-based servers. We select parallel workloads from the NAS Parallel Benchmarks (NPB), which are representative of HPC and workloads from the NU-MineBench benchmarks, which are representative of data mining workloads. Since we consider five firmware options, each server in the cluster can be configured in total of $2^5 = 32$ different ways. Since we run an application on the entire cluster, we enforce any chosen firmware configuration on all of our servers. For instance, consider four exemplary workloads from our two benchmark datasets. Figure 6.2 shows the runtime and power of these workloads under the 32 possible firmware configurations. All results are normalized with respect to the results when all firmware options are enabled. The trends in the figure lead us to our first observation.

Observation #1:. Firmware settings can have a large impact on the performance and power consumption of applications since each application has its own unique characteristics. For instance, from Figure 6.2, we see that application `CG` shows 173% variation in runtime as a function of the firmware settings. However, application `SP` shows 59% variation in runtime as a function of the firmware settings. Given these large application-dependent variations in runtime and power, it is imperative to ask if there are any shared

Runtime-optimal Configurations							**Energy-optimal Configurations**						
NAME	CTB	MTB	HT	HP	CP	Runtime	NAME	CTB	MTB	HT	HP	CP	Energy
BT	✓	✓	✓	✓	✗	0.982	BT	✗	✓	✓	✗	✗	0.962
CG	✓	✓	✓	✓	✗	0.937	CG	✗	✓	✓	✗	✗	0.915
LU	✓	✓	✗	✓	✗	0.629	LU	✗	✓	✗	✗	✓	0.569
SVM	✓	✓	✗	✓	✓	0.701	SVM	✗	✓	✗	✗	✓	0.655

Table 6.2: Optimal firmware settings are different for each of the four applications considered in Figure 6.2. Further, different settings optimize runtime and energy consumption.

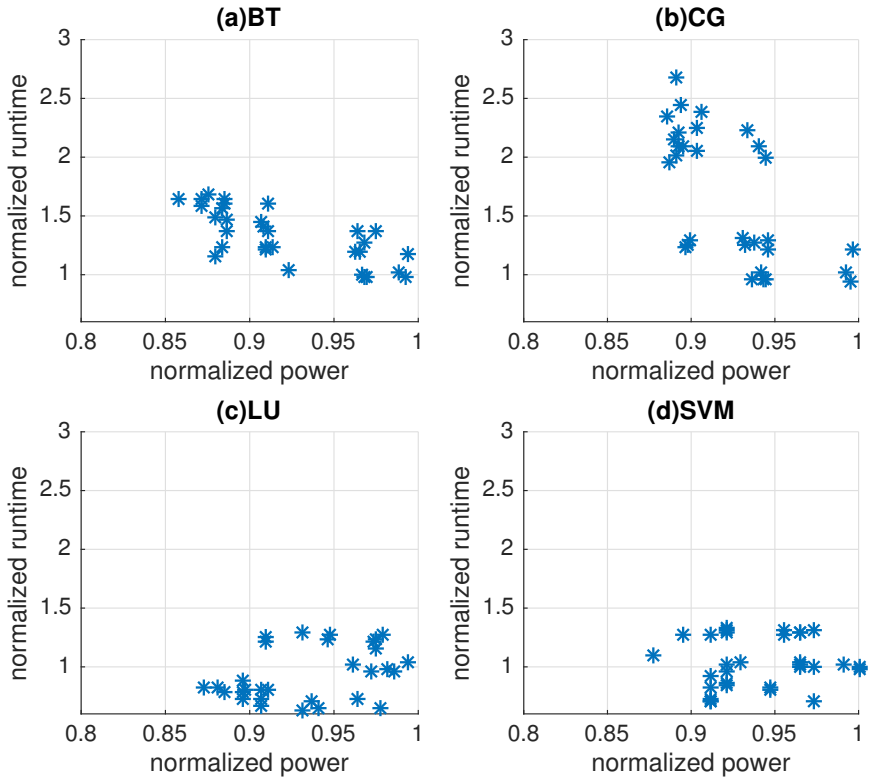

Figure 6.2: Firmware configurations can have a large impact on the runtime and power consumption of a server depending on the application characteristics.

optimal settings among various applications or whether there is a simple characterization for the optimal runtime and energy efficiency based on the firmware settings.

Observation #2:. The optimal firmware configurations vary by application, where each application could have its own distinct optimal configuration for performance and energy efficiency. Table 6.2 illustrates how the optimal configurations for minimizing runtime and energy consumption of the four workloads can be drastically different. The results interestingly show that enabling all firmware settings does not necessarily lead to the best runtime or energy efficiency. One possible reason for this behavior is that the enabled options could conflict with each other. Further, different applications have different sen-

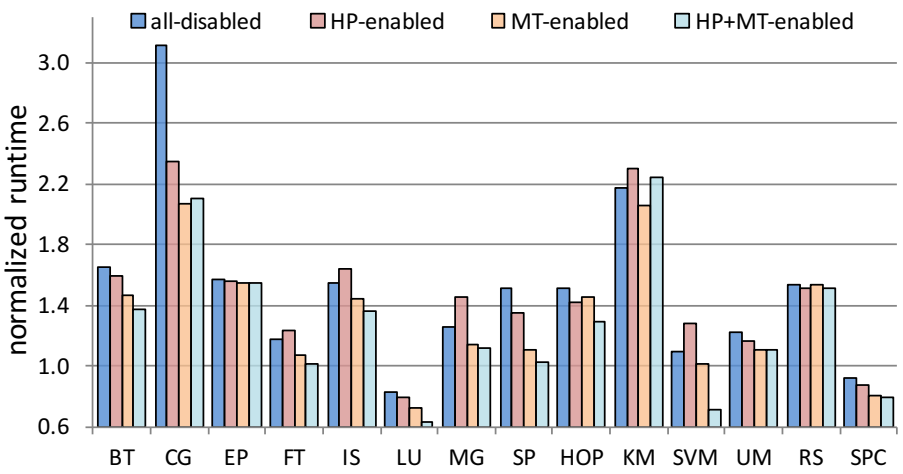

Figure 6.3: The normalized runtime of workloads under firmware configurations with only HP enabled, only MT enabled, and both HP and MT enabled shows how the interdependence between firmware options is application specific. The baseline (*i.e.*, runtime = 1) case is where all five options are enabled.

sitivity to the firmware settings. Turning on some firmware settings might not yield good performance improvement or even not have a positive impact. For instance, enabling HT reduces the runtime of many workloads, but it hurts `LU` and `SVM`. The results also show that the optimal settings for runtime minimization may not be the best settings for energy minimization.

Observation #3:. There are subtle interactions among the firmware configurations that do not necessarily add up to provide or diminish gains. Thus, combining settings that yield better results individually may not necessarily further improve the results, and similarly combining settings that have a negative impact individually might surprisingly lead to a positive impact. Figure 6.3 illustrates the inter-dependency between MT and HP. In the figure, we compare the runtime under three different firmware configurations: enabling only MT, enabling only HP, and enabling both MT and HP. The runtime is normalized against the case that all five settings are enabled. The runtime of disabling all firmware options is also plotted for reference to show the impact of the individual options. From

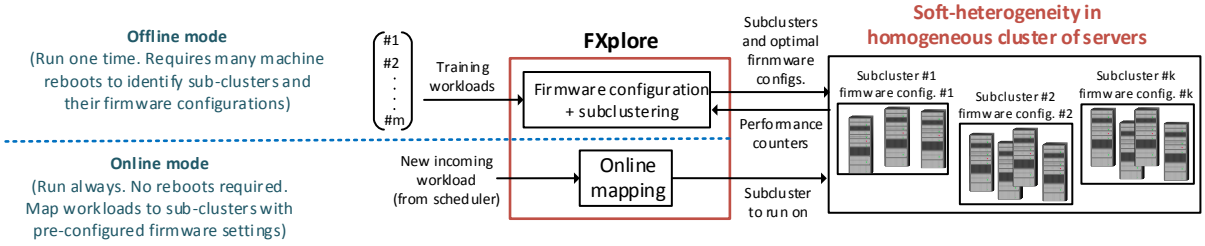

Figure 6.4: Overview of *FXplore*. It comprises two operation modes. First, is an offline mode where we use sub-clustering and sequential search to find the optimal firmware configurations for groups of servers. This requires rebooting the servers several times. Second, is the online mapping stage, where we use machine learning algorithms to map incoming workloads to appropriate sub-clusters requiring no rebooting of the servers.

the results, for most of the applications such as `BT`, `LU`, `SP`, `HOP`, and `SPC`, enabling HP and MEM individually improves the runtime over all-disabled case and enabling both of them delivers greater improvement. However, for `CG` and `KM`, enabling both of them yields worse runtimes compared to the cases where only MT is enabled. And for `FT`, `IS`, `MG`, and `SVM`, although only enabling HP makes the runtime worse than the all-disabled, when HP is enabled together with MT, the runtime is improved, comparing to the case where either HP or MT are exclusively enabled.

We thus conclude that there is no ideal firmware configuration that is optimal for all kinds of workloads. Also, there is no simple rule that could give us the optimal firmware configuration for a given application. Thus, an intelligent and efficient method to determine the optimal configuration for any given application is highly desirable. Next, we present *FXplore* that precisely addresses this problem.

6.3 FXplore Methodology

In this section, we present the methodology of *FXplore*. We describe how it enables system administrators to partition servers into sub-clusters such that each sub-cluster is configured with a different firmware configuration to improve performance and/or energy efficiency for target workloads. Figure 6.4 gives an overview of our methodology. There are two modes of operation. First is the offline mode, where given a set of M target workloads, we partition them into κ sub-clusters, afforded by the administrator, and derive the optimal configurations for the sub-clusters in a fast and effective manner. Second is the online mode, where incoming workloads are profiled using hardware performance counters (PMCs) and mapped to one of the existing sub-clusters with a pre-determined optimal configuration. This process requires no reboots of the servers and is invoked during regular use of the servers. Note that if $M = \kappa$ then each workload can afford its own sub-cluster with optimal settings. However, it is expected that $\kappa < M$ because a large κ can complicate cluster management.

Today, for system administrators, the only available alternative to *FXplore* is brute-force enumeration, where the outcomes (*i.e.*, runtime and energy) under all possible firmware settings need to be enumerated for every workload. Among these settings, the one that gives the best results needs to be chosen and used for the server. For N firmware options, brute-force enumeration requires an exploration of 2^N settings per workload (*e.g.*, $32 \times M$ reboots of the server are required for M workloads and 5 firmware options), which is not scalable especially when modern high-end servers provide 10-15 firmware options that could impact the power/performance of applications [19]. Through *FXplore*, we propose to reduce this firmware exploration time complexity from 2^N to $O(N^2)$. Further, once the one-time exploration process is completed offline, we provide a methodology to map new incoming workloads in real time to sub-clusters of servers with pre-determined

optimal configurations. We describe the details of our offline heuristic exploration and sub-clustering methodologies in Sections 6.3.1 and 6.3.2, respectively. The details of the on-line operation are given in Section 6.3.3. In Section 6.3.4, we extend our methodology to handle the case when co-runners are enabled; *i.e.*, when multiple applications are run on the same server.

6.3.1 Identifying the BIOS configurations

This is the first part of the offline process in *FXplore*, which needs to be run one time with $O(N^2)$ server reboots. In this method, we follow an iterative sequential approach to optimize our cost function (*i.e.*, performance or energy) with the aim to minimize the search space of firmware options. Our approach seeks to capture the subtle interactions of the firmware configurations as outlined in observation #3 in Section 6.2, which concluded that the improvements in performance or energy obtained by a combination of firmware options is not equal to a simple superposition of improvements due to each option. The procedure for our proposed sequential search method, *FXplore-S*, is given in Algorithm 7. At the outset, we enable all candidate firmware options and label them as *free* (step 1). Then, in each iteration (step 2), we profile the input workload by temporarily disabling one *free* firmware option at a time (steps 3, 4, and 6). During this time, for each option, we also measure and register the cost function *i.e.*, runtime or energy consumption (step 5). After this, we disable the option that, when it is disabled, the cost function is minimized. We also label that option as *locked* for all subsequent iterations of the procedure (item 7). At the k^{th} iteration, $N - k + 1$ *free* options are evaluated and the one with the highest impact on the cost function is disabled and locked. Thus, to disable and lock N firmware options, we will need N iterations. After completing N iterations, we rank the results from all iterations by their cost-function value, and set the combination of firmware options that

globally minimize the cost function.

The exploration time complexity of *FXplore-S* is $N + (N-1) + \cdots + 1 = O(N^2)$, which is a substantial improvement compared to the exponential complexity (*i.e.*, 2^N) of brute-force enumeration. Our approach can also be extended to handle non-binary firmware options. We convert a non-binary firmware option to a group of binary options and configure them using *FXplore-S*. However, during the locking stage, we lock the group as a whole instead of considering the binary options in the group separately.

6.3.2 Deriving the Subclusters

This is the second part of the offline process, which also needs to be run one time during configuration of the servers. If the system administrator can afford a dedicated sub-cluster for every target workload, then the firmware configurations identified by *FXplore-S* can be directly used. In reality the number of sub-clusters is likely to be smaller than the number

input : Input workload and n candidate firmware options.
output: Optimal firmware configuration for input workload.
1. Enable all n candidate firmware options and label them as *free*;
2. **for** $k = 1 \ldots n$ **do**
    3. **for** *each free firmware option* **do**
        4. Disable the firmware option;
        5. Run the workload, measure, and record runtime or energy;
        6. Enable firmware option;
        7. Disable the option that, when it was disabled, workload achieved the best results in Step 5 and label it as *locked*;
    end
    8. Repeat 2-7 until all the firmware option are disabled;
end
9. Search the firmware configurations have been explored and find the one gives the best results;

Algorithm 7: *FXplore-S*: Offline sequential-search algorithm to determine the optimal firmware configuration.

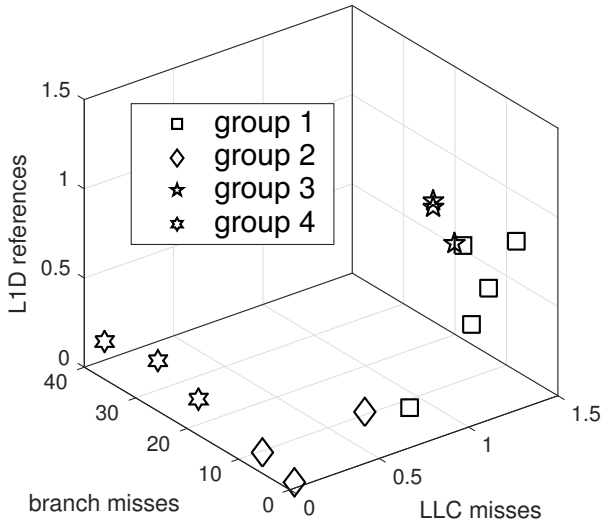

Figure 6.5: Plot of truncated feature vectors (only three performance counters out of the five) and their natural clusters.

of target workloads. Thus, in this section we propose a method, *FXplore-SC* that aims to partition the target workloads into groups with consistency so that they can be executed together in the same sub-cluster configured with the same firmware configurations.

FXplore-SC proceeds by (1) grouping applications based on their system-level performance characteristics, (2) identifying the optimal configurations for a representative application from each group using *FXplore-S*, and (3) applying the firmware configurations from the representative application of each group to the remaining workloads in the group. The insight behind this procedure is that workloads that exhibit similar system-level performance characteristics should have similar firmware settings.

To measure similarity of workload characteristics, we resort to PMCs. PMC values reveal subtle characteristics of the workloads since they are directly associated with their hardware interactions. For each workload, we compute the average PMC values over time and then per core. For our benchmarks, we collected 13 PMCs covering a diverse range of characteristics. We then computed the principal component analysis (PCA) of the PMC

measurements and analyzed the PCA scores of the performance counters. The PCA analysis reveals that the most relevant performance counters are the number of instructions retired, L1 data references, L2 data-cache misses, last-level cache (LLC) misses, and mispredicted branch instructions. As it is difficult to visualize high dimensional data, we selected three of the five counters: LLC misses, branch prediction misses and L1 data cache references and plotted their three-dimensional feature vectors in Figure 6.5. In the figure, the PMC values are normalized. We can clearly see clustering of the benchmarks even in this lower-dimensional space. *FXplore-SC* is based on PMC values. Its procedure is shown in Algorithm 8. Given a set of workloads, we first quantify the characteristics of each workload in the set by running it on a baseline configuration (*e.g.*, where all firmware options are enabled). We do this quantification by collecting the PMCs mentioned above (step 1). Thus, the average PMC values comprise the feature vector for every workload (step 2). After this, we apply the k-means algorithm to group the workloads into κ groups (step 3). The clustering algorithm works by minimizing the distance between the group members (*i.e.*, workloads) and maximizing the distance between the group centroids. For each group (step 4), we pick a representative workload and determine its optimal configuration using *FXplore-S* (step 5). Finally, we employ these settings for all other workloads of the group (step 6).

input : Workload set W and desired clustering granularity k.
output: Optimal configurations for the subclusters.
1. Run each workload on a baseline configuration and collect its PMC values;
2. Average the PMC values for each application to produce each workload unique feature vector;
3. Apply k-means on the feature vectors to cluster the workloads into k clusters based on their feature vectors;
4. **for** *each cluster* **do**
    5. Pick an application from the cluster and apply *FXplore-S* to determine its optimal firmware configuration;
    6. Use this firmware configuration for the remaining workloads in the cluster;
end

Algorithm 8: *FXplore-SC*: Offline process to determine sub-clusters of servers and fix their optimal configurations.

The value of κ is chosen by the system administrators, which provides them with the flexibility to trade-off the amount of heterogeneity and improvement in performance/power against the amount of management overhead. At its extremes, $\kappa = 1$ leads to one nominal firmware configuration that is used for all servers, and $\kappa = M$ provides a high degree of customization, where every application gets it own sub-cluster that is optimally configured according to its characteristics. It is up to the system administrators to decide the acceptable amount of heterogeneity in the cluster. We will evaluate the impact of κ in Section 6.4.

6.3.3 On-line Operation

This is the online process in *FXplore*, which is run all the time and it requires no server reboots. Once the sub-clusters of servers have been identified and set to have optimal configurations using a number of representative or training workloads, the mapping of a new workload is done based on the similarities to the training workloads. A number of works in the literature tackle this problem [42, 17]. For instance, Liao *et al.* evaluate a number of ML techniques, such as decision trees (DT) and support vector machines (SVM), to map new workloads to a target set of trained workloads as a function of the features of the workloads [42]. We follow a similar approach in this paper. In our case, a new workload is profiled on a baseline server to get its PMC-based feature vector. Then it is mapped using nearest-neighbor (NN) search into one of the existing groups based on the distance between its feature vector and the centroids of the groups, where the cluster with minimum distance is chosen. Once, we map a new workload to a cluster, we use the configuration of the cluster for the incoming workload.

6.3.4 Workload Co-location

In many modern clusters, a single server might host multiple workloads concurrently to leverage the availability of the large number of cores or processor sockets. A number of prior works consider the interference arising from allocating workloads on the same server and provide techniques to identify the optimal pairing of co-running workloads to minimize the degradation in performance [17, 50, 52, 30]. We address the co-runner problem by leveraging these prior works to extend our framework so that co-runners are enabled as follows.

1. A desired workload allocation algorithm is first run to identify the optimal pairing of application co-runners on a server with a baseline firmware configuration.

2. If (w_1, w_2) are identified to be an optimal pairing of two applications w_1 and w_2, then *FXplore-S* is executed to identify the best firmware configuration for the pair simultaneously.

3. If clustering is required, then the average per-core PMC vector from running both workloads is first profiled. This average vector is then used as part of the feature space in the sub-clustering algorithm *FXplore-SC*.

We evaluate the effectiveness of *FXplore* in handling co-runners in Section 6.4.4. Note that we do not propose any new workload co-location or scheduling algorithm. Instead, we observe that co-runner algorithms can have cyclic dependencies with firmware tuning; *i.e.*, there is a possibility that the results of the co-runner algorithm depend on the firmware configuration chosen in the first place. We broke that dependency by choosing to identify the optimal co-runners on a baseline server using existing co-runner algorithms;

however, the possibility of co-optimizing the firmware configuration and co-runner pairs of applications can lead to further improvements in future work.

6.4 Experimental Results

We present experimental results that validate the ability of *FXplore* to determine the runtime- and energy-optimal firmware configurations for various workloads, while providing system administrators with the ability to control the degree of soft-heterogeneity through sub-clustering. To evaluate *FXplore*, we use a cluster of 8 Dell PowerEdge C1100 servers. Each server is equipped with dual Xeon L5520 quad-core processors (total 8 cores) and 40 GB of DRAM. They run Ubuntu 12.4 and AMI BIOS version 2.66. We use the `perfmon2` tool to collect PMC values from the servers. To determine the total power consumption of the cluster, we created a measurement environment that senses the current flow through the power cord of each server.

We profiled a large set of benchmarks to evaluate the effectiveness of *FXplore*. In particular, we consider workloads from the following two sets of parallel benchmarks: (1) eight workloads from NPB representing computational fluid-dynamics applications: `BT`, `CG`, `EP`, `FT`, `IS`, `LU`, `MG`, and `SP`) [3] and (2) six workloads from NU-MineBench representing data mining applications: `HOP`, `kmeans(KM)`, `SVM`, `Utility Mining(UM)`, `RSearch(RS)` and `SaclParC(SPC))` [57]. NPB is designed to characterize HPC clusters, while NU-MineBench comprises datacenter-like workloads. We configure each workload as 64 threads (8×8) to execute on our experimental cluster. We double the number of threads when hyper-threading is enabled.

To establish ground truth for the optimal configurations, we exhaustively search through

all possible firmware configurations. For every configuration, we collect PMC values, power consumption and the runtime of each workload, and compute the energy consumption and exploration time for exhaustive search. Note that to collect the PMC values in our experiments, we run our parallel workloads to completion. We next assess the effectiveness of *FXplore*.

6.4.1 Sequential search results

In this subsection, we present experimental results for the offline exploration mode of *FXplore i.e.*, *FXplore-S*. We demonstrate the effectiveness of *FXplore-S* in finding the optimal configurations for a given workload, while attaining large reductions in exploration time compared to brute force. We consider the following configurations:

1. **all-enabled**: We enable all the five firmware options. The runtime (or energy) under other firmware configurations are normalized with respect to this runtime (or energy). We choose it as baseline configuration in all experiments of Section 6.4.
2. **FXplore-S**: We run our sequential search algorithm to find the best configuration. We consider two cases: a *runtime-driven case* that seeks to minimize runtime, and an *energy-driven case* that seeks to minimize energy.
3. **Brute-force**: We use a brute-force enumeration on all configurations to find the optimal configuration for optimizing runtime or energy consumption.

Optimizing runtime: We report the normalized runtime of all workloads using the different configuration methods above in Figure 6.6. From the results, we can draw the following conclusions. Enabling all options does not necessarily deliver near-optimal results as is obvious in the cases of `LU`, `KM`, `SVM`, and `SPC`. One possible reason for this

behavior is that the enabled firmware options conflict with one another. For instance, if a workload contains several spin-wait loops, enabling HT can create memory-order conflicts and slow down execution. Now, in such a case, if we enable HP in addition to HT, the server can end up with a large number of unusable memory fetches, which may lead to thermal issues and throttling. Thus, the overall performance of the workload may suffer substantially. Figure 6.6 shows that *FXplore-S* always finds the optimal or near-optimal firmware configuration for all the workloads except for the runtime of `UM`, which is 2% longer than the optimal. *FXplore-S* finds the exact optimal configurations for the rest of the workloads. Overall, *FXplore-S* improves runtime by 11% over the `all-enabled` configuration on average. In terms of exploration time, the results in Figure 6.8 demonstrate that the average exploration time of *FXplore-S* is only 46% of brute-force search; that is, *FXplore-S* speeds up the exploration time by 2.2×. Note that the speed-up will increase with more firmware settings because the exploration time of *FXplore-S* grows quadratically, while that of brute-force search grows exponentially.

Optimizing energy efficiency: Energy efficiency is another important objective for clusters. Therefore, in this second experiment, we switch *FXplore-S* to the energy-driven case where we determine firmware configurations that minimize the energy consumption for various workloads. For the five options, unlike the impact of the firmware configuration on runtime, the effects on power consumption are more or less predictable. In particular, disabling any of the five options always saves power. Since energy is the product of power and runtime, the impact of the firmware configurations on energy is determined by how pronounced are the savings in power. Figure 6.7 gives the energy consumption of the workloads for various firmware exploration methods, where we normalize the energy numbers with respect to the energy measurements from `all-enabled`. From the figure, we observe that by using *FXplore-S* we can almost always find the energy-optimal or near

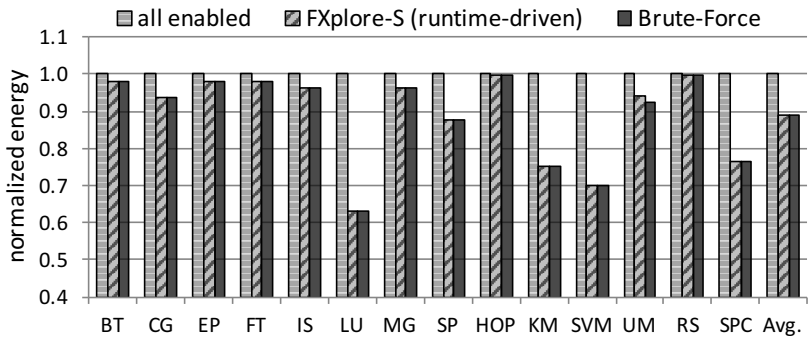

Figure 6.6: Normalized runtime improvement of workloads with different firmware configuration methods. Normalized to all-enabled.

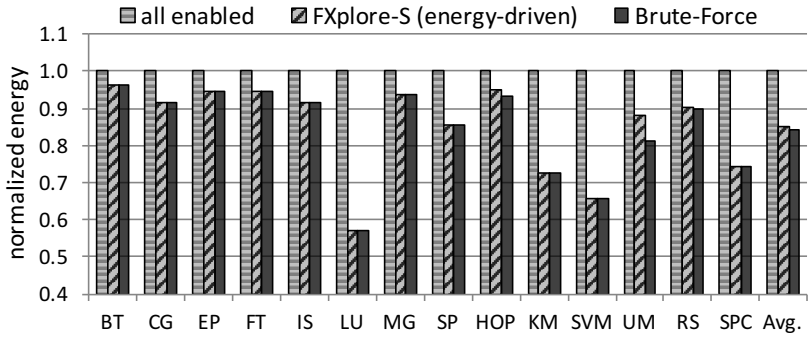

Figure 6.7: The normalized energy of workloads under different BIOS configurations.

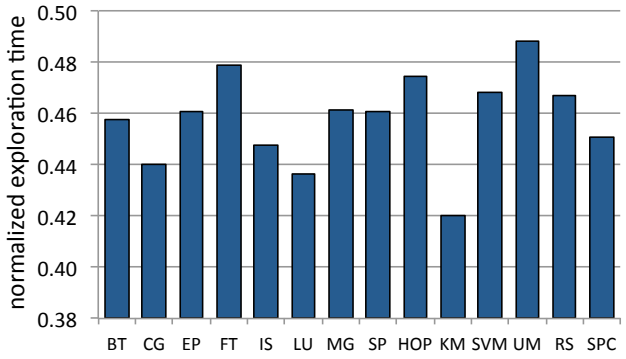

Figure 6.8: The normalized exploration time of each workload, normalized to `Brute-force`.

energy-optimal configuration. The speed-up in exploration time in the energy-driven case is similar to the runtime case.

Scalability of FXplore with number of firmware settings. Since *FXplore-S* is a heuristic algorithm. In this section, we study its effectiveness and scalability in finding the optimal configuration as a function of the number of available settings N. Since our evaluation servers provide only a limited number of firmware options that we can tune, we consider the cases of $N = 2$-5. We use *exploration error* as a metric to evaluate the effectiveness of *FXplore-S*. For any workload N, let $T_o(N)$ be its runtime under its optimal configuration (found by brute-force search) and $T_s(N)$ be the runtime under the firmware configuration identified by *FXplore-S*. Then, we define *exploration error* as:

$$exploration\ error(N) = \frac{T_s(N) - T_o(n)}{T_o(N)}$$

and *exploration accuracy* as equal to 1 minus the *exploration error*. We report this number using the y-axis on the right hand side of Figure 6.9. We observe a near negligible degradation in exploration accuracy as we go from 1 to 5 firmware options. On the y-axis on the left hand side of Figure 6.9, we also report the average number of mispredictions that we get for all the candidate workloads using *FXplore-S*. Again a small degradation occurs because of the increase in depth of the search path; however, the inaccuracy is considered almost negligible from a practical perspective. Unlike the cases of $N = 3$ and $N = 4$

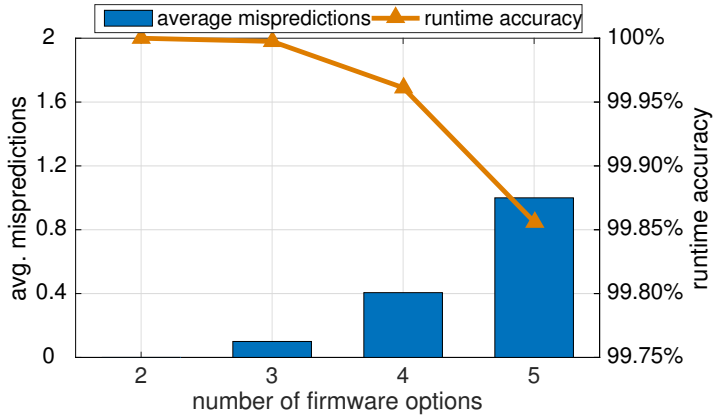

Figure 6.9: The scalability of *FXplore-S* as a function of the number of firmware options for optimizing runtime.

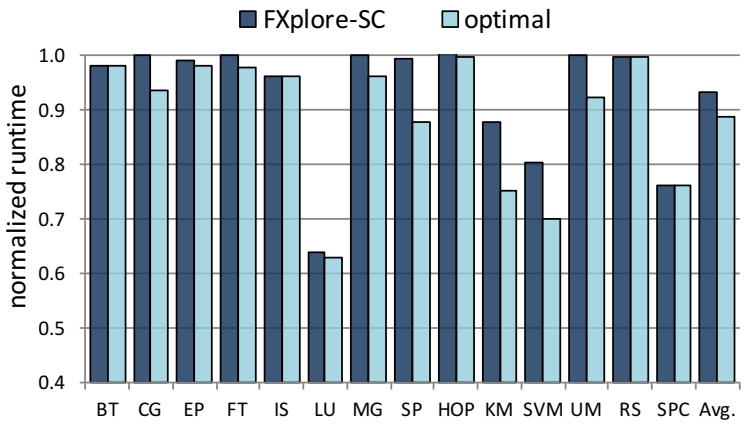

Figure 6.10: Workload runtime under sub-clusters firmware configurations created by *FXplore-SC*, when $\kappa = 4$, *i.e.*, four sub-clusters.

firmware options, where we have $\binom{5}{3} = 10$ and $\binom{5}{4} = 5$ estimation samples, respectively, the case of $N = 5$ options has only one sample, which makes it sensitive to experimental noise. The overall trend is that exploration error grows linearly, which means the accuracy will not decrease drastically when more firmware options are considered.

6.4.2 Clustering results

In this subsection, we evaluate the second part of the offline mode in *FXplore*, *i.e.*, *FXplore-SC*. In particular, we assess how well can *FXplore-SC* derive sub-clusters with heterogeneous configurations. The number of sub-clusters κ is a configurable parameter and is chosen by the system administrator depending on management costs. Since we have 14 benchmarks, we can potentially vary κ from 1 to 14 and evaluate the resulting runtimes of the workloads under different clustering results. Note that when $\kappa = 14$, the *FXplore-SC* method becomes essentially the *FXplore-S* method since no clustering is involved, and each workload gets it optimal configuration. However, choosing the optimal number of sub-clusters in general is a challenging task in itself. Choosing a small number of clus-

ters will minimize the heterogeneity of the servers, while a large number of them will complicate system management. Thus, to maintain a good tradeoff between management overheads and the amount of heterogeneity, we choose four sub-clusters for our experiments.

Figure 6.10 shows the runtime of each workload in four sub-clusters, *i.e.*, $\kappa = 4$. Again we normalize runtimes to the all-enabled firmware option. In the figure, we compare the normalized runtime of each benchmark under the optimal configuration of the sub-cluster it belongs to and its own individual optimal configuration (determined by brute-force search). We observe that the average runtimes of the workloads under optimal sub-cluster level firmware configurations are only 5% higher than under individually optimal configurations.

6.4.3 Evaluation of On-line Mapping Methods

In this subsection, we evaluate the on-line mode of *FXplore*, where incoming workloads from a scheduler get mapped to sub-clusters that are statically pre-programmed with different firmware configurations. For this mode, we leverage some of the existing ML techniques in workload mapping [42, 17, 50]. For high statistical confidence, we employ leave-one-out cross validation of the workloads. Accordingly, we use the offline mode of *FXplore* for all but one workload to determine the sub-clusters and their optimal configurations. Then, we use the PMC-based feature vector of the isolated workload to map it to the sub-clusters using different ML techniques such as DT and SVM, which have been tested in [42] as well as NN, which is used in *FXplore*. We repeat this process to isolate every workload in our test set. Our results presented in Table 6.3 show that NN works as well as any other classifier and there is no best mapping algorithm. Overall using NN-based online mapping and four sub-clusters, we find that there is an average of 3%

	runtime (×)			runtime (×)
	Classification Method			Optimal
name	NN (FXplore)	SVM	DT	
BT	0.982	1.371	1.371	0.982
CG	0.937	2.101	2.101	0.937
EP	0.982	0.982	1.547	0.982
FT	1.000	1.000	1.010	1.000
IS	0.962	1.359	1.359	0.962
LU	0.629	0.629	0.629	0.629
MG	0.968	1.116	1.116	0.968
SP	1.029	1.029	0.879	0.879
HOP	1.043	1.043	1.292	0.999
KM	1.161	2.242	2.242	0.908
SVM	0.711	0.711	0.982	0.711
UM	0.999	1.103	1.096	0.999
RS	1.000	1.000	1.514	1.000
SPC	0.789	0.789	1.038	0.789
Average	**0.942**	**1.177**	**1.298**	**0.910**

Table 6.3: Effectiveness of different machine-learning algorithms in mapping new workloads to sub-clusters

discrepancy in runtime compared to the optimal.

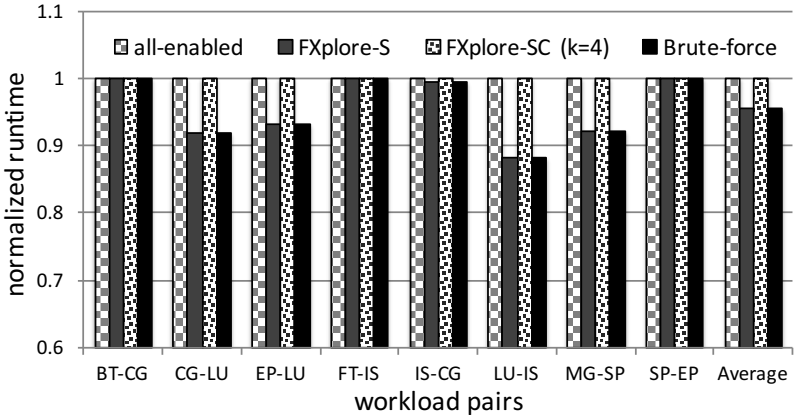

Figure 6.11: The average of co-located workloads under different firmware configurations, normalized to their average runtime under the baseline configuration.

6.4.4 Results with workload co-location

In this subsection, we show that *FXplore* works for mixture of workloads allocated on the same server. Note that we do not propose workload co-location or scheduling algorithm. But, rather we study the impact of *FXplore*-based mapping and firmware configuration when we have a mixture of workloads running on a server. To simulate co-located workloads in a realistic setting, we configure each benchmark to occupy four threads per server. Since the first step is to run a co-runner interference algorithm to identify good pairings, we profiled all possible combinations of NPB benchmarks to find, for each benchmark, which other benchmark offers least interference when co-located on the same server. By profiling all possible pairs, we identify the optimal pairs and reduce the "noise" that can be introduced in the experiment if a heuristic interference method is used. We have 8 pairs of benchmarks and run them on all the firmware configurations to evaluate the ability of *FXplore* to deal with a mixture of workloads. Figure 6.11 gives the average normalized runtime of the two co-located workloads under different firmware configurations, normalized to the runtime of benchmarks co-running under the baseline firmware configuration. The results show that *FXplore-S* can always find the optimal settings. If four subclusters are desired, then *FXplore-SC* results in only a 4.4% increase in runtime compared to the optimal case when each pair gets their own optimal configuration.

Compared to the case of a single workload, we have found that *FXplore* tends to enable more firmware settings when there are co-runners. The reason is that optimal pairing usually pairs workloads that have distinct characteristics to reduce interference; for example, a co-runner pair might include a CPU-intensive application and a memory-intensive application to reduce competition for hardware resources and have minimum interference with each other. However, pairing the workloads in this way will average their performance characteristics and make the combination of them both CPU-intensive

and memory-intensive, which leads to enabling more firmware options. However, in many cases, there are possibly subtle interactions between the firmware setting preferences and co-location interference. For example, it might be better to run a workload under its own optimal configuration with a sub-optimal co-runner, instead of scheduling it together with an optimal co-runner on a server with a sub-optimal configuration. A future direction is to find the best matching between servers and workloads after creating the heterogeneity.

6.5 Summary

Heterogeneity is a powerful capability in a cluster of servers. It allows workloads to fully exploit the potential of hardware. Unfortunately, commodity servers are expected to handle a diverse range of workloads, which makes it infeasible to customize hardware on these servers such that every workload's performance and energy-efficiency is maximized. In this paper, we demonstrated that firmware options provide relatively strong knobs to introduce soft heterogeneity in a cluster of servers. We showed that both the performance and energy consumption of workloads can be highly sensitive to the firmware settings. However, determining the optimal configuration is not an easy task, because of the exponential complexity in the number of configurations. Thus, we proposed *FXplore* to intelligently explore the firmware configuration design space and reach the optimal configuration with a fast exploration time. Our methodology involves two modes of operation: (1) a one-time offline mode that requires multiple server reboots to explore firmware settings and determine the server sub-clusters and (2) online mapping mode, where incoming workloads from a scheduler get profiled using PMCs and mapped to the sub-clusters wherein a group of servers share the same optimal firmware settings. We demonstrate that *FXplore* can improve runtimes by 11% and energy efficiencies by 15% in average, compared to baseline firmware configurations. It can accelerate firmware configuration exploration by $2.2\times$

compared to brute-force exploration, which is expected to be substantially higher as the number of options increase in emerging server. We also showed how *FXplore* can identify optimal configurations when co-runners are used. Furthermore, we evaluated the online mapping mode of ML techniques that map new incoming workloads, without requiring a reboot of any server, to existing sub-clusters with pre-determined firmware configurations.

Although some CPU-related hardware settings can be set through the OS and VMMs, many memory and storage related options could usually only be set from firmware. In this paper, we wanted to highlight that there is an opportunity beyond OS tuning to achieve heterogeneity in servers through the firmware. The fact that we can control some of the hardware-software options *via* the OS and VMMs makes our work even more relevant because we can get rid of some of the reboot overheads during the offline mode of *FXplore*. The overall algorithm, however, remains intact even when changing options through the OS or VMMs. Thus, *FXplore* brings us one step closer to realizing a heterogeneous cluster of servers out of an originally homogeneous set of servers with no additional costs or management overheads generally associated with equivalent hardware changes.

Chapter 7

Summary and Future Extensions

This dissertation presents several energy-efficiency optimization techniques for computing clusters. We proposed optimal solutions for cluster power budgeting, in both centralized and distributed fashion, addressing both computing power and cooling power. Substantial improvements are demonstrated with a large set of experiments. We also motivated novel computing cluster optimization opportunities such as layout planning and soft heterogeneity. Optimal and close-to-optimal algorithms for solving these problems are presented and evaluated. Section 7.1 summarizes our contributions. We discuss the potential future extensions in Section 7.2.

7.1 Summary of the Dissertation

In Chapter 3, we proposed an algorithm to find the optimal partition of the total power budget between the computing and the cooling power and accelerated it with heat cross-interference coefficient matrix. With a given computing power budget, we proposed an

optimal power budgeting algorithm for servers hosting heterogeneous workloads. The computing power budgeting algorithm allocates power cap to servers such that many system performance metrics can be efficiently optimized. The power budgeting system for heterogeneous workload makes decisions according to the workload characteristics. A throughput predictor is proposed to estimate the changes in throughput as function of potential changes to allocated power caps. The throughput predictor provides accurate throughput predictions under power cap. The optimal computing power budgeting algorithm, which is based on multiple-choice knapsack algorithm, can efficiently optimize the SNP of the cluster. A computing cluster with thousand nodes is simulated with our experimental setup. We demonstrate that our proposed method improves SNP and slowdown norm by 4.1% over previous methods. When considering fairness, our method achieves 51.7% improvement to uniform method and 90.0% improvement to greedy method.

In Chapter 4, we extend the computing power budgeting problem into very large-scale computing clusters with the power budget constraint, where the centralized method is not scalable and reliable. We proposed *DiBA*, a fully distributed framework to optimally allocate the computing power budget for each node locally. Each node exchanges its power consumption state to the neighboring nodes within a defined communication topology. The communication is very light-weighted and the algorithm converges fast. We evaluate *DiBA* with simulations. *DiBA* finds the optimal power allocation as well as the state of art centralized method, while outperforms the uniform power allocation by 14.5% in average. We simulated clusters with up to 6400 nodes and demonstrate that *DiBA* always converges in milliseconds which is $272\times$ faster than centralized method. For large scale clusters, the fast convergence and fully scalable feature of *DiBA* is particularly ideal for dynamic workloads as well as dynamic power budget with fast time scales.

Besides workload heterogeneity, server heterogeneity also can be exploited to improve

the system energy-efficiency. In Chapter 5, we motivated and formulated a novel optimization problem: the computing cluster rack layout planning problem for heterogeneous clusters. Heterogeneous racks offer different performance- power specifications. The objective is to identify the optimal positions of the server racks during the planning phase of the cluster such that the expected cooling power is minimized. Utilization patterns of servers and job scheduling mechanisms can significantly impact the minimum cooling power. We reformulated the rack layout planning problem to probabilistically identify the optimal layout that minimizes cooling power under different operating conditions. We solve the optimization problem using integer linear programming. For the experimental results, we devised a realistic model for clusters using state-of-the-art CFD modeling tools and demonstrated that our methods lead to significant improvements in the thermal characteristics and cooling power. We demonstrate the effectiveness of our approach in reducing total cooling power between $15.5\% - 38.5\%$ based on the cluster utilizations with an average of 23.3%

Heterogeneity allows job scheduler to fully exploit the characteristics of server hardware and workloads. Unfortunately, commodity servers are expected to handle a diverse range of workloads, which makes it infeasible to customize hardware on these servers to optimize the computing cluster energy-efficiency. The workload might change periodically so that the static customization is not flexible to adapt to new workloads. In Chapter 6, we motivated a novel approach of introducing soft heterogeneity through firmware reconfiguration. Firmware options provide configurable knobs to customize servers for target application in clusters. Both the performance and energy consumption of workloads can be highly sensitive to the firmware options. Firmware configurations can impact the runtime for 43% and power for 16% for given applications. We proposed *FXplore* to efficiently explore the firmware configuration design space. *FXplore* finds the optimal configuration with minimum amount of profiling. In this work, we demonstrated the advantage

in exploration time using five firmware options across a range of workloads. We show that *FXplore* can improve runtimes by 11% and energy efficiencies by 15% in average, compared to baseline firmware configurations. It can accelerate firmware configuration exploration by $2.2\times$ compared to brute-force exploration, which is expected to be substantially higher as the number of options increase in emerging server. We also proposed techniques to manage heterogeneity by partitioning the cluster as desired in subclusters, each with its own unique firmware configuration. We showed how *FXplore* can identify optimal firmware configurations when co-runners are used. Furthermore, we evaluated machine learning techniques that can map new incoming workloads into existing subclusters based on similarity with training workloads.

7.2 Possible Research Extensions

There are many other optimization problems in computing clusters. Along the domain of energy-efficiency optimization, two main categories of optimizations are natural extensions of our work. First, in this dissertation, we consider mainly the HPC workloads. The other important workload is latency-critical workload. Web service datacenters consists of multiple layers where each has different functionality and provides different services. Minimizing the power consumption of this type of infrastructure is a challenging problems. This is because, 1) the server nodes have various functionalities and have performance dependency with each others, modifying the status of one server will impact the performance of others; 2) the performance metrics, such as service level agreement (SLA), are much harder to be modeled as a function of power consumption. Secondly, incorporating energy-efficiency with job scheduling under many practical constraints forms more complex optimization problems. For example, in an heterogeneous cluster, scheduling jobs onto right server node to maximize the performance under power cap is an interest-

ing problem. As approximation techniques becomes widely used in various applications, coordinating the performance, power and quality of results would be an important optimization problem to investigate.

Bibliography

[1] F. Ahmad and T. Vijaykumar. Joint Optimization of Idle and Cooling Power in Data Centers while Maintaining Response Time. In *Proceedings of Architectural Support for Programming Languages and Operating Systems*, pages 243–256, 2010.

[2] H. Amur, K. Schwan, and M. Prvulovic. Towards Optimal Power Management: Estimation of Performance Degradation due to DVFS on Modern Processors. Technical Report GIT-CERCS-10-02, Georgia Tech, 2010.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. Technical report, 1991.

[4] L. A. Barroso and U. Holzle. *The Datacenter as a Computer*. Morgan and Claypool Publishers, 2009.

[5] J. Beckett. Bios performance and power tuning guidelines for dell poweredge 12th generation servers, 2012.

[6] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[7] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers*, 82(2):47–111, 2011.

[8] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar. The need for speed and stability in data center power capping. In *Proceedings of the 2012 International Green Computing Conference (IGCC)*, IGCC '12, pages 1–10, 2012.

[9] C. Bienia and K. Li. PARSEC 2.0: A New Benchmark Suite for Chipmultiprocessors. In *In Proceedings of the Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[10] B. Bollobás. *Random graphs*. Springer, 1998.

[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, 2001.

[12] Y. Cho, Y. Kim, S. Park, and N. Chang. System-level power estimation using an on-chip bus performance monitoring unit. In *Proceedings of the 2008 IEEE/ACM international conference on computer-aided design*, pages 149–154, 2008.

[13] R. Cochran, C. Hankendi, A. Coskun, and S. Reda. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *ACM/IEEE International Symposium on Microarchitecture*, pages 175–185, 2011.

[14] G. Contreras and M. Martonosi. Power prediction for intel xscale/spl reg/processors using performance monitoring unit events. In *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*, pages 221–226, 2005.

[15] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, 2011.

[16] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *ACM international conference on Autonomic computing*, pages 31–40, 2011.

[17] C. Delimitrou and C. Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Architectural Support for Programming Languages and Operating Systems*, pages 77–88, 2013.

[18] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 127–144, 2014.

[19] Dell. Dell poweredge r930 system owner's manual. 2015.

[20] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: Active low-power modes for main memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 225–238, 2011.

[21] EN. M. Elnozahy, M. Kistler, and R. Rajamony. *Power-Aware Computer Systems*. Springer, 2003.

[22] S. Eranian. Perfmon2: a flexible performance monitoring interface for linux. In *Proceedings of the 2006 Ottawa Linux Symposium*, pages 269–288. Citeseer, 2006.

[23] S. Eyerman and L. Eeckhout. A Counter Architecture for Online DVFS Profitability Estimation. *IEEE Transactions on Computers*, 59(11):1576–1583, 2010.

[24] X. Fan, W. Weber, and L. Barroso. Power Provisioning for a Warehouse-sized Computer. *International Symposium on Computer Architecture*, pages 13–23, 2007.

[25] G. Fursin, C. Miranda, O. Temam, M. Namolaru, E. Yom-Tov, A. Zaks, B. Mendelson, E. Bonilla, J. Thomson, H. Leather, et al. Milepost gcc: machine learning based research compiler. In *GCC Summit*, 2008.

[26] FutureFacilities. 6sigmaroom lite. http://www.futurefacilities.com/.

[27] A. Gandhi, M. Harchol-Balter, and R. Das. Optimal Power Allocation in Server Farms. In *International Conference on Measurement and Modeling of Computer Systems*, pages 157–168, 2009.

[28] J. Glanz. Power, pollution and the internet. *The New York Times*, 22, 2012.

[29] 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*, EuroSys '09, pages 317–330, 2009.

[30] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '11, pages 22:1–22:14, 2011.

[31] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, 2008.

[32] R. E. Grant and A. Afsahi. Characterization of multithreaded scientific workloads on simultaneous multithreading intel processors. In *In Workshop on Interaction between Operating System and Computer Architecture, IOSCA*, 2005.

[33] J. Heo, P. Jayachandran, I. Shin, D. Wang, T. Abdelzaher, and X. Liu. Optituner: On performance composition and server farm energy minimization application. *IEEE Transactions on Parallel and Distributed Systems*, 22(11):1871–1878, 2011.

[34] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, 2011.

[35] C.-H. Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the ACM/IEEE Supercomputing Conference*, pages 1–1, Nov 2005.

[36] N. S. Hussien, S. Sulaiman, and S. M. Shamsuddin. A review of intelligent methods for pre-fetching in cloud computing environment. In *Recent Advances on Soft Computing and Data Mining*, pages 647–656. Springer, 2014.

[37] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *International Symposium on Microarchitecture*, pages 347 –358, 2006.

[38] C. Lefurgy, X. Wang, and M. Ware. Power Capping: A Prelude to Power Shifting. *Cluster Computing*, 11:183–105, 2008.

[39] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, and R. Rooholamini. An empirical study of hyper-threading in high performance computing clusters. In *Linux HPC Revolution*, 2002.

[40] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010.

[41] N. Li and J. R. Marden. Decoupling coupled constraints through utility design. *IEEE Transactions on Automatic Control*, 59(8):2289–2294, 2014.

[42] S.-W. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou. Machine learning-based prefetch optimization for data center applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 56:1–56:10, 2009.

[43] S.-W. Liao, T.-H. Hung, D. Nguyen, H. Zhou, C. Chou, and C. Tu. Prefetch optimizations on large-scale applications via parameter value prediction. In *Proceedings of the international conference on Supercomputing*, pages 519–520, 2009.

[44] J. Liberman and G. Kochhar. Optimal bios settings for high performance compting with poweredge 11g servers, 2010.

[45] 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 International Symposium on Computer Architecture*, ISCA '14, pages 301–312, 2014.

[46] 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, 2014.

[47] D. Lo and C. Kozyrakis. Dynamic management of turbomode in modern multi-core chips. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*, pages 603–613, 2014.

[48] S. H. Low and D. E. Lapsley. Optimization flow control-I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[49] P. Maldikar. *Adaptive Cache Prefetching using Machine Learning and Monitoring Hardware Performance Counters*. PhD thesis, University of Minnesota, 2014.

[50] J. Mars and L. Tang. Whare-Map: Heterogeneity in "Homogeneous" Warehouse-Scale Computers. In *International Symposium on Computer Architecture*, pages 619–630, 2013.

[51] J. Mars, L. Tang, and R. Hundt. Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity. *IEEE Computer Architecture Letter*, 10(2):29–32, July 2011.

[52] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259, 2011.

[53] D. Meisner, B. Gold, and T. Wenisch. PowerNap: Eliminating Server Idle Power. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, 2009.

[54] D. Meisner, J. Wu, and T.F. Wenisch. Bighouse: A simulation infrastructure for data center systems. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 35–45, 2012.

[55] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

[56] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers. In *Proceedings of USENIX Annual Technical Conference*, pages 61–75, 2005.

[57] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. Minebench: A benchmark suite for data mining workloads. In *Workload Characterization, 2006 IEEE International Symposium on*, pages 182–188, Oct 2006.

[58] R. Nathuji, C. Isci, E. Gorbatov, and K. Schwan. Providing platform heterogeneity-awareness for data center power management. *Cluster Computing*, 11(3):259–271, September 2008.

[59] C. Patel and A. Shah. Cost Model for Planning, Development and Operation of a Data Center. *Hewlett-Packard Laboratories Technical Report*, 2005.

[60] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. Technical report, Rutgers University, 5 2001.

[61] D. Pisinger. Algorithms for knapsack problems, 1995.

[62] A. Coskun R. Cochran, C. Hankendi and S. Reda. Identifying the Optimal Energy-Efficient Operating Points of Parallel Workloads. In *ACM/IEEE International Conference on Computer-Aided Design*, pages 608–615, 2011.

[63] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "Power" Struggles: Coordinated Multi-Level Power Management for the Data Center. In *Architectural Support for Programming Languages and Operating Systems*, pages 48–59, 2008.

[64] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. Application-aware power management. In *IEEE International Symposium on Workload Characterization*, pages 39–48, 2006.

[65] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *International Symposium on Computer Architecture*, pages 66–77, 2006.

[66] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. de Supinski. Practical performance prediction under dynamic voltage frequency scaling. In *International Green Computing Conference and Workshops*, pages 1–8, 2011.

[67] J. Sartori and R. Kumar. Three scalable approaches to improving many-core throughput for a given peak power budget. In *International Conference on High Performance Computing (HiPC)*, pages 89–98, 2009.

[68] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[69] C. D. Spradling. Spec cpu2006 benchmark tools. *SIGARCH Comput. Archit. News*, 35(1):130–134, March 2007.

[70] B. Sprunt. The basics of performance-monitoring hardware. *IEEE Micro*, 22(4):64–71, 2002.

[71] B. Sprunt. Pentium 4 performance-monitoring features. *Ieee Micro*, 22(4):72–82, 2002.

[72] Q. Tang and S. Gupta. Thermal-Aware Task Scheduling for Data Centers through Minimizing Heat Recirculation. *Cluster Computing*, pages 129–138, 2008.

[73] Q. Tang, T. Mukherjee, S. K. S. Gupta, and P. Cayton. Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters. In *2006 Fourth International Conference on Intelligent Sensing and Information Processing*, pages 203–208, 2006.

[74] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller. Ship: Scalable hierarchical power control for large-scale data centers. In *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*, pages 91–100, 2009.

[75] J. Wilkes. More Google cluster data. Google research blog, November 2011.

[76] Q. Wu, Q. Deng, L. Ganesh, C. H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song. Dynamo: Facebook's data center-wide power management system. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture*, pages 469–480, June 2016.

[77] X. Zhang, S. Dwarkadas, G. Folkmanis, and K. Shen. Processor hardware counter statistics as a first-class system resource. In *USENIX Workshop on Hot Topics in Operating Systems*, HOTOS'07, pages 14:1–14:6, 2007.