1-1-2011

Qos-aware fine-grained power management in networked computing systems

Jiayu Gong
Wayne State University,

QOS-AWARE FINE-GRAINED POWER MANAGEMENT IN NETWORKED COMPUTING SYSTEMS

by

JIAYU GONG

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2011

MAJOR: COMPUTER ENGINEERING

Approved by:

_____

Advisor Date

_____

_____

_____

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor, Dr. Cheng-Zhong Xu. It has been an honor to work with him. I appreciate all his invaluable guidance, support, and encouragement throughout this work. The joy and enthusiasm he has for the research was contagious and motivational for me in the Ph.D. pursuit, even during tough times.

I am also grateful to my committee members: Dr. Song Jiang, Dr. Nabil Sarhan, and Dr. Hongwei Zhang for their time, interest, and helpful suggestions to improve this work.

The memebers in Cloud and Internet Computing (CIC) group have contributed immensely to my personal and professional time at Wayne State University. The group has been a source of friendships as well as great advice and collaboration. I am privileged for having Xiliang Zhong, Song Fu, Jia Rao, Bojin Liu, Bo Yu, Xiangping Bu, Kun Wang, Zhen Kong, Xuechen Zhang, Yizhe Wang, Yuehai Xu, Haikun Liu, Yaqin Luo, Cheng Wang, Yudi Wei, Duan Hu, and Mumtaz Dawoodi as my colleagues and friends.

Finally, I would like to thank my family for all their love, encouragement and understanding. I wish to thank my parents supporting me in all my pursuits. I wish to express my gratitude to my wife, Jing Qian, for her continuous love, understanding and support for these days and nights. Especially, the time she spared for me on our little Alvin (son) and from her own studies was precious.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Power is emerging to be the key challenge in networked computing systems, from real-time embedded systems to enterprise server environments. Meanwhile, a guaranteed level of Quality-of-Service (QoS) is required to be provided. Our work is motivated by the challenge of power management with demands of QoS guarantee in networked computing systems. In this chapter, we first introduce the background and motivation of our work in power management in networked computing systems. Then, we discuss the major design challenges and present the features of our solution. We will outline the rest of the dissertation in the end of this chapter.

1.1 Background and Motivation

Power management is one of the key design issues in networked computing systems. The motivation of the proposed research will be discussed from the perspectives of both real-time embedded and enterprise environments.

In the context of real-time embedded systems, energy is a critical resource, especially for the networked portable systems such as PDAs, pocket PCs, and cellu-

lar phones. On one hand, many popular applications designed for these systems consume significantly high energy, such as video processing and wireless communications. On the other hand, most of these devices are power by batteries which is finite. Limitations in battery capacities and demands for longer device operation time have motivated research for networked real-time systems, which can be interpreted as minimizing total energy consumption for a given workload. However, some wireless devices may rely on renewable energy sources. These devices often work in periodic cycles during which the energy collection is limited. In this case, the most important considerations is not to minimize energy consumption but to maximize the utility of energy available during each cycle [72].

The energy expenditure on networked real-time systems consists of two parts: the expenditure on the circuit board and the expenditure for wireless communication. With advances on semiconductor and circuit design, the power dissipation on circuit board is becoming less significant in the overall system power consumption. But the power consumption in wireless communications, especially the radio power components, does not benefit much from Moore's Law and has become a major source of power consumption in low-power networked real-time systems [87]. This motivates our work to be emphasized on seeking energy efficiency on wireless communication rather than on circuit board for networked real-time systems.

The concern of power consumption is not limited to the real-time embedded systems only. Such a concern rises in the enterprise server environments as well. Modern enterprise servers are featured with increasing performance capabilities accompanied by a dramatic rise in power consumption. For enterprise systems, online power management is necessary due to several significant reasons. First, the electricity bill keeps increasing in data centers and online power management can reduce this cost. By 2011, U.S. data centers will consume 100 billion kWh at a cost of $7.4 billion in

contrast to 61 billion kWh in 2006 [38]. Second, power provisioning is constrained by the cooling and power delivery limitations. It shows the nearly 60 Amps per rack provisioned currently in data centers could be a bottleneck for high density configuration [94]. Third, high power consumption in a data center can lead to tremendous environment pollutions. According to the U.S. Environmental Protection Agency (EPA), each 1000 kWh of energy consumption generates 0.72 tons of CO_2 emission. The CO_2 emission due to U.S. data centers in 2006 was estimated to be 44 million tons, which is equivalent to the output of 8 million passenger vehicles [38]. Additionally, power consumption can impact reliability and availability of the systems as well.

Thus, there are two key design issues existing for effectively operating today's enterprise servers. First, power consumption has to be capped within the capacity of the power supplies and cooling facilities in order to reduce operating costs and avoid system failures caused by power capacity overload or system overheating. Second, service owners need to be assured by meeting their required Service-Level Agreements (SLAs), such as response time and throughput. However, there are a few challenges in power and performance management in enterprise server environments. The challenge of power budgeting remains on how to dynamically plan system resource or transition the hardware components from high-power states to low-power states in a responsive and accurate manner whenever the system power consumption exceeds a given power budget. As the characteristics of workloads in servers are usually heterogeneous and time-varying, the impact on system performance due to power management varies greatly. Improving energy efficiency without significant effect on performance as possible becomes challenging. Hence, both performance and power management algorithms must be self-adaptive to workload variations for optimal overall system performance. With proliferation of virtualization, the density

of server configuration can be higher using consolidation. It leads to a more severe power provisioning problem. Furthermore, power and performance management becomes more challenging due to resource sharing and interference among co-residing virtual machines (VMs).

Although there are a number of existing work providing power management strategies to meet different requirements, the lack of a general framework of power management solutions motivates us to design and implement a power management middleware for networked computing systems.

1.2 Problems and Objectives

In this dissertation, we aim to build power-efficient networked computing systems. Nowadays, one general principle in system design is to design the systems based on their peak performance requirement. However, their capacities often exceed the average throughput demand and their busy time usually accounts for only a small fraction of the application time. For example, in [14], the authors analyzed average CPU utilization of a sample of 5,000 Google servers over a period of 6 months. Though the distribution of CPU utilization may vary across different clusters and workloads, a common trend is that that servers spend most of time within the 10-50% CPU utilization range and relatively little aggregate time at high utilization levels. This makes it possible for processor or transmitter slowdown to reduce the processing speed and still ensure no tasks miss their deadlines. Both processor speed slowdown and wireless transmitter slowdown save energy with the penalty of performance loss.

In real-time systems, applications are usually delay sensitive. More and more wireless real-time systems are being built with renewable energy resources. These devices often work in periodic cycles during which the energy collection is limited.

Due to the real-time and energy limitation, it is not always possible for a wireless node to deliver all data in its transmission buffer at a time. It is more desirable to transit more valuable data first in case a wireless node cannot send all its data. To quantify the level of importance of a packet, we associate a reward to each packet transmitted. Thus, the object becomes to maximize system rewards under given time and energy constraints.

The power consumption from circuit board is a major resource of power consumption of the entire system thus it affect the system-wide power consumption significantly. In server environments, processor is still one dominant energy-consuming components although its significance is decreasing. Dynamic Voltage and Frequency Scaling (DVFS) is the most effective slowdown technique to reduce power consumption by scaling the processor voltage and frequency when the system is not fully loaded. Ideally, processor slowdown leads to cubic power saving on processor with a linear performance loss. DVFS can thus regulate system-level power consumption effectively. Generally, system-level power consumption increases with system processing, networking and storage usage. It is difficult to plan the system usage in advance in a networked computing system due to the nature of variability. To regulate the power consumption within the power cap in an accurate, stable and responsive manner is necessary to reduce operating costs and avoid system failures without significant performance loss. As the existing black-box method may not be responsive over a variety of workloads, we provide responsive control over system-level power consumption using information of system behaviors.

On the other track, performance could be the first-class constraint to meet. Thus, servers are often over-provisioned significantly to meet target performance constraints under peak loads. However, this design leads to poor overall energy efficiency since servers are typically underutilized. Thus, power consumption is desirable to be re-

duced in order to improve energy efficiency. Soft real-time services require to maintain QoS quantile to a set point. The challenges rely on the difficulty on planning system usage to meet the set point while minimizing power consumption.

Additionally, it is also desirable to provide guarantees on both power and performance. As the characteristics of workloads in servers are usually heterogeneous and time-varying, it is required that power management scheme should be self-adaptive to workload variations for optimal overall system performance. Virtualization technologies bring Due to resource sharing of VMs, the techniques imposed on shared hardwares, such as DVFS, may not be applicable. Instead, re-allocating CPU resources by limiting processing time to VMs can both regulate power consumption and retain performance isolation.

We notice there is few work providing a general-purposed, practical and comprehensive power management middleware for networked computing systems. We develop a power management middleware, named PMM. PMM has the functionalities of power and performance monitoring, power management (PM) policy selection and control, as well as energy efficiency analysis. PMM is a general framework of power management middleware for different platforms, from servers to real-time embedded systems. We deploy PMM on Base Band Units (BBUs) to supply the gap of lack of power management solutions in BBUs.

1.3 Summary of Contributions

We have studied power management for wireless transmitters, enterprise servers, virtual machines, and Base Band Units (BBUs) under different power and workload models. The main contributions of this dissertaion are as following:

1. For real-time embedded system, we formulated the reward maximization prob-

lem in wireless networks under delay and energy constraints as a Multidimensional Multi-Choice Knapsack Problem (MMKP). We developed a branch-and-prune algorithm to solve this problem optimally in pseudo-polynomial running time. Then we proposed a heuristic approach, named Clustering to approximate the optimal solution with polynomial time complexity. Simulation results show the effectiveness of proposed optimal solution and Clustering algorithm. Meanswhile, the Clustering approach can closely approximate the optimal solution at a much smaller computational cost. The results were presented in [50].

2. Circuit board power consumption is still the major resource of system power consumption and needs to be managed. Thus, to provide a general-purposed and practical power management solution to networked computing system, we developed a power management middleware, named PMM, to regulate system power using circuit board actuators. A real-time power metering tool is developed and integrated in PMM. This tool can estimate system-level power consumption in a real-time and accurate manner with high accuracy. Meanwhile, it provides the information of power consumption of subsystems as well. Such a software-based power metering solution is crucial for the deployment of PMM in a large scale. We designed PMM using a client-server model. PMM has the functionalities of power and performance monitoring, power management (PM) policy selection and control, as well as energy efficiency analysis. A PM policy library is included in PMM to provide PM policies for different purposes. This library is flexible to extend. A prototype of PMM is implemented and deployed on BBUs. To the best of our knowledge, this middleware is the first software to provide comprehensive and practical solution to power management in BBUs. Three PM policies for different purposes have been included in the PM pol-

icy library of this prototype. These policies have been validated on different platforms, such as enterprise servers, virtual environments and BBUs.

3. In the enterprise environments, we proposed a system-level power estimation model for different CPU running speeds and a performance impact model for a web server by using performance events. Based on these two models, we developed a model-predictive feedback controller to control system-level power consumption while maximizing system performance. In contrast to the exiting feedback control, our proposed approach is more responsive by restricting more than 75% of power overloading settle to power cap within 2 control periods. Meanwhile, the system performance degradation is minimal. Details have been published in [48].

4. To coordinate power and performance in virtualized data centers, we developed vPnP, a feedback control based coordination system, to provide guarantees on a service level agreement (SLA) with respect to both performance and power. In this system, we proposed self-tuning model predictors to estimate the performance and power, respectively and a utility function optimizer to achieve different levels of tradeoff between power and performance. We implemented vPnP in a Xen-based infrastructure. The experimental results show the flexibitilty of vPnP to achieve different levels of tradeoff between performance and power. Our proposed vPnP system is more robust than existing two-layer feedback controller over a variety of workloads by reducing as large as 15% performance deviation. The results were published in [49].

5. To improve energy efficiency in BBUs, we proposed a power management strategy for controlling delay and minimizing power consumption using DVFS. We use the Robbins-Monro (RM) stochastic approximation method to estimate de-

lay quantile. We couple a fuzzy controller with the RM algorithm to obtain the CPU frequency for the receiver side BBU that will maintain performance within the specified QoS. This controller is non-workload-specific and self-adaptive.

1.3.1 Outline

The rest of this plan is organized as follows. Chapter 2 reviews the existing work of power management techniques for wireless transmitters, server environments and virtual machines.

In Chapter 3, we study the reward maximization problem for wireless real-time systems. We prove this problem is NP-hard. But by exploiting the properties of the reward optimization problem, we propose the optimal solution in pseudo-polynomial time. And we provide a sub-optimal approach with polynomial time complexity.

We developed a power management middleware in Chapter 4 for networked computing systems. It can be easily adapted to different platforms, such as servers and real-time embedded systems. This power management middleware has the functionalities of power and performance monitoring, power management (PM) policy selection and PM control, as well as energy efficiency analysis. More details of the PM policies in this middleware will be discussed in Chapter 5, 6, and 7, under different contexts.

In the context of enterprise environments, we develop a model-predictive feedback controller to control system-level power consumption in Chapter 5. This controller can dynamically adapt processor frequency to limit system-wide power consumption within power budget in a responsive manner.

In Chapter 6, we propose a feedback control based coordination system to provide guarantee in SLA with respect to power and performance in virtualized data centers. This coordination system includes two adaptive predictors to correlate CPU resource

allocation to power and performance. It employs an optimizer to find solution optimizing the utility function of power and performance.

In Chapter 7, We propose a power management strategy for controlling delay and minimizing power consumption using DVFS for BBUs. We use the Robbins-Monro (RM) stochastic approximation method to estimate delay quantile. We couple a fuzzy controller with the RM algorithm to obtain the CPU frequency for the receiver side BBU that will maintain performance within the specified QoS.

Chapter 8 concludes the dissertation with summary of our work and directions for future work.

Chapter 2

Related Work

This chapter reviews recent research work on power management in networked computing systems. In the context of real-time embedded system, we discuss the delay-constrained packet transmission in wireless communications by using packet transmission speed adaptation. In the enterprise environments, we give an overview of existing results on controlling system-level power consumption and system performance.

2.1 Transmission Speed Adaptation in Communication Systems

Power consumption in packet transmission can be reduced significantly by transmitting packets at a lower bit rate. Most existing studies on energy-efficient packet transmission focused on minimizing the energy expenditure subject to a time constraint. The time constraint can be in terms of average response time [16, 29, 91] and a single deadline [44, 104, 121] to all packets. Collins and Cruz proposed optimal transmission scheme in a fading channel with average delay constraint and a peak

transmitter power [29]. They used a simplistic channel model with two-state Markov chain and assumed the energy expediture is linear with transmitted data. Berry and Gallager considered the energy minimization problem with average buffer delay in a block-fading channel [16]. The energy minimization was turned into a convex optimization problem and dynamic programming was used to find the optimal solution. When the time constraint is considered as a single deadline for all packets, the indirect bound on packet delay requires all packets arrive before T to be transmitted no later than T. Uysal-Biyikoglu *et al.* [104] proposed off-line optimal and on-line near-optimal algorithms for a single transmitter-receiver pair. Same delay constraint was used in their later extension to multiple users [44].

However, both of the above time constraints do not put a limit to response time of individual packets, which may lead to unexpected large delay. To guarantee timely packet deliveries, it is more desirable to put a delay constraint to each packet [26, 67, 91, 98, 119, 124]. For example, Khojastepour and Sabharwal considered strict maximum delay constraint for each packet [67]. They established the connection between maximum delay scheduling and a linear filter for an i.i.d. input. Two optimal scheduling approaches were proposed. One is a time-variant policy which makes scheduling decisions according to each new packet arrival and uncompleted arrivals in the queue backlog. Zhong and Xu [124] studied delay constrained packet scheduling with a focus on quality of service control. They derived relation between maximum packet transmission rate and packet arrival patterns so as to provide statistical response time and packet drop control.

There are a few existing work studying packet transmission with individual delay constraints to guarantee packet transmission time. A general scenario could be a wireless transmitter communicates with multiple receivers. The transmitter generates data periodically and send the data to corresponding receivers. The wireless trans-

mitter is powered by renewable energy sources such as solar panel. As a result, the transmitter needs to finish packet transmission subjected to both delay and energy constraints. Due to the constraints, it may not be able to send all the data. To ensure that the most valuable information is transmitted to the receivers, a reward (value) is associated to each packet. The objective is reduced to maximizing the total reward under the time and energy constraints. This kind of problems have been investigated in existing studies [109, 43, 110, 121].

Wang and Mandayam [109] tried to maximize throughput in a block fading channel under time/energy constraints. They considered a binary rate control with only an on/off model in which the transmitter either transmits with a constant power or remains silent. Under the same model and constraints, the authors later considered the transmission of a fixed size file with the objective to maximizing the probability of successful delivery of the entire file [110]. Fu *et al.* tried to maximize throughput in a fading channel by considering the effect of variable transmission rates [43]. All the three studies are limited to a single-transmitter-single-receiver scenario. Zhang and Chanson [121] presented optimal scheduling algorithms for the reward maximization problem in a scenario of multiple receivers under given time limits and energy constraints. They assumed all packets are ready to transmit at beginning time and this assumption is not applicable to the periodic data model where messages are released in a regular interval. In addition, they assumed a wireless transmitter with continuous transmission rate while it may only work at a limited number of rate levels in practice.

2.2 Power Management in Enterprise Environments

We will first review the approaches for power consumption measurement, modeling and profiling. Then we introduce power management mechanisms. After that, we discuss the related work from the perspectives of objectives, scopes and methodologies.

2.2.1 Power Consumption Measurement, Modeling and Profiling

External power meters (analyzers) connected with the power source can be used to measure power consumption online. External power measurement may not achieve a high sampling rate. For example, WattsUp Pro [3] measures RMS power with a sampling rate of 1Hz and Extech 380801 [4] has a sampling rate of 2Hz. In some today's blade, a power measurement circuitry is employed. This kind of in-blade power monitor firmware can measure the blade's power with a sampling rate of as large as 1000Hz [70]. The power consumption can also be measured by employing resistors connected with the power source and capturing voltage drop across the resistor using data acquisition (DAQ) hardware.

In addition to measuring the power consumption of a computer in total, we may also need to know where and when the power/energy has been spent [15]. A variety of power consumption profiling techniques have been proposed.

Simulation has already employed in profiling power consumption of microprocessors [21], memories [20], disks [88], and system-level [51]. However, the main drawbacks of simulation approaches rely on that the simulation models cannot match the real systems accurately and simulation itself is time-costly.

Online power measurement can profile the system-level power consumption [70,

83, 111]. Bircher and John [18] measured power in five subsystems by employing resistors connected in series with the power source. The voltage drop across the resistor is captured by data acquisition (DAQ) hardware with a sampling rate of 10000Hz. However, the online power measurement may not be practical as it needs dedicated measuring equipments for each machine or physical decomposition of a machine.

Performance Monitoring Counters (PMCs) can assist power consumption profiling and modeling, which can be traced back to [15]. Contreras et al. [31] proposed a linear power estimation model using hardware performance counters to estimate run-time CPU and memory power consumption using Intel PXA255 processor. The number of counters used is more than that the processor can provide concurrently thus this model is not feasible for online power estimation. Fan et al. [40] proposed a linear model considering CPU utilization and an empirical term which minimize squared root to improve accuracy. Heath et al. [52] presented a linear model using disk utilization in addition to CPU utilization. Both models ignore the impact on power consumption from other subsystems. Bircher and John [18] proposed power models for entire systems and subsystems using PMCs. It needs to decompose the measurement of power for each subsystem physically. Economous et al. [36] presented Mantis providing a full-system power prediction model by adding CPU performance counters to the OS-reported CPU and disk utilization. Notice although PMCs are embedded into most current commodity processors, the number of hardware counters that can be counted concurrently is quite limited without multiplexing technique.

2.2.2 Power Management Mechanism

Essentially, power management is done by transitioning hardware components back and forth between high and low-power states or modes. The components are fully active and operational in high-power mode while the functionality associated with the low-power modes depends on the particular component. Switching between power modes may introduce non-negligible overhead in terms of both energy and performance [55].

Multiple classes of execution states are supported in today's server processors for the purpose of power management. These states include the frequency and voltage (P-state) in active mode, sleep states (C-states) in idle time, and throttle state (T-state). Dynamic Voltage and Frequency Scaling (DVFS) is used to switch among different P-states. DVFS relies on the fact that the dynamic power consumed by microprocessor is a cubic function of its operating voltage. Thus, reducing the operating voltage/frequency provides substantial saving in power at the cost of slower execution [105]. Most of today's processors have well-documented interfaces for DVFS, such as AMD's Cool'n'Quiet technology [10] and Intel's Enhanced SpeedStep technology [59]. However, the number of voltage or frequency stages is very limited. In multi-core processors, it is not flexible to manipulate P-states due to the dependencies among the cores residing in the same die. T-states can further throttle down a CPU by inserting stop clock signals and thus omitting duty cycles. The mechanism to enter different T-states is to manipulate the processor clock modulation setting (throttling) by modulating the duty cycle of the processor clock, which changes the effective frequency of the processors [60]. Sleeping states (C-state) can be utilized when the CPU is idle. In ACPI standard, C0 is the active state and the sleep states are called C1, C2, ..., Cn [55]. The deeper C-state is, the more power the processor can save. C-states can

cause relatively large switch overhead and might not be effective when the system is not idle but in a low utilization state.

Current disks also enable power management by deactivation. In active mode, the disk is being actively used and consumes more power. In idle mode, the disk can still spin at its regular speed and accesses can be performed without delay. In low-power mode, relatively high transition overhead will be involved, such as turning the spindle motor off (standby) and turning the disk interface off (sleep) [28, 90]. Multi-speed disk [22] can also be employed to manage power consumption of disk subsystems.

Power dissipated by a memory subsystem largely depends on its capacity and bus frequency. In practice, the power consumed by periodic refresh is very small. Most of the power is consumed by row and column decoders, sense amplifiers, and external bus drivers due to large arrays with very long and high capacitance internal bus lines. To reduce power consumption, one or more of these subcomponents can be disabled by switching a device to one of several pre-defined low-power states when it is not being actively accessed. Memory controllers and chipsets can switch the subcomponents to low-power states [62] or switch a memory rank's power on and off [86]. The non-negligible performance penalty, called re-synchronization cost, is incurred to transition from current low-power state to an active state before access. In addition, multi-frequency memory can dynamically scale the working frequency and consequently the data rate [58].

At system-level, the entire computer can be managed as active, sleep (suspended, hibernated), and power-off states with time and energy overhead for the transitions between these states [55].

In virtualized environments, although P-states are still useful to regulate power consumption when they are enabled, problems might occur when manipulating P-states. Since multiple virtual machines may share a single core, tuning P-states of

a core could threaten desired performance isolation. T-states can further throttle down a CPU by inserting stop clock signals and thus omitting duty cycles. However, T-states are not always well documented; access to the T-states may need to modify the clock modulation register. In contrast, C-states can be utilized when the CPU is idle. But it incurs relatively large switch overhead and might not be effective when the system is low utilized. Instead, in virtualized environments, re-allocating CPU resources, by limiting processing time, to virtual machines can both regulate power consumption and meanwhile retain performance isolation brought by virtualization. This functionality is provided by a hypervisor scheduler, such as Credit Scheduler [2] in Xen [13]. One more important power management mechanism introduced by virtualization is virtual machine migration. Virtual machine migration lead to power consumption migration. Consolidation enabled by migration can idle servers so that the idle servers can enter low-power states for more power saving.

2.2.3 Power Management Scope

A lot of existing work focused on how to reduce power consumption by improving the energy-efficiency of individual server components [69], from processors [54], memory [39, 86], to disk [126].

At system-level, Lu *et al.* [76] presented a power reduction technique at OS-level using task-based power management. Zeng *et al.* [120] proposed to build an explicitly energy-aware operating system by introducing a system-wide abstraction for the energy used. Both work budget the energy available to individual processes. Felter *et al.* [41] studied *power shifting*, a technique to reduce peak power with minimal performance impact that is based on dynamically re-allocating power to the most performance critical components. Carrera *et al.* [22] focused on the disk subsystems,

proposed four disk energy management techniques which includes combining laptop and server-class disks and the use of multi-speed disks.

Previous work [19] shows that the power consumption of processors is usually the dominant factor of power in servers. This is particular true in dense blade server environments. Thus, a large number of research work on system-level power management manipulate system power by DVFS or processor throttling. Elnozahy *et al.* [37] presented a soft real-time feedback control-based DVFS policy combined with request batching. Sharma *et al.* [99] investigated adaptive algorithms for DVFS in QoS-enabled web servers to minimize energy consumption subject to service delay constraints. The control is done at session-level by using synthetic utilization bounds as control set point for DVFS. The authors also address the impact on energy and performace due to turning on/off a server in addition to DVFS. In [81], a feedback controller is developed to manage the average power consumption of a laptop to prolong battery lifetime by DVFS. This study relies on experiments to find the best control parameters regardless of current system usage which can affect power consumption. Lefurgy *et al.* [70] presented a technique to control the peak power consumption of a server using a feedback controller using processor throttling and precise, system-level power measurement. This controller is based on the nominal model without awareness of system behavior, which may lead to a longer settling time in a dynamic system.

At the cluster or datacenter-level, the goals of power management can be categorized into two classes: improving energy-efficiency and power capping. These two categories of work will be discussed later.

2.2.4 Power Management Objectives

In general, there are two types of objectives in power management. One focuses on average power optimization by minimizing the power needed to achieve the performance target. This can be translated to a tracking problem which means the consumed power should track the resource demands of the applications. This optimization problem can be translated as improving energy efficiency. Power management for improving energy efficiency can be conducted for different objectives: minimizing energy consumption with performance guarantee, maximizing performance under power budget, and making tradeoff between power and performance. The other category emphasizes on peak power in order to optimize the power provisioning delivery and cooling in data centers. This is essentially a capping problem that ensures the power consumption of a system will not violate a power budget.

Energy Minimization with Performance Guarantee

To improve energy efficiency of a data center, a popular design methodology is to minimize the energy consumption for processing applications. From data center administrators' perspective, a primary concern is application performance, which can be represented as Quality of Service (QoS), Service-Level Agreement (SLA), etc. It is required to meet performance target and meanwhile minimize energy consumption.

Early power management study focused on power-on/off scheme. Pinheiro *et al.* [89] proposed a load concentration strategy to manage cluster-wide power consumption, in which the nodes in a cluster are turned on or off to ensure the expected performance is just about acceptable according to the workload.

Recently, DVFS became widely employed because it could provide significant energy saving while avoiding the comparatively large switching overhead between power-

on and power-off states. Sharma *et al.* [99] investigated adaptive algorithms for DVS in QoS-enabled web servers to minimize energy consumption subject to service delay constraints which can be represented by different deadlines for different client classes. Further, for multi-tier server clusters, for example, a 3-tier web server Horvath *et al.* [56] presented a coordinated DVFS strategy.

Horvath *et al.* extended the work [56] to combine both DVFS and power-on/off mechanism in a cluster with dynamic configuration [57]. They made performance and power tradeoff decisions using end-to-end delay as a simple SLA metric and took into account the overheads for each transition between multiple sleep modes and standby power levels.

It is observed that the portion of power due to processors is small in comparison with total system power in some recent clusters. There are still a number of work using power-on/off mechanism only due to the non-trivial power consumption for active idle servers. Chen *et al.* [24] proposed a dynamic provisioning technique to turn on a minimum number of servers required to satisfy application-specific SLA with consideration of time taken for turning on/off a server and load dispatching algorithms.

In virtualization environments, the resource allocation, such as CPU, memory and disk I/O, to each virtual machine should be taken into account as well in order to achieve performance guarantee. Wang *et al.* [113] proposed a two-layer control architecture to provide response time guarantees for virtualized enterprise servers. The primary control loop uses a multi-input-multi-output (MIMO) control over CPU resources to balance load among virtual machines so that they can achieve roughly the same normalized response time. The secondary loop controls the normalized response time of all virtual machines to a desired level by DVFS for power efficiency.

There are a few work studying statistical QoS guarantee for power management

schemes [118, 56, 71]. Both [56, 71] consider multi-tier web servers enabling DVFS. However, in [56], the statistical performance guarantee is achieved by a queueing model which requires *a priori* knowledge of workload arrival rate and service rate. In [71], stochastic approximation is coupled with PID feedback controller to guarantee delay quantile of an unknown distribution. In [118], the statistic correlation between resource allocation and QoS was studied but this work was limited to the *a priori* workload knowledge and simulation.

In addition to control-theoretic approach, there are few work in an attempt to solve this problem using direct methods, such as reinforcement learning. For example, Tan *et al.* [102] proposed an approach to learn the power management policy to minimize power consumption for a given performance constraint by RL in a model-free manner. We still believe the indirect methods should be more efficient for this kind of problems.

Performance Maximization under Power Budget

In addition to reducing energy consumption, another energy-efficient design obejective is to control power consumption to adapt to a given power budget so as to reduce the power (then the performance) of the components when actual power consumption of the server or cluster exceeds the budget [70]. As a result of controlling power consumption, the performance should be maximized without using power more than the budget.

For example, given a budget of power, Gandhi *et al.* [45] studied the problem of how to allocate power among a server farm so that the performance can be optimized. A queueing theoretic model is developed to predict the optimal power allocation in a variety of scenarios. The optimal power allocation scheme depends on many factors, such as power-to-frequency relationship, the arrival rate of jobs, etc.

Most of the work of performance maximization under power budget have overlap

with the solutions to power capping, which will be discussed later.

Tradeoff between Power and Performance

In data centers, it is required to enable resource provisioning in accordance with flexible SLAs that specify dynamic tradeoff between performance and cost that can be translated as power consumption. Indirect methods are mostly employed towards this purpose.

The work [23] by Chase *et al.* is one of the earliest focusing on energy-conscious cluster-wide resource management. It is based on an economic model in which the amount of resource is a function of service quality.

Chen *et al.* [27] used SLA to direct tradeoff between energy and performance as well. Unlike [23], they considered power-on/off as well as DVFS. They proposed three online strategies based on steady state queuing analysis, feedback control theory, and a hybrid mechanism of both.

Kephart *et al.* [66] proposed an approach to address tradeoff between power and performance, which designed an agent to deal with each aspect of system behavior, such as power, performance and availability. Such a multi-agent approach employs a utility function defined as a joint of power and performance for tradeoff decision.

Kusic *et al.* [68] presented an online resource provisioning framework for combined power and performance management with consideration of switching costs incurred when provisioning virtual machines. The authors formulated this management problem as one of sequential optimization under uncertainty and provided a limited lookahead control approach. This work relies on the workload-specific performance models requiring *a priori* workload knowledge.

On another track, virtual machine migration provides an alternative way to save energy by consolidation which reduces number of hosts. Verma *et al.* [106, 107, 108]

investigated static/semi-static/dynamic job placement to achieve different goals in power and performance, such as power minimization and performance maximization, enabled by virtual machine migration.

Power Capping

There are a few work that proposed control-theoretic approaches for power capping at system-level [70, 48]. Wang and Chen [111] extended the work [70] from single server to multiple servers. A Multi-Input-Multi-Output (MIMO) control algorithm is developed to control multiple servers simultaneously. In each control period, the controller collects the power measurement and CPU utilization of each server, computes a new CPU frequency for each server, and directs the servers to scale CPU frequencies in a coordinated manner.

At cluster-level, Femal *et al.* [35] noticed the uneven distribution of workload among nodes. Thus they allocated power non-uniformly according to demands. A cluster-level manager collects information of all nodes and allocates power to each node while satisfying total power budget. A node-level manager allocates power to each device at a fine level of granularity.

Further, the long-term data from real-world servers have been studied in [94]. Two trends were summarized: 1) low resource utilization with infrequent and short-lived bursts; 2) the probability of synchronized spikes on all servers at the same time is rather low. Based on these observations, Ranganathan *et al.* [94] proposed a power budgeting approach across an ensemble of servers by leveraging statistical properties of concurrent resource usage.

In virtualized environments, the power capping problem is also addressed. Multi-layer control approach is widely used in this scenario. Nathuji *et al.* [82] developed a two-layer feedback controller for power budgeting with QoS management in virtual-

ized servers. One loop monitors power consumption and determines a platform-level CPU allocation to meet power budget. The other loop is distributed across virtual machines to bid resource based on shadow price for each virtual machine. In [112], both power budget and performance are guaranteed. The power consumption is constrained in a cluster-level power control loop by scaling CPU frequency. The performance guarantee is achieved by allocating CPU resources among virtual machines in a performance control loop.

2.2.5 Power Management Methodologies

Power management methodologies can be categorized into two classes: indirect methods and direct methods.

Indirect Methods

An indirect method assumes an explicit model to capture the behaviors of a target system. This method is model-based. It mostly relies on a system identification procedure to build analytical models of the controlled system and determine the control rule from the model. Such models can abstract the power consumption and performance as analytic functions on a set of system and application parameters. These model can even be used for prediction. Base on the models, both optimization and control approaches can be applied for power management. The most widely-used indirect method is the control-theoretic approach. Control theory provides a powerful mechanism to handle disturbances, uncertainties, and unpredictable changes in systems using feedback [127].

Lu *et al.* [77] described a formal feedback control algorithm combined with DVFS for multimedia systems. Zhu and Muller [128] combined feedback control theory with

DVFS schemes for hard real-time systems with dynamic workloads. Both work are on the processor level without considering system-level power consumption.

Sharma *et al.* [99] applied control theory to control application-level quality of service requirements. Chen *et al.* [27] developed a controller to control SLA, which is response time, in a server cluster. Both of these work provide no guarantee to the power consumption of a computing system.

Single-input-single-output (SISO) controller is mostly used when there is only a single control output employing control over a single resource. Lefurgy *et al.* [70] designed a Proportional (P) controller to cap the peak power of a server by throttling processor. Wu *et al.* [116] managed power by controlling the synchronizing queues in multi-clock-domain processors. Zhang *et al.* [122] adjusted the resource demands of virtual machines based on resource availability.

Multi-input-multi-output (MIMO) control can be employed when there are multiple control outputs in power management for enterprise servers, which might need control over multiple resources. The MIMO control can be translated to an optimization problem. Extending the power capping technique in [70] from a single server to a cluster, Wang *et al.* [111] developed a MIMO control algorithm for cluster-level power control in a non-virtualized environment. Kusic *et al.* [68] presented a power and performance management strategy using lookahead control. In addition, Kephart *et al.* [66] have proposed a coordinated management scheme to achieve tradeoffs between power and performance by optimizing a utility function for a non-virtualized server.

Multilayer control can be applied to address similar problems. Representative work includes [113, 82, 112], which employ two-layer feedback controller. In [113], one loop controls the CPU resource to each virtual machine to guarantee performance while the other one controls CPU frequency for power efficiency. In [82], one loop lim-

its the power consumption and the other one bids resource based on shadow price for each virtual machine. However, the work in [113, 82] are either performance-oriented or power-oriented without explicit coordination of power and performance. In [112], power and performance can be coordinated using a two-layer feedback controller depending on off-line system identification.

Model Predictive Control (MPC) is quite useful when the control action has dead times, such as performance impact of an application due to resource re-allocation. Xu *et al.* [117] presented an end-to-end QoS provisioning framework to monitor and control user-perceived QoS. Both work rely on queueing model. To manage power consumption of enterprise servers, Wang *et al.* [113] designed a MIMO power control algorithm based on MPC control theory. Kusic *et al.* [68] used Limited Lookahead Control (LLC) to formulate models for the cost of control.

Adaptive control copes with the dynamic of a system by modifying the control law. To meet the performance SLA, a self-tuning admission controller was designed for a 3-tier web sites in [64]. In [115], a self-tuning fuzzy controller was proposed to guarantee client-perceived end-to-end QoS. To manage resource in virtualized environments, Padala *et al.* [85] proposed an adaptive estimator to capture the relationship between allocated system resource and performance. In [63], the Kalman filter was integrated into feedback controllers to dynamically allocate CPU resources to virtual machines.

There are also a number of challenges when applying control theory to power management. Modeling is difficult because most inter-relationships in the system are non-linear. Modeling itself requires system identification which may not cover all relevant correlations. Classical control theory only deals with continuous inputs while the input variables for power management can only take on discrete values. Though adaptive control can deal with dynamic, a limitation still exists on how fast the workloads or system behaviors can change. In addition, using dynamic models

may not provide theoretic guarantee on the properties of controllers.

Direct Methods

In contrast, direct methods determine control rules without needs for an explicit model of a system. One representative example is reinforcement learning (RL), which learns the impact on system behaviors, such as performance and power consumption, due to the action taken on the system, such as power state change. Compared with control theory, reinforcement learning is model-free. RL is fundamentally a sequential decision theory that properly treats dynamics in the system. It can improve decision-making policy over time, similar to adaptive control [101].

Rao *et al.* [95] proposed a reinforcement learning based approach for virtual machine configuration which is adaptive to heterogeneous virtual machines. Tesauro *et al.* [103] presented a reinforcement learning approach to developing effective control policies for real-time power management in application servers. Tan *et al.* [102] proposed a online power management technique using reinforcement learning. The best power management policy can be learned without knowing the workload information *a priori*.

However, there are also a few challenges in reinforcement learning. One is the tradeoff between exploitation, which is to select from what it already knows, and exploration, which is to make better action selection from unknowns in the future. Performance of the initialization phase that explores without any knowledge during live online training may be unacceptably poor. The convergence is hard to prove and the convergence rate is low. In addition, RL can suffer from poor scalability in large state spaces.

Chapter 3

Maximizing Rewards in Wireless Networks with Energy and Timing Constraints

Energy is a critical resource of wireless devices powered by battery with limited capacity. Reliable content delivery over a wireless channel is a major source of energy expenditure. The increasing wireless transmission rate results in an exponential increase of the energy consumption of wireless devices [16]. However, applications are usually delay-sensitive. With renewable energy resources, such as solar power, wind power, and mechanical power from the environments, wireless nodes are subjected to limited amount of energy collected in each period. Meanwhile, wireless nodes powered by renewable battery often generate a significant amount of data during each cycle. Due to the limitation of energy and delay, it is often impossible to deliver all the data generated during each period. Instead, the data with higher level of importance should have high priority to deliver. A reward is associated to each packet to qualify the level of importance. In this chapter, we will present our solutions to reward

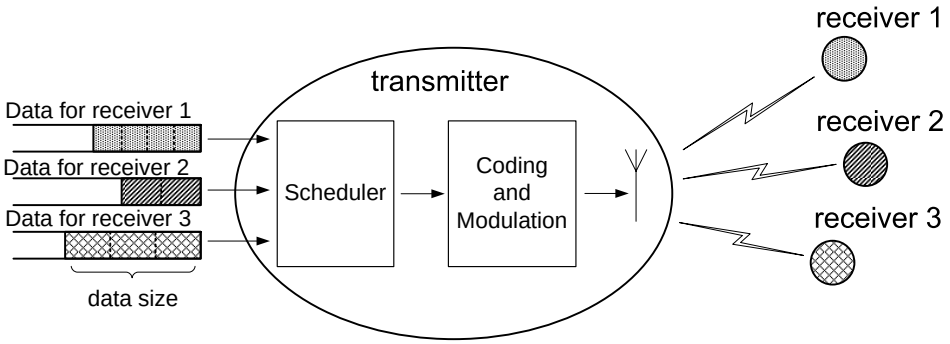


Figure 3.1: One-transmitter-multiple-receiver model in a single-hop wireless network.

maximization problem in a single-transmitter-multiple-receiver wireless networks with energy and delay constraints.

3.1 System Model and Problem Formulation

In this part, we first define data and energy consumption models for the reward maximization problem. Then we present a formulation of the reward maximization problem under given time and energy constraints.

3.1.1 Data Model

Early studies of energy-efficient problem in wireless networks were largely targeted at communication channels over a single-transmitter-single-receiver model; see [29] [16] [104] [43] for examples. A single-transmitter-single-receiver model is also known as point-to-point communication where there is only one transmitter which will communicate with a single receiver. In recent years, we have seen the extension of the studies to a more general single-transmitter-multiple-receiver model [44] [121] [84] [125] in which a wireless transmitter communicates with multiple receivers periodically, as shown in Fig. 3.1. In this model, the transmitter can only communicate with

one receiver at a time and has an energy budget in each transmit cycle. Each receiver will receive data from the transmitter periodically. Every transmitter-receiver pair has a maximal amount of data to be transmitted in each time period. The receivers are located with distances from the transmitter. The data to different receivers can be transmitted at different transmission rates.

We regard the transmission between each transmitter-receiver pair as a periodic data stream and refer this as a task. It is a sequence of packet transmissions with the same characteristics that occurs at a regular interval. We use $Task = \{\tau_1, \tau_2, \ldots, \tau_N\}$ to denote the set of N tasks. The number of tasks is always equal to the number of receivers in our model. All the transmission tasks are assumed to be independent and preemptive, scheduled by the Earliest Deadline First (EDF) policy [74].

EDF is a popular scheduling policy for delay sensitive wireless packet scheduling [42][119]. We choose EDF as the underlying policy because it has been shown to be optimal for deadline constrained scheduling over optimal link conditions under various modeling assumptions [47][75]. The preemption can take place between transmissions of packets of different tasks. Practically, it can be implemented in a transmitter by re-arranging packets in the transmission buffer by choosing those with the earliest deadlines. Similar preemptive model was assumed in [42][47][100].

We characterize a task τ_i by a tuple $\{C_i, T_i, D_i, P_i, R_i\}$, where C_i denotes the size of data to be transmitted, T_i is the task period, P_i and R_i are power function and reward function respectively. The relative deadline D_i is assumed to be equal to task period T_i. We assume for each data stream i, there are several discrete levels of sizes of data to be transmitted, denoted by $\{c_i^1, c_i^2, \ldots, c_i^K\}$. We have $C_i \in \{c_i^1, c_i^2, \ldots, c_i^K\}$, which means the actual amount of data to be transmitted is not always the maximum and should be determined due to time and energy constraints. In general, the data stored in the transmit buffer can be either generated by local host or forwarded from

other nodes. Different data may have different priorities and rewards related to the corresponding receiver and data size. For notional brevity, we use τ_i^j to represent the task when data c_i^j is selected for transmission.

The transmitter has a set of discrete levels of transmission rates. We define the set of available transmission rates of a transmitter as $Speed = \{s_1, s_2, \ldots, s_M\}$ in which the available rates are indexed in an ascending order.

3.1.2 Power Consumption Model

The power consumption of a wireless transmitter can be divided into two parts: circuit power and transmission power. The transmission power usually dominates since long-range communications (over 100m) are common in wireless networks. In order to maintain the same transmission rate, the required transmission power needs to increase with the distance between the transmitter-receiver pair to offset the propagation loss. In addition, the circuit power is expected to decrease as the IC technology advances. This part of power only occupies a small portion of the whole power consumption. So in our work, we assume the transmission power dominates the negligible circuit power.

In our power model, we assume the channel is slowly time-varying, which means the channel condition will not change during transmission. Proper channel coding can reduce the energy consumption effectively during transmission. We take the Additive White Gaussian Noise (AWGN) channel model as an example, which explains how energy, rate, and data size are related. With optimal channel coding, the maximum transmission rate is [32]:

$$S = \frac{B}{2}\log_2(1 + \frac{P'}{N_0 B}), \tag{3.1}$$

where S is the transmission rate, P' is the received signal power, N_0 is the spectral density, and B is the channel bandwidth. From this equation, we can describe the relationship between the transmission rate S and the received power P' by the following equation:

$$P' = N_0 B \cdot (2^{\frac{2S}{B}} - 1). \tag{3.2}$$

As we aforementioned, the power will increase with distance between transmitter and receiver in order to maintain the same transmission rate. Considering this power attenuation, we have:

$$P = \frac{P'}{A} = \frac{N_0 B}{A} \cdot (2^{\frac{2S}{B}} - 1), \tag{3.3}$$

where P is the transmission power and A is the attenuation factor for the transmitter-receiver pair. The attenuation factor A is generally inversely proportionally to a function of the distance, denoted by l. For example, this function could be a square function, $A \propto 1/l^2$, in [32]. In this work, we do not assume any specific form of the relationship between attenuation factor and distance except that all transmitter-receiver pairs have the same fading functions which are only affected by distance. It is easy to see that the required transmission power P is strictly increasing and strictly convex in the transmission rate S. This power function $P(S)$ is continuous

in S though we only consider the discrete cases for this function in this work.

Let P_i denote the power consumption function for task τ_i. Let C_i and S_i represent the size and rate of data transmission for τ_i, respectively. The transmission time to transmit data C_i equals to $\frac{C_i}{S_i}$. Therefore it consumes $P_i(S_i)\frac{C_i}{S_i}$ units of energy. The energy consumed for τ_i for transmission in one period, denoted by E_i, with data size C_i at transmission rate S_i becomes

$$E_i(C_i, S_i) = P_i(S_i)\frac{C_i}{S_i} = \frac{N_0 B}{A_i} \cdot (2^{\frac{2S_i}{B}} - 1) \cdot \frac{C_i}{S_i}, \tag{3.4}$$

where the coefficient A_i for each transmitter/receiver pair differs depending on the distance between them. Similar power models were defined in [121] [125], as well. As the channel states and receiving nodes are assumed to be static during the transmission period, the power attenuator factor A_i is also static. We fix the bandwidth B and assume it is the same for all the streams. To simplify the problem, we assume the overhead of switch among different transmission rates is tiny and can be ignored.

3.1.3 Problem Formulation

We consider the transmission in a hyper-period T which is defined as the Least Common Multiple (LCM) of task periods $T_1,\ T_2, \ldots,\ T_N$. The consideration of a hyper-period ensures all tasks can finish their periodic transmissions at least once. Let E_{max} represent the units of energy budget allocated to the transmitter during this hyper-period T. Our objective is to maximize the total reward while all tasks meet their deadlines and the total energy consumption does not exceed the budget E_{max}. In other words, the optimization problem in this work is to find a speed and a data size for each task to maximize the overall rewards while satisfying delay and

energy constraints.

The reward R_i can be a function of multiple variables such as data size, transmission rate, etc. In this work, we define reward function R_i as a generic function of data size c_i^j to be transmitted, represented by $R_i(c_i^j)$. It is conceptually the same as the reward functions in [11][33] and the utility function in [92]. In this work, however, we do not assume any specific form of the reward function. It can be reduced to different forms in different application contexts. In its simplest form, the reward function R_i can be interpreted as the amount of data transmitted, with respect to the data size; the reward maximization problem is then reduced to throughput maximization, as in [43] [121]. In an image sensing application, it could be interpreted as the amount of information transmitted using different image formats [73]. In a file transferring application, it can be reduced to the probability of successful file delivery [110].

In general, we formulate the reward maximization problem as

$$\text{maximize} \sum_{i=1}^{N} \frac{T}{T_i} R_i(C_i) \tag{3.5}$$

$$\text{subject to } \sum_{i=1}^{N} \frac{T}{T_i} \cdot \frac{C_i}{S_i} \leq T \tag{3.6}$$

$$\sum_{i=1}^{N} \frac{T}{T_i} E_i(S_i, C_i) \leq E_{max} \tag{3.7}$$

$$S_i \in \{s_1, s_2, \ldots, s_M\}, 1 \leq i \leq N \tag{3.8}$$

$$C_i \in \{c_i^1, c_i^2, \ldots, c_i^K\}, 1 \leq i \leq N. \tag{3.9}$$

Constraint (3.6) guarantees that all the data streams can be completed under the EDF scheduling [74]. Whenever there is a schedule that maximizes the reward and can guarantee all streams transmitted under constraints, we can always convert that

schedule to EDF scheduling with the same reward, according to the proof of EDF optimality in [75]. When all the packets in a hyper period are ready for transmission at time 0, (3.6) reduces to a discrete case as in [121]. If we enable continuous data size and transmission rate in (3.8) and (3.9), respectively, this problem is reduced to a continuous case, similar to that in [121]. In constraint (3.9), we enable multiple choices of data sizes; the problem is reduced to the one in [97] when we fix available data size of each task i in constraint (3.9) to be $\{c_i^1, c_i^2\}$, with $c_i^1 = 0$ and $c_i^2 \neq 0$, for all $1 \leq i \leq N$. Theorem 1 shows the reward-maximization problem for periodical tasks defined by (3.5)-(3.9) is NP-hard.

Theorem 1. *The reward-maximization problem for periodical tasks defined by* (3.5)-(3.9) *is NP-hard.*

Proof. Let x_{ij} and x_{il} denote two 0-1 decision variables, respectively. Let $\Omega_i = \{s_1, s_2, \ldots, s_M\}$ and $\Lambda_i = \{c_i^1, c_i^2, \ldots, c_i^K\}$ for all $1 \leq i \leq N$. If a task i is assigned to transmit data at rate $j \in \Omega_i$, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$. If a task i is assigned to transmit data with data size $l \in \Lambda_i$, then $x_{il} = 1$; otherwise, $x_{il} = 0$. The optimal problem ((3.5)-(3.9)) can be rewritten as:

$$\text{maximize} \sum_{i=1}^{N} \sum_{l \in \Lambda_i} \frac{T}{T_i} R_i(C_{il}) x_{il} \tag{3.10}$$

$$\text{subject to} \sum_{i=1}^{N} \sum_{j \in \Omega_i} \sum_{l \in \Lambda_i} \frac{T}{T_i} \cdot \frac{C_{il}}{S_{ij}} x_{ij} x_{il} \leq T \tag{3.11}$$

$$\sum_{i=1}^{N} \sum_{j \in \Omega_i} \sum_{l \in \Lambda_i} \frac{T}{T_i} E_i(S_{ij}, C_{il}) x_{ij} x_{il} \leq E_{max} \tag{3.12}$$

$$\sum_{j \in \Omega_i} x_{ij} = 1, 1 \leq i \leq N \tag{3.13}$$

$$\sum_{l \in \Omega_i} x_{il} = 1, 1 \leq i \leq N. \tag{3.14}$$

By viewing $\frac{T}{T_i} R_i(C_{il})$ as the profit of item l out of class i, terms $\frac{T}{T_i} \cdot \frac{C_{il}}{S_{ij}}$ and $\frac{T}{T_i} E_i(S_{ij}, C_{il})$ as the corresponding weights, the optimal problem formulation ((3.10)-(3.14)) becomes to an instance of well-known NP-hard Multidimensional Multiple-Choice Knapsack Problem (MMKP) [65].

□

3.2 Branch-and-Prune for the Optimal Solutions

A general method of solving the optimal MMKP problem is to search the solution space until an optimal solution is found and confirmed [65]. We can use breadth-first search to generate partial solution along with the sequence of receivers. This algorithm enumerates all possible data sizes and transmission rate for each receiver. This process can be visualized as a state space branch where each non-leaf node in this tree has $M \times K$ children if there are M transmission rate levels and K data size levels for each receiver. Therefore a naive algorithm would generate $(M \times K)^i$

nodes at level i. The state space can grow exponentially with the task number. To reach the solution in practical run time, most researchers relied on heuristics to obtain approximated solutions [61][79][93] or adapted approaches to reduce the computational complexity [34][25][123]. These approaches are not readily applicable to our problem as our problem involves more decision factors. In the following, we develop a branch-and-prune algorithm for the reward maximization optimization problem with 2-dimension multiple choices of data size and transmission rate.

3.2.1 Branch-and-Prune Algorithm

Consider a tuple $(\bar{r}_{ik}, \bar{t}_{ik}, \bar{e}_{ik})$ as a state, where $\bar{r}_{ik}$ denotes the reward sum of the first i tasks corresponding to the accumulated transmission time $\bar{u}_{ik}$ and energy sum $\bar{e}_{ik}$. We use a list to record all the states of task i

$$L_i = \langle (\bar{r}_{i1}, \bar{t}_{i1}, \bar{e}_{i1}), (\bar{r}_{i2}, \bar{t}_{i2}, \bar{e}_{i2}), \ldots, (\bar{r}_{in_i}, \bar{t}_{in_i}, \bar{e}_{in_i}) \rangle,$$

where n_i is the number of states after the ith iteration. Due to the delay and energy constraints, branching from unpromising states that generate infeasible or non-optimal solutions can be avoided. We summarize three circumstances, referred to pruning criteria, in which branches from a certain state will be pruned.

1. Temporal Criterion: The state $(\bar{r}_{ik}, \bar{t}_{ik}, \bar{e}_{ik})$ in list L_i will be pruned if it satisfies that

$$\bar{t}_{ik} + \sum_{j=i+1}^{N} \frac{T}{T_j} \frac{C_j^{min}}{s_M} > T.$$

   The inequality means the partial solution will not meet the deadline by transmitting the smallest size of data to the remaining receivers even at the maximal

Table 3.1: An example for 4 data streams with 2 data size levels and 2 transmission rate levels.

Data Stream	(r,t,e) at c_1 and s_1	(r,t,e) at c_1 and s_2	(r,t,e) at c_2 and s_1	(r,t,e) at c_2 and s_2
1	(1,2,1)	(1,1,4)	(2,4,2)	(2,2,8)
2	(1,2,1)	(1,1,5)	(2,4,2)	(2,2,10)
3	(2,2,1)	(2,1,4)	(4,4,2)	(4,2,8)

transmission rate. Therefore the solutions generated from this state are infeasible.

2. Energy Criterion: The state $(\bar{r}_{ik}, \bar{t}_{ik}, \bar{e}_{ik})$ in list L_i will be pruned if it satisfies that

$$\bar{e}_{ik} + \sum_{j=i+1}^{N} \frac{T}{T_j} E_i(s_1, C_j^{min}) > E_{max}.$$

The inequality means the partial solution will not meet the energy constraint by transmitting the smallest size of data to the remaining receivers even at the minimal transmission rate. Therefore the solutions generated from this state are infeasible.

3. Dominance Criterion: If two states $(\bar{r}_{ik}, \bar{t}_{ik}, \bar{e}_{ik})$ and $(\bar{r}_{ij}, \bar{t}_{ij}, \bar{e}_{ij})$ in a list L_i satisfy $\bar{r}_{ik} > \bar{r}_{ij}$ and $\bar{t}_{ik} \leq \bar{t}_{ij}$ and $\bar{e}_{ik} \leq \bar{e}_{ij}$, or $\bar{r}_{ik} \geq \bar{r}_{ij}$ and $\bar{t}_{ik} \leq \bar{t}_{ij}$ and $\bar{e}_{ik} < \bar{e}_{ij}$, or $\bar{r}_{ik} \geq \bar{r}_{ij}$ and $\bar{t}_{ik} < \bar{t}_{ij}$ and $\bar{e}_{ik} \leq \bar{e}_{ij}$, then the latter state is said to be dominated by the former one. Intuitively, if a state uses more energy, requires larger transmission time but achieves smaller reward than another state, it can always be replaced by the latter state.

In the following, we use an example to show how these three pruning criteria work. Suppose there are three data streams with two transmission rate levels and two data

size levels each, as shown in Table 3.1. Assume the time and energy constraint is $(7, 7)$. A root node 0 at level-0 is represented by a 3-tuple $(0, 0, 0)$ which means reward, time, and energy are all zero since no data stream has been enumerated. Start from the root node, we first enumerate data stream 1. Note that the order of enumerating the data streams will not have impact on the final solution. Since there are two transmission rate levels and two data size levels for each data streams, we have four different nodes generated from root node after enumerating data stream 1, shown in Fig. 3.2. We label these four nodes at level-1 as 1, 2, 3 and 4.

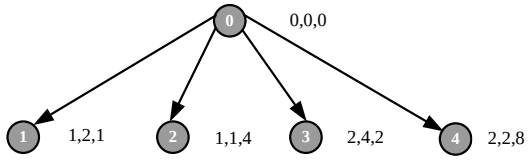


Figure 3.2: Partial state space tree after one enumeration.

Then we proceed to the next level. We examine every node at level-1 and enumerate data stream 2. Start from node 1, we find nodes 1 cannot be pruned based on all above three pruning criteria. So we can generate 4 new nodes at level-2 from node 1 by enumerating data stream 2 and calculate the 3-tuple (r, t, e) for the new nodes. In the same way, we can derive another 8 new nodes at level-2 starting from node 2 and 3. Node 4 will be pruned due to energy criterion (the current energy consumption is 8 which exceeds the energy budget 7). So there will be no level-2 nodes derived from node 4. After pruning and enumeration, we obtain 12 nodes at level-2, shown in Fig. 3.3.

When we examine the level-2 nodes, we find node 8, 10, 12, and 16 can be pruned according to the energy criterion. Node 15 is pruned due to the temporal criterion. Nodes 6 and 14 will be pruned as well because they are dominated by nodes 9 and 11,

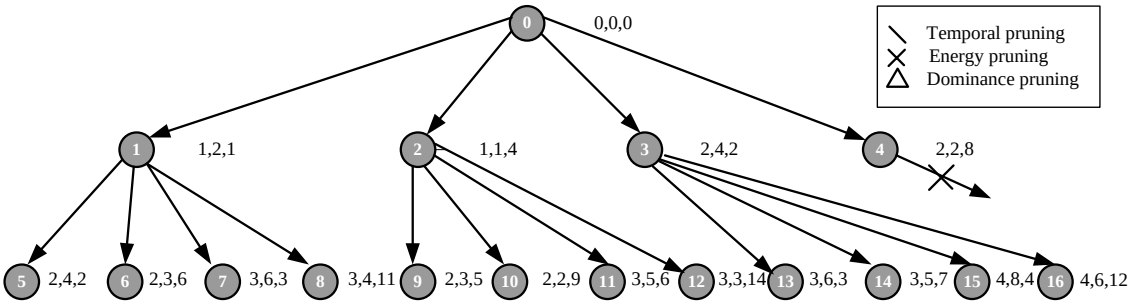


Figure 3.3: Partial state space tree after two enumerations.

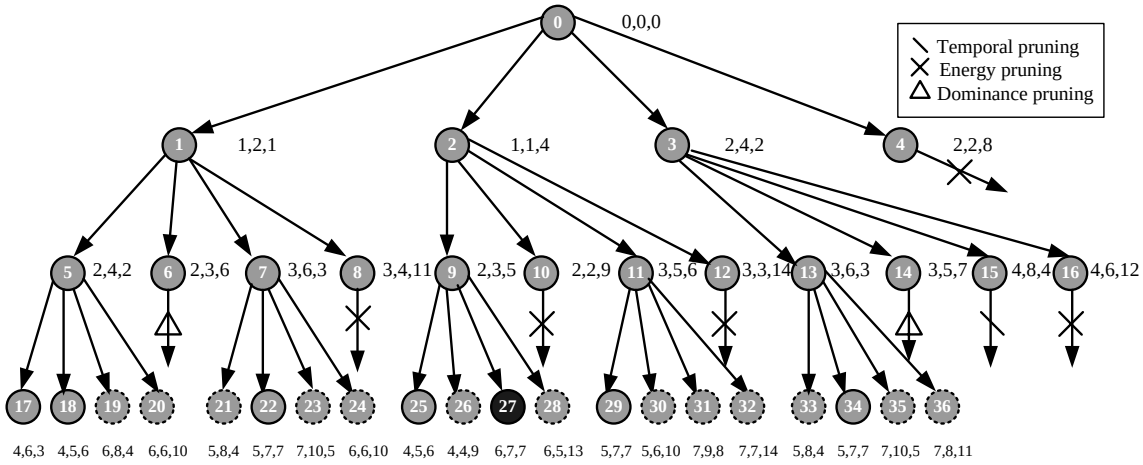


Figure 3.4: The complete state space tree by pruning.

respectively. Iteratively performing this sequence of steps (nodes generation, 3-tuple calculation, and pruning), we can obtain all nodes at level-3. Thus we generate the complete state space tree for the given task set. The complete state space tree is shown in Fig. 3.4. The generated nodes at level-3 with solid line are feasible solutions while those with dotted line are infeasible solutions. Among those feasible solutions, the dark node numbered 27 represents the optimal solution $(6, 7, 7)$ because it has the largest reward value. The number of final nodes (level-3 nodes) now is 20. It has been reduced dramatically in contrast to that without any pruning, which is 64.

Algorithm 1 shows the pseudo-code. Initially, we have list L_0 with zero reward, zero time and zero energy. List L_i is obtained by the following steps. We first get a number of lists L'_{ijk} by adding reward, transmission time and energy values of task i under each data size $j \in \{c_i^1, c_i^2, \ldots, c_i^K\}$ and each transmission rate $k \in S$ to the states in list L_{i-1}. The component-wise addition is denoted by $L'_{ijk} = L_{i-1} \oplus (\bar{r}_{ijk}, \bar{t}_{ijk}, \bar{e}_{ijk})$. Secondly, we merge the lists L'_{ijk} into a list L'_i in decreasing order of reward value. Thirdly, we prune all non-feasible state by the Temporal Criterion and Energy Criterion. Finally, we use the procedure Prune-Lists() to find the undominated states in L' and returns it as L''. In this procedure, we check each pair of states to eliminate the dominated states as many as possible in order to save more space.

3.2.2 Algorithm Analysis

Now consider the running time for the ith iteration when adding task i. In Algorithm 1, lines 3-7 are linear in the number of the states in L'_i, denoted by $|L'_i|$. $|L'_i|$ equals to $|L'_{i-1}|$ multiplied by the number of data sizes and the number of transmission rates for task i. The time complexity for merging sorting is $O(|L'_i| \log(|L'_i|))$.

Line 9 can be completed in $O(|L'_i|)$. Line 10 can be completed in $O(|L'_i|^2)$ because we check each pair of states to eliminate the dominated ones. This part dominates the whole time complexity. As a result, the running time of adding task i is $O(|L'_i|^2)$. We denote the upper bound of states in L'_i by U, i.e., $max_{1 \leq i \leq N}\{|L'_i|\} \leq U$. Thus an upper bound of running time for adding N tasks is $O(NU^2)$. The value of U may increase significantly with the increase of the size of task set, the number of data size levels, and the number of transmission rate levels. So this algorithm has a pseudo-polynomial time complexity. The necessary memory requirement of this algorithm is bounded by the number of states in a list as well. Since it is not necessary for us to keep all the lists in the algorithm, the space requirement would be equal to the size of the last list, which is in the order of $O(U)$.

3.3 Time-Efficient Approximation

Although the above three pruning conditions are effective in removing unpromising states, the state space in Algorithm 1 can still expand significantly and it will be computationally expensive to get the optimal solution with a large number of receivers, data sizes, and transmission rates. In practice, it is not always necessary to find the optimal solution with limited time and computation resources. A near-optimal solution is more desirable if it can be completed in reasonable time while consuming reasonable computation resources.

In this section, we will first propose a polynomial-time heuristic approach. Then we will analyze the complexity of this algorithm.

3.3.1 Polynomial-time Approximated Approach (Clustering)

We develop a heuristic algorithm, named Clustering algorithm, to approximate the optimal solution to the proposed problem with a polynomial computational time complexity. This Clustering algorithm is novel for the proposed problem. The general idea of this algorithm can be traced back to data clustering in mathematics.

This Clustering algorithm is based on a clustering property of the final states after we enumerate all the tasks. Fig. 3.5 shows the result if we plot the all of the final states for Table 3.1 after enumerating all possible combination of data sizes and transmission rates into a 3-D space with the coordinates of $(reward, energy, time)$. We can find the nodes, representing the final states, are clustered instead of randomly scattering. This is attributed to the discrete levels of data sizes and transmission rates. Those nodes in the same cluster tend to have close reward values, energy consumption, and transmission time. We have similar observations about the intermediate results after each enumeration.

During each enumeration, if we start from one cluster, the best solution generated from this cluster will have close reward values. If one node in this cluster can generate the optimal solution, the others in the same cluster can produce near-optimal solutions as well. Keep only one node in each cluster and remove the others, we can still get the near-optimal solution though the optimal solution might be removed. If none of the nodes in this cluster can lead to the optimal solution, there would be no impact on the final solution if we keep one and remove the others. The benefit of this method is that it can reduce the state space significantly.

We create clusters of nodes by dividing the reward value axis and energy axis evenly to $mbins$ and $nbins$ number of ranges, respectively. Thus we construct $mbins \times nbins$ rectangle prisms (clusters) in a 3-D space. In Fig. 3.5, cluster B represents one

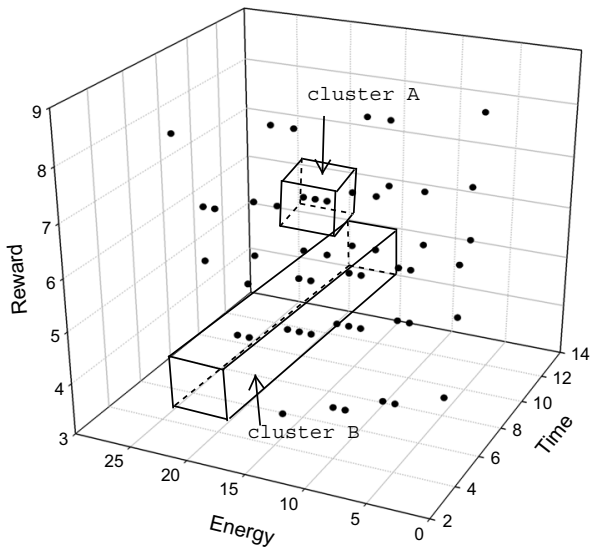


Figure 3.5: The reward-time-energy relationship in a 3-D space.

of the clusters. Suppose the maximal and minimal sums of energy consumption for all tasks are E_{max} and E_{min}, respectively. We set the length of the bottom of the cluster to be $\epsilon_e = (E_{max} - E_{min})/nbins$. Similarly, we set the width of the bottom of the rectangle prism (cluster) to be $\epsilon_r = (R_{max} - R_{min})/mbins$, where R_{max} and R_{min} are the maximal and minimal sums of reward values for all tasks, respectively. The scaled reward of task i under data size j, transmission rate k, and reward group size ϵ_r is rounded to the next integer, i.e., $\lceil \frac{r_{ijk}}{\epsilon_r} \rceil$, which represents the scaled reward value in each cluster. In the same way, we can get the rounded scaled energy value, $\lceil \frac{e_{ijk}}{\epsilon_e} \rceil$, which represents energy values in each cluster. We introduce $\lceil \frac{r_{ijk}}{\epsilon_r} \rceil$ and $\lceil \frac{e_{ijk}}{\epsilon_e} \rceil$ into the 3-tuple representing a state in Algorithm 1 to construct a 5-tuple state used in Algorithm 2.

When enumerating a task, we generate all states for one task with pruning conditions. the Clustering approach divides all the states into buckets, which represent the clusters, and keep one node for each bucket, as shown in Algorithm 2. In Algorithm 2,

the strategy in determining which node should be kept in each cluster depends on the factor of transmission time. While the nodes in the same cluster have close rewards and energy consumption, we always want to keep the one with the smallest transmission time in order to save time for the remaining tasks. Besides this strategy, we have other criteria to decide which node should be reserved in each cluster. Different metrics, such as energy, reward, and energy-delay product, can be employed. We conduct simulations to compare the impact of performance when employing different strategies for selecting representative node in the following evaluation part. According to the evaluation results, we choose time as the only metric for this selection since it can lead to the best approximation.

Clearly, if the intermediate states themselves can cluster well, it will lead to small number of non-empty clusters after each enumeration so that the state space can be reduced more thus the execution time can be shortened. Even in the worst case, as a result of rounding up and keeping one node per cluster, the number of states can also be reduced greatly so are the running time and required space to solve the scaled problem compared with the optimal approach. The number of nodes at each enumeration is bounded by the largest number of clusters, which is equal to $mbins \times nbins$.

3.3.2 Algorithm Analysis

For given numbers of ranges for energy and reward, we have the upper bound for the number of states in each iteration by $U = nbins \times mbins$. In Algorithm 1, the running time for adding task i is $O(|L_i'|^2)$ because we compare each pair of elements in the list to eliminate the dominated states. Here, we don't have this step. So the running time for adding task i is $O(|L_i'| \log(|L_i'|))$, dominated by sorting. Thus

an upper bound of the running time is $O(NU \log U)$, where U denotes the upper bound of states in L'_i. Replace U by $nbins \times mbins$, we get the time complexity of $O(N(nbins \times mbins) \log(nbins \times mbins))$. Since the values of $nbins$ and $mbins$ are initialized according to the problem size (see evaluation part) which is determined by a polynomial function with variables of task set size, data size levels and rate levels, the total running time cost is polynomial in the size of task set, data size set and rate set. Similar to Algorithm 1, the space cost is in order of $O(U)$, which is $O(nbins \times mbins)$, as well.

3.4 Performance Evaluation

We simulate the following four algorithms in our experiment:

- Branch-and-prune algorithm for optimal solution: We simulate this algorithm to obtain the optimal solution for reward maximization problem.
- Polynomial-time Clustering algorithm (Clustering): This is the proposed time-efficient Clustering algorithm for a near-optimal solution.
- Greedy-pack and greedy-unpack algorithms: These two algorithms are adapted from [97] as the competitors of proposed polynomial-time Clustering algorithm. Though the reward maximization problem in [97] is different from that in our work, the method of designing heuristic approaches is a general principle and can still be adapted to the problem studied in this work.

We compare the solutions of Clustering, greedy-pack, and greedy-unpack with the optimal solution obtained by the branch-and-prune algorithm. We use the metric of normalized system reward to show how close these algorithms can approximate

the optimal solution. We study the effect of different parameters on the simulation results and investigate the execution time cost for different algorithms. We conduct simulations to show to what degree our proposed branch-and-prune algorithm can restrict the explosion of state space. In addition, we investigate the impact on approximation by choosing different numbers of clusters and strategies of representative node selection for Clustering algorithm.

3.4.1 Simulation Setup

The wireless channel settings we used are similar to those described in [12][121]. The channel bandwidth is 1MHz. The bits per transmission is set to be 1 bit/Hz, 2 bits/Hz, 4 bits/Hz, and 6 bits/Hz (BPSK). Each receiver will receive data from the transmitter periodically at the above transmission rates. These periodic data streams are generated based on the following parameters: distance between receiver and transmitter, the number of different sizes of data to be transmitted, the sizes of data to be transmitted, period, and reward values. The distances between transmitter and receivers are uniformly distributed in a range of $[20, 200]$ meters. Under this setup, a transmission power of 20 mW is required to reliably transmit data at 1 bit/Hz (BPSK) at a distance of 100 m. According to power function (3.3), we can calculate the power consumption for different receivers with different transmission rates.

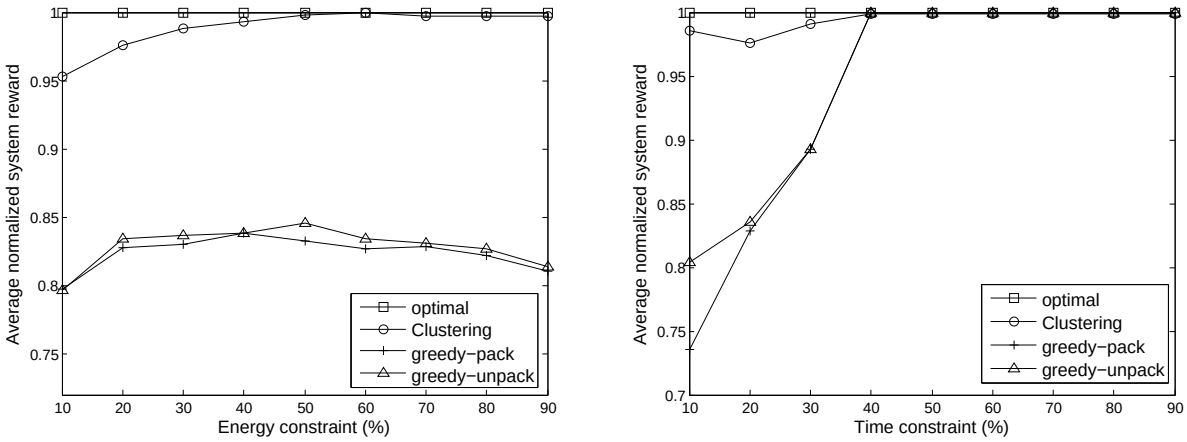
We assumed the number of data size levels to be 5 for all receivers. The data sizes are uniformly distributed in the range of $[1, 16]$ Mb. The periods for each receiver can be represented by the number of jobs transmitted to each receiver, denoted by $\{job_1, job_2, \ldots, job_N\}$, within the time interval T. So the period T_i can be calculated as $T_i = \frac{T}{job_i}$. The value of job_i is a random integer value in the range of $[1, 12]$.

Based on the above parameters, we can calculate the energy consumptions by the energy function (3.4). Each receiver has its own reward function. In this simulation, we define a linear reward function in order of the size of transmitted data. The coefficient h_i for $R_i(C_i) = h_i C_i$ is the multiplication of h_{i-1} and a random value in the range of $[1.2, 3.0]$ for $i = 2, 3, \ldots, N$. The coefficient h_1 is a random value in the range of $[0.1, 0.2]$.

The energy constraint E_{max} and the length of hyper-period T for a task set are defined in ways similar to those in [97]. E_{max} is a multiplication of a parameter α and E^*, where $0 < \alpha < 1$ and E^* is the total energy consumption for all data streams to transmit the largest size of data at the largest transmission rate within the interval, i.e., $E^* = \sum_{i=1}^{N} job_i E_i(C_i^{max}, s_M)$. T is the multiplication of a parameter β and T^*, where $0 < \beta < 1$ and T^* is the total transmission time for all data streams to transmit the largest size of data at the smallest transmission rate, i.e., $T^* = \sum_{i=1}^{N} job_i \frac{C_i^K}{s_1}$. A lower value of α means more stringent energy constraint and a lower value of β means tighter deadline. We refer to α and β as energy constraint factor and time constraint factor, respectively. In Clustering algorithm, we assumed $nbins = mbins = 10 \times N \times (logM + logK)$ by default.

3.4.2 Simulation Results

In this part, we firstly study the impact of time and energy constraint factors on the performance of the four algorithms. Secondly, we investigate the relationship between the performance and execution time. Thirdly, we further study the effectiveness of our branch-and-prune algorithm for optimal solution. Finally, we simulate our Clustering algorithm with different parameters to the effect on the extent of the approximation to the optimal solution.



(a) Energy constraint

(b) Time constraint

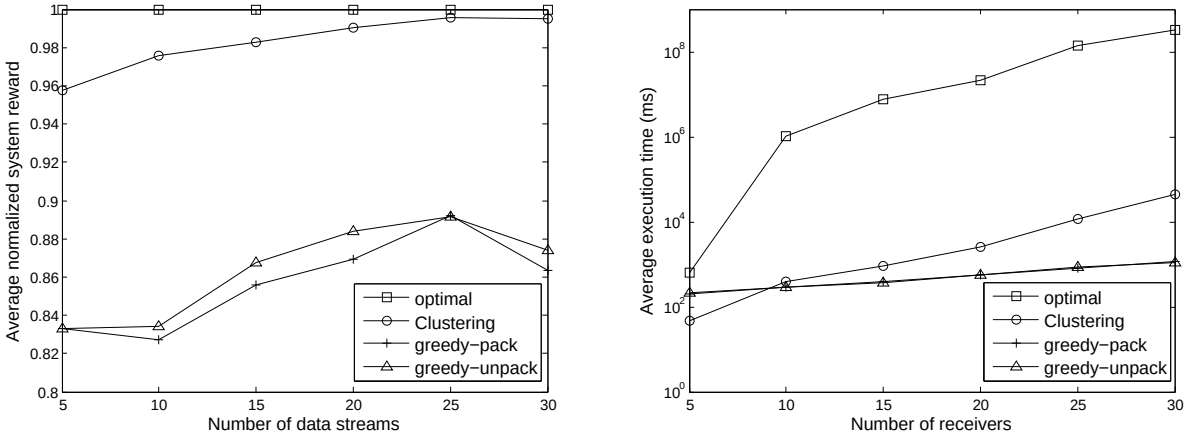
Figure 3.6: Reward under different time and energy constraints.

Impact of time and energy constraint factors

The first experiment is to study the impact of time and energy constraint factors on the performance. We simulated optimal, Clustering, greedy-pack, and greedy-unpack algorithms for this reward maximization problem with different time and energy constraint factors. The system reward normalized to the optimal value is presented in Fig. 3.6.

Fig. 3.6(a) demonstrates the impact of energy constraint factor on the performance. We assumed the number of receivers to be 10 and the time constraint factor α to be 0.2. We increased the energy constraint factor β from 0.1 to 0.9 with a step size of 0.1. The normalized reward of Clustering increases with the increase of energy constraint factor. It is around 95% when $\beta = 0.1$ but can reach more than 99% when $\beta \geq 0.5$. The normalized rewards of two greedy algorithms always have a big gap larger than 15% from the optimal value even if the energy constraint is loose. The best normalized reward is bounded by 85%.

Fig. 3.6(b) shows the impact of time constraint factor on the performance. We



(a) Normalized reward

(b) Execution time

Figure 3.7: Reward and execution time with different numbers of receivers.

increased the time constraint factor α from 0.1 to 0.9 with each step equal to 0.1 and assumed a constant task size of 10 receivers and a constant energy constraint factor $\beta = 0.2$. The results of Clustering are always larger than 95%. With a loose time constraint, it may increase to more than 99%. We find the normalized system reward of the two greedy algorithms always has a gap more than 10% from the optimal value when α is less than 0.4. They increase rapidly with the increase of time constraint factor and can be more over 99% of the optimal value when time constraint factor goes beyond 0.4. There are two reasons for this result.

On one hand, we observe the optimal solutions are always the same when β is fixed to 0.2 and α varies from 0.4 to 0.9. This can be explained as follows. Since the energy consumption grows exponentially with the increase of transmission rate, we can decrease the energy consumption greatly by slowing down the transmission rate if the time constraint is loose, which means linear increase of time can lead to exponential decrease of energy consumption. In this way, the time constraint has insensitive impact on the solution. That is why we can see the optimal solutions are always same since α is 0.4.

On the other hand, the greedy algorithms can always have good performance with a loose time constraint when the only constraint is energy consumption. Since the only constraint is energy consumption after we loose the time constraint, the greedy algorithms can have good performance in this case.

As comparison, for the case in Fig. 3.6(a), a loose energy constraint will only lead to small decrease of time. These two constraints will always have sensitive impact on the solution though energy constraint is very loose. In simulation, we notice the optimal solutions are not the same for the cases in Fig. 3.6(a) with different β values.

From these results, we can see Clustering algorithm always obtains near-optimal solution when the constraints are not very tight. With looser constraints, it can even achieve the optimal solution.

Performance and execution time

The second experiment is to investigate the performance and execution time for these four algorithms in this simulation. The simulation was run on a machine with Pentium D 2.8GHz CPU and 2GB RAM. We assumed both the time and energy constraint factor to be 0.2. We increased the number of receivers from 5 to 30 with each step equal to 5.

Fig. 3.7(a) demonstrates the average normalized system reward. The normalized system reward for Clustering for different task size is typically more than 96% at the worst case. With more receivers, the normalized system reward will increase and can reach more than 99%. This is because we set the values of $nbins$ and $mbins$ related to the number of receivers. The larger these values are, the more precise solution we can obtain. For greedy-pack and greedy-unpack, the normalized system reward is usually around $82\% - 90\%$. There is always a gap more than 10% with all task sizes. We notice these two greedy algorithms usually can reach quite similar solution for the

data size and transmission rate in the end although they search for the near-optimal solution in an opposite way. That is why they have similar normalized system reward.

Fig. 3.7(b) shows the execution time of these four algorithms. It is expected that the execution time for the optimal solution is the most and increases rapidly with the increase of the number of receivers. The reason is that the increase of number of receivers will lead to tremendous increase of the state space though we have already employed pruning techniques. Compared with the branch-and-prune algorithm for optimal solution, Clustering can always have tiny execution time cost while achieving near-optimal solution. It can obtain more than 99% reward while consuming less than 0.1% of execution time of the optima solution. With the increase of task size, the execution time of Clustering increases but is still tiny compared with that for the optimal solution. Greedy-pack and greedy-unpack have very close execution time costs. They have the smallest cost but much worst performance than Clustering, shown in Fig. 3.7(a). We notice that the execution time of Clustering is close to or even smaller than that of the greedy algorithms when the number of receivers is under 10. It is because the number of clusters in Clustering depends on the number of receivers. When the number of receivers is small, the number of clusters is small, which leads to less execution time. While more receivers can lead to more clusters, more execution time will be consumed. In the case of 30 receivers, the execution time of Clustering is almost 100 times of that of greedy algorithms.

Combining the results from Section 5.2.1 and Section 5.2.2, we can see Clustering algorithm can outperform the greedy algorithms in terms of normalized reward. It is because that Clustering algorithm always keeps intermediate states which lead to optimal or near-optimal solution when the number of the cluster is large enough. In the greedy algorithm, when enumerating tasks, only the data size and transmission rate that can satisfy the criteria or lead to local optimization will be selected. This

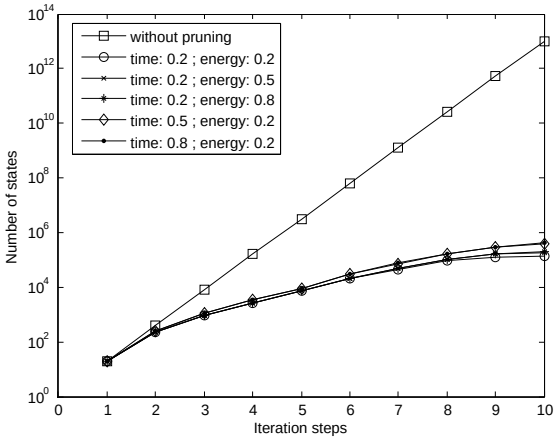


Figure 3.8: Effectiveness of states pruning with 10 periodic data steams.

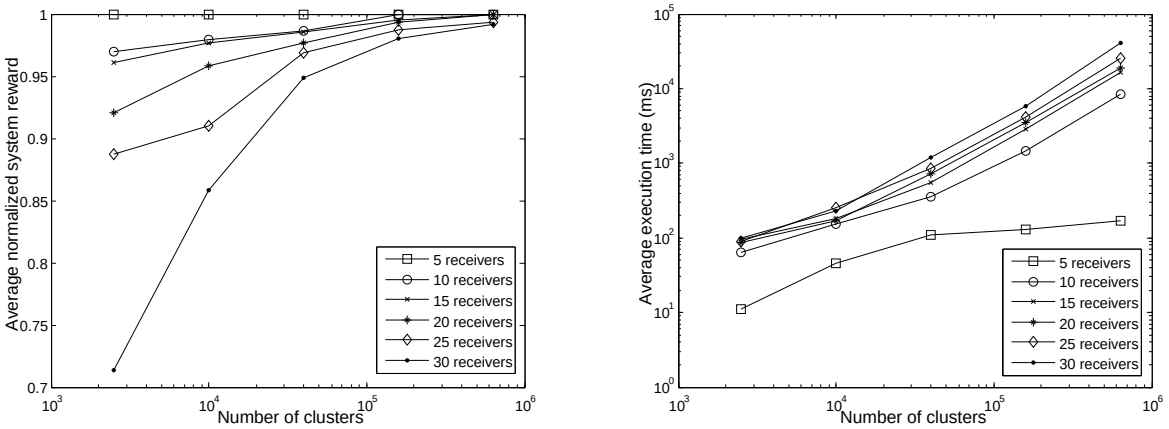
scheme may be more likely to miss the optimal or near-optimal solution thus leads to worse performance. From another point of view, the greedy algorithms will reduce the state space more significantly compared with Clustering algorithm but it suffers from more opportunities to miss the optimal or near-optimal solutions. In addition, the increase of the number of receivers will not incur the incredible increase of state space for greedy algorithms. This explains why the greed algorithms can always have smallest execution cost in most cases and the execution time for greedy algorithms will not increase greatly with the number of receivers.

In conclusion, Clustering can balance the tradeoff of the performance and the running time cost better than the greedy algorithms.

Effectiveness of branch-and-prune algorithm

Fig. 3.8 shows the effectiveness of states pruning when we searched the optimal solutions for 10 periodic data streams. Simulation results with different time and energy constraint factors are presented.

In an exhausting search algorithm, the number of states observed in each iteration



(a) Impact on accuracy

(b) Impact on execution time

Figure 3.9: Impact of number of clusters.

increases rapidly. After we employed pruning criteria, the state space is reduced dramatically though it is still expanding as more iterations are processed. Especially, the total number of states is reduced to less than 0.00001% of its original value when the last iteration is reached. Though the number of states still increases as the iterations go on, the increasing rate is much smaller than the original one. The state space generally expands in the enumeration of first several tasks as few nodes can be pruned by the pruning criteria. More non-feasible states will be pruned as more tasks are enumerated. With looser time and energy constraints, more states will be preserved in the space.

Impact of number of clusters for Clustering

In the Clustering approach, the size of state space depends on the number of clusters, which affects the accuracy of final result. In this part, we will study the impact of number of clusters on the performance for the Clustering algorithm.

In all the simulations, we assume $mbins = nbins$, and both the time and energy constraint factors to be 0.2. First, we study the impact of different number of clusters

on the accuracy approaching optimal solution and time cost. We tune the value of $nbins$ to be $50, 100, 200, 400, 800$, which means the numbers of clusters can be $2500, 10000, 40000, 160000, 640000$. The number of receivers is increased from 5 to 30 with each step equal to 5. Fig. 3.9 shows the impact of different number of clusters on both the accuracy and execution time of Clustering algorithm. We notice that both the normalized system reward and execution increase with the increase of cluster number in Fig. 3.9(a) and Fig. 3.9(b). This is because larger number of clusters preserves more states in each iteration. So the states that can lead to optimal or near-optimal solution will be more likely preserved for future search, which will lead to more execution time cost. More receivers lead to larger state space. This requires us to have a large enough number of clusters to get more precise solution. If we set the number of clusters too small for a large task set, we cannot get the solution close to the optimal one. For example, we can get the normalized system reward around 99% for the case containing 5 receivers by setting 2500 clusters ($nbins = 50$). But the same setting leads to smaller system reward with more receivers. When there are 30 receivers, this setting can only achieve a normalized system reward less than 75%.

So the number of clusters should be chosen carefully with the number of receivers in order to get near-optimal solution. We conduct simulations to study the impact of ratio of problem size over cluster number on the performance of Clustering algorithm. For a problem with N receivers, M rate levels and K data size levels, the size of state space is $(M \times K)^N$. We define problem size as $log(M \times K)^N = N \times (logM + logK)$. We represent the number of clusters by $nbins$. Then the ratio can be calculated by $N \times (logM + logK)/nbins$. We set the value of this ratio to be $2, 1, 0.5, 0.1, 0.02$. Clearly, the smaller the ratio, the larger number of clusters. We can see small ratio can always lead to more accurate result. From Fig. 3.10, we find we can obtain more than 97% normalized reward if we set the ratio to be 0.1. This can approximate

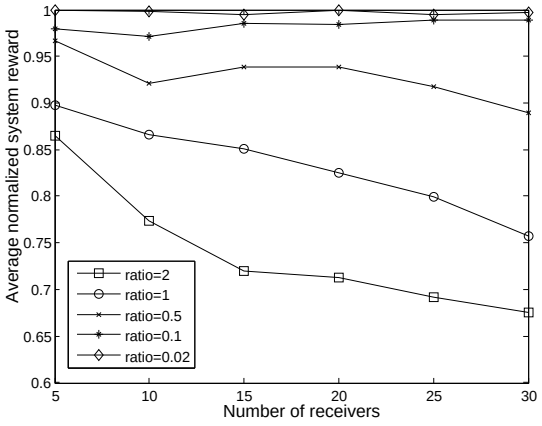


Figure 3.10: Impact of ratio of problem size over cluster number.

the optimal solution well. Though a better approximation can be achieved if the ratio is 0.02, the execution time will increase tremendously. Compromising both the effect of approximation and execution time cost, in our simulation, we set $nbins = 10 \times N \times (logM + logK)$ to achieve a near-optimal solution with affordable computation cost.

Strategy of representative node selection

As shown in Algorithm 2, we keep the node with the smallest transmission time in each cluster. We use this strategy to save time for the remaining tasks. Besides this strategy, there are other methods to determine which node should be kept in each cluster. In this part, we conduct simulations to show the results of employing different strategies to select representative node for each cluster.

We compare four different strategies using different metrics when selecting a representative node in each cluster. These metrics are time, energy, reward and energy-delay product. If we use the metric of time/energy/energy-delay, we will keep the node with the smallest value of this metric in each cluster. The node with the largest

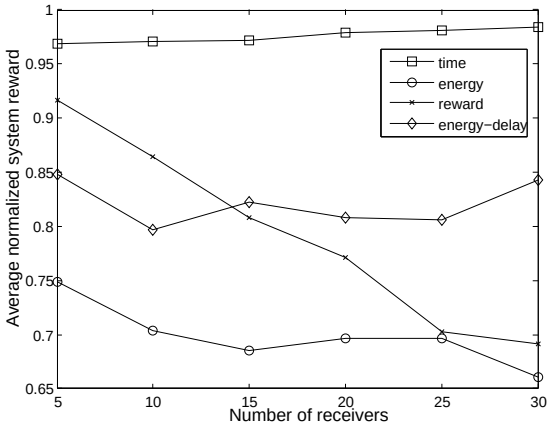


Figure 3.11: Impact of different strategies selecting representative node.

reward will be kept in each cluster if we use the metric of reward. From Fig. 3.11, we can see using time as the metric can always guarantee the best performance. It can achieve a normalized system reward more than 97%, which can beat the second best strategy significantly, no matter how many receivers are taken into account in the simulation setup. Based on this result, we always employ time as the only metric when determining which node in each cluster should be kept in our Clustering algorithm.

Algorithm 1 Reward maximization using branch-and-prune.

$L_0 = \langle (0,0,0) \rangle$
for i=1 to N **do**
  for all data size $j \in \{c_i^1, c_i^2, \ldots, c_i^K\}$ **do**
    for all transmission rate $k \in \{s_1, s_2, \ldots, s_M\}$ **do**
      $L'_{ijk} = L_{i-1} \oplus (\bar{r}_{ijk}, \bar{t}_{ijk}, \bar{e}_{ijk})$
    end for
  end for
  merge L'_{ijk} into a list L'_i in a decreasing order of reward
  delete all states in L'_i with $\bar{t} + \sum_{j=i+1}^{N} \frac{T}{T_j} \frac{C_j^{min}}{s_M} > T$ or $\bar{e} + \sum_{j=i+1}^{N} \frac{T}{T_j} E_i(s_1, C_j^{min}) > E_{max}$
  L_i= Prune-Lists(L'_i)
end for
return the largest state in L_N

procedure PRUNE-LISTS(L')
  $L'' = \emptyset$
  while $L' \neq \emptyset$ **do**
    choose and delete the largest state $(\bar{r}', \bar{t}', \bar{e}')$ from L'
    $flag = true$
    if $L'' == \emptyset$ **then**
      add $(\bar{r}', \bar{t}', \bar{e}')$ to the end of list L''
    else
      for all states $(\bar{r}'', \bar{t}'', \bar{e}'')$ in L'' **do**
        if ($\bar{t}'' < \bar{t}'$ and $\bar{e}'' \leq \bar{e}'$) or ($\bar{t}'' \leq \bar{t}'$ and $\bar{e}'' < \bar{e}'$) **then**
          $flag = false$
          break
        end if
      end for
      if $flag == true$ **then**
        add $(\bar{r}', \bar{t}', \bar{e}')$ to the end of list L''
      end if
    end if
  end while
  return L''
end procedure

Algorithm 2 A polynomial-time heuristic approach (Clustering).

$L_0 = \langle(0,0,0,0,0)\rangle$
calculate E_{min}, R_{max} and R_{min}
$\epsilon_r = \frac{R_{max}-R_{min}}{mbins}$
$\epsilon_e = \frac{E_{max}-E_{min}}{nbins}$
for i=1 to N **do**
    $L_i = \emptyset$
    for all data size $j \in \{c_i^1, c_i^2, \ldots, c_i^K\}$ **do**
        for all transmission rate $k \in \{s_1, s_2, \ldots, s_M\}$ **do**
            $L'_{ijk} = L_{i-1} \oplus (\lceil \frac{\bar{r}_{ijk}}{\epsilon_r} \rceil, \bar{r}_{ijk}, \bar{t}_{ijk}, \lceil \frac{\bar{e}_{ijk}}{\epsilon_e} \rceil, \bar{e}_{ijk})$
        end for
    end for
    merge L'_{ijk} into a list L'_i in a decreasing order of scaled reward
    delete all states in L'_i with $\bar{t} + \sum_{j=i+1}^{N} \frac{T}{T_j} \frac{C_j^{min}}{s_M} > T$ or $\bar{e} + \sum_{j=i+1}^{N} \frac{T}{T_j} E_i(s_1, C_j^{min}) > E_{max}$
    construct $mbins \times nbins$ buckets $B_1, B_2, \ldots, B_{mbins \times nbins}$
    for all states $st = (\lceil \frac{r}{\epsilon_r} \rceil, r, t, \lceil \frac{e}{\epsilon_e} \rceil, e)$in L' **do**
        $isNew = true$
        for all non-empty buckets B_i in $(B_1, B_2, \ldots, B_{mbins \times nbins})$ **do**
            select a state st'=$(\lceil \frac{r'}{\epsilon_r} \rceil, r', t', \lceil \frac{e'}{\epsilon_e} \rceil, e')$ from B_i
            if $(\lceil \frac{r}{\epsilon_r} \rceil == \lceil \frac{r'}{\epsilon_r} \rceil)$ and $(\lceil \frac{e}{\epsilon_e} \rceil == \lceil \frac{e'}{\epsilon_e} \rceil)$ **then**
                put st into B_i
                $isNew = false$
                break
            end if
        end for
        if $isNew == true$ **then**
            put st into an empty bucket
        end if
    end for
    for all non-empty bucket n $(B_1, B_2, \ldots, B_{mbins \times nbins})$ **do**
        select one element with smallest $\bar{t}$ and add it to the end of L_i
    end for
end for
return the largest state in L_N

Chapter 4

PPM: A Power Management Middleware for Networked Computing Systems

In networked computing systems, energy saving is always a critical design issue because of both the cost of electricity bills and the reliability and compliance with environmental standards. Despite the great engineering and technical advances that have been made, power management is still necessary to address the energy consumption issue for further possible saving in today's networked computing systems, from real-time embedded servers to enterprise servers. Although there are a number of existing work providing power management strategies to meet different requirements, the lack of a general framework of power management solutions motivates us to design and implement a power management middleware, name PPM, for networked computing systems.

We will present the design of PPM in the scenario of Base Band Units (BBUs). Currently the wireless service operators are facing serious challenges from surging

power consumption and high operating expense of radio access network. For example, China Mobile's power consumption doubled in the last 5 years, while over 70% of its power consumption came from base stations consisting of Radio Remote Units (RRUs) and Base Band Units (BBUs). In current RAN, BBU usually features hundreds of watts during the operation, which dominating the power consumption of base stations. To reduce power consumptions and operating expense of operators, energy saving is a critical design criteria for BBUs. However, there is few such power management solutions or frameworks developed for BBUs. It is much more urgent to provide such a power management middleware for BBUs.

In this chapter, we will first give a brief overview of PPM. Then, we start presenting the design issues with one key component, power metering, which is the foundation of PPM. After that, we will discuss the architecture of this middleware. Finally, we will leave the details of power management policies to the subsequent chapters.

4.1 Overview

A power management software is required to provide solutions for emerging challenges in power consumption for BBUs. The crucial criterion for us to design this power management software is that it should be easily to deploy on different platform without touching upon the underlying OS or the running workload. So we follow the design method of a middleware to design this power management software. It can provide a practical and effective solution for power management for BBUs.

The information of real-time power consumption is necessary for the design and deployment of power management strategies. As the external power measurement may not always be available, the fundamental challenges rely on how to obtain power consumption measurement without external measurement equipment as well as how

to quantitatively understand power consumption at a system level. This motivates us to design and implement a power metering prototype for BBUs. Such a software-based power metering tool is crucial for the further deployment of PMM in a large scale. Thus, we decompose PMM into two major parts: real-time power metering tool and power management middleware.

4.2 Real-Time Power Metering

We designed and implemented a power metering prototype for BBU. The major objectives of this tool are two-fold. First, we need a software-based solution to estimate power consumption for BBU. Without the knowledge of current power consumption of BBUs, the power management policies cannot be accurately applied. However, the number of BBUs in practice could be huge, it is not feasible to equip each BBU with its own external power measurement equipment. Thus it is required to estimate the power consumptions of each BBU in a software manner. Second, understanding how the workload impacts the power consumption of a system is crucial to guide the potential energy saving strategies. With external power measurement equipment, it is still hard to obtain the information of component-level power consumption unless it is allowable to decompose the measurement of power consumption for each subsystem physically. Our proposed power metering solution provides such an insight of power dissipation amongst components without physical decomposition of power measurement.

4.2.1 Design of Power Metering Tool

In general, system power consumption increases with system resource usage. The system usage can be represented by corresponding system performance metrics. Thus

we propose our power metering solution by building such a model correlating system performance metrics with power consumption. Using this model, we can estimate power consumption using this power model based on real-time system performance metrics. There are three major components of this power metering tool, as shown in Fig. 4.1.

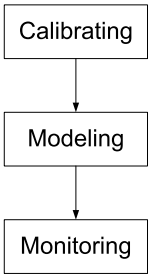


Figure 4.1: Power metering design.

A one-time, offline calibration phase is necessary to obtain the system-wide power consumption and relevant system performance metrics. This calibration component should be able to stress each major component in order to find the basic correlation between its utilization and power consumption. The modeling component is to formulate a power model based on the collected data of power consumption and system performance metrics. Finally, using this derived power model, the real-time power consumption can be estimated in the monitoring component. It is the observable function of power metering tool that we deliver to the end-users.

Clearly, the core of power metering is the modeling component. We will emphasize on this component.

4.2.2 Power Model

The key of power metering is to find the relevant system performance metrics to represent system power consumption and formulate a power model based on this correlation. This is the major function of modeling component.

We first describe the target platform. Then we present the methodology to find the correlated system performance metrics in order to build power model. Finally, we discuss the training set and construction of power model.

Experiment Platform Selection

We evaluated a number of development kits to select the testbed as the experiment platform to validate proposed power metering solution for BBUs. The candidacy testbed should consist of both general-purposed CPU and DSP in order to process both control-plane and data-plane workloads.

The specific platform we model is Beagle Board [1], which is a low-power and low-cost single-board computer produced by Texas Instruments (TI) in association with Digi-Key. Beagle Board is based on an OMAP3530 application processor featuring an ARM Cortex-A8 running at up to 720MHz and delivering over 1200 Dhrystone MIPS of performance via superscalar operation with highly accurate branch prediction and 256KB of L2 cache. There are a number of Linux distributions available for Beagle Board, including Ångström, Debian, Ubuntu, Gentoo, etc.

The specification of its major hardware components can be summarized in Table 4.1. Although there are a number of other peripheral connectors provided by Beagle Board, such as 2 LCD connectors and audio connectors, we only focus on modeling based on the minimum number of components which implement the key functionalities of Beagle Board. Those peripheral connectors can considered as sup-

Table 4.1: Beagle Board major hardware components specification.

Hardware component	Specification
Processor	TI OMAP3530 (ARM Cortex-A8 core and TMS320C64x+ DSP)
Memory	256 MB NAND and 256 MB MDDR SDRAM
Storage	microSD card slot
Network	Ethernet adapter connected with HS USB 2.0 port

plement to the power model when necessary.

We connected the Beagle Board to a power supply with stable voltage. It was not feasible to employ multimeters for measuring current since their precision is low and coarse-grained. Thus, we used a sense resistor of low resistance to determine the current level of the system. The voltage drop was measured across the sense resistor and sent into a Data Acquisition Card. A National Instruments PCI-6024E DAQ card was used to sample data at a sampling rate of 1000 Hz, shown in Fig. 4.2. The sample data can be collected for the purpose of model construction.

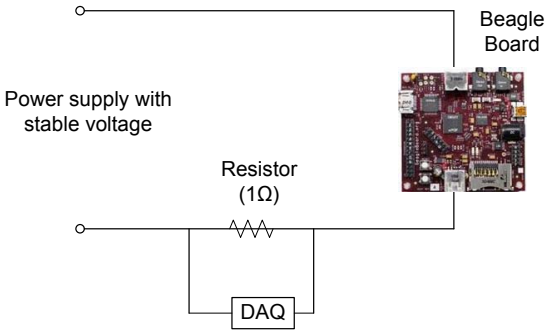


Figure 4.2: Power measurement circuit.

Hardware Component Selection

Different hardware components may contribute different amount of power consumption. It is necessary to determine which components should be taken into account for power model construction.

The most straightforward solution is by monitoring the system power consumption variation when we run an individual component intensive application with different intensity. The assumption here is that the power consumption for other components should be constant during this stress testing. In practice, this assumption cannot perfectly hold since the running applications will still involve the usage of other components slightly. However, this method is applicable to identify the hardware components that should be considered to build power model.

From our experiments, we found not all the list major hardware components have significant impact on system power consumption. For example, CPU has the largest impact while network can only contribute quite a little. We include all these major hardware components for power model construction.

Performance Metrics Selection

System power consumption varies with system resource usage. Intuitively, more resource usage leads to larger power consumption. OS-level performance metrics can represent the resource usage for each component in a fine-grained timely manner. There are hundreds of performance metrics. Amongst them, we need to select those most relevant to the system power consumption.

There are a large number of performance metrics reflecting the usage of hardware components. We pre-select a set of them based on our expertise. Then we further reduce the number of performance metrics that should be used for power

model construction. We collected performance metrics and power measurement data when running different applications stressing different components. We applied two attribute selection algorithms on this collection of data. One is Best First and the other is Greedy Stepwise, implemented in Weka [46]. Both algorithms gave the same result of attribute selection. We will discuss these relevant performance metrics with their corresponding hardware components in the following.

- CPU: There is a strong correlation between the power consumption and CPU utilization together with CPU frequency. The relation between power and frequency has been validated to be approximately cubic. In our work, we focus on build power model with consideration of CPU utilization given a CPU frequency. Admittedly, this model varies with CPU frequency. There are 6 stages of CPU frequencies for ARM Cortex-A8, from 125MHz to 720MHz. We need the model for each CPU frequency. Given a CPU frequency, the relevant performance metrics of CPU affecting power consumption are CPU utilization (u_{cpu}) and CPU I/O wait (u_{iowa}). CPU utilization consists of utilizations occurred while executing at user level, at system level, at user lever with priority, and to wait for I/O (idle). When the CPU stalls due to an outstanding of disk I/O requests, the power consumption in the circuit may decrease. So we need to distinguish this portion of CPU time.

- Memory: The power consumed by periodic refresh is very small. Most of the power is consumed by memory access (read/write). Usually, the memory activity is indicated by last level cache (LLC) miss, which is L2 cache miss in our experimental platform. L2 cache miss implies that memory will be accessed if L2 cache cannot hold the whole data set for a workload. However, reading L2 cache miss is costly and may not be acceptable in our platform in order

to achieve a fine-grained timely manner. Instead, we use the percentage of committed memory (u_{mem}) to represent the activity of memory. This metrics indicate the percentage of memory needed for current workload in relation to the total amount of memory.

- Disk: The power consumption due to disk is related to the disk read/write rate (u_{disk}). This portion of power consumption is relatively small compared with overall full system power according to existing research.

- Network: The power consumed on network depends on the network I/O rate (u_{net}). It is noticeable that the power consumption due to wired network is insignificant.

Training Suites and Model Derivation

To determine the relationship between performance metrics and power consumption for each relevant hardware components, it is necessary to have a set of training programs so that performance metrics and power consumption during run can be logged for model derivation. The workloads in this training set should be able to stress each individual hardware component without involving other components, ideally. However, this perfect case can never achieved. Instead, we selected individual component intensive workloads for this set of training suites. In addition to the completeness, one more concern of the selection of training suites is that the set of training suites should be easy to be deployed. Specifically, we used qsort in mibench [6] to stress CPU, memtester to stress memory, dbench to stress disk, and netperf to stress network, respectively. To accurately model the power consumption in the system idle case, we should include an idle run into the training suites.

When running the training suites, we use SAR [9] to collect OS-level performance

metrics at a sampling rate of 1Hz. We use an external circuit to measure power consumption of the target system. In practice, due to the limit computation resource of an embedded device, we drop the work of constructing power model to an auxiliary server.

We assume the power consumption of a system is the summation of the static power and the dynamic power due to each hardware components, that is:

$$P_{sys} = P_{cpu} + P_{mem} + P_{disk} + P_{net} + P_{static}, \tag{4.1}$$

where P_{sys} is the system power consumption, P_{cpu}, P_{mem}, P_{disk} and P_{net} are the dynamic power consumption due to each component, respectively, and P_{static} is the power consumption in the idle state.

We further assume the relationship between the relevant performance metrics and the corresponding power consumption of each component is linear. Then Eq.(4.1) can be represented as:

$$P_{sys} = c_{cpu} * u_{cpu} + c_{iowa} * u_{iowa} + c_{mem} * u_{mem} + c_{disk} * p_{disk} + c_{net} * p_{net} + P_{static}, \tag{4.2}$$

where c_{cpu}, c_{mem}, c_{disk} and c_{mem} represents the coefficients of this linear model to determined for each components.

We used Least Square Regression (LSR) to derive the power model at the auxiliary server side. For example, the modeled power equation with fixed CPU frequency of 720MHz is:

$$P_{sys} = 0.00496 * u_{cpu} - 0.00348 * u_{iowa} + 0.000507 * u_{mem} + 0.000507 * p_{disk} + 0.000036 * p_{net} + 1.195095. \quad (4.3)$$

After the model is derived, it can be sent back to Beagle board to estimate the power consumption according to the monitored performance metrics. Notice the power model is related to the CPU frequency. Different power model is needed for different CPU frequency.

4.2.3 Power Model Evaluation

Test Set Selection

The experimental platform, Beagle Board, is to mimic the real-world Base Band Units(BBUs). The major function of BBU is signal processing.

Take VoIP application running on OpenBTS [7] for instance. OpenBTS (Open Base Transceiver Station) is a software-based GSM access point, allowing standard GSM-compatible mobile phones to make telephone calls without using existing telecommunication providers' networks. It processes the functions from physical layer to network layer. In the scenario of VoIP application, the data plane tasks to be processed on BBU consist of codec/decodec on DSP and packetization/depacketization on ARM processor. The ideal test set could be a VoIP workload to represent real-world application.

However, as we haven't included DSP in our power metering solution, we focus on the workload on ARM processor partially at first stage. In addition, porting OpenBTS to Beagle Board is in progress. So we select other relevant compute-bound

workloads as the test set.

We validated our proposed power model on three representative telecommunication applications from mibench: crc32, fft and gsm.

- CRC32: This benchmark performs a 32-bit Cyclic Redundancy Check (CRC) on a file. CRC checks are often used to detect errors in data transmission. The data input is the sound files from the ADPCM benchmark.

- FFT(/IFFT): This benchmark performs a Fast Fourier Transform and its inverse transform on an array of data. Fourier transforms are used in digital signal processing to find the frequencies contained in a given input signal. The input data is a polynomial function with pseudo-random amplitude and frequency sinusoidal components.

- GSM encode/decode: The Global Standard for Mobile (GSM) communications is the standard for voice encoding/decoding in Europe and many countries. It uses a combination of Time- and Frequency-Division Multiple Access (TDMA/FDMA) to encode/decode data streams. The input data is small and large speech samples.

Experimental Results

To quantify the estimation accuracy of this power model, we introduced two metrics: mean absolute percentage error (MAPE) and root mean square error (RMSE). MAPE is defined as the average of the absolute values of the errors, i.e.,

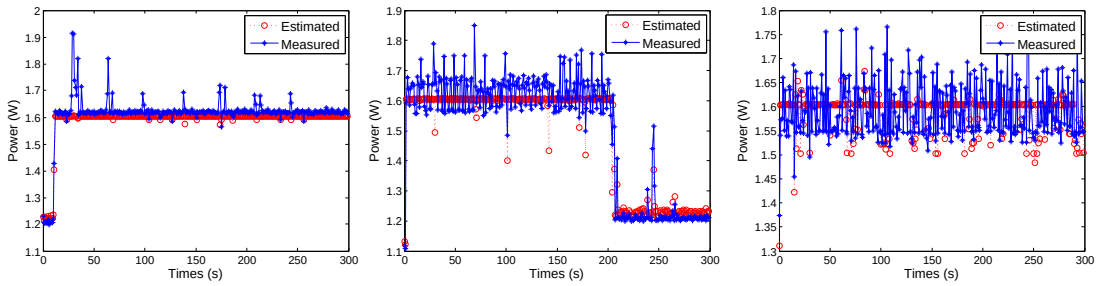
$$MAPE = avg(\frac{|estimated - measured|}{measured}). \tag{4.4}$$

RMSE is employed to study the proportion of the errors. RMSE is defined as the square root of the mean square error:

$$RMSE = \sqrt{avg((\frac{|estimated - measured|}{measured})^2)}. \tag{4.5}$$

We validated the accuracy of power model when we fixed the CPU frequency to be 720 MHz. This power model is shown in Eq.(4.3). Similar results can be observed for other CPU frequency settings.

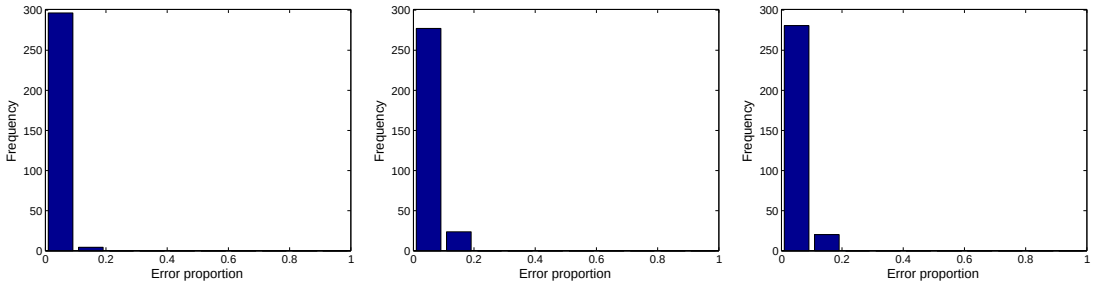
Fig. 4.3 shows the modeled and measured power consumption for each application. Power consumption is estimated for 1 second interval. Fig. 4.5 depicts the error histograms. We can see most errors will be limited into 10%.



(a) crc32 (b) fft (c) gsm

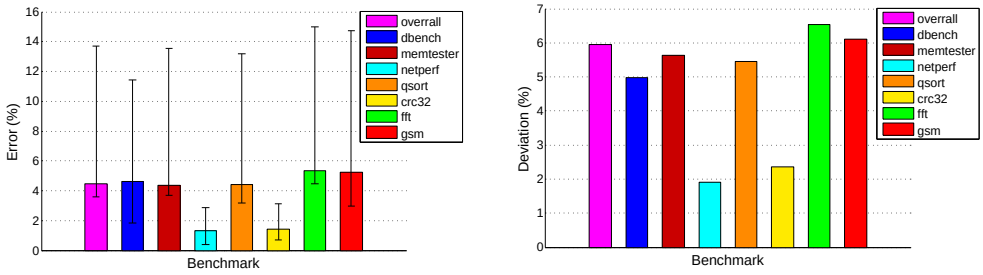
Figure 4.3: Power estimation of selected applications.

We further show MAPE and RMSE of power model for both training set and testing set in Fig. 4.5(a) and Fig. 4.5(b), respectively. The 95 percentile error bar is also shown in Fig. 4.5(a). We can see the average error of this power model can be bounded to 7%. And RMSE can be limited to 20%. These results indicate high accuracy of this power metering solution for online power estimation.



(a) crc32 (b) fft (c) gsm

Figure 4.4: Power estimation error proportion of selected applications.



(a) Average error of power model with 95 percentile.

(b) RMSE of power model.

Figure 4.5: Average error and RMSE of power model.

4.2.4 Integration with DSP

DSP is included in Beagle Board to accelerate digital signal processing. This portion of power consumption is noticeable when DSP is used for processing. To our best knowledge, our work should be the first one to address power model in the scenario of heterogeneous multicore environment including ARM and DSP. We present the methodology to estimate this portion of power consumption via experiments. Then we provide the guideline to refine the existing power model to include DSP.

We conducted experiments to show the impact on power and performance due to DSP in comparison with ARM processor. The applications/benchmarks selected for this experiment include matrix multiplication (m-m), and fast Fourier transform

Table 4.2: Power consumption and execution time of DSP (in comparison with ARM).

Applications		ARM		DSP	
		Exec time (s)	Avg power (W)	Exec time (s)	Avg power (W)
M-M	600	12.62	1.7611	2.191	1.4123
	900	50.47	1.7331	20.517	1.4071
	1200	132.12	1.719	49.286	1.3923
FFT	4096	3.595	1.7251	0.5086	1.4432
	8192	7.888	1.7385	1.07	1.4353
	16384	17.214	1.7064	2.43243	1.4297

(fft). All these applications have two versions. One is to run on ARM processor. The other is to run on DSP.

Table 4.2 shows the experimental results in terms of execution time and system-wide power consumption when running applications on DSP and ARM. As these applications are compute-bound, the processor is always fully utilized. We can see the power consumption when running on DSP is much less than that on CPU. We assume the idle power for the system is around 1.2W according to Eq.(4.3). Then the dynamic power consumption due to running workloads on DSP is around 0.2W, which is around 17% compared with the idle power. And this portion when running on ARM can be as large as 45%.

Using DSP can accelerate the execution of compute-bound workloads if DSP library is used. We can see the speedup when using DSP can be at least 2.5X compared with the execution time using ARM processor.

From our experimental results, we can see running DSP will always lead to the increase of system-wide power consumption by around 0.2W. Thus we can assume that the power consumption due to DSP can be regarded as a constant once it is running. Thus, we can refined our power model Eq.(5.5) to the following equation:

$$P_{sys} = c_{cpu} * u_{cpu} + c_{iowa} * u_{iowa} + c_{mem} * u_{mem}$$
$$+c_{disk} * p_{disk} + c_{net} * p_{net} + c_{dsp} * p_{dsp} + P_{static}, \quad (4.6)$$

where p_{dsp} is a constant representing the running power of DSP and c_{dsp} is a $\{0, 1\}$ variable indicating whether DSP is running or not. However, the current lack of capability of detecting DSP running, which is due to the limits of up-to-date experiment environment, restricts us from implementing this solution into our power metering prototype.

4.3 Power Management Middleware

Power management is always with consideration of performance. To provide the function of power management in BBUs, we proposed a general-purposed power management middleware and deployed it on BBUs. It can achieve different levels of tradeoffs between power and performance upon users' request.

4.3.1 Architecture

The power management module for BBUs is designed as a middleware which lies between the application level and OS level. The major function of this middleware is to provide the functionality to achieve different levels of tradeoff between power and performance. It also can monitor the power consumption and performance in a real-time manner. In our testbed, we consider beagle board as the target platform (BBU). The underlying OS is Ångström7 and the workload running at application

level can be data processing for VoIP or streaming applications.

This power management middleware is designed based on a client-server model, as shown in Figure 4.6. The client side, named Power Management Client (PMC) resides within the target platform, which is beagle board in our environment. The server side, named Power Management Server (PMS), can be deployed on an external server or board. The message passing between PMC and PMS is over a TCP connection. The reason for us to design PM middleware in a client-server model-based manner is three-fold. First, this distributed design can alleviate the overhead on the client, which is the target platform. Second, with this design, sleep state of the target platform can be applied. Third, this distributed design can be adapted to energy-aware BBU cluster design in the future.

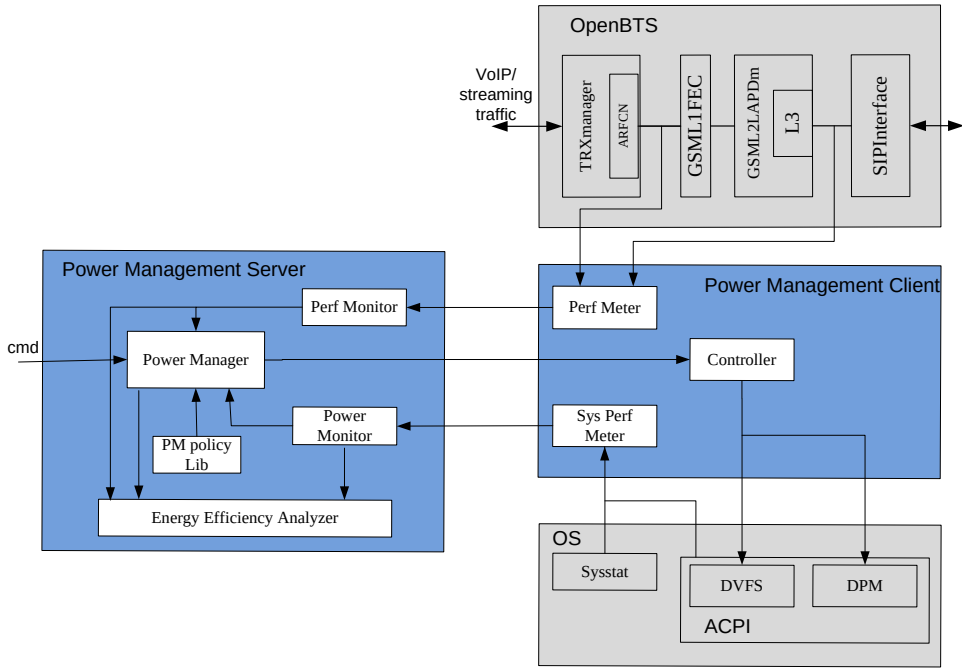


Figure 4.6: Architecture of PMM.

4.3.2 Design of Power Management Client (PMC)

PMC consists of the following major function units:

- Perf Meter: the input of this meter is the performance reported from application layer. The reported performance metrics could be delay, packet loss, etc. Considering the VoIP application processed by OpenBTS, one of the most important performance metrics is delay. We consider the delay caused by packet processing from L1 to L3 as the major performance metric in this scenario. The output of performance collector is the samples of performance metrics which will be sent to the Performance Monitor at server side periodically.

- Sys Perf Meter: the input of this meter is the OS-level system performance metrics reported from the underlying platform. We pre-select which OS-level metrics should be reported from OS by using sysstat (SAR). These metrics should be related to the power metering solution that we will use at the server side. The OS-level performance metrics that need to be monitored include CPU utilization, memory utilization, network I/O rate, and disk I/O. The current CPU frequency is also directed to this collector. The output of this collector is the OS-level metrics with CPU frequency which will be sent to server end for the purpose of power metering.

- Controller: this component gets the input from Power Manager from PMS. The input is the power management decision. Then this control component can map the decision to the action that should be taken by the system. For example, this action could be frequency scaling using the command:

  "`echo 550000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed`",

which is to set CPU frequency to 550MHz or entering sleep state by using "`echo mem > /sys/power/state`".

4.3.3 Design of Power Management Server (PMS)

PMS includes the following components:

- Performance Monitor: this monitor can retrieve the performance metrics from Performance Collector periodically. Then it can get the average performance for a user-defined period. These performance metrics will be sent to Power Manager to make the decision of power management.

- Power Monitor: the real-time power consumption can be either measured by an external power meter or estimated using a power metering tool. In this Power Monitor, we estimate the power consumption by the OS-level metrics passed from client side using our power metering tool. The output of this monitor is the estimate (or real measurement) of current power consumption of the system. The power consumption is also an input of Power Manager to guide the power management decision.

- Power Manager: it has two major functions. One is to select the proper power management policy from PM policy component according to the operators requirement (it can be input from command line). The other is to make power management decision (determine the CPU frequency or whether to sleep or wake-up of the BBU) according to the policy and reported performance metrics as well as power consumption. The performance metrics used in Power Manager might be normalized according to the pre-defined tolerable threshold.

- PM policy Lib: This power management middleware may contain a number of

power management policies for different purposes. We store these policies in PM policy Lib providing various options of tradeoff between power and QoS. For example, the policies we consider may include "shallow", "medium" and "high" in terms of saving of energy. Currently, we have designed three policies for this middleware. These policies are to limit power consumption, or provide power-aware statistical QoS guarantee, or achieve tradeoff between power and performance. The details of the included power management policies will be discussed in Chapter 5, 6 and 7 in different contexts.

- Energy efficiency analyzer: This component provides the analysis of energy consumption saving and performance loss. It obtains power and performance data from corresponding monitors. Then energy efficiency can be analyzed for the selected power management policy.

4.3.4 Cross-Layer Message Passing

The message passing from application layer to PMM is the application performance. In our testbed, GNURadio [5] is running at application level. The L1 to L2 packet processing delay could be one important performance metric for power management decision, which will be sent to Performance Collector in PMC. There is usually a specific QoS requirement for the performance metrics. This QoS metric, for example, in terms of delay, is related to the specific application. For example, in VoIP application, this user-acceptable end-to-end delay is usually 150ms.

OS layer has the interactions with PMM as well. If the power consumption is estimated by our previous power metering tool, the sysstat (SAR) tool will report the OS-level performance metrics (CPU, memory, etc.) as well as current CPU frequency periodically to OS Metrics Collector in PMS. When the Control component

in PMC gets power management decision from PMS, it will contact OS layer to take corresponding actions.

We employ a sequence diagram to illustrate the message interactions in PMM. To simplify the demonstration of these interactions, we only consider how the PM policy is selected and then the PM decision is made to be sent back to PMC. Figure 4.7 shows the sequence diagram of the message interactions for this case. We don't split the PMC into different function components to communicate with PMS for the simplicity of the diagram. We also omit the interaction between PMC and platform. We emphasize on the interaction mostly related to PM decision making. Notice the PM decision will be conducted periodically. In our case, this control interval can be 1 second.

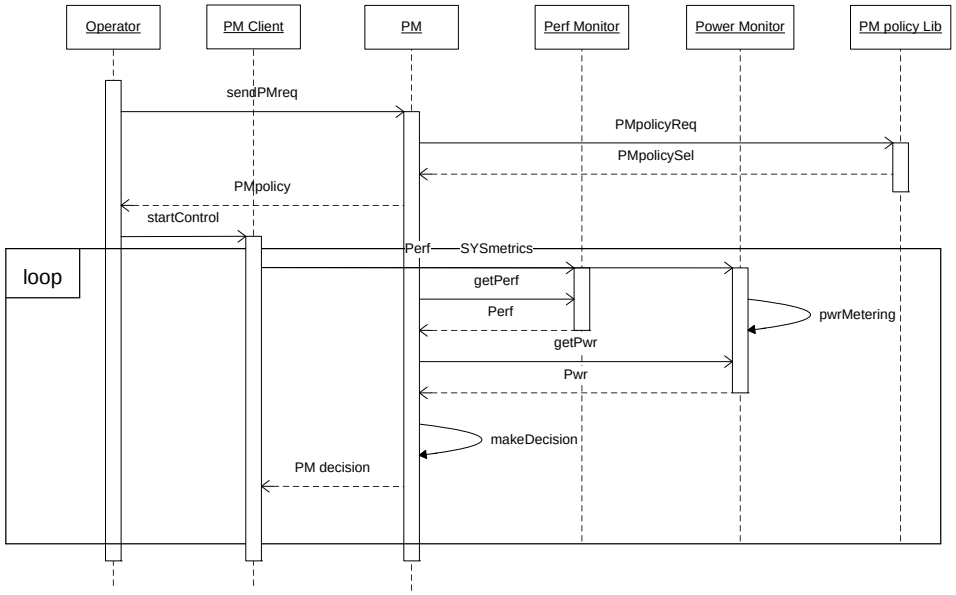


Figure 4.7: Sequence diagram of PM decision making.

Chapter 5

System-Level Peak Power Management

Traditionally, servers are designed to provide for the worst-case scenario by over-provisioning which adds costs with only few benefits for the real environments. In contrast, a "better-than-worst-case" design approach has been adopted by limiting power consumption (power capping) [30]. So that system failure due to power capacity overload or overheating can be avoided and contractually negotiated system load can be guaranteed. In addition, power capping is a key element for implementing power shifting, which is a dynamic setting of power budgets for individual servers so that a global power cap for a cluster can be maintained. Power consumption relies on system resource usage. However, system usage is hard to predict in a system with variability. Feedback control approach is widely used to regulate system configuration dynamically. In this chapter, we provide a strategy to control system-level power consumption in an accurate and responsive manner.

5.1 Black-box Feedback Control for Power Management

We first discuss the feedback control in the context of power management. Then we present the design of a PID controller for power capping.

5.1.1 Overview of Feedback Control

Feedback control is to use measurements of a system's outputs to adjust the system control inputs in order to achieve externally specified goals. In the context of using black-box feedback control for power management, the output is the measured power and the reference input is the power cap. The control input, also known as the control knob, mostly focuses on the power management techniques supported by processors. This control knob can be frequency and voltage (P-state) using Dynamic Voltage/Frequency Scaling (DVFS), or sleep states (C-state) that reduce power consumption when parts are idle, or the throttle state (T-state) [70]. In this work, we use CPU frequency (P-state) as the control knob, called CPU speed. The power is measured and fed back for use in the control computation to get the control input for the next control period so that the measured power in the next control period can be settled to the power cap.

Applying feedback control in power capping, we need to consider the following properties. First, applying the controller should make the system stable. In a server with highly variable traffic, the measured power oscillates due to the change of resources required by the traffic. Applying a feedback controller, this oscillation still exists since the dynamics of the system cannot be removed. This system is Bounded-Input-Bounded-Output (BIBO) stable since the output is bounded.

Second, the measured power can converge to the power cap with the controller, which is called accuracy. However, it is difficult for a system to always operate sufficiently close to the power cap due to the system dynamics. In this scenario, we would emphasize on that the measured power could be settled below the power cap since the main purpose of the power controller is to avoid power overloading.

Third, this convergence process should be short. The time taken for this process is the settling time. On one hand, fast settling to power cap when the current power consumption is below power cap can fully utilize server resources earlier. On the other hand, responsive adjustment to satisfy power cap when current power consumption is beyond power cap will reduce the possibilities of both outage and exceeding of negotiated load. For a web server, this property is of particular importance in the presence of highly variable traffic. The dynamic of traffic requires different amount of system resources over time which leads to fluctuation of system power consumption. In the context of power management in dynamic systems, the settling time would be emphasized on how long it will take for a system to operate below the power cap in order to avoid power overloading.

Fourth, using the controller will not let the measured power exceed the power cap greatly.

The first two properties were widely studied in previous studies while the last two were overlooked. In our work, we focus on the property of settling time since it is closely related to the system performance and system failure due to power overloading.

Lefurgy et al. [70] designed a Proportional (P) controller with a first-order delta-sigma modulator to control peak power for a server, based on a linear model between maximal power consumption and CPU speed. This controller works well for the system with static workload. In case of dynamic workloads, this controller may not perform well; see the following example.

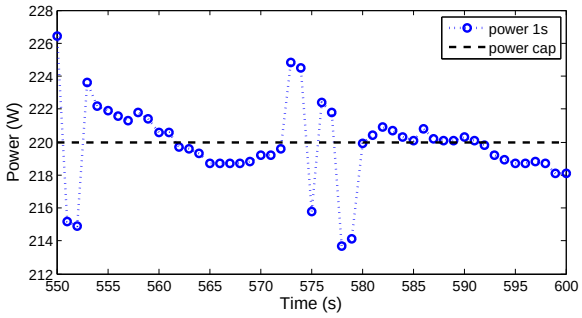


Figure 5.1: Example of a long settling time of P controller.

We applied this P controller on a web server running SPECweb2005 benchmark [8]. SPECweb2005 defines a representative workload cluster of applications for web servers. The system power consumption changes with workload which can be reflected by the number of clients. We used the power model for 1500 clients as the nominal system model and found this P controller theoretically can let power settle in 5 seconds (assume the control period is 1 second) for the scenario of 900 clients. But from experiment results, it may take be as long as 8 seconds or even longer to settle power consumption below power cap, shown in Fig. 5.1. This is due to the variability of Internet traffic that leads to remarkable variability of power consumption in contrast to the static power consumption when running CPU-intensive workloads used in [70]. In this case, the controller should not be designed based on the maximal power model. Using the first-order delta-sigma modulator will scale the CPU speed frequently even within one control period, which is not expected in a stable system. Thus we design the controller to limit power consumption without this modulator.

5.1.2 Design of the PID Controller

We design a PID controller, as illustrated in Fig. 5.2, based on the model that studies the relationship between current power consumption and CPU speeds. This

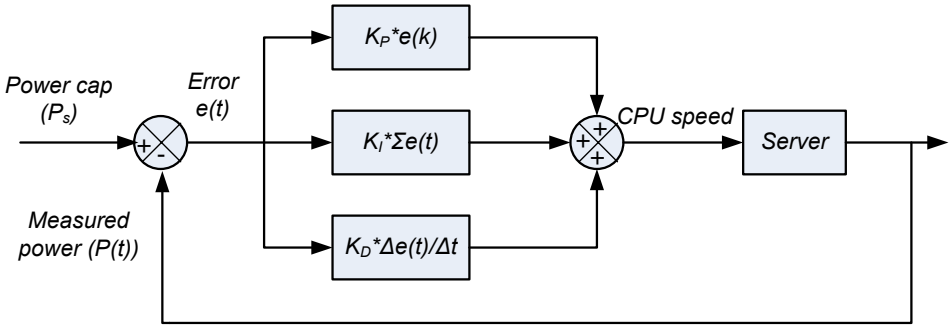
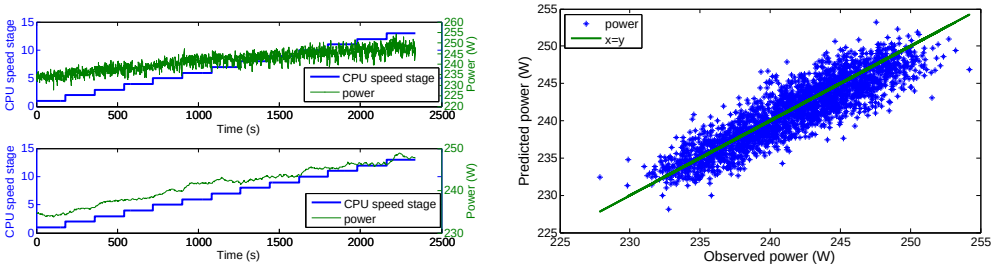


Figure 5.2: PID based power cap controller.



(a) Data used in system identification

(b) Model evaluation

Figure 5.3: Data for system identification and model evaluation.

relationship for a web server running SPECweb2005 with 1500 clients is depicted in Fig. 5.3(a). Fig. 5.3(a) plots the instant and average power (averaged over a 1-minute interval) under different CPU speed settings after ramp and warm-up phases. The CPU speed stage is increased by one stage every 180s.

Let $p(k)$ and $s(k)$ be the power consumption and CPU speed of the server in the kth control period, respectively. In the k^{th} control period, given $p(k)$, the control goal is to choose a CPU speed $s(k)$ so that $p(k+1)$ can converge to the power cap in a finite number of control periods. We use the differences of $p(k)$ and $s(k)$ with their operating points to build a linear model. The control output of this model is $\hat{p}(k) = p(k) - \bar{p}$, the control input is $\hat{s}(k) = s(k) - \bar{s}$, and the desired value of the control output is $\hat{P}_s = P_s - \bar{p}$, where $\bar{p}$ and $\bar{s}$ are operating points, and P_s is the power

cap, also known as set point. The operating points are desired steady values of CPU speed and power consumption in the power management context.The system model is:

$$\hat{p}(k) = a\hat{p}(k-1) + b\hat{s}(k-1), \tag{5.1}$$

where a and b are system parameters.

The time domain form of this PID controller is:

$$\hat{s}(k) = K_P e(k) + K_I \sum_{j=1}^{k} e(j) + K_D[e(k) - e(k-1)], \tag{5.2}$$

where $e(k) = \hat{P}_s - \hat{p}(k)$, K_P, K_I and K_D are control parameters.

We estimated the system parameters a and b in Eq. (5.1) by system identification using the nomial model shown in Fig. 5.3(a). The middle value of a range of possible CPU speed settings is selected as the operating point of CPU speed, $\bar{s}$, while the average power consumption under setting is $\bar{p}$. By system identification, we get the system parameters for Eq. (5.1), $a = 0.4093$ and $b = 0.6450$. Fig. 5.3(b) plots the measured and predicted power. There are a number of approaches to indicate how well the model fits. Amongst them, the coefficient of determination is a widely-used term, denoted as R^2. The R^2 value in this model is 0.84.

We use Root-Locus method [53] to design this PID controller. Constraining the settling time to be 1 control period can lead to the maximum overshoot of around 3%. By studying the root locus of this PID controller, we find the settling time can be 1 if the derivative term is zero or very close to zero. So we set $K_D = 0$. Thus this PID controller is reduced to a PI controller with $K_P = 0.6322$ and $K_I = 1.5478$. The

poles [53] that determine the stability of the closed-loop system are $0.00016 \pm 0.0241i$, which are inside the unit circle. Thus this closed-loop system is stable. The accuracy is guaranteed by the PI controller itself.

We analyzed the performance of this PI controller for the scenarios of 900 and 2100 clients as well. The stability and accuracy are guaranteed. The settling time is 7 for 900 clients and 2 for 2100 clients, similar to the theoretical results using previous P controller. In practice, we can see this PI controller can lead to shorter settling time in contrast to the P controller in Section 5.4. However, it may still need more than 5 seconds occasionally. Thus we need a controller to control system power more responsively.

5.2 A Gray-box Approach

In this section, we will first present the architecture and basic idea of the gray-box approach to manage peak power of a web server while maximizing performance. Then a model-predictive feedback controller is developed based on this approach. We present the model prediction component of this controller by proposing two models on which our gray-box approach is based.

5.2.1 Architecture

The architecture of this gray-box approach is shown in Fig. 5.4. It consists of two loops. One is a feedback loop that sums up the errors between the power cap and the measured power and adjusts the future CPU speed setting. The other is a model prediction loop that predicts the control input for the next control period.

The feedback loop employs a controller to sum up the error and adjust the future CPU speed. An external power meter is used to measure power. To predict

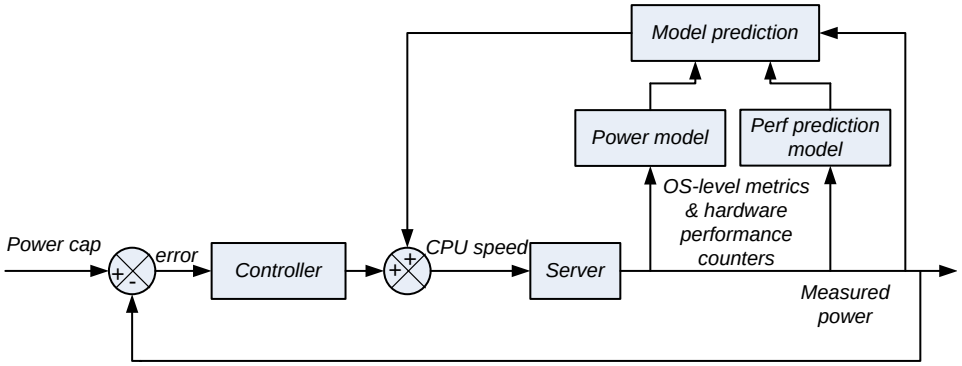


Figure 5.4: Power control loop.

the future CPU speed setting, this approach relies on gray-box monitoring to collect performance events, including both OS-level metrics and hardware performance counters. A light-weight profiling engine residing in the server was implemented for the data collection. These collected performance events will be used by a power model and performance prediction model. The CPU speed for the next control interval is predicted by estimating power consumption at different speeds based on these two models. The one which can lead to the largest power consumption but below the power cap is the predicted speed. This prediction will combine with the adjustment from error. Thus the CPU speed for the next control period can be determined. Because of the model prediction function, this approach is proactive, which means it is not only reactive on the measured power but also can predict the control input based on the usage statistics.

To distinguish the power controller based on this gray-box approach from other ones, we name it M-controller since it is a model predictive feedback controller.

5.2.2 Controller Design

Based on the results in Section 5.1, the derivative term can be omitted for the proposed PID controller. Thus, Eq. (5.2) can be re-written as:

$$s(k) = \bar{s} + K_P e(k) + K_I \sum_{j=1}^{k} e(j), \tag{5.3}$$

where K_I and K_P determine the adjustment of the CPU speed corresponding to the current and past errors. Let f denote a model predictive CPU speed based on the power model and performance prediction model of a server. It follows that the CPU speed of the k^{th} control period is:

$$s(k) = f(k) + \alpha \hat{g}[K_P e(k) + K_I \sum_{j=1}^{k} e(j)], \tag{5.4}$$

where $\hat{g}$ is a model-predictive scaling factor of CPU speed with respect to the change of output error in the power consumption, and α is to tune the weight of this scaling factor. A controller with a large control factor can respond quickly to the errors. But it may cause oscillation. To reduce the overshoot, we prefer a smaller control factor by setting α to be a small value (e.g., $\alpha = 0.1$).

5.2.3 Model Prediction

The gray-box approach periodically predicts the future power consumption under different CPU speed settings to improve the controller's agility by monitoring performance events. This prediction is based on two models: power model and performance prediction model.

Power Model

We correlate a few OS-level metrics and hardware performance counters with power consumption of a system to build a power model. The purpose is to use this derived model to predict power consumption based on collected performance events.

Economous et al. [36] presented Mantis which provides a fast and accurate full-system power prediction model by low-overhead OS utilization metrics and performance counters. In our work, we adapt the approach in Mantis to construct power model with more analysis on the details of CPU utilization. We use the following OS-level metrics and performance counters to construct the power model:

- CPU utilization (u_{cpu}): It consists of utilizations occurred while executing at user level, at system level, at user lever with priority, and to wait for I/O (idle). It can be represented by $100\% - idle\%$.

- CPU I/O wait (u_{iowa}): When the CPU stalls due to an outstanding of disk I/O requests, the power consumption in the circuit will decrease. We distinguish this portion of CPU time, which is different from Mantis.

- L2 cache miss (u_{L2miss}): Memory will be accessed if L2 cache cannot hold the whole data set for a workload. This metric indicates the activity of memory.

- Disk read/write rate (u_{disk}): Existing research showed the variation between power of standby mode and that of active mode is relatively small compared with overall full system power.

- Network rate (u_{net}): The power consumed on network depends on the network I/O rate.

We assume this power model is linear as follows:

$$P_{system} = C \bigotimes U + P_{leak}, \tag{5.5}$$

where $C = \left[c_{cpu}, c_{iowa}, c_{L2miss}, c_{disk}, c_{net}\right]$, $U = \left[u_{cpu}, u_{iowa}, u_{L2miss}, u_{disk}, u_{net}\right]^T$, and $\bigotimes$ is an inner product operator. The measurements of the metrics in vector U will be discussed in Section 5.3. Clearly, C represents the coefficients of this linear model to be determined. Different CPU speeds have different power models.

Performance Prediction

The system behaviors would be affected due to CPU frequency scaling, which can be observed by performance events.

We designed a model to predict the future OS-level metrics and hardware counters under different CPU frequencies. Let $U(k)$ denote the vector of metrics in the kth control period. Without loss of generality, we index the elements in $U(k)$ by an integer from 1, i.e., $U(k) = \left[u_1^k, \cdots, u_n^k\right]^T$. CPU speed stages are indexed by h_i, $i \in \{1, 2, \cdots\}$. The size of $U(k)$ is determined by the number of metrics monitored. We conduct the following formula to represent the transition from current metrics vector to future metrics vector if we change CPU speed from h_i to h_j:

$$U(k+1) = A^{i,j} \times U(k) \tag{5.6}$$

where $A^{i,j} = \begin{bmatrix} a_{11}^{i,j} & \cdots & a_{1n}^{i,j} \\ \vdots & \ddots & \vdots \\ a_{n1}^{i,j} & \cdots & a_{nn}^{i,j} \end{bmatrix}$. $A^{i,j}$ defines the effect on metrics if CPU speed is scaled from f_i to f_j, which can be obtained using Least Square Method (LSM) given $U(k+1)$ and $U(k)$.

The transition matrix $A^{i,j}$ $(i < j)$ can be calculated as:

$$A^{i,j} = A^{i,i+1} \times A^{i+1,i+2} \times \cdots \times A^{j-1,j} \tag{5.7}$$

5.3 Model Construction

In this section, we first present the experimental environment for the construction of the models. Then we show the estimation of parameters of the models.

5.3.1 Experiment Environment

Experiments were conducted on a Dell PowerEdge1950 server with two Intel quad-core Xeon E5450 processors, 8GB memory, one 250GB SATA hard disk, and 1Gb Ethernet interface. The processors could be operated at four frequencies ranging from 2.0GHz to 3.0GHz. All the cores can run at different speed in pair. There are 13 possible CPU speed stages in total, indexing from 1 to 13 in an ascending order. To scale CPU speed, we can write the new frequency level in to a system file.A BIOS routine will periodically check this file and reset the CPU frequency accordingly. The average overhead for the BIOS to change frequency was around 1ms according to our experimental results.

We patched Linux kernel 2.6.18 with perfctr patch so that we can read hardware performance counters on-line with relatively small overhead compared with OProfile. We use SAR to collect run-time OS-level metrics for CPU utilization, hard disk I/O, and network I/O. The power meter we use to obtain the power consumption value is WattsUp Pro [3]. This power meter has an accuracy of $\pm 1.5\%$ of the measured RMS power with sampling rate of 1Hz. The workload was SPECweb2005.

5.3.2 Model Parameters Estimation

While modern processors provide counters for measuring a large number of different events, the number of events that can be counted concurrently is typically quite small (2-4) if we don't use the multiplexing technique. In our experimental environment with Intel Xeon 5450 Quad core which is Intel Core 2 architecture, there are only two programmable counters. Since both the L2 cache miss and L1 cache miss can only be counted in a specific performance counter (pmc0) in Intel Core 2 architecture, we can only use the counts of L2 cache miss instead of L2 cache miss ratio.

The hard disk will be in standby state if there is no workload for 30 seconds, typically. In our experiment, SPECweb2005 benchmark will read from hard disk frequently and let it always be active. The variation of power consumption for hard disk is very small when it is active. In our power model, we can see the coefficient for this variable is almost 0. So we can simplify the model (5.5) by removing u_{disk} from U and c_{disk} from C.

The related performance events with power consumption were collected during a set of runs under different CPU speed settings in order to obtain the coefficients of the power model. We use pace regression to solve this regression problem. Pace Regression [114] improves on classical ordinary least squares (OLS) regression by evaluating the effect of each variable and using a clustering analysis to improve the statistical basis for estimating their contribution to the overall regression. WEKA [46] provides an API for user to draw the pace regression model from an input data set. The linear models fit well ($R^2 > 95\%$) for all CPU speed stages.

To obtain the transition matrices for power prediction model, We collected traces by scaling the CPU speed to its next step and scaling back to the original setting periodically for off-line training when running workload. The difference of performance

events between two periods can be attributed to both the CPU frequency scaling and the change of workload. Due to the variability of the workload, the effect caused by the workload might not be negligible. Thus we need to pre-process the collected trace to reduce the noise with the assist of measured power. Since the power consumption will increase monotonically with scaling up CPU speed in an ideal scenario, we will remove the data that violate this principle. In addition, we filter the traces by checking the gap between the system power before scaling and that after scaling as well. If this gap is larger than a threshold, e.g. 5%, relative to before-scaling system power, we attribute this big gap to the significant variability of workloads and will not use the corresponding data for training model.

5.4 Evaluation

In this section, we present the experimental results for the gray-box approach. We validate the power model and performance prediction model. We compare our approach against several other representative power controllers in terms of responsiveness and impact on application performance.

5.4.1 Experimental Methodology

The M-controller was implemented within the controlled server instead of using another dedicated machine to communicate with the controlled server periodically. The CPU utilization of running this controller is less than 0.5%.

We conducted experiments to show the responsiveness and impact on application performance of M-controller compared with the existing P, PID and an ad-hoc controller which is a representative industrial power controller. We followed [70] to design P controller and the PI controller is designed in Section 5.1. The basic idea

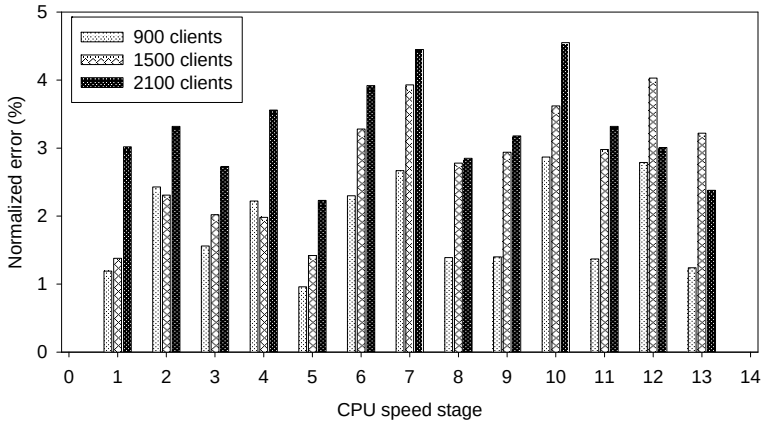


Figure 5.5: Average error of power model.

of ad-hoc controller is to simply raise or lower CPU speed by one step depending on if the measured power is lower or higher than the power cap. All the controllers are designed based on the system identification for a run of SPECweb2005 with 1500 clients. The execution time for a run of SPECweb2005 was set to be 5000 seconds with 180 seconds ramp-up time and 300 seconds warming time. All the controllers were evaluated in the scenarios of 900, 1500, and 2100 clients. The number of clients reflects the load and bursts of a server.

5.4.2 Model Validation

Fig. 5.5 shows the accuracy of our proposed power model. The accuracy is defined as the ratio of the mean absolute error between the predicted power and measured power to the measured power. For SPECweb2005 benchmark, the average error ranges from around 1% to 4.8%. The results show this power model can estimate power consumption accurately based on the collected metrics.

We evaluated the accuracy of the performance prediction model as well. Due to space limit, we only take one scenario, from speed stage 5 to 6, for example. Fig. 5.6

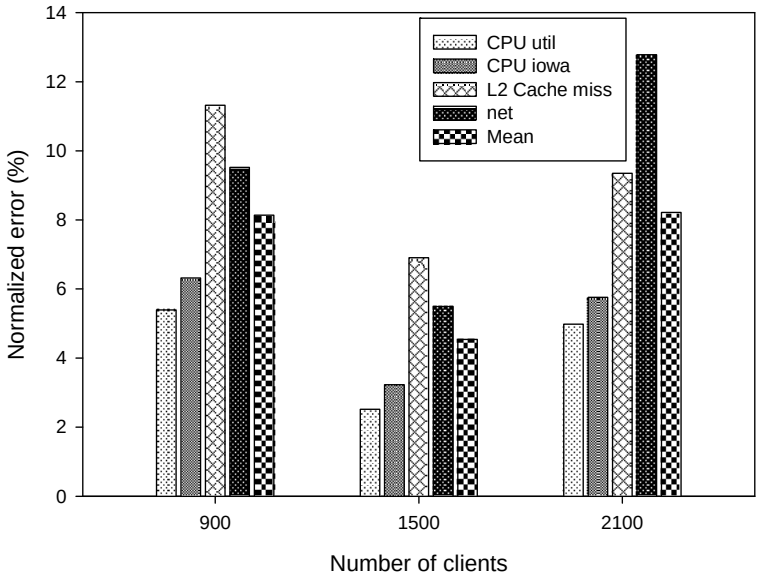


Figure 5.6: Average error of performance prediction model.

shows results of performance prediction. We can see the prediction error on the metrics related to CPU ranges from around 2% to 6%. The prediction error on other metrics can be larger than 10%. We use arithmetic mean as a collective metric to summarize the errors of all elements of the vector due to the linearity of the power model. The arithmetic mean of normalized error for predicting a vector of metrics can range from 4% to 8%.

5.4.3 Controller Responsiveness

In this part, we investigated the responsiveness of M-controller in comparison with other controllers.

The power caps are selected close to the average power of running workload at the middle CPU speed stage. The power caps are set to be 220W, 240W, and 260W for 900, 1500, and 2100 clients, respectively. The data were collected after ramp-up and warming time. We take the scenario of 900 clients for example. The results for

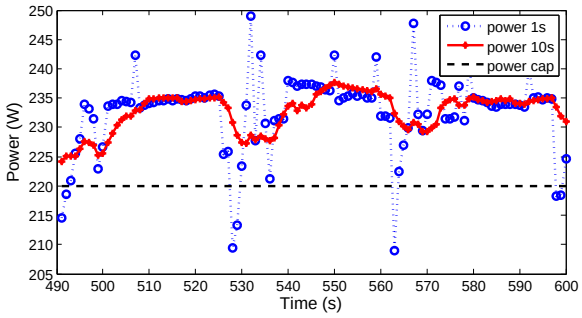
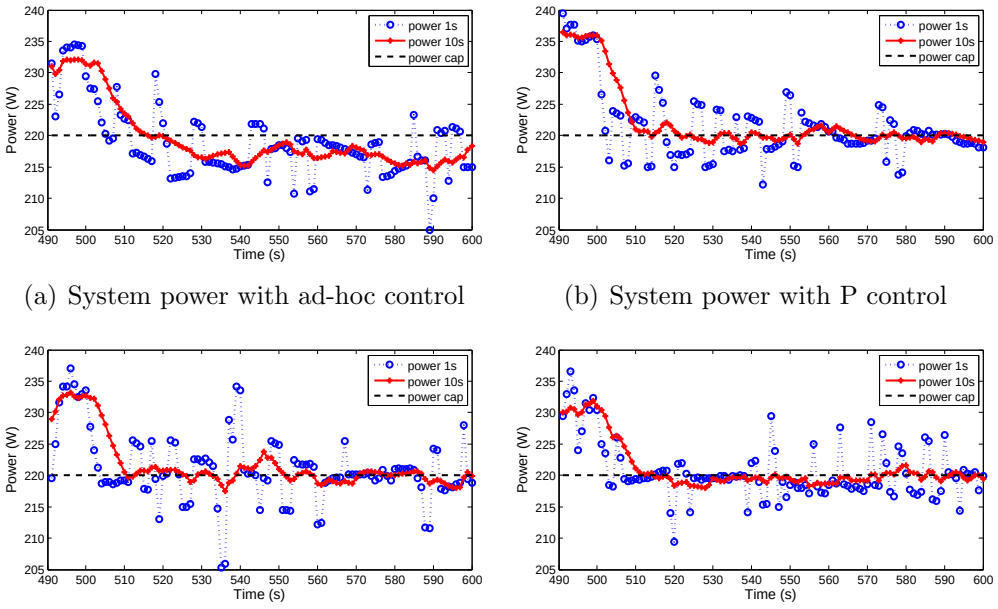


Figure 5.7: System power without power controller: 900 clients.

1500 clients and 2100 clients are not presented for brevity since similar observations can be found.

The power consumption of a server running at full speed without any power controller may exceed the power cap for a long period, shown in Fig. 5.7. Both the instant power consumption (1 second) and average power consumption (10 seconds) are plotted. The power consumption would fluctuate due to the high dynamics.



(a) System power with ad-hoc control

(b) System power with P control

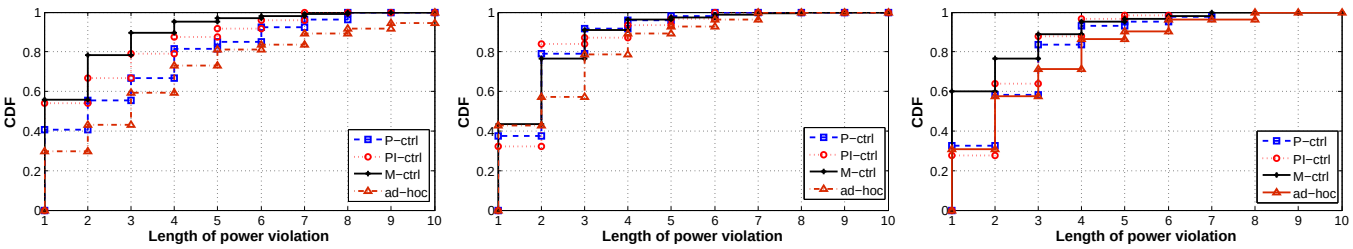
(c) System power with PI control

(d) System power with M control

Figure 5.8: Performance of controllers: 900 clients.

Fig. 5.8 plots the system power when applying controllers. With controllers applied on 500s, the peak power consumption of a system can be controlled as most time the average power is below the power cap. Because of the oscillations of power caused by the workload itself, the system power consumption cannot always stay at the power cap even with power controllers.

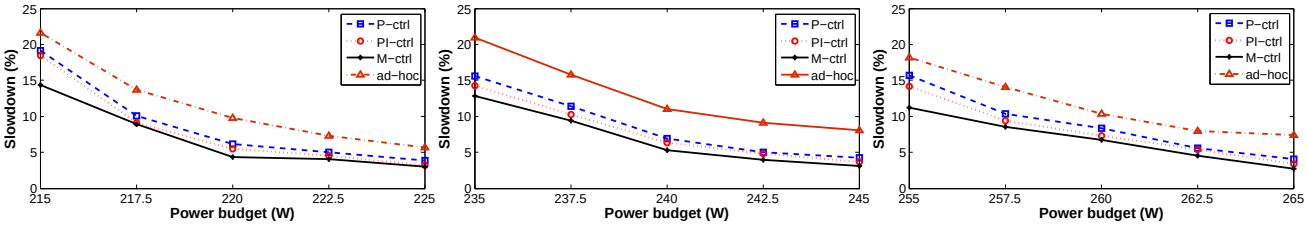
The ad-hoc controller raises or lowers CPU speed by one step at each control period. Thus it is slow to response the power change. The average power is far below the power cap, which shows the ad-hoc controller is more conservative. A P controller can get the system to operate close to power cap faster but it sometimes may need a long period to settle, e.g., from 552s to 560s in Fig. 5.8(b). A PI controller has similar effect to the P controller. Using an M-controller can accelerate the process of settling to power cap while the oscillations still exist. The average power by using M-controller is close to the power cap and the instant power could be pulled back to the power cap more quickly when it exceeds the power cap in contrast to those using P and PI controller.



(a) 900 clients (b) 1500 clients (c) 2100 clients

Figure 5.9: CDF of length of power violations

To show the responsiveness of these controllers, we analyzed the lengths of power violations after the controllers have been applied. Power violation is defined as that the measured power exceeds the power cap. Since the main purpose of these controllers is to limit the power usage, more responsiveness means shorter length of



(a) Slowdown: 900 clients (b) Slowdown: 1500 clients (c) Slowdown: 2100 clients

Figure 5.10: Impact on application performance

power violations. The effect of settling to power cap when measured power is below power cap can be reflected in the impact on performance, which is not discussed here. Fig. 5.9 shows the cumulative distribution function (CDF) of lengths of power violations for different settings. We can see more than 75% of power violations will have the length of no longer than 2 for M-controller, which means M-controller is responsive. In case of 900 and 2100 clients, M-controller is more responsive than the others and PI controller is more responsive than P controller. In case of 1500 clients, we can see the P and PI controllers are almost as responsive as, or more responsive than M-controller. It is because these two controllers are built based on system identification of 1500 clients. Overall, M-controller is the more responsive than others.

5.4.4 Impact on Performance of Application

We investigated the impact of these controllers on the performance of running application from two perspectives. One is the measured performance directly. The other is the system power consumption.

Take the scenario of 900 clients for example. We varied the values of power cap from 215W to 225W in a step length of 2.5W. Fig. 5.10 plots the impact on application performance in terms of slowdown. Slowdown is defined as a percentage of the average response time increase compared with that at in a full-speed run. When the power

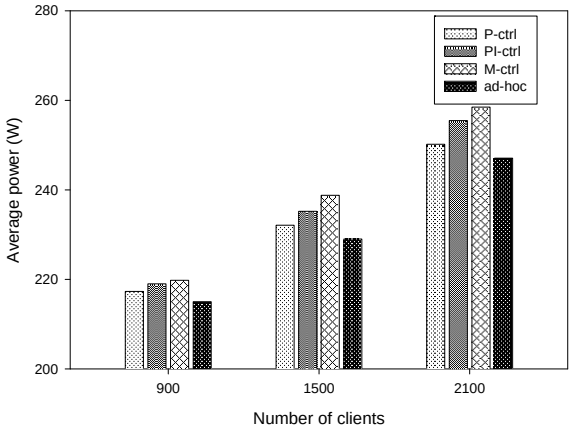


Figure 5.11: Average power.

budget constraint is stringent, the performance degradations for all controllers are remarkable, which are all over 14%. M-controller outperforms others in terms of slowdown by 5% when the power budget is 215W. The performance degradation for all controllers decreases with power budgets. Amongst all controllers, M-controller leads to the least performance degradation. When the power budget constraint is loose, the gap among the slowdown of P, PI and M-controller is marginal. Ad-hoc controller always has the largest performance loss. Similar results can be observed for the scenarios of 1500 and 2100 clients.

Fig. 5.11 shows the average power consumption for all controllers. A system with M-controller always has the largest average power.The results validate that more power lead to better performance. M-controller maximizes the power usage so it can lead to the least performance degradation. Ad-hoc controller has the smallest average power which makes it have worst performance. P and PI controllers have close average power which explains why the performance degradation of these two controllers are very close.

Chapter 6

Automated Coordination of Power and Performance in Virtualized Data centers

In Chapter 5, we developed a responsive power management controller to limit power consumption for an enterprise server. However, in server environments, one primary concern is the service-level agreement (SLA) in performance. Both power and performance are important issues in today's data centers.

With the proliferation of virtualization technology, a number of benefits have been brought to data centers, such as performance isolation and server consolidation. Meanwhile, as all co-residing virtual machines (VMs) share a hardware platform, the power management policies imposed on hardware may affect the performance of all hosted VMs. It becomes even more complicated with the consideration of time-varying workloads.

In this chapter, we presented vPnP, a feedback control based coordination system, to provide guarantees on SLA with respect to both power and performance in vir-

tualized environments. This framework provides a feasible and effective solution to achieving different levels of tradeoff between power and performance in data centers.

6.1 System Architecture

In this section, we first discuss the mechanism to regulate the performance and power of a server. Then we present the design of our vPnP system with detailed description of the key components.

6.1.1 Control Power and Performance with VCPU Caps

Multiple classes of execution states are supported in today's server processors, which can be used for the purpose of power management. These states include the frequency and voltage (P-state) in active mode, sleeping states (C-states) in idle time, and throttle state (T-state). P-states are well documented and can significantly impact active power consumption. But it has only very limited speed stages. With the multi-core technology, it is not flexible to manipulate due to the dependencies of the cores residing in the same die. Things are getting worse when it comes to virtual environments. Since multiple virtual machines may share a single core, tuning P-states of a core would threaten desired performance isolation properties. C-states can be utilized when the CPU is idle. But it leads to relatively large switch overhead and might not be effective when the system is not idle but in a low utilization. T-states are not always well documented and may need to modify clock modulation register. In this work, we regulate the power consumption by re-allocating CPU resources to virtual machines (VMs).

We assume a hypervisor scheduler to limit processing time to guest VMs. More specifically, non-work-conserving scheduling is employed so as to allow hardware to

enter low-power idle state. This capability is enabled in Xen by capping the CPU time of VCPUs allocated to a VM. By capping VCPUs, the utilization of underlying physical processors can be constrained so as to regulate power consumption. In this work, we use VCPU capping as the actuator for power management.

6.1.2 Design of vPnP

Figure 6.1 shows the architecture of our power and performance coordination system, named vPnP. A physical server can host multiple VMs on which different applications are running. A power monitor is employed to measure the power consumption for the whole physical server. The real-time performance of each VM can be collected by a separate performance monitor. A number of performance data, such as mean response time, CPU utilization and throughput, can be collected by this performance monitor. In our system, we currently only measure performance in terms of throughput. All the power and performance data are reported to the resource coordinator synchronously. The power and performance predictors can predict the future power consumption and performance, respectively. A multi-criteria utility function is defined to coordinate the power and performance requirement. The ultimate goal is to find a utility-optimizing policy which will be conveyed to the VM hypervisor to regulate the VCPU cap for each VM.

Power and Performance Predictor

Since the system behaviors could be non-linear, time-varying and workload-dependent, it is not accurate to represent the relationship between power/performance and VCPU cap by a static linear function. One feasible approach to capture the system behavior is to use a linear model to approximate locally on the neighborhoods (operating

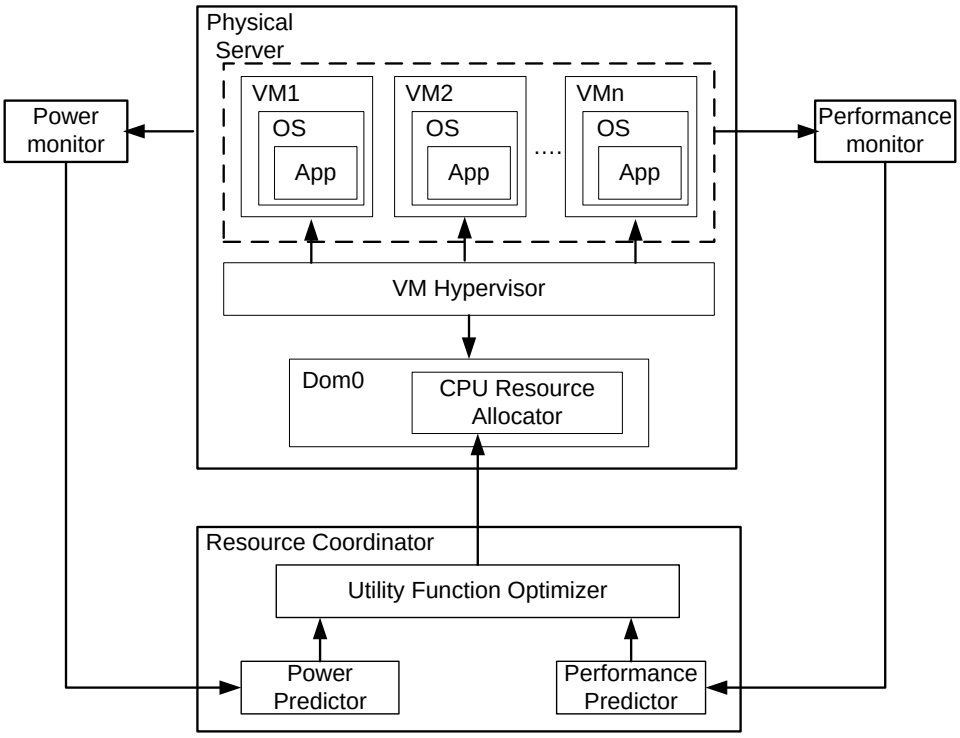


Figure 6.1: System architecture of vPnP.

range) of a point (operating point). In order to let this linear model adapt to different operating ranges, real-time update of this model is required.

Autoregressive-moving-average (ARMA) models are mathematical models for persistence, or autocorrelation, in a time series. In [85], an ARMA model is used to approximate the quantitative relationship between allocated resource and the normalized performance. In our work, we extend this model to predict power consumption of a physical server. A self-adaptive ARMA(2,2) model is defined to represent the relationship between the power consumption of the physical server and the VCPU caps of all hosted VMs in a control interval.

Let k denote the k^{th} control interval. $p(k)$ is the average power consumption of the physical server, $C(k)$ is the vector of VCPU caps of all hosted VMs, $g_i(k)$ and vector $H_i^T(k)$ capture the correlation between power and VCPU caps. It follows that:

$$p(k) = \sum_{i=1}^{2} g_i(k)p(k-i) + \sum_{i=0}^{1} H_i^T(k)C(k-i), \tag{6.1}$$

Let $r(k)$ and $c(k)$ represent the throughput and VCPU cap of a VM, respectively, $a_i(k)$ and $b_i(k)$ be time-varying model parameters to capture the relationship between throughput and VCPU cap. A self-adaptive ARMA(2,2) model which is similar to that in [85] represents the relationship between throughput and VCPU cap for a VM:

$$r(k) = \sum_{i=1}^{2} a_i(k)r(k-i) + \sum_{i=0}^{1} b_i(k)c(k-i), \tag{6.2}$$

Notice the model parameters $g_i(k)$, $H_i^T(k)$ in (6.1), and $a_i(k)$, $b_i(k)$ in (6.2) vary in different interval. These parameters will be updated every control period using

the data collected in a given range of past intervals, say M intervals, similar to the sliding window size. These data include the measured performance, power, and CPU resource allocated to each VM in the past M intervals. Least-square regression can be used to obtain these model parameters. The assumption to use ARMA model is that significant workload disturbance, which may lead to tremendous change in the model parameters estimation, seldom occurs. In this case, the convergence of the model parameters can be achieved. However, the dynamics of workloads retard this process. If the workload is changed, the model parameters may not be estimated accurately during a period. It may take as long as M intervals to get the accurate model parameters estimation for the system with new workload since all the data collected for the past M intervals can be replaced. It implies a smaller M is apt to adapt to the change in the system responsively but it can easily be affected by the infrequent disturbance.

Utility Function Optimization

Coordinating power and performance for resource allocation can avoid the ineffective management caused by the autonomous actions caused by isolated design.

Let $\hat{r_i}$ denote the SLA of performance for the i^{th} VM. Let p_s denote the power budget for the physical server. Two step functions are defined to quantify the SLA of performance and the power consumption:

$$\Theta_i(r_i(k)) = \begin{cases} 1 & r_i(k) \geq \hat{r_i}; \\ r_i(k)/\hat{r_i} & \text{otherwise}; \end{cases} \tag{6.3}$$

$$\Gamma(p(k)) = \begin{cases} 1 & p(k) \leq p_s; \\ p(k)/p_s & \text{otherwise}, \end{cases} \tag{6.4}$$

In our work, a utility function should be defined in such a manner that optimizing the utility function is to determine the resource allocation to each VM to meet performance SLAs and power budget. One example of such utility functions is:

$$U_1 = \alpha \sum_{i=1}^{n} (1 - \Theta_i(r_i(k)))^2 + (1 - \alpha)(\Gamma(p(k)) - 1)^2, \tag{6.5}$$

where the parameter α represents the weight of SLAs of power and performance for different tradeoffs. The goal is to minimize the utility function (6.5) by finding a column vector $C(k)$ representing the VCPU caps for all hosted VMs. Intuitively, this utility function can achieve its optimal value, 0, by meeting both performance SLA and power budget. In case this optimal cannot be achieved, there is no solution to guarantee both power and performance. The solution to minimize this utility function represents the tradeoff of the power and performance.

Another example could be:

$$U_2 = \alpha \sum_{i=1}^{n} (\Theta_i(r_i(k)))^2 - (1 - \alpha)(\Gamma(p(k)))^2, \tag{6.6}$$

Using the function (6.6), the goal is to maxmize the value U_2. Besides similar observation to (6.5), we can see these two utility functions achieve the optimization goal in different manners in case the optimal cannot be achieved and the power factor is dominant. (6.5) would be minimized by balancing the performance SLA for each

VM in this case while (6.6) would unbalance the performance SLAs. This can easily be proved by using Cauchy-Schwarz inequality.

A utility function optimizer is designed to determine the VCPU caps to optimize the defined utility function. Take (6.5) for example, since $r_1(k), r_2(k), \ldots, r_n(k)$ and $p(k)$ can all be represented by linear functions using $c_1(k), c_2(k), \ldots, c_n(k)$, the utility function (6.5) can ultimately be represented by a quadratic function $Q(c_1(k), c_2(k), \ldots, c_n(k))$. Thus the VCPU caps can be found by solving the following problem:

$$\text{minimize } Q(c_1(k), c_2(k), \ldots, c_n k) \tag{6.7}$$

$$\text{subject to } c_{low} \leq c_i(k) \leq c_{up}, i \in \{1, 2, \ldots, n\}, \tag{6.8}$$

Constraint (6.8) ensures the allocated VCPU caps will not be out of the range. In practice, we set the c_{up} to 100% and c_{low} to be 10% to avoid starvation. The objective function is quadratic and convex. The ellipsoid method can solve this problem in polynomial time. In practice, we can use an off-the-shelf quadratic programming solver to calculate the solution. Similar approach can be applied to the utility function (6.6). In this case, the objective function is quadratic but not convex. Thus this optimization problem is NP-hard.

Notice here we treat all VMs with same priority. In order to differentiate the performance priority, we can add weights to the performance gained in each VM.

6.2 System Implementation

In this section, we present the implementation details of each components in vPnP as follows:

Power monitor: The power consumption of a physical server is measured by WattsUp Pro [3] power meter. This power meter has an accuracy of ±1.5% of the measured RMS power with sampling rate of 1Hz. The power measurement data are sent to the resource coordinator in a real-time manner.

Performance monitor: A small daemon program is running at each application to record the time stamps of incoming requests. When the request is finished, another time stamp can be obtained. The difference of time stamps is response time. The number of requests finished during a unit interval is throughput. In our work, we only measure performance in terms of throughput. The throughput will be reported to the resource coordinator every control interval which is 30 seconds.

Resource coordinator: The resource coordinator consists of three parts: performance predictor, power predictor and utility function optimizer. The performance and power predictors update the control parameters at the end of every control period using least-square regression. The data in the past 20 intervals will be used in this regression. The overhead for both regression will be around 10ms. The utility function optimizer employs an off-the-shelf quadratic programming solver to calculate the VCPU caps for next interval. The order of magnitude of this calculation overhead is 10ms. The solution will be sent to the CPU resource allocator.

CPU resource allocator: Credit Scheduler [2], Xen's proportional share scheduler, is used to allocate CPU resource. More specifically, we use the VCPU cap to limit the CPU time that can be allocate to a VM. Though in Credit Scheduler, the weight can also specify the CPU resource allocation to each VM, we here only use VCPU cap to implement this allocation by setting same weights to all VMs. Notice a VCPU cap of 0 means there is no upper cap.

6.3 Evaluation

In this section, we present the experimental results for vPnP. We compare vPnP with the existing two-layer feedback controller, Co-Con [112], from the perspective of flexibility and robustness of coordinated control of power and performance.

6.3.1 Experimental Methodology

vPnP is evaluated in an experimental testbed consisting of five physical machines connected through a gigabit networks. Two physical servers are used to host VMs. A third storage server is used for Network File System (NFS) to host all VM images so that live migration might be integrated to our system in the future. Each server has two quadcore processors, Intel Xeon5450, for a total of eight cores. In addition, each server is equipped with 8GB RAM. CentOS Linux 5.0 with kernel 2.6.18 is running on all servers. The virtualization is enabled by Xen 3.4.

We selected TPC-W [80] as the host application. TPC-W is an E-Commerce benchmark that models after an online book store. It provides workloads with different mixes. There are three predefined workload mixes: browsing (B), shopping (S) and ordering (O), indicating different resource requirements. TPC-W consists of two tiers, i.e., the front application (APP) tier and the back database (DB) tier. Each tier is hosted in a VM with 2 VCPUs and 2GB memory. Two TPC-W applications are running in our testbed. We consolidate two DB VMs on a `DB_HOST` and two APP VMs on an `APPS_HOST`. We only evaluated vPnP on the `DB_HOST` since the DB tiers are bottlenecks of this application.

In our testbed, the resource coordinator resides in dom0 of `APPS_HOST` in order to alleviate the computational stress at `DB_HOST` with the consideration of consolidation as well. The CPU resource allocator resides in dom0 of `DB_HOST`.

The client-side workload generators are located on another two servers. The default concurrency level is 400, 400, and 1000, for browsing, shopping, and ordering, respectively. The default throughput targets for these workloads with default concurrency levels are 1200, 1500, and 3000 per interval, respectively. The clients machines are guaranteed to be powerful enough to create resource stress on any of the VMs.

We conducted three experiments on the platform.

First, we evaluated the correctness of our self-adaptive power and performance predictors. Second, we conducted experiments to show the agility of vPnP to achieve different tradeoffs between performance and power. In case performance and power cannot be both guaranteed simultaneously, it is desirable to make a balance between them. This tradeoff can be power-preferred, performance-preferred or other user-defined criterion. Third, we investigated the robustness of vPnP. vPnP is designed in a workload-independent manner. It can perform well for different applications or in the scenarios with dynamics.

We adapted an existing two-layer controller from Co-Con [112] as the baseline. The controllers were designed using the off-line data during a run of TPC-W with browsing mix following the methods in [112].

Each run of the workload lasted 200 intervals by default. Since vPnP used the data of the past 20 intervals for prediction, we started it at the 25^{th} interval. The results from the 30^{th} interval to the 100^{th} interval will be presented and the results thereafter will be omitted due to the similarity.

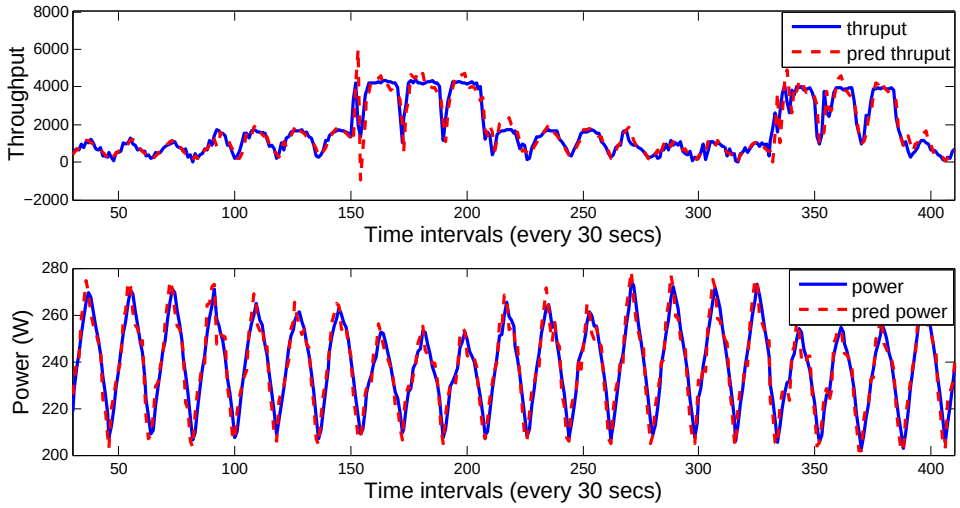


Figure 6.2: Performance and power prediction.

6.3.2 Experimental Results

Predictor Accuracy

The online performance and power predictors can adapt the model parameters with the dynamics in the system. The accuracy of these two predictors affects the CPU resource allocation.

We conducted experiments to show the adaptivity of these two predictors by running a sequence of different workloads. We allocated the VCPU cap to the VM from 100% to 10% in a step length of 10% every interval back and forth. Starting from the 90^{th} interval, we change the workload type every 60 intervals (30 minutes) following this order: browsing→ shopping→ ordering→ shopping→ browsing→ ordering→ browsing. Figure 6.2 plots the results from the 30^{th} interval to the 410^{th} interval when all workload changes finish.

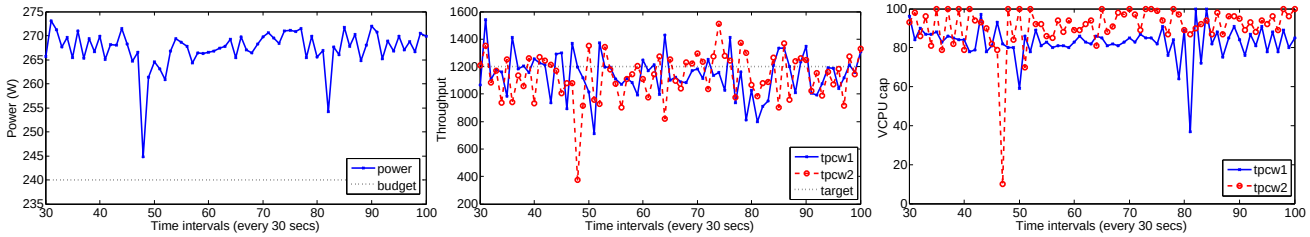
To quantify the prediction accuracy of these predictors, we used two metrics, mean absolute error, defined as $\sum_{k=1}^{n} |y(k) - \tilde{y}|/n$, and error deviation, defined as $\sqrt{\sum_{k=1}^{n}((y(k) - \tilde{y})/\tilde{y})^2/n}$, where $y(k)$ is the predicted power/throughput at k^{th} in-

Table 6.1: Prediction accuracy.

prediction	metrics	no change	B→ S	S→ O	O→ S	S→ B	B→ O	O→ B
performance	mean	0.1830	0.2135	0.5252	0.6232	0.2832	0.4526	0.5060
	deviation	0.2882	0.2704	0.8592	1.1809	0.3856	0.6433	0.8370
power	mean	0.0192	0.0230	0.0144	0.0213	0.0170	0.0209	0.0188
	deviation	0.0251	0.0309	0.0182	0.0251	0.0216	0.0316	0.0248

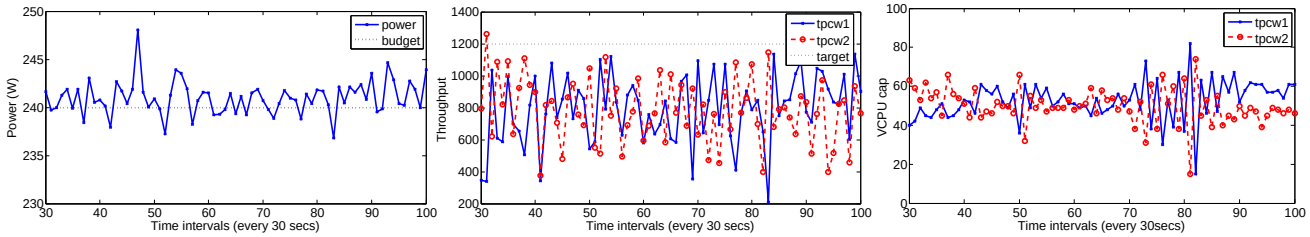
terval and $\tilde{y}$ is the measured power/throughput. We compared the metrics when the predictors have already adapted to the workload with those when the predictors are adapting to the new workload. To adapt to a new workload may take up to M intervals (M is the sliding window size). So we consider these M intervals when workload changes as the adapting phase. The results are shown in Table 6.1.

The power predictor can predict the power consumption at all the time accurately, with mean absolute error below 3% and deviation below 0.04. The mean absolute error for performance prediction may reach around 20% even if there is no workload change. This is because the power consumption only depends on current resource utilization and resource allocation. But the performance in terms of throughput will be affected by the process delay [115] and workload dynamics as well in addition to the control over resource allocation. Even if the resource allocation remains stable, the observed performance cannot be as stable as the power consumption due to the workload dynamics. During the adapting process, the accuracy of performance prediction is impaired. If the workload change is not significant (for example, B→S), the performance predictor can adapt gracefully to workload change. However, the performance predictor may take up to the whole adapting phase to adapt to the new workload in case of significant workload change (for example, S→O).



(a) Power consumption (b) Throughput (c) VCPU cap

Figure 6.3: Results of utility function U_1 with performance-preferred policy ($\alpha = 0.9$).



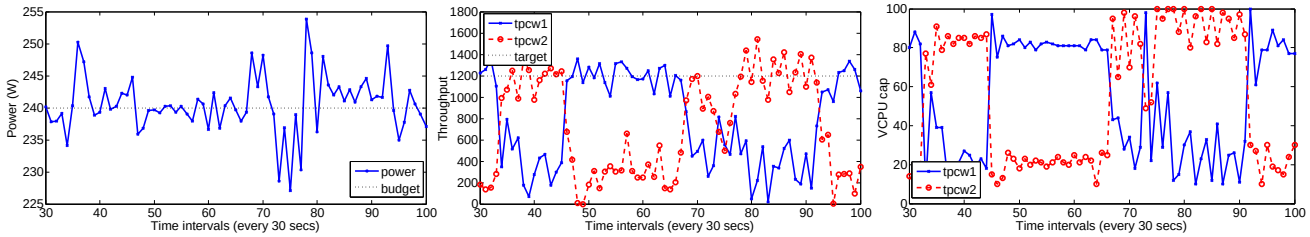
(a) Power consumption (b) Throughput (c) VCPU cap

Figure 6.4: Results of utility function U_1 with power-preferred policy ($\alpha = 0.1$).

Tradeoff of Performance and Power

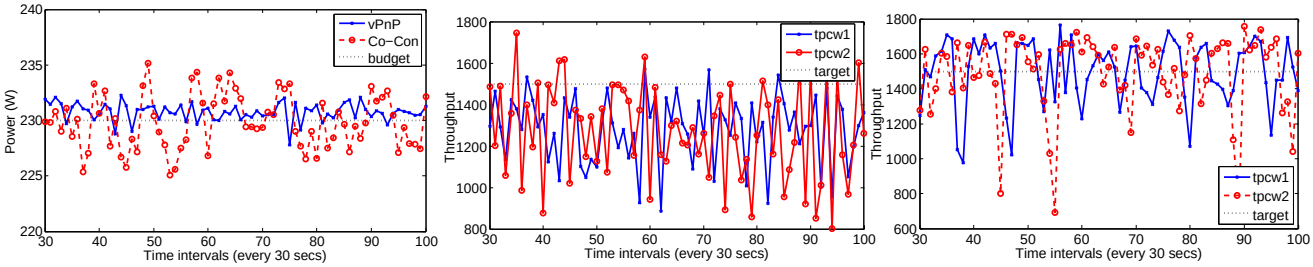
Using vPnP, different tradeoffs between power and performance can be achieved in an agile way by tuning the weight in the utility function. Using the two utility functions defined in Section 6.1.2, in case both of power and performance SLAs cannot be met, the policy depends on the weights in the utility functions. A larger α represents the tendency to meet the performance SLA while a smaller α means satisfying power budget is more important. We defined two policies here for the purpose of evaluation: performance-preferred ($\alpha = 0.9$) and power-preferred ($\alpha = 0.1$). We conduct experiments using TPC-W browsing workload.

First we applied our vPnP framework using the performance-preferred policy. Since power is not dominant, vPnP will allocate large VCPU caps to meet performance SLA while power consumption is beyond budget. The oscillations of throughput are caused by both the disturbance of workload and the adjustment to the VCPU



(a) Power consumption (b) Throughput (c) VCPU cap

Figure 6.5: Results of utility function U_2 with power-preferred policy ($\alpha = 0.1$).



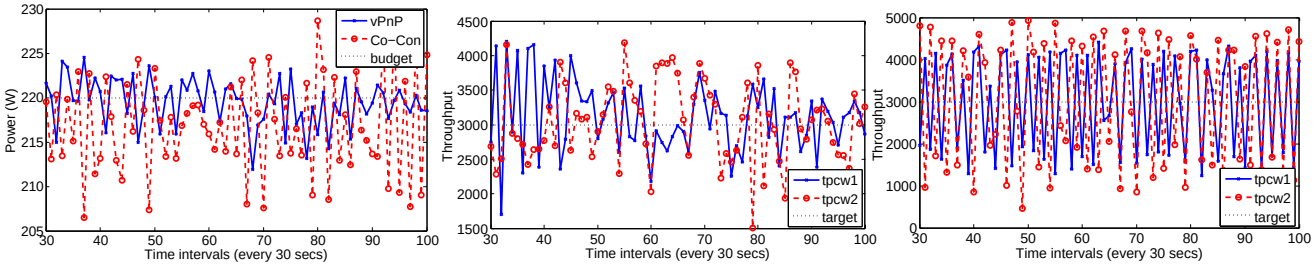
(a) Power consumption (b) Throughput under vPnP (c) Throughput under Co-Con

Figure 6.6: Comparison between vPnP and Co-Con running TCPW shopping workload.

caps. The utility function U_2 performs in a similar way. For lack of space, we omit this result.

Then the power-preferred policy is employed. Results are shown in Figure 6.5 and Figure 6.4. Both applications cannot meet the performance SLA mostly but power consumption is close to the budget. These two utility functions act in different ways. The utility function U_2 would allocate enough VCPU cap to one VM rather than to allocate CPU resource in a relatively fair manner as the utility function U_1 does. For the sake of fairness, in the rest of experiments, we only use utility function U_1.

In contrast, the power control is always primary in Co-Con. It cannot make tradeoff between power and performance.



(a) Power consumption (b) Throughput under vPnP (c) Throughput under Co-Con

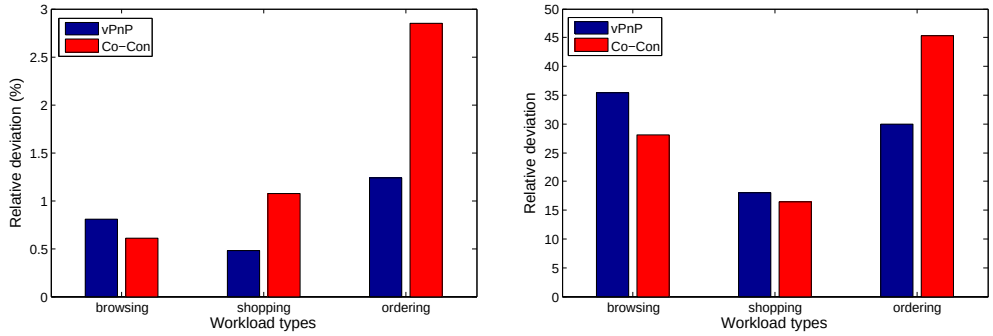
Figure 6.7: Comparison between vPnP and CoCon running TPC-W ordering workload

System Robustness

We investigated the robustness of vPnP by conducting experiments running different TPC-W workloads: browsing, shopping and ordering. 2 identical applications will be hosted for each run. Co-Con is evaluated as a baseline. To make this comparison fair, we use a strict power-preferred policy for vPnP by setting $\alpha = 0.01$.

The two-layer controller in Co-Con is designed statically based on the off-line collected during a run of TPC-W browsing mix. When running shopping and ordering workloads, we find vPnP can limit the power consumption closer to the budget than Co-Con. The results are shown in Figure 6.6 and Figure 6.7. The static two-layer controller in Co-Con cannot adapt its control parameters to the workload change thus it may not perform well when the workload changes. As vPnP doesn't rely on any off-line trained model, it can adapt to a large variety of workloads. Since workload disturbance and controller both affect the power and performance, the oscillations occur and it is hard to evaluate directly from the figures.

Thus, to quantify the performance of Co-Con and vPnP, we define relative deviation, which is based on root-mean square error, as the metric. The relative deviation for power is $\sqrt{\sum_{k=1}^{n}(p(k)-p_s)^2/n}/p_s$. Similarly, the relative deviation for throughput of one application is $\sqrt{\sum_{k=1}^{n}(r(k)-r_s)^2/n}/r_s$.



(a) Power deviation (b) Performance deviation

Figure 6.8: Performance of vPnP and Co-Con.

The quantified results are shown in Figure 6.8. Both vPnP and Co-Con can achieve very small relative deviation for power for all workloads. It implies they can guarantee the power when the power budget is defined within a specific range (we will see what will happen if power budget varies in a large range). For performance, we can find Co-Con outperforms vPnP by around 5% when running browsing workload. The difference is marginal when running shopping workload. Using ordering workload, we can see the performance relative deviation of vPnP is more than 15% less than that of Co-Con. Overall, the performance relative deviation of vPnP can be limited to 35% with relatively small variation (less than 15%). To the contrary, the performance relative deviation of Co-Con may vary over a large range, from 15% to 45%. The results show the adaptivity of vPnP for a variety of workloads.

In practice, the power budget might change due to thermal condition or temporary reductions in cooling or power delivery capacity. We investigate how vPnP and Co-Con reacts to this power budget change. Two TPC-W applications are running browsing workload in our testbed with initial power budget 240W for `DB_HOST`. From the 100^{th} interval, the power budget changes every 50 intervals following this order: 240W→230W→240W→250W.

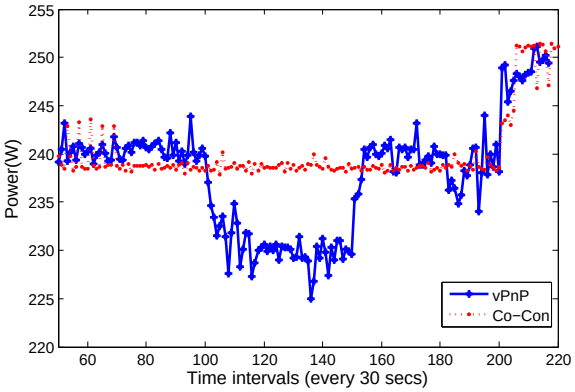


Figure 6.9: Reaction to power budget change.

As shown in Figure 6.9, when power budget decreases to 230W, Co-Con cannot limit the power consumption to this budget. It is due to the limited stages of CPU frequency. Only 4 CPU frequencies are available for Intel Xeon5450 in our testbed. It limits the range of power control. When the power budget is 240W, the frequency has already been set to the lowest. If the budget decreases more, Co-Con cannot be effective to control power since there is no lower frequency can be chosen. In contrast, vPnP can work in a large range of power budget since it use VCPU cap to regulate power, which provides a large difference in power consumption between the highest and the lowest cap.

To sum up, vPnP is more agile than Co-Con since it can achieve different tradeoffs between power and performance while Co-Con always sticks to the power budget. vPnP also outperforms Co-Con in terms of robustness over a variety of workloads.

Chapter 7

Statistical QoS Guarantee on Processing Delay in BBUs

In this chapter, we consider the energy efficiency of BBUs supporting soft real-time applications with guaranteed end-to-end delay. Similar to enterprise web servers, BBUs are often significantly over-provisioned in order to meet target delay constraints even under peak loads. However, this design principle suffers poor overall energy efficiency as BBUs are typically underutilized. When the scale of BBU comes large, the loss could be huge. Thus, it is desirable for solutions that reduce system's power consumption without significant effect on performance guarantees.

We consider the scenario of BBUs running three kinds of workloads: VoIP, Web and video streaming. Voice application and video streaming can be categorized as periodic tasks with arbitrary release. Web service application can be regarded as aperiodic tasks. Soft real-time constraints should be met. That is we need to maintain user-defined QoS quantile to a set point. The techniques by scheduling on individual workload is not feasible as only single BBU task processes user workload. Thus we prefer control-theoretical approach to achieve statistical QoS guarantee in BBUs while

minimizing power consumption. This approach should require no *a priori* knowledge of workload. In this chapter, we present a power management strategy for controlling delay and minimizing power consumption using DVFS for BBUs.

7.1 Policy Design

Take the streaming application for example. One of the most concerned performance metrics is delay. This delay includes processing delay, queueing delay, transmission delay and propagation delay. We assume the stream sender is very powerful, such as a blade server. We employ BBUs to receive the packets of streaming. As the delay on the sender end and propagation delay can be small enough, we only consider the delay on the receiver side. More specifically, the delay we consider here at receiver end include queueing and processing delay, which involves demodulation and decode.

There are a few work studying statistical QoS guarantee for power management schemes [118, 56, 71]. However, these work either rely on *a priori* knowledge on workloads [56, 118] or is validated by simulation [118, 71] or is not self-adaptive [71]. To deal with heterogeneous workloads, such a scheme should be adaptive.

We propose a power management strategy for controlling delay and minimizing power consumption using DVFS for BBUs. We use the Robbins-Monro (RM) stochastic approximation method to estimate delay quantile. We couple a fuzzy controller with the RM algorithm to obtain the CPU frequency for the receiver side BBU that will maintain performance within the specified QoS. We use the normalized delay, which is the ratio of the measured delay to the reference delay, in this design.

7.1.1 Robbins-Monro Method

The Robbins-Monro method estimates the quantile of an unknown distribution [96]. We assume $M(x)$ is the expected value at x of the system delay, where M is a monotone function of x. For each x there corresponds a random variable $Y = Y(x)$ with a distribution function $Pr[Y(x) \leq y] = H(y|x)$, such that

$$M(x) = \int_{-\infty}^{\infty} y \, dH(y|x). \tag{7.1}$$

Let F(x) be an unknown distribution function, and

$$F(\Theta) = \alpha (0 < \alpha < 1), F'(\Theta) > 0, \tag{7.2}$$

or equivalently,

$$Pr[F \leq \Theta] = \alpha, \tag{7.3}$$

where α is the desired delay quantile.

Adapted from [17], we let $z(t)$ be the independent random variable and the outcome of the experiment, the delay, in our case, with distribution function $Pr[z(t) \leq x] = F(x)$, and let $y(t)$ be defined as:

$$y(t) = \begin{cases} 1 & z(t) \leq x(t); \\ 0 & \text{otherwise.} \end{cases} \tag{7.4}$$

Notice we only simply use a binary metric $\{0, 1\}$ to indicate whether the delay has met the requirement or not. The reason to employ this binary metric is that we only concern whether the delay requirement has been satisfied for single request or

the requests during an interval instead of indicating how close the delay is to meet the requirement. To further investigate how close the delay is, we need to study the statistics or quantile of the delays. This is the major motivation for us to design this controller with statistic QoS guarantee while minimizing power consumption. In addition, the definition of $y(t)$ depends on the performance metric. In the scenario of a streaming server, the performance metric can be frames per second (fps) thus a different definition of $y(t)$ should be given, which will be shown in the evaluation section.

Let $x(0)$ be an initial guess of Θ, and let

$$x(t+1) = x(t) + a(t) \cdot (\alpha - y(t)), \tag{7.5}$$

where $a(t)$ should satisfy that $0 < \sum_1^\infty a_t^2 = A < \infty$, and $\sum_2^\infty \frac{a_t}{a_1 + \cdots + a_{t-1}} = \infty$. It can then be proven that $M(x) = F(x)$ and that $\lim_{t \to \infty} x(t) = \Theta$.

The sequence $\{x\}$ can be proven to converge to Θ as the solution to $Pr[X \leq \Theta] = \alpha$. The parameter $a(t)$ can be set two ways: (a) decreasing as t goes to infinity, with some restrictions to guarantee convergence, as true to the original form of the Robbins-Monro method, or (b) to a small fixed value ϵ. We use the latter in order to assure that the system can adapt to changing distributions arising from time-varying workloads.

Notice the above formulation is based on the assumption that only a single request is submitted at time t, such as batch jobs. However, this assumption may not always hold. For example, multi-threaded web applications make it impossible to split each request without overlap. In the context of packet processing, as there would be hundreds and thousands of packets processes per second, it is too costly to adjust CPU frequency upon the completion of each packet. Instead, we prefer the delay of

finishing a certain amount of packet processing, say every 1000 packets.

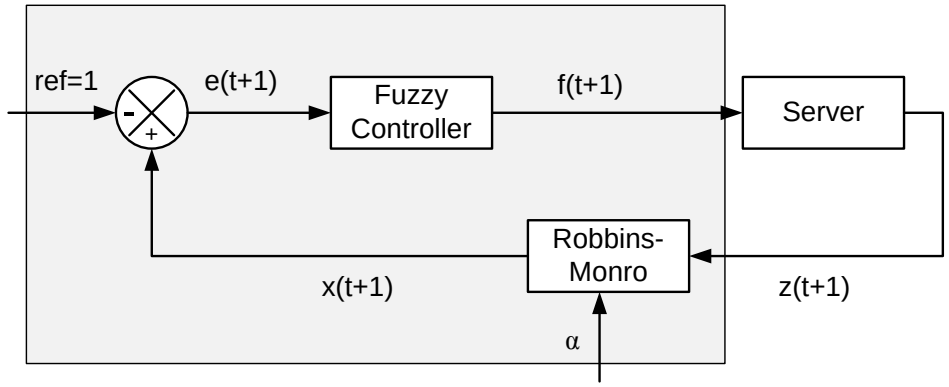


Figure 7.1: Power management controller design.

7.1.2 Fuzzy Controller

We apply the updated delay estimate $x(t+1)$ obtained from the Robbins-Monro algorithm as the new set point to regulate the processor frequency, f, via a single-input single-output (SISO) fuzzy controller, as shown in Fig. 7.1. For a value of $\alpha = 0.95$ to indicate a target performance of having 95% of requests meet QoS deadline, it follows that, on average, $y(t)$ will also be equal to 0.95 (for 95% of responses, $y = 1$, and for 5% of responses, $y = 0$).

Figure 7.2 illustrates the structure of the Self-Tuning Fuzzy Controller (STFC). It consists of three components, namely the fuzzy logic controller, the scaling-factor controller and the output amplifier. The frequency in control interval $k+1$, denoted by $f(k+1)$, is adjusted according to its error $e(k)$ (i.e., the normalized difference between the reference value and the achieved one) and change of error $\Delta e(k)$ in previous control interval k using a set of control rules embedded in the fuzzy logic controller.

Based on these, the controller calculates frequency adjustment $\Delta f(k)$ for next

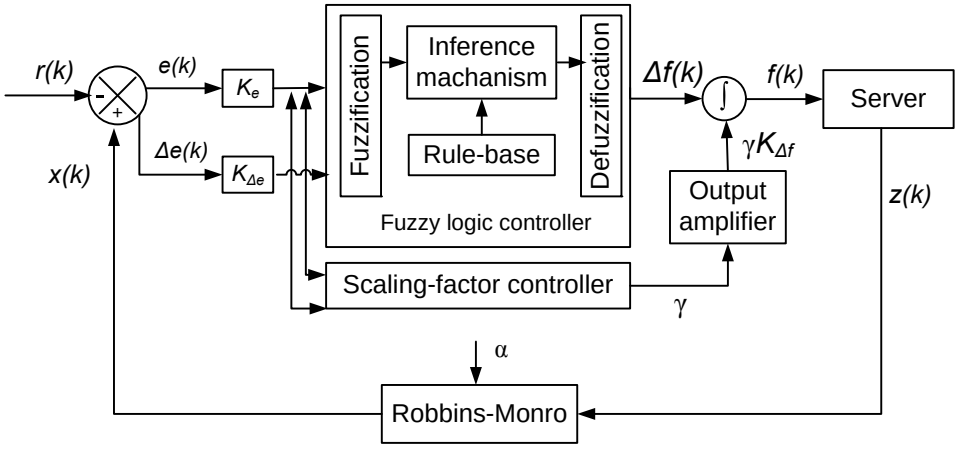


Figure 7.2: Fuzzy controller design.

control interval. The calculated resource adjustment is then fed into the next layer gain scheduler.

The fuzzy logic controller contains four building blocks. The actual fuzzy logic is implemented as a set of *If-Then* rules about quantified control knowledge about how to adjust the frequency according to $e(k)$ and $\Delta e(k)$. The fuzzification interface converts controller inputs into certainties in numeric values of the input membership functions. The inference mechanism activates the rule-base and applies fuzzy rules according to the fuzzified inputs and generates the fuzzy conclusions for the defuzzification interface. The defuzzification interface converts fuzzy conclusions into the change of frequency in numeric value.

The STFC is built on the static fuzzy logic controller by adding the self-tuning scaling factors and the output amplifier. There are three scaling factors: input factors K_e and $K_{\Delta e}$, output factor γ and output amplifier $K_{\Delta f}$. The change of input scaling factors changes the connection of input values to suitable rules, The change of output scaling factor and the amplifier together adjust the amplitude of the control input. The actual inputs of the fuzzy logic controller are $|K_e|e(k)$ and $|K_{\Delta e}|\Delta e(k)$. Thus,

the adjustment of frequency in control interval $k+1$ is

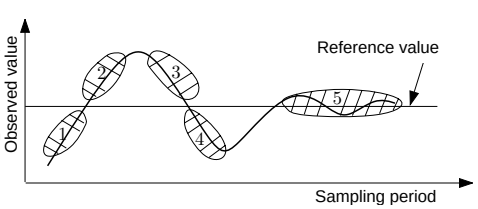
$$f(k+1) = f(k) + \gamma |K_{\Delta f}| \Delta f(k) = \int \gamma K_{\Delta f} \Delta f(k) dk.$$

The design objective is to translate human expert's knowledge into a set of control rules to control the frequency without a system model. In the fuzzy logic controller, the control rules are defined using linguistic variables. For brevity, linguistic variables "$e(k)$", "$\Delta e(k)$", and "$\Delta f(k)$" are used to describe $e(k)$, $\Delta e(k)$, and $\Delta f(k)$, respectively. The linguistic variables assume linguistic values NL(negative large), NM (negative medium), NS (negative small), ZE (zero), PS (positive small), PM (positive medium), and PL (positive large).

Figure 7.3(a) gives an simple illustration of typical control effect. In this figure, we identify five zones with different characteristics. Zone 1 and 3 are characterized with opposite signs of $e(k)$ and $\Delta e(k)$, where the error is self-correcting and the achieved value is moving toward the reference value. Thus, $\Delta f(k)$ needs to be set either to speed up or to slow down current trend. Zone 2 and 4 are characterized with the same signs of $e(k)$ and $\Delta e(k)$, where the error is not self-correcting and the achieved value is moving away from the reference value. Therefore, $\Delta f(k)$ should be set to reverse current trend. Zone 5 is characterized with rather small magnitudes of $e(k)$ and $\Delta e(k)$. Therefore, the system is at a steady state and $\Delta f(k)$ should be set to maintain current state and correct small deviations from the reference value.The resulted control rules are summarized in Figure 7.3(b). For example, when "$e(k)$" and "$\Delta e(k)$" are NL and PS, "$\Delta f(k)$" is set to PM.

We adapted the same design for the membership function and inference mechanism from [115].

The fuzzy logic controller only defines the basic control rules according to the



(a) The control effect

"$\Delta u(k)$"	"$e(k)$"	"$\Delta e(k)$"						
		NL	NM	NS	ZE	PS	PM	PL
②	NL	PL	PL	PL	PL	PM	PS	ZE ③
	NM	PL	PL	PL	PM	PS	ZE	NS
	NS	PL	PL	PM	PS	ZE	NS	NM
	ZE	PL	PM	PS	ZE	NS	NM	NL ⑤
	PS	PM	PS	ZE	NS	NM	NL	NL
①	PM	PS	ZE	NS	NM	NL	NL	NL ④
	PL	ZE	NS	NM	NL	NL	NL	NL

(b) The rule table

Figure 7.3: Design of the fuzzy control rules.

inputs of $e(k)$ and $\Delta e(k)$. It outputs the sign and magnitude of the frequency adjustment $\Delta f(k)$. However, there could be a few fluctuations due to workload change or inaccuracy from the controller.

A self-tuning controller is designed to address this problem using adaptive output magnitude and flexible control rules. The self-tuning features consider dynamically changing the input, output scaling factors and the output amplifier. The output scaling factor γ and the output amplifier $K_{\Delta f(k)}$ together determine the magnitude of the allocation adjustment. The amplifier implements heuristic control knowledge as follows:

$$K_{\Delta f(k)} = |\frac{f}{2} \cdot e(k)|,$$

where f is the current frequency. The amplifier follows a heuristic rule that the maximum resource adjustment should not exceed half of current capacity for stability and should be proportional to the control error for adaptability. The direction of the adjustment is still determined by the fuzzy logic.

In real systems, there are only a few discrete levels of CPU frequencies. To mimic the continuous CPU frequency scaling, we adapt the method of "CPU dithering". For example, suppose we have 4 discrete CPU frequency levels, say 3, 6, 8, and 10 if we divide them by 100MHz. If we want to mimic the frequency of 7, we can let the CPU run half time at speed 6 and half time at speed 8. In our experiment, we

will round the frequency level to integer. And we set the minimal interval for scaling frequency is 1 second.

We notice this controller can provide user-defined QoS guarantee while minimizing power consumption. However, there is no guarantee on the power consumption. If we want to guarantee the power consumption, similar controller can be design with the power consumption as the reference and outcome.

7.2 Implementation

We implemented this statistical QoS controller on two test beds.

The first test bed on a BBU. More specifically, it is a BeagleBoard-xM embedded system with USRP. BeagleBoard-xM has a TI DM3730 MPSoC which includes an ARM Cortex-A8 General Purpose Processor (CPP) and C64x+ DSP. It has 512MB DDR RAM. The ARM processor on BeagleBoard-xM has four CPU frequencies: 300MHz, 600MHz, 800MHz and 1GMHz. Universal Software Radio Peripheral (USRP) is a universal platform for software radios. GNURadio [5] provides the signal processing runtime and processing blocks to implement software radios external RF hardware (USRP) and commodity processors (ARM). We connect a USRP with a multimedia server. At receiver end, there is a BeagleBoard-xM with USRP. With GNURadio installed, the receiver can receive stream from sender via wireless communication. The application we used in evaluation is streaming application. We consider the signal processing at receiver end. This work includes demodulation, deinterleaving, and decoding. The test bed is set up as Figure 7.4.

The second test bed is Dell PowerEdge1950 servers with 4 CPU frequencies. The available benchmarks and applications on our BBUs are limited, we evaluate the above power management policy on servers due to the similarity between the target

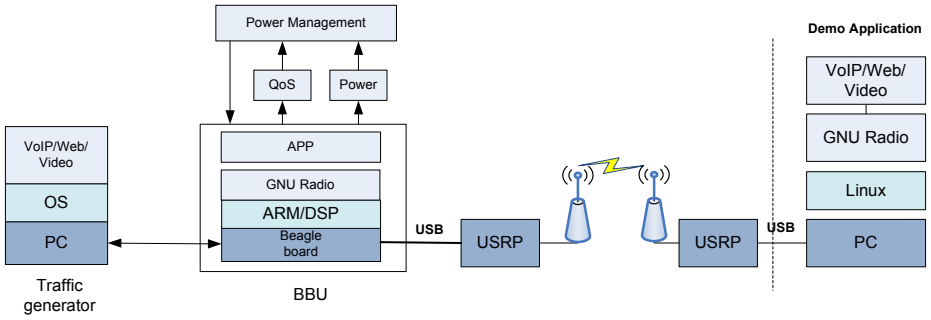
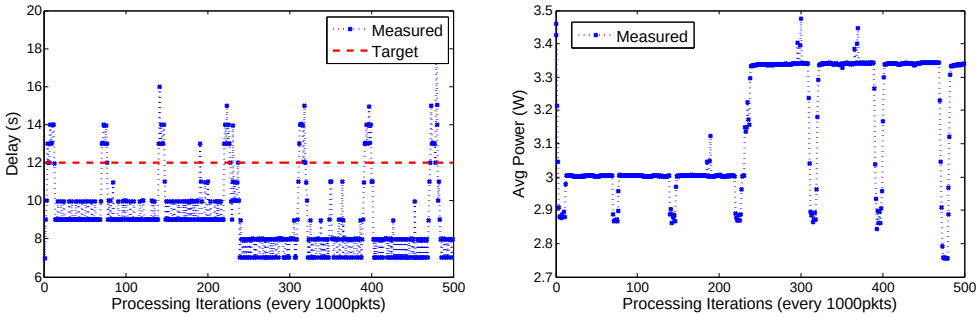


Figure 7.4: System Overview.



(a) Delay (b) Power consumption

Figure 7.5: Statistic QoS controller on streaming on BBU.

applications in BBUs and the workloads we selected on servers. One application is TPC-W, which is similar to data applications in BBUs. TPC-W is an E-commerce benchmark modeling after an online book store. It is a CPU-bound workload.

7.3 Experimental Results

First, we evaluate this statistic QoS controller on BBU running streaming benchmark. We set the rate to be 200kbps and the packet size to be 100 bytes. We assume the tolerable delay for BBU to process 1000 packets is 12 seconds. And the target QoS quantile is 90%.

Table 7.1: Comparison of three settings (streaming on BBU).

CPU setting	QoS satisfaction	Avg power consumption (W)
max freq	100%	3.436
min freq	0%	2.869
with controller	90.6%	3.158

Table 7.2: Comparison of three target delay with same QoS quantile (streaming on BBU).

Target delay (s)	QoS satisfaction	Avg power consumption (W)
10	91.4%	3.358
12	90.6%	3.158
15	89.8%	3.038

Figure 7.5 shows the results of applying this controller on streaming workload for 500 iterations. We can see the delay will be bounded under the target, 12 seconds, for most intervals. There are a few spikes of delay plots as CPU frequency scaled down. It also leads to a few dropoffs on power consumption.

We further summarized the above results in a quantitive manner. As comparison, we ran the same streaming application at maximal and minimal CPU frequency, respectively. Results are shown in Table 7.1. Using this statistic QoS, we can reduce the power consumption by around 10% while still meeting target statistic QoS. Considering the relatively large static power consumption of a server, which is usually 2.6W, we can achieve round 30% dynamic power saving with 10% QoS loss.

We also study the impact on power consumption when the target delay is different while QoS quantile is fixed. With looser delay constraint, more power can be saved. Results are shown in Table 7.2. Similar observation can be found when QoS quantile is changed. We omit the results here.

Then, we investigate the performance of this statistic QoS controller on TPC-W

Table 7.3: Comparison of three running scenarios (TPC-W on server).

CPU setting	QoS satisfaction	Avg power consumption (W)
max freq	100%	267.5
min freq	69.625%	237.1
with controller	90.375%	253.9

benchmark. We assume that the control target can always be achieved by a proper control solution. However, this target should be selected carefully as well according to the workload characteristics. We ran TPC-W with shopping workload mixes. The concurrency level was 300. We defined tolerable response time is 3000ms and target QoS quantile is 90%. The performance metric is reported every 3 seconds and control period is 3 seconds.

We ran this application over 800 intervals when the controller was applied. We summarized the above in a quantitive manner. As comparison, we ran TPC-W with same workload mixes and concurrency level at maximal and minimal CPU frequency, respectively. Results are shown in Table 7.3. Using this statistic QoS, we can reduce the power consumption by more than 5% while still meeting target statistic QoS. Considering the relatively large static power consumption of a server, which is usually more than 196W, we can achieve round 20% dynamic power saving with 10% QoS loss.

Chapter 8

Conclusions and Future Work

This dissertation aims to build power-efficient networked computing systems. In this chapter, we summarize the approaches presented in this dissertation and discuss the directions of future work.

8.1 Conclusions

Power is emerging to be key challenges in networked computing systems, from real-time embedded systems to enterprise server environments. A design challenge for embedded systems is power efficiency because most embedded systems are powered by battery with limited capacity. The concern of power expenditure rises as well in enterprise server environments. On one hand, the increasing power consumption leads to tremendous increase spending on cooling and delivery equipments. On the other hand, the electricity bill keeps increasing due to popularity of high-density enterprise servers. Furthermore, power consumption in data centers can also lead to tremendous environment pollutions.

The energy expenditure on networked real-time systems consists of two major

parts: the expenditure on the circuit board and the expenditure for wireless communication. In our work, we emphasize on seeking energy efficiency on wireless communication rather than on circuit board for networked real-time systems. We consider transmission energy optimization over an additive white Gaussian noise (AWGN) channel with delay constraint for each packet. We assume such a transmitter has renewable energy resources so that it is often impossible to deliver all the data generated during each period. Instead, the data with higher level of importance, represented by reward, should have high priority to deliver. We formulate this problem as a reward maximization problem. We propose the optimal solution to this problem in pseudo-polynomial time. And we provide a sub-optimal approach with polynomial time complexity.

Regulating on circuit board can affect system-wide power consumption significantly. We provide a general-purposed, practical and comprehensive power management middleware for networked computing systems to manage circuit board power consumption. It has the functionalities of power and performance monitoring, power management (PM) policy selection and control, as well as energy efficiency analysis. It is a general framework of power management middleware for different platforms, from servers to real-time embedded systems. A prototype is implemented and deployed on BBUs. To the best of our knowledge, this middleware is the first software to provide comprehensive and practical solution to power management in BBUs. Three representative PM policies are included. We present these three strategies in different contexts to exhibit the ability of adaptation.

In enterprise server environments, limiting system power consumption (power capping) is an effective solution to avoid system failure due to power capacity overload or overheating. The system-wide power consumption includes that from different subsystems, such as processors, memory, etc. Although there are a few power man-

agement mechanisms on the subsystems other than processors, we use DVFS as the main mechanism to adjust system-wide power consumption as other mechanisms may not always be feasible and processors consumes relatively large portion of power. To achieve a responsive control on system-wide power consumption, we present a model-predictive feedback controller to regulate processor frequency so that power budget can be satisfied without significant loss on performance.

The proliferation of virtualization technology brings a number of benefits to data centers, such as performance isolation and server consolidation. However, challenges arise as all co-residing virtual machines (VMs) share a hardware platform so that power management policies imposed on hardware may affect the performance of all hosted VMs. It becomes even more complicated with the consideration of time-varying workloads. In addition to power budget, service-level agreement (SLA) in performance one primary concern. We presented vPnP, a feedback control based coordination system, to provide guarantees on SLA with respect to both power and performance in virtualized environments. Rather than DVFS, we use VCPU capping in this scenario as power management mechanism to alleviate the interference among VMs.

Finally, considering the Base Band Units (BBUs) in practice, to improve energy efficiency in BBUs, we proposed a power management strategy for controlling delay and minimizing power consumption using DVFS. A Robbins-Monro (RM) stochastic approximation method is to estimate delay quantile. A fuzzy controller is coupled with the RM algorithm to obtain the CPU frequency for the receiver side BBU that will maintain performance within the specified QoS.

8.2 Future Work

There are several issues and challenges along the line of this dissertation.

First, we plan to extend our current work for power management of wireless transmitters. In our previous work, we studied packet transmission in an AWGN channel. In practice, the state of the wireless channel may vary from time to time. We will consider more realistic wireless fading channels and multi-user environments in our future work.

Second, the power model presented in Chapter 4 and Chapter 5 is learned offline. As this learning process requires a comprehensive data sets, it may take a long to obtain an accurate power model. An online solution can be provided by regression using insufficient samples while the model can keep refining as most samples are collected. In addition, we did not consider the power model in virtualized environments. How to estimate and validate the power consumption for each hosted VM remains a challenge.

Third, the coordination of power and performance in virtualized environments need to be studied at cluster-level. Currently, we did not take into account the knowledge of the entire cluster. The cluster-level power and performance coordination solution involves the application deployment as well as power allocation across the physical servers, and resource allocation of each co-residing VMs on a host. This study can be further explored when we enable power-on/off in cluster. When investigating the usage of the entire cluster, existing study shows that the average utilization could be below 20% [78]. The servers nowadays are not energy-proportional [106] thus a significant amount of power is still consumed at low levels of utilization. By consolidation, more servers can be idle then more power can be saved if these idle servers are shut down. Consolidation can achieve more aggressive power saving with

the risk of SLA violations. From the users' perspective, the SLA should be guaranteed. This aggressive mechanism make the problem more complicated.

Fourth, PMM needs to be enriched with more PM policies and may take more components into account (for example, DSP in BBUs). So far, we have integrated three representative PM policies into PM library in PMM. More policies could be added to meet different requirements.

REFERENCES

[1] Beagle board. http://www.digikey.com/beagleboard.

[2] "credit scheduler". http://wiki.xensource.com/xenwiki/CreditScheduler.

[3] Electronic educational devices inc., "watts up pro power meter". http://www.wattsupmeters.com.

[4] Extech instruments corporation, 'extech 380801 power analyzer". http://www.extech.com/.

[5] Gnuradio. http://gnuradio.org/.

[6] mibench. http://www.eecs.umich.edu/mibench/.

[7] Openbts. http://openbts.sourceforge.net/.

[8] Specweb2005. http://www.spec.org/web2005/.

[9] sysstat. http://pagesperso-orange.fr/sebastien.godard/.

[10] AMD. White paper publication 26094: Bios and kernel developer's guide for ame athlon 64 and amd opteron processorss. 2006.

[11] H. Aydin, R. G. Melhem, D. Mossé, and P. Mejía-Alvarez. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Trans. Computers*, 2001.

[12] S. Banerjee and A. Misra. Adapting transmission power for optimal energy reliable multi-hop wireless communication. In *Proc. of Wireless Optimiazation Workshop (WiOpt)*, 2003.

[13] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03*. ACM, 2003.

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

[15] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *In Proceedings of the 9th ACM SIGOPS European Workshop*, 2000.

[16] R. A. Berry and R. G. Gallager. Communication over fading channels with delay constraints. *IEEE Trans. on Information Theory*, 2002.

[17] L. Bertini, J. C. B. Leite, and D. Mosse. Generalized tardiness quantile metric: Distributed dvs for soft real-time web clusters. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 227–236, Washington, DC, USA, 2009. IEEE Computer Society.

[18] W. L. Bircher and L. K. John. Complete system power estimation: A trickle-down approach based on performance events. In *ISPASS '07*, 2007.

[19] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case for power management in web servers. 2002.

[20] E. Brockmeyer, M. Miranda, H. Corporaal, F. Catthoor, M. Mir, H. Corporaal, and F. Catthoor. Layer assignment techniques for low energy in multi-layered memory organisations. In *DATE'03*, 2003.

[21] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA'00*, 2000.

[22] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *ICS'03*, 2003.

[23] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *SOSP'01*, 2001.

[24] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI'08*. USENIX Association, 2008.

[25] J.-J. Chen and T.-W. Kuo. Voltage scaling scheduling for periodic real-time tasks in reward maximization. In *RTSS'05*, 2005.

[26] W. Chen and U. Mitra. Energy efficient scheduling with individual packet delay constraints. In *Infocom'06*, 2006.

[27] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. *SIGMETRICS Perform. Eval. Rev.*, 2005.

[28] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *SC '02*. IEEE Computer Society Press, 2002.

[29] B. Collins and R. L. Cruz. Transmission policies for time varying channels with average delay constraints. In *Proc. of Allerton Conference on Communnication, Control, and Computing*, 1999.

[30] B. Colwell. We may need a new box. *Computer*, 2004.

[31] G. Contreras and M. Martonosi. Power prediction for intel xscale®processors using performance monitoring unit events. In *ISLPED '05*. ACM, 2005.

[32] T. M. Cover and J. A. Thomas. Elements of information theory. *New York: Wiley*, 1991.

[33] J. K. Dey, J. F. Kurose, and D. F. Towsley. On-line scheduling policies for a class of iris (increasing reward with increasing service) real-time tasks. *IEEE Trans. Computers*, 1996.

[34] K. Dudzinski and S. Walukiewicz. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 1987.

[35] M. E. Femal and V. W. Freeh. Boosting data center performance through non-uniform power allocation. In *ICAC '05*. IEEE Computer Society, 2005.

[36] D. R. S. Economous, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *MoBS '06*, 2006.

[37] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[38] EPA. Epa report to congress on server and data center energy efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[39] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for dram power management. In *ISLPED '01*. ACM, 2001.

[40] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07*. ACM, 2007.

[41] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *ICS '05*. ACM, 2005.

[42] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 1990.

[43] A. Fu, E. Modiano, and J. N. Tsitsiklis. Optimal energy allocation for delay-constrained data transmission over a time-varying channel. In *Infocom'03*, 2003.

[44] A. E. Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu, and S. Zahedi. Energy-efficient scheduling of packet transmissions over wireless networks. In *Infocom'02*, 2002.

[45] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *SIGMETRICS '09*. ACM, 2009.

[46] S. R. Garner. Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, 1995.

[47] L. Georgiadis, R. Guerin, and A. Parekh. Optimal multiplexing on a single link: delay and buffer requirements. *Information Theory, IEEE Transactions on*, 1997.

[48] J. Gong and C.-Z. Xu. A gray-box feedback control approach for system-level peak power management. In *ICPP'10*, volume 0. IEEE Computer Society, 2010.

[49] J. Gong and C.-Z. Xu. vpnp: Automated coordination of power and performance in virtualized datacenters. In *IWQoS '10*, 2010.

[50] J. Gong, X. Zhong, and C.-Z. Xu. Energy and timing constrained system reward maximization on wireless networks. In *ICDCS'08*, 2008.

[51] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software

power estimation: The softwatt approach. In *HPCA '02*. IEEE Computer Society, 2002.

[52] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP '05*. ACM, 2005.

[53] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[54] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED '07*. ACM, 2007.

[55] Hewlett-Packard, Intel, Microsoft, Pheonix, and Toshiba. Advanced configuration and power interface specification revision 4.0, 2009.

[56] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput.*, 56(4), 2007.

[57] T. Horvath and K. Skadron. Multi-mode energy management for multi-tier server clusters. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008.

[58] M. T. Inc. Calculating memory system power for ddr3, 2007.

[59] Intel. White paper publication 301170: Enhanced intel speedstep technology for the intel pentium m processor. 2004.

[60] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3A: System Programming Guide Part 1*, 2009.

[61] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *ISLPED '04*, New York, NY, USA, 2004. ACM.

[62] Y. Joo, Y. Choi, H. Shim, H. G. Lee, K. Kim, and N. Chang. Energy exploration and reduction of sdram memory systems. In *DAC '02*, New York, NY, USA, 2002. ACM.

[63] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *ICAC '09*. ACM, 2009.

[64] A. Kamra. Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites. In *IWQoS*, 2004.

[65] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2004.

[66] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. Rawson, and C. Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *ICAC '07*. IEEE Computer Society, 2007.

[67] M. A. Khojastepour and A. Sabharwal. Delay-constrained scheduling: Power efficiency, filter design, and bounds. In *Infocom'04*, 2004.

[68] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *ICAC '08*. IEEE Computer Society, 2008.

[69] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *Computer*, 36(12), 2003.

[70] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *ICAC '07*. IEEE Computer Society, 2007.

[71] J. C. Leite, D. M. Kusic, D. Mossé, and L. Bertini. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, New York, NY, USA, 2010. ACM.

[72] D. Li and P. H. Chou. Maximizing efficiency of solar-powered systems by load matching. In *ISLPED '04*, New York, NY, USA, 2004. ACM.

[73] D. Li and P. H. Chou. Application/architecture power co-optimization for embedded systems powered by renewable sources. In *DAC'05*, 2005.

[74] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 1973.

[75] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[76] Y.-H. Lu, L. Benini, and G. De Micheli. Operating-system directed power reduction. In *ISLPED '00*. ACM, 2000.

[77] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *CASES '02*. ACM Press, 2002.

[78] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *ASPLOS '09*. ACM, 2009.

[79] P. Mejia-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Trans. Embed. Comput. Syst.*, 2004.

[80] D. A. Menasce. Tpc-w - a benchmark for e-commerce, 2002.

[81] R. J. Minerick, V. Freech, and P. M. Kogge. Dynamic power management using feedback. In *Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2002.

[82] R. Nathuji, P. England, P. Sharma, and A. Singh. Feedback driven qos-aware power budgeting for virtualized servers. In *FeBID '09*, 2009.

[83] R. Nathuji and K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *SOSP '07*. ACM, 2007.

[84] M. J. Neely. Optimal energy and delay tradeoffs for multi-user wireless downlinks. In *Infocom'06*, 2006.

[85] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *EuroSys '09*. ACM, 2009.

[86] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini. Dma-aware memory energy management. In *HPCA '06*, 2006.

[87] M. Pedram. *Power Aware Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[88] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *ICS '04*. ACM, 2004.

[89] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *In Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2001.

[90] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting redundancy to conserve energy in storage systems. *SIGMETRICS Perform. Eval. Rev.*, 34(1), 2006.

[91] D. Rajan, A. Sabharwal, and B. Aazhang. Delay-bounded packet scheduling of bursty traffic over wireless channels. *IEEE Trans. on Information Theory*, 2004.

[92] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A resource allocation model for qos management. In *RTSS'97*, 1997.

[93] D. Rakhmatov and S. Vrudhula. Energy management for battery-powered embedded systems. *ACM Trans. Embed. Comput. Syst.*, 2003.

[94] P. Ranganathan, P. Leech, D. Irwin, J. Chase, and H. Packard. Ensemble-level power management for dense blade servers. In *ISCA '06*, 2006.

[95] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *ICAC '09*. ACM, 2009.

[96] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[97] C. A. Rusu, R. Melhem, and D. Mossé. Maximizing the system value while satisfying time and energy constraints. *IBM J. Res. Dev.*, 2003.

[98] C. Schurgers, V. Raghunathan, and M. B. Srivastava. Power management for energy-aware communication systems. *ACM Trans. Embedded Comput. Syst.*, 2003.

[99] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron. Power-aware qos management in web servers. In *RTSS '03*, 2003.

[100] V. Sivaraman, F. M. Chiussi, and M. Gerla. End-to-end statistical delay service under gps and edf scheduling: A comparison study. In *Infocom'01*, 2001.

[101] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. In *the American Control Conference*, 1991.

[102] Y. Tan, W. Liu, and Q. Qiu. Adaptive power management using reinforcement learning. In *ICCAD '09*. ACM, 2009.

[103] G. Tesauro, R. Das, H. Chan, J. O. Kephart, D. Levine, F. L. R. III, and C. Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *NIPS*. MIT Press, 2007.

[104] E. Uysal-Biyikoglu, B. Prabhakar, and A. E. Gamal. Energy-eficient packet transmission over a wireless link. *IEEE/ACM Trans. on Networking*, 2002.

[105] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3), 2005.

[106] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08*. Springer-Verlag New York, Inc., 2008.

[107] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of hpc applications. In *ICS '08*. ACM, 2008.

[108] A. Verma, G. Dasgupta, T. Kumar, N. Pradipta, and D. R. Kothari. Server workload analysis for power minimization using consolidation. In *USENIX ATC'09*. USENIX Association, 2009.

[109] H. Wang and N. B. Mandayam. Delay and energy constrained dynamic power control. In *GLOBECOM'01*, 2001.

[110] H. Wang and N. B. Mandayam. Opportunistic file transfer over a fading channel under energy and delay constraints. *IEEE Transactions on Communications*, 2005.

[111] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *HPCA'08*. IEEE Computer Society, 2008.

[112] X. Wang and Y. Wang. Co-con: Coordinated control of power and application performance for virtualized server clusters. In *IWQoS '09*, 2009.

[113] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS '08*. IEEE Computer Society, 2008.

[114] Y. Wang and I. H. Witten. Pace regression. *Hamilton, New Zealand: University of Waikato, Department of Computer Science, 1999*.

[115] J. Wei and C.-Z. Xu. eqos: Provisioning of client-perceived end-to-end qos guarantees in web servers. *IEEE Trans. Computers*, 55(12), 2006.

[116] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark. Formal control techniques for power-performance management. *IEEE Micro*, 2005.

[117] C.-Z. Xu, B. Liu, and J. Wei. Model predictive feedback control for qos assurance in webservers. *Computer*, 2008.

[118] M. Xu and C.-Z. Xu. Decay function model for resource configuration and adaptive allocation on internet servers. In *IWQoS '04*, 2004.

[119] M. Zafer and E. Modiano. A calculus approach to minimum energy transmission policies with quality of service guarantees. In *Infocom'05*, 2005.

[120] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: Managing energy as a first class operating system resource. In *ASPLOS '02*, 2002.

[121] F. Zhang and S. T. Chanson. Improving communication energy efficiency in wireless networks powered by renewable energy sources. *IEEE Trans. on Vehicular Technology*, 2005.

[122] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West. Friendly virtual machines - leveraging a feedback-control model for application adaptation. In *VEE'04*. ACM, 2004.

[123] X. Zhong and C.-Z. Xu. System-wide energy minimization for real-time tasks: lower bound and approximation. In *ICCAD '06*, New York, NY, USA, 2006. ACM.

[124] X. Zhong and C.-Z. Xu. Energy-efficient wireless packet scheduling with quality of service control. *IEEE Trans.on Mobile Computing*, 2007.

[125] X. Zhong and C.-Z. Xu. Online energy efficient packet scheduling with delay constraints in wireless networks. In *Infocom'08*, 2008.

[126] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In *HPCA '04*. IEEE Computer Society, 2004.

[127] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin. What does control theory bring to systems research? *SIGOPS Oper. Syst. Rev.*, 43(1):62–69, 2009.

[128] Y. Zhu and F. Mueller. Feedback edf scheduling exploiting dynamic voltage scaling. In *RTAS '04*, 2004.

ABSTRACT

QOS-AWARE FINE-GRAINED POWER MANAGEMENT IN NETWORKED COMPUTING SYSTEMS

by

JIAYU GONG

DECEMBER 2011

Advisor: Dr. Cheng-Zhong Xu

Major: Computer Engineering

Degree: Doctor of Philosophy

Power is a major design concern of today's networked computing systems, from low-power battery-powered mobile and embedded systems to high-power enterprise servers. Embedded systems are required to be power efficiency because most embedded systems are powered by battery with limited capacity. Similar concern of power expenditure rises as well in enterprise server environments due to cooling requirement, power delivery limit, electricity costs as well as environment pollutions.

The power consumption in networked computing systems includes that on circuit board and that for communication. In the context of networked real-time systems, the power dissipation on wireless communication is more significant than that on circuit board. We focus on packet scheduling for wireless real-time systems with renewable energy resources. In such a scenario, it is required to transmit data with higher level of importance periodically. We formulate this packet scheduling problem as an NP-hard reward maximization problem with time and energy constraints. An optimal solution with pseudo-polynomial time complexity is presented. In addition, we propose a sub-optimal solution with polynomial time complexity.

Circuit board, especially processor, power consumption is still the major source

of system power consumption. We provide a general-purposed, practical and comprehensive power management middleware for networked computing systems to manage circuit board power consumption thus to affect system-level power consumption. It has the functionalities of power and performance monitoring, power management (PM) policy selection and PM control, as well as energy efficiency analysis. This middleware includes an extensible PM policy library. We implemented a prototype of this middleware on Base Band Units (BBUs) with three PM policies enclosed. These policies have been validated on different platforms, such as enterprise servers, virtual environments and BBUs.

In enterprise environments, the power dissipation on circuit board dominates. Regulation on computing resources on board has a significant impact on power consumption. Dynamic Voltage and Frequency Scaling (DVFS) is an effective technique to conserve energy consumption. We investigate system-level power management in order to avoid system failures due to power capacity overload or overheating. This management needs to control the power consumption in an accurate and responsive manner, which cannot be achieve by the existing black-box feedback control. Thus we present a model-predictive feedback controller to regulate processor frequency so that power budget can be satisfied without significant loss on performance.

In addition to providing power guarantee alone, performance with respect to service-level agreements (SLAs) is required to be guaranteed as well. The proliferation of virtualization technology imposes new challenges on power management due to resource sharing. It is hard to achieve optimization in both power and performance on shared infrastructures due to system dynamics. We propose vPnP, a feedback control based coordination approach providing guarantee on application-level performance and underlying physical host power consumption in virtualized environments. This system can adapt gracefully to workload change. The preliminary results show

its flexibility to achieve different levels of tradeoffs between power and performance as well as its robustness over a variety of workloads.

It is desirable for improve energy efficiency of systems, such as BBUs, hosting soft-real time applications. We proposed a power management strategy for controlling delay and minimizing power consumption using DVFS. We use the Robbins-Monro (RM) stochastic approximation method to estimate delay quantile. We couple a fuzzy controller with the RM algorithm to scale CPU frequency that will maintain performance within the specified QoS.

AUTOBIOGRAPHICAL STATEMENT

JIAYU GONG

Jiayu Gong is a Ph.D. candidate of Department of Electrical and Computer Engineering at Wayne State University. He received the B.S. degree and the M.S. degree both in Computer Science from Nanjing University, Nanjing, China, in 2002 and 2005, respectively.

His research interests include power management in server environments and embedded systems, mobile computing, and resource management in cloud computing. He has published one journal paper in IEEE Transactions of Mobile Computing, 4 papers in proceeding of referred conferences and workshops, as well as one book chapter. He is the receiver of the prestigious *Thomas C. Rumble Fellowship* in 2010 and the *Summer Dissertation Fellowship*. He is also two times receiver of *Travel Award* for Excellence in Graduate Student Research (2008 and 2010). All these rewards are from Wayne State University.