5-2016

Feedback-Directed Management of Performance and Power for Emerging Computer Systems

Xing Fu
University of Tennessee - Knoxville, xfu1@vols.utk.edu

Recommended Citation

Fu, Xing, "Feedback-Directed Management of Performance and Power for Emerging Computer Systems. " PhD diss., University of Tennessee, 2016.
https://trace.tennessee.edu/utk_graddiss/3694

To the Graduate Council:

I am submitting herewith a dissertation written by Xing Fu entitled "Feedback-Directed Management of Performance and Power for Emerging Computer Systems." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Xiaorui Wang, Major Professor

We have read this dissertation and recommend its acceptance:

Gregory D. Peterson, Seddik Djouadi, Mingjun Zhang

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Feedback-Directed Management of Performance and Power for Emerging Computer Systems

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Xing Fu

May 2016

Dedicate to my family

Acknowledgements

I appreciate my advisor Dr. Xiaorui Wang for his training preparing me well for research and academic career. Thanks for his patience while I am overcoming personal difficulties. I would like to express my sincere gratitude to Dr. Gregory Peterson and Dr. Seddik M. Djouadi. I learned a lot from their courses and obtained their recommendation letters when I applied for a faculty position of University of South Alabama. This dissertation would not have been possible without Dr. Mingjun Zhang's generous support. His insightful feedback improved the quality of dissertation significantly.

I am glad to have the opportunities to collaborate with Ming Chen, Yefu Wang, Xiaodong Wang at University of Tennessee. My experience at a VMWare core technology group will have profound impact on my career. I obtain deeper understanding of computer system research by collaborating with reseachers from almost all well-known labs in the related field.

Abstract

Emerging computing systems face the critical management challenge of both performance and power simultaneously. For example, distributed real-time embedded systems such as cyber-physical systems need to reduce power consumption while enforces CPU utilization bounds on multiple uniprocessors in order to meet end-to-end deadlines. Data centers operators attempt to oversubscribe data center power delivery networks to reduce throughput penalty during a power overload.

This dissertation presents latest development of a control framework which adopts novel control theoretic approaches to address emerging computing systems including multi-core real-time embedded systems, distributed real-time embedded systems, and data centers. The dissertation leverages task migration and cache partitioning mechanisms in multi-core systems to reduce power and energy consumption while control CPU utilizations. We then present a control algorithm for simultaneous temperature and utilization control for distributed real-time embedded systems. Algorithms and optimizations are presented to extend state-of-the-art model predictive control technique to overcome technical challenges such as scalability. We also adapt the parameter of a power controller widely adopted by IBM servers to improve its performance significantly. A power capping algorithm for an entire data center by shifting power between data center cooling systems and IT equipments is proposed for improving performance. Finally, we present a hierarchical heuristic to minimize energy consumption of Virtual Desktop Infrastructure without violating its performance constraints. Both theoretic analysis, hardware, and simulation experiments demonstrate that our control algorithms can achieve better performance for those power-aware emerging computing systems compared to state-of-the-art baselines. The control framework also found successful applications in other systems such as cyber-physical surveillance systems. Results of feedback-directed management of performance and power based on control frameworks reveal wide potential applications in autonomic management in the era of cloud computing which consists of enormous mobile embedded devices and data centers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Traditionally, academic and industry focused on increasing CPU processor performance by shrinking feature sizes and increasing transistor density. With the continued scaling, power related issues now are first-priority design constraints of CPU design. Moreover, the number of huge data centers grows rapidly to accommodate the demand of cloud computing and supercomputing. The power related issues such as energy efficiency and thermal management and power capping rise in various computing systems from multi-core systems to data centers and are major hurdles on the road to future computing system. In this dissertation, several integrated control solution are proposed to effectively manage both power and performance by adopting novel control-theoretic methodologies for various emerging computing systems.

Task Consolidation in Multi-Core Real-Time Systems

Multi-core processors have become a primary trend in the current processor development due to well-known technological barriers such as the "Power Wall" and "Instruction-level Parallelism Wall". As a result, future high-performance real-time embedded systems are anticipated to be equipped with multi-core processors, or even many-core processors (i.e., processors with tens or hundreds of cores). However, power consumption still remains the major constraint for the further throughput improvement of multi-core processors. Therefore, new scheduling algorithms must be developed to minimize power consumption while achieving the desired timeliness guarantees for multi-core (and many-core) real-time embedded systems.

Although various power/energy-efficient scheduling algorithms have recently been proposed for multi-core real-time embedded systems (e.g., [119]), existing studies focus on open-loop solutions such as *static* speed scheduling and offline DVFS (dynamic voltage and frequency scaling) configurations. While those open-loop solutions can work effectively for traditional real-time embedded systems deployed in closed execution environments, they may incur degraded performance in terms of power/energy efficiency and real-time guarantees when applied to real-time embedded systems that execute in open and *unpredictable* environments in which workloads (*e.g.*, WCETs) are unknown and may vary significantly at runtime. Therefore, in order to achieve runtime power optimization and real-time guarantees, novel online strategies must be designed to dynamically respond to execution time variations for multi-core real-time embedded systems running in unpredictable environments.

Recently, feedback control techniques have been demonstrated to be a valid tool in providing timeliness guarantees for real-time embedded systems by adapting to workload variations based on dynamic feedback. In particular, feedback-based *CPU utilization control* [93] has been shown to be an effective way of providing real-time guarantees for soft real-time systems. The goal of utilization control is to enforce appropriate schedulable utilization bounds on all CPU cores in a real-time embedded system, despite significant uncertainties in system workloads. As a result, utilization control can guarantee all the real-time deadlines of the system without accurate knowledge of the workload, such as task execution times. However, existing utilization control algorithms are not designed to provide online power minimization for multi-core real-time systems. A recent study [141] proposes a power-aware utilization control approach that adopts DVFS to achieve utilization control and power efficiency. While this solution can effectively reduce *dynamic* power consumption, it cannot minimize *static (leakage)* power consumption because it does not minimize the number of active CPU cores in response to workload variations. As chip feature sizes continue to shrink, it becomes increasingly important to minimize leakage power since leakage power consumption is becoming a major contributor to the total power consumption of a multi-core processor [73].

To minimize the number of active CPU cores, it is necessary to migrate tasks among the cores for consolidation. In traditional multiprocessor real-time systems, tasks are often assigned to processors in a static way, at design time, due to the large overheads of online task migrations. A key advantage of the shared L2 caches in many multi-core real-time systems is that the overhead of migrating a task among cores is less than 40 microseconds, which is sufficiently small in many real systems [12][153]. This feature allows multi-core real-time systems to be more power-efficient since the leakage power consumption can be minimized by dynamic task consolidation. Although task migrations in multi-core processors may cause L1 cache misses, the typical penalty of an L1 cache miss is only 10-30 CPU cycles. In contrast, in traditional multiprocessor real-time systems, task migrations can be expensive by having frequent L2 cache misses, whose penalty is approximately 100-300 CPU cycles [12].

In this dissertation, we propose a novel *online* solution that integrates feedback control with optimization strategies to minimize (both dynamic and leakage) power consumption and guarantee timeliness for multi-core real-time embedded systems. Our solution monitors the utilization of each CPU core in the system and dynamically responds to execution time variations by conducting per-core DVFS and task consolidation among the cores in a multi-core processor. In our solution, each CPU core has a utilization controller that throttles the DVFS level of the core so that its utilization stays slightly below the schedulable bound for minimized dynamic power with real-time guarantees. To minimize leakage power, we dynamically consolidate real-time tasks onto a few of the most power-efficient cores on a longer timescale by utilizing the small overhead of migrating tasks among different cores within a multi-core processor. The migration is subject to the schedulable utilization bounds of the active cores. We then shut down unused CPU cores for minimized leakage power.

Cache-Aware Utilization Control for Energy Efficiency in Multi-Core RT Systems

Despite a significant amount of existing work on power management for traditional multi-processor real-time systems, existing power management algorithms are not designed to sufficiently utilize the new features available in many multi-core processors, such as shared L2 caches and per-core DVFS (Dynamic Voltage and Frequency Scaling), to effectively minimize processor energy consumption while providing real-time guarantees. For example, most current power/energy management algorithms assume that all the cores of a processor can only have a uniform DVFS level while per-core DVFS is already available (*e.g.*, AMD's Independent Dynamic Core Technology) to allow better power/energy efficiency. Intel's new 48-core processor also features per-tile DVFS with two cores within each tile. In addition, the current algorithms are not designed to dynamically partition the shared L2 caches among the different cores for better real-time performance and to conduct dynamic cache resizing to place rarely accessed cache units into low-power modes for minimized cache leakage power consumption. Therefore, novel power management algorithms are needed to utilize the shared L2 caches and per-core DVFS for maximized energy savings.

The existing research on power-aware utilization control primarily relies on DVFS by assuming that the task execution times can be adapted linearly with the CPU frequency. While this assumption is valid for real-time tasks that are computation intensive, memory-intensive tasks can have approximately 75% of their instructions that are load or store [104, 60]. Consequently, when a processor core is running memory-intensive tasks and the CPU frequency is set to the highest level, the utilization can still be above the desired schedulable bound, resulting in undesired deadline misses. In this case, the cache size partitioned to the core can be increased to reduce the cache miss rate and cache access latency due to reduced main memory access delay. As a result, the CPU utilization can be lowered for better real-time performance. Similarly, if the utilization is lower than the bound, even when the frequency is already throttled to the lowest level, the active cache size can be reduced and rarely accessed cache units can be put into low-power modes to minimize cache leakage power.

In this dissertation, we propose a two-level utilization control solution for energy efficiency in multi-core real-time systems. At the core level, our solution utilizes both per-core DVFS and dynamic L2 cache partitioning to address two (often conflicting) optimization objectives: controlling the CPU utilization of each core to its desired schedulable bound and minimizing the core energy consumption. Since the utilization contributed by a periodic real-time task is determined by both its CPU frequency-dependent and frequency-independent execution times [24], per-core DVFS and cache partitioning can be used to adapt the frequency-dependent and independent portions, respectively. A key challenge in our design is that traditional control theory, such as PID (Proportional-Integral-Derivative) and MPC (Model Predictive Control), cannot effectively handle multiple optimization objectives. Therefore, we propose a novel utilization controller, based on advanced Multi-Objective MPC control theory [96][22], to achieve both optimization objectives. At the processor level, a cache demand arbitrator is proposed to coordinate the cache size demand from each core and conduct dynamic cache resizing to minimize the leakage power consumption of the shared L2 caches.

Power-Aware Utilization Control for Distributed RT Systems

Traditional approaches to handling end-to-end real-time tasks, such as end-to-end scheduling [129] and distributed priority ceiling [111], rely on schedulability analysis, which requires *a priori* knowledge of the tasks' Worst-Case Execution Times (WCET). While such open-loop approaches work effectively in the closed execution environments of traditional real-time systems, they may violate the desired timing constraints or severely underutilize the system when task execution times are highly unpredictable. In recent years, a new category of real-time applications called Distributed Real-time Embedded (DRE) systems has been rapidly growing. DRE systems commonly execute in open and *unpredictable* environments in which workloads are unknown and vary significantly at runtime. Such systems include data-driven systems whose execution is heavily influenced by volatile environments. For example, task execution times in vision-based feedback control systems depend on the content of live camera images of changing environments [62]. DRE systems call for a paradigm shift from classical real-time computing that relies on accurate characterization of workloads and platform.

Recently, feedback control techniques have shown a lot of promise in providing real-time guarantees for DRE systems by adapting to workload variations based on dynamic feedback. In particular, feedback-based CPU utilization control [93][144] has been demonstrated to be an effective way of meeting the *end-to-end deadlines* for soft DRE systems. The primary goal of utilization control is to enforce appropriate schedulable utilization bounds (e.g., the Liu and Layland bound for RMS) on all the processors in a DRE system, despite significant uncertainties in system workloads. In the meantime, it tries to maximize the system utility by controlling CPU utilizations to stay slightly below their schedulable bounds so that the processors can be utilized to the maximum degree. Utilization control can also enhance system survivability by providing overload protection against workload fluctuation [139].

However, previous research on CPU utilization control focuses exclusively on task rate adaptation by assuming task rates can be continuously tuned within specified ranges. While rate adaptation is an effective actuator for some DRE systems, it has several limitations. First, it is often infeasible to achieve desired utilization set points by rate adaptation alone [140]. For example, many DRE systems are configured based on tasks' WCETs. Consequently, even when all the tasks are running at their highest rates, CPU utilizations are still way below the desired set points, resulting in severely underutilized systems. In that case, CPU frequency scaling can be used for power savings while still guaranteeing task schedulability. Second, many tasks in DRE systems only support a few discrete rates. While optimization strategies [40][77] are developed to handle discrete task rates, they rely on the common assumption that task WCETs are known *a priori* and accurate, which makes them less applicable to DRE systems running in *unpredictable* environments. Third, the model of task rate in many applications could be complex and vary at runtime based on application evolution [55][28]. As a result, the estimated task rate ranges are often inaccurate and may change significantly online, which may lead to unexpected rate saturation and even deadline misses when CPU utilizations are higher than the schedulable bounds and can be lowered down only by rate adaptation. Finally, some DRE systems may not allow rate adaptation for any tasks but their CPU utilizations still need to be controlled. Therefore, it is important to explore complementary ways for effective CPU utilization control.

In this paper, we propose to use Dynamic Voltage and Frequency Scaling (DVFS) in conjunction with rate adaptation for utilization control. Since the CPU utilization contributed by a real-time periodic task is determined by both its rate and its execution time, CPU frequency scaling can be used to adapt task execution time for power-efficient utilization control. The integration of DVFS in utilization control introduces several new challenges. First, a centralized controller for simultaneous rate adaptation and DVFS would have a Multi-Input-Multi-Output (MIMO) nonlinear model. Therefore, multiple linear control loops are more preferable for acceptable runtime overhead. Second, different control loops need to be carefully designed to coordinate together for the desired control functions. Finally, the control accuracy and global system stability of the coordinated control solution must be analytically assured.

This dissertation presents a two-layer coordinated CPU utilization control architecture. The primary control loop uses DVFS to locally control the CPU utilization of each processor. In the meantime, the secondary control loop adopts rate adaptation to control the utilizations of all the processors at the cluster level on a finer timescale.

The rest of this chapter is organized as follows. We formulate the new CPU utilization control problem in Section 5.1. Section 5.2 presents the system model and control architecture. Section 5.3 briefly introduces the rate adaptation loop while Section 5.4 provides the detailed design and analysis of the CPU frequency scaling loop. Section 5.5 discusses the implementation of the control architecture in a real-time middleware system. Section 5.6 presents our empirical results on a physical testbed.

Coordinated Temperature and Utilization Control for Distributed Real-Time Embedded Systems

A new class of real-time applications called distributed real-time embedded (DRE) systems has been rapidly growing. DRE systems include wireless sensor networks and cyber-physical systems. They commonly execute in open and *unpredictable* environments, in which both workloads and system conditions are unknown and may vary significantly at runtime. For example, task execution times in vision-based surveillance systems depend on the content of live camera images of changing environments [62]. Therefore, DRE applications commonly require runtime control and guarantees of end-to-end timeliness for their proper operation.

However, existing work on utilization control can only provide timeliness guarantees, while today's DRE systems face an increasing probability of overheating and even thermal failures, due to their continuously decreasing feature size and increasing demand for computation capabilities. For example, recent studies show that 50% of all electronics failures are related to overheating [152]. More specifically, the lifetime of a processor can be approximately halved if it runs 10-15°C higher than its normal temperature range [134]. Furthermore, a 15°C increase in temperature could double the failure rate of a disk drive [10]. Therefore, thermal constraints also need to be strictly enforced for DRE systems. Although some recent research has proposed optimization algorithms based on task allocation and configurations of processor voltage/frequency to achieve minimized system temperature and guaranteed real-time performance [37, 135], those open-loop solutions cannot be directly applied to DRE systems where workloads and system conditions may vary at runtime. While some dynamic thermal managment (DTM) approaches have been proposed for general computer systems (*e.g.*, [30]), they cannot provide desired real-time guarantees for

DRE systems. Therefore, existing work can only provide either timeliness guarantees or thermal control in an *isolated* manner.

Simultaneous thermal and utilization control is challenging because the desired guarantees *cannot* be achieved by simply putting the two control loops together. Without effective coordination, individual control solutions may conflict with each other. For example, many thermal management methods rely on dynamic voltage and frequency scaling (DVFS), which may significantly impact the execution times of the real-time tasks running in the systems. As a result, the timeliness guarantees provided by existing control solutions may be severely violated. In addition, although each control loop can be proven to be stable individually, system stability must be theoretically guaranteed for the entire system. Although previous work has approached the coordination problem by forcing one control loop to run on a significantly longer timescale than the other loop [142], both the thermal and utilization control loops prefer to run on small timescales for DRE systems, because both the thermal and timing constraints are critical and must be promptly enforced upon any violations. Hence, a new kind of coordination methodology must be designed and analyzed.

This dissertation proposes a novel coordinated thermal and utilization control solution to provide simultaneous thermal and timeliness guarantees for DRE systems. The thermal control loop locally controls the temperature of each processor, while the utilization control loop provides end-to-end timeliness guarantees at the cluster level.

Power Oversubscription in Data Centers

Server power consumption has become a first-order concern for modern enterprise data centers. In order to amortize the non-recurring investments in the power supply facility of a data center, it is preferable to operate the facility as close as possible to its maximum capacity [46]. An additional pressure on facility operators is that upgrades in power delivery systems are extremely expensive and often lag behind required increases in hosted servers to support new business. Both of these reasons result in pressure to load as many servers as possible on the branch circuits that supply power to computer racks. Traditionally, branch circuits are provisioned conservatively based on server nameplate power ratings, which results in significant waste of the branch circuit's power supply capacity.

A promising solution is to oversubscribe the branch circuit. This involves placing more servers on it than it can support if all the servers use their maximum power consumption at the same time. To prevent overload of the circuit, *power capping* has been proposed to limit the aggregate server power to the branch circuit capacity. This provides better performance when power demand is below the branch circuit capacity and prevents undesired shutdowns by slowing down servers occasionally when the power demand is over the branch circuit capacity. Server manufacturers have responded by providing power capping as a standard feature to limit the power draw to a user-defined limit (power cap) [64][67].

An important issue for all power capping solutions is to select an appropriate power cap. In order to maximize the number of hosted servers in a data center, a common practice is to set the server power cap as the rated current limit of the branch circuit divided by the number of servers [64][57][46][113]. The main rationale of this practice is that *peak* power should never exceed the branch circuit capacity, otherwise the branch circuit's circuit breaker (CB) might trip and cause undesired server shutdowns, or even power outages. If the peak power becomes higher than the cap at runtime due to workload increases, immediate actions (such as processor throttling) are taken to maintain the power below the cap as soon as possible. Some studies even suggest having a safety margin below the cap to avoid any instantaneous power overloads [136].

We argue that this common practice is too conservative, even though power capping is already a step ahead of traditional power provisioning based on nameplate power ratings. This conservativeness can result in an unnecessarily low system performance because even a small, short-lived power overload causes servers to slow down in spite of the fact that the circuit breakers will not trip. If harmless power overloads could be tolerated by power capping, then we can have higher performance, as well as more hosted servers with the same circuit capacity. We systematically study the tripping characteristics of a typical CB used in data centers. Our results on a physical test bed show that instantaneous violations of the rated CB power limit are not necessarily fatal because CBs are designed to sustain a certain amount of power overload. Whether a CB trips or not depends primarily on the transient behaviors of a power overload, such as the magnitude and time duration. The time interval for a CB to sustain a power overload is determined by the magnitude of the overload and normally, a higher magnitude leads to a shorter interval. Generally, a CB will trip only when the duration of an overload is longer than the allowed time interval. The allowed interval is also affected by the ambient temperature.

Based on those observations, we propose an adaptive power control strategy that utilizes the tripping characteristics of the equipped CB to aggressively optimize the system's performance without causing the CB to trip. The power controller is designed based on an advanced adaptive control theory for parameter tuning and to adapt to variations in ambient temperature.

Data Center Level Power Control

Power consumed by data centers has become a serious concern in era of Cloud computing. In addition to high electricity bills and negative environmental implications, increased power consumption may lead to system failures caused by power capacity overload or system overheating, as data centers increasingly deploy more and more servers for a higher utilization of their power budget. The goal of power control (also called power capping) is to have run-time measurement and control of the power consumed by a data center, so that the servers can achieve the highest system performance while keeping the power consumption below a given power budget.

Wang et al [138] proposed Scalable HIerarchical Power control (SHIP) to prevent system failures while allowing data centers to operate at peak efficiencies for a higher return on investment. While only the server power consumption is capped in SHIP, in this work, we propose to shift power between server racks and cooling systems (*e.g.*, a set of Computer Room Air Conditioners (CRACs)) for further optimized system performance within a desired power cap for an *entire* data center. We mainly focus on the case that the total power consumption of the entire data center exceeds the power distribution capacity of the facility, and thus we must throttle the power draw of servers or/and cooling systems. This situation can be expected to occur soon as many data centers rapidly deploy new servers, while their power distribution and cooling systems have already approached the peak capacities [138][52][46]. Within the foreseeable stringent power budget, if we give too much power to servers and too little to the cooling systems, some servers may have overheating or even undesired shutdowns. On the other side, if we allocate too much power to the cooling systems, many servers have to be turned off while the data center is overcooled. Therefore, it is challenging to have a globally optimal power allocation that maximizes the data center's performance.

The remainder of the chapter is organized as follows: we describe the system architecture consisting of an inner power control loop and an outer power control loop in Section 8.1 and the detail design of the outer power control in Section 8.2. Further improvement of the outer power control loop based on air-side economizer are presented in Section 8.3. We presents extensive large-scale simulations in Section 8.4.

End-to-End Energy Management of Virtual Desktop Infrastructure

Virtual infrastructure allows data center operators to reduce IT costs, including electricity. Virtual machine consolidation increases the utilization of physical infrastructure, making the data center more efficient and reducing its carbon footprint. Closely following on the heels of server consolidation, enterprises are fast adopting virtual desktop infrastructure (VDI) to replace and consolidate existing physical desktops as well. In a VDI environment, a user's operating system instance and applications are run on a virtual machine hosted in the enterprise data center.

Users remotely control the virtual machines using thin clients such as stateless hardware terminals, smartphones or tablet PCs.

In this work, we minimize energy consumption by manipulating various knobs such as CPU DVFS levels and consolidating virtual machines. A key challenge is to guarantee that performance will not be adversely affected, leading to undesired violations of service level agreements. To address this challenge, we establish a performance model which predicts end-to-end performance of VDI workloads, given CPU DVFS levels and consolidation ratios etc. We select a collection of typical applications used by VDI users, and define a relevant end-to-end performance metric. We do this instead of adopting well-known CPU utilization or throughput metrics because they don't sufficiently reflect a user's experience with interactive applications (which is of prime importance in a VDI deployment).

The remainder of the paper is organized as follows. In Section 9.1 we describe end-to-end energy management with performance guarantees. In Section 9.2, we present details of the system implementations and experimental results.

Contributions

Specially, this dissertation has the following contributions.

For Power-Aware Utilization Control:

- We derive an analytic model that captures the system dynamics of the new CPU utilization control problem.
- We design a two-layer coordinated control architecture and present detailed coordination analysis.
- We implement our control architecture in a real-time middleware system.
- We present empirical results to demonstrate that our control solution outperforms a state-of-the-art utilization controller that relies solely on rate adaptation.

For Task Consolidation:

- We propose a control theoretic solution for timeliness guarantees that minimizes both dynamic and leakage power consumption. Compared with traditional open-loop solutions, our solution can achieve better power efficiency and real-time performance when task execution times vary significantly at runtime in unpredictable environments.

- We design a two-level power optimization architecture that analytically integrates core-level utilization control with processor-level task consolidation to eliminate the complexity of one-level hybrid model-predictive control. The task consolidation problem is formulated as a bin-packing problem and several solutions are comparatively studied.

- While the majority existing work relies solely on simulations for evaluation, we present empirical results on a hardware multi-core testbed to demonstrate the efficacy of our integrated solution with the Mibench benchmarks [60]. Extensive simulation results also show that our solution can achieve more power savings than state-of-the-art algorithms in many-core systems.

For Cache Partitioning:

- We derive an analytic model that captures the system dynamics of the new cache-aware multi-core utilization control problem.

- We propose a two-level utilization control solution for energy efficiency that includes a core-level utilization controller and a processor-level cache demand arbitrator.

- We apply the recent advance in control theory, Multi-Objective MPC (MOMPC) theory, to design the utilization controller for achieving the two (often conflicting) optimization objectives.

- We present extensive experimental results (using the well-known *Mibench* [60] benchmarks) to demonstrate that our solution outperforms two state-of-the-art power management algorithms that do not consider L2 caches or per-core DVFS by having more accurate utilization control and less energy consumption.

For Temperature Control:

- While most existing work relies on open-loop optimization to minimize power and temperature for DRE systems with the assumption that task execution times and system thermal condition do not change significantly at runtime, we analytically model the temperature and CPU utilizations of a DRE system and design a feedback control solution for dynamic thermal and real-time guarantees for DRE systems running in *unpredictable* environments.

- While most existing closed-loop solutions provide either thermal or timeliness guarantee in an isolated manner, our solution coordinates the thermal and utilization control loops to provide simultaneous runtime guarantees. To our best knowledge, our solution is the first one that adopts robust control theory as a theoretical foundation such that both control loops can run on their respective desired timescales for prompt control actions with guaranteed system stability.

- While most existing work assumes that the processors support DVFS for thermal management, we design a task rate adaptation based thermal control loop for legacy processors without DVFS support. Our solution can achieve thermal guarantee for a heterogeneous cluster.

- While most existing work relies solely on simulations for evaluation, we present empirical results on a physical testbed to demonstrate the efficacy of our control solution and extensive simulations for a large-scale heterogeneous cluster.

For Power Oversubscription:

- We present a systematic study to investigate the tripping characteristics of a typical CB used in many data centers. While previous solutions simply assume that power can never exceed the CB's capacity, to the best of our knowledge, our work is the first that utilizes transient CB tripping behaviors to optimize server performance or host additional servers. We also consider the impacts of ambient temperature on transient CB behaviors.

- In contrast to most existing power capping solutions that rely on simplistic heuristics, we use control theory to design an adaptive power controller that precisely controls the transient response of power overload to follow the designed CB trip curve. We also propose a proactive control solution to explore the practical upper bound of power oversubscription.

- Our extensive hardware results with the SPEC CPU2006, SPECJBB, and LINPACK benchmarks show that the proposed CB-aware power control solutions achieve 38% better performance, on average, than a state-of-the-art baseline that simply uses the CB capacity as the power cap without considering the CB's tripping characteristics.

- We conduct analyses to show that our adaptive power capping solutions allow a server rack to host three times more servers than traditional static power provisioning schemes and 54% more servers than the current power capping practice widely used in the industry.

For Data Center Level Power Control,
Specifically, this work makes several major contributions:

- while previous power control solutions assume that a data center cooling system always runs at its full capacity. Our work shifts power between a cooling system and servers.

- we adopt a two-level control technique to design an outer cooling system power controller and an inner server power controller. We present a systematic study of a data center cooling system and formulate the controller as a nonlinear constrained problem.

- our large-scale simulation results demonstrate the efficacy of the proposed solution and show the advantages compared to baselines.

For End-to-End Energy Management:

- We derive an accurate performance model using a black-box modeling approach.

- Based on the model, we formulate an optimization problem to minimize the energy consumption while guaranteeing performance, and transform it into a canonical form which can be solved by standard optimization solvers. However, no polynomial time solvers exist to obtain the optimal solution. To scale the proposed solution in VDI deployments, which can have thousands of seats (VMs), a two-step heuristic algorithm is designed to reduce the algorithmic complexity significantly.

- We prototype the proposed solution and analyze the overhead of each component of the implementation to ensure that the overall solution will not introduce significant performance degradation or increased energy consumption. Experimental results from a hardware test bed demonstrate the efficacy of the proposed solution in terms of energy and performance. It significantly outperforms the state-of-the-art baseline widely adopted in industry today.

The rest of this dissertation is organized as follows. Chapter 2 discusses the related work. Chapter 3 presents Integrating Utilization Control with Task Consolidation for Power Optimization in Multi-Core Real-Time Systems and 4 presents Cache-Aware Utilization Control for Energy Efficiency in Multi-Core RT Systems, respectively. Chapter 5 presents Power-Aware Utilization Control for Distributed RT Systems. Chapter 6 presents Coordinated Temperature and Utilization Control for Distributed Real-Time Embedded Systems. Chapter 7 presents Power Oversubscription in Data Centers. Chapter 8 presents Data Center Level Power Control. Chapter 9 presents End-to-End Energy Management of Virtual Desktop Infrastructure. Chapter 10 concludes the dissertation.

Chapter 2

Related Work

Power-Aware Utilization Control for Distributed RT Systems

A survey of feedback performance control in computing systems is presented in [6]. Many projects that applied control theory to real-time scheduling and utilization control are closely related to this paper. Steere et al. and Goel et al. developed feedback-based schedulers [127][56] that guarantee desired progress rates for real-time applications. Abeni et al. presented control analysis of a reservation-based feedback scheduler [7]. Lu et al. developed a middleware service that adopts feedback control scheduling algorithms to control CPU utilization and deadline miss ratio [92]. Feedback control has also been applied to power control [82] and digital control applications [33].

Various CPU utilization control algorithms (e.g., [92] [126][87][139]) have been recently proposed to guarantee real-time deadlines. For example, Lu et al. designed constrained MIMO utilization control algorithm for multiple processors that are coupled due to end-to-end tasks [93]. Wang et al. proposed decentralized utilization control algorithm for large-scale distributed real-time systems [144]. Yao et al. developed an adaptive utilization control algorithm [151]. However, all those algorithms assume that task rates can only be continuously tuned. Hybrid control theory [77] and optimization strategies [40] are adopted to handle discrete task rates based on the assumption that task WCETs are known *a priori* and accurate, which makes them less applicable to DRE systems running in *unpredictable* environments. In contrast to all the existing work that relies exclusively on rate adaptation, we present a two-layer control architecture that uses both rate adaptation and DVFS for power-efficient utilization control.

Energy-efficient real-time scheduling algorithms have been proposed [17, 149, 37, 116, 16]. Most existing work relies on detailed knowledge (e.g., WCETs) of workloads to minimize the energy consumption or temperature, or maximize the system reward in an *open-loop* manner. While they can effectively guarantee task schedulability in closed environments without a feedback loop for adaptation to workload variations, they may not be directly applied to DRE systems whose workloads may significantly change at runtime. In contrast, we use DVFS as a knob to dynamically react to unpredictable workload variations instead of minimizing the energy consumption of the entire DRE system. To our best knowledge, our paper is the first effort that adopts DVFS for end-to-end CPU utilization control.

Integrating Utilization Control with Task Consolidation for Power Optimization in Multi-Core Real-Time Systems

Several projects have addressed the scheduling problems for multi-core real-time embedded systems. Anderson et al. proposed a cache-aware scheduling technique to avoid cache thrashing for real-time tasks on multi-core platforms [12]. Guan et al. presented test conditions for non-preemptive EDF and fixed priority scheduling [58]. However, these studies do not migrate tasks for power optimization. Sarkar et al. studied the impact of task migrations on the WCETs of real-time tasks [117]. Their work focused on WCET analysis instead of real-time scheduling. In addition, they assume non-shared L2 caches which can incur a much higher task migration overhead. Chattopadhyay et al. studied the WCET analysis for a unified cache multi-core processors [34]. All of these studies do not address the power optimization problem in multi-core real-time systems. In contrast, we attempt to minimize the power consumption of multi-core real-time systems in addition to providing real-time guarantees.

Power management is an important problem for real-time embedded systems. Multiple projects have studied real-time scheduling with power management for uniprocessor systems (*e.g.*, [75]). Aydin et al. considered the energy-aware partitioning of real-time tasks for multiprocessor systems [18]. However, the power models of [18] did not consider leakage power consumption. Chen et al. extended the power models adopted in [18] and proposed a real-time scheduling method that minimizes both dynamic and leakage energy consumption [36]. However, these studies focus on multi-processor real-time systems where task migration can be expensive due to state maintenance. Seo et al. studied energy efficient multi-core real-time scheduling [119]. Their assumption is that all cores must run at the same frequency (chip-wide DVFS). In contrast, we utilize the availability of per-core DVFS for further power savings. As a result, the problem formulation is significantly different. All the aforementioned studies assume that task execution times are known a priori. While these studies can optimize the system power consumption when execution times do not change dynamically, the optimality is not guaranteed under execution time variations. Although much work on feedback control scheduling exists, to the best of our knowledge, our work is the first one which integrates the utilization control with DPM. Recently, [50] proposed to dynamically partition shared last-level caches of multi-core processors to control the utilization while reduce the power consumption. [50] is complementary to this work and can be integrated to further reduce the power consumption while guarantee real-time.

Cache-Aware Utilization Control for Energy Efficiency in Multi-Core RT Systems

In recent years, scheduling for multi-core real-time systems has received much attention. Many multiprocessor scheduling algorithms (*e.g.*, [43, 103]) can be applied to multi-core processors. Bini et al. [25] proposed two abstractions to facilitate multi-core adoption for real-time systems and the corresponding schedulability analysis.

Nelis et al. [103] studied slack reclamation schemes to reduce the power of a multi-core real-time system. Block et al. [26] proposed an adaptive framework based on feedback which controls each task instead of the utilization of the task system. Seo et al. [119] studied energy efficient multi-core real-time scheduling using a chip-wide DVFS. However, all these studies do not explicitly consider the impact of shared L2 caches.

Several cache-aware multi-core real-time scheduling algorithms have been recently proposed. Anderson's group proposed various open-loop cache-aware global scheduling algorithms for multi-core real-time systems (*e.g.*, [11]). Lakshmanan et al. [79] studied partitioned fixed-priority preemptive scheduling. Bui et al. [31] optimized the impact of cache partitioning on a multi-core real-time system. Guan et al. [59] also studied cache-aware scheduling. Yan et al. [150], Li et al. [84] and Hardy et al. [61] analyzed the impact of a shared L2 instruction cache on WCET estimation for shared L2 cache multi-core systems. Paolieri et al. [106] used L2 cache partitioning to solve the multi-core WCET problem. Suhendra et al. [128] proposed a similar cache partitioning and locking approach. All the aforementioned studies are different from ours because they do not address the power consumption of a shared L2 cache.

Coordinated Temperature and Utilization Control for Distributed Real-Time Embedded Systems

Extensive work has been done to investigate power models of CPUs of computing systems. Power models are closely related to thermal managements because instantaneous temperatures are determined by instantaneous powers according to physic laws of thermal dynamics. Power models have been approached in several ways. [69] proposed a power model capturing architecture features of a processor. [35] observed that the power consumption of a CPU increases linearly with its CPU utilization under a particular workload pattern. [46] proposed a nonlinear power model and a simplified linear power model relating CPU utilizations to powers.

In this dissertation, we try to explicitly control the processor temperatures for DRE systems. Previous research on thermal management focuses mainly on general computer systems. For example, Brooks et al. [30] propose a dynamic thermal management scheme based on heuristics. Skadron et al. [122] present several DTM schemes including a control-theoretic algorithm. Donald et al. [42] develop a control-theoretic thermal management approach for multi-core processors. However, all the aforementioned work cannot provide timeliness guarantees for real-time systems.

Several studies have proposed thermal management algorithms for real-time systems. Bansal et al. [19][20] present online algorithms to solve real-time scheduling problems while guaranteeing thermal constraints. Chen et al. [39][38] design real-time scheduling algorithms with reactive CPU speed assignment and develop several optimization algorithms to minimize the maximum system temperature in a static way. Different from their work that relies on heuristics or optimizations, we propose a coordinated solution based on control theory to provide simultaneous thermal and timeliness guarantees despite various runtime thermal and execution time variations. Most recently, [53] proposed a thermal controller which handles input constraints. They use the linear power model which is similar to the linear power model in [46]. Our thermal controller based on rate adaptations use more accurate nonlinear model to improve the control performance. In addition, the thermal model in [53] can not capture transient process of temperature. One of disadvantages is that inevitable long control period of thermal controller increase chance of thermal failure.

Control-theoretic techniques have been applied to many computing systems. For example, various CPU utilization control algorithms (e.g., [92][93][144][139]) have been recently proposed to guarantee real-time deadlines. However, those algorithms cannot provide thermal guarantees. Recently, coordinated control solutions have been proposed for power/energy management. For example, Raghavendra et al. [110] propose a multi-layer controller for data center power management. Heo et al. [63] study the incompatibilities problems of conflicting control systems and propose a formal methodology to analyze conflicts. Another coordination strategy has been proposed in [142] by forcing different control loops to run on different timescales. In contrast, our solution is designed based on robust control theory to allow the thermal and utilization control loops to run on their respective desired timescales for prompt control actions and simultaneous guarantees.

Power Oversubscription in Data Centers

Recently, the power management issue has attracted a large amount of attention from both academia and industry. For example, Meisner et al. [99] proposed a PowerNap scheme to reduce the server's idle power. Ahmad et al. [8] optimized the idle and cooling power in a data center. However, these studies focus primarily on power minimization instead of power provisioning.

Power provisioning is an important technique for data centers to avoid expensive upgrade costs and to maximize the power infrastructure utilization; thus, it becomes an important, practical issue in data center operation. Fan et al. [46] investigated the workload characteristics of the data center and demonstrate the existence of a great potential for oversubscription in the production data center. Lefurgy et al. [81] proposed a control-theoretic approach to power provisioning and showed the advantages of this method in terms of performance as compared with commercial ad hoc solutions. Pelley et al [108] proposed a novel power router to make the flexible power budget usable. Femal et al [47] investigated how to improve throughoutput given a fixed power budget. Yet, each of these studies still does not answer the question of how much power can be over-subscribed. Govindan et al. [57] adopted statistical profiling-based techniques to power provisioning. They considered the sustainable power budget; however, their soft fuse method is essentially an ad hoc approach.

The control-theoretic approach is a promising adaptation mechanism in power and thermal management. Donald et al.[41] proposed a PI-controller based solution for multicore thermal management. Skadron et al. [120] designed a PID controller approach for accurate and localized dynamic thermal management. Srikantaiah et al [125] adopted a reinforced oscillation resistant controller for shared cache management. Wang et al. [145] designed a model prediction controller. Those studies focus on power and thermal management issues for individual computer systems.

None of them consider the adaptation of control parameters since there is no design constraint on the settling times of controllers in those studies.

Power Capping in Data Centers

Ahmad et al. [8] proposed a hill climbing algorithm to minimize the total power consumption of cooling and servers, however, it assumes fixed CRAC flow rates. In contrast, our work enforces the power budget of a data center and configures the cooling and server power consumption optimally to maximize performance. Huang et al. [66] adjusts CRAC Output Temperature to reduce the power consumption of a data center cooling system. Their assumption is that a higher server fan speed is needed to remove the heat generated by a server since the CRAC Output Temperature increase will increase a server inlet temperature. Their adjustment is very coarse and they choose one CRAC output temperature among two based on the data center utilization which leads to a moderate power reduction. In contrast, our cooling system optimizer achieves more fine-grained control and adjusts CRAC flow rate as well. The inner control loop will control the total power consumption of servers.

Some existing work [8] adopts an ad hoc approach to minimize the cooling power. Specifically, CRACs are chosen to lower their output temperature step-by-step. It is extremely difficult to determine the step size. If the step is selected to be too small, it will take a long time for the proposed solution in [8] to settle. On the other hand, if the the step is chosen to be too big and CRAC output temperatures are throttled too aggressively, the servers may be overheating. In contrast, we formulate the following optimization problem to adjust CRAC flow rate and output temperatures to minimize the cooling power consumption based on the model derived in Subsection 8.2.1.

End-to-End Energy Management of VDI

Many have studied aspects of energy management for pieces of a VDI system, such as networking [78][44], embedded and mobile devices [65][90][143] and data centers [71][52] etc. However, all existing work tackled these separately, as isolated components. An integrated solution for VDI energy management does not exist in industry today. A challenge in extending existing work and applying it to VDI energy management is that they enforce performance (such as CPU utilization [65] or throughput) at the granularity of a server [78][52]. In contrast, a VDI deployment requires integrated management of end-to-end energy and performance.

Chapter 3

Task Consolidation in Multi-Core Real-Time Systems

3.1 System Architecture

In this section, we present our system architecture. As shown in Figure 3.1, our system architecture features a task consolidation manager for the entire multi-core processor and a utilization control loop for each core in the processor.

First, for every core, a utilization controller exists that controls the CPU utilization of the core by scaling the core frequency. The controller is a Single-Input-Single-Output (SISO) controller since the change of core frequency only affects the utilization

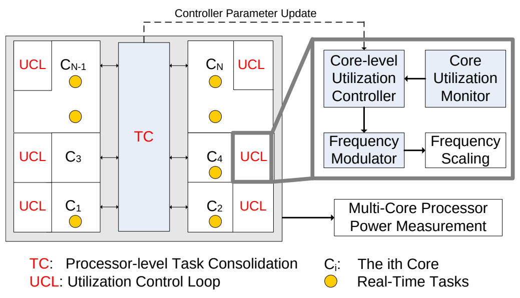

Controller Parameter Update
UCL
C$_{N-1}$
C$_N$
UCL
TC
UCL
C$_3$
C$_4$
UCL
UCL
C$_1$
C$_2$
UCL
Core-level Utilization Controller
Core Utilization Monitor
Frequency Modulator
Frequency Scaling
Multi-Core Processor Power Measurement
TC: Processor-level Task Consolidation
UCL: Utilization Control Loop
C$_i$: The ith Core
● Real-Time Tasks

Figure 3.1: System architecture

of the core. This control loop works as follows: (1) the utilization monitor on each core sends the utilization of the core to the local controller; (2) the controller computes a new CPU frequency and sends it to the frequency modulator on the core; and (3) the frequency modulator then changes the core frequency using DVFS.

Second, the processor-level task consolidation manager dynamically allocates tasks among the cores for task consolidation. It works as follows: (1) the task consolidation manager monitors all the tasks $\{T_i | 1 \leq i \leq m\}$ and measures their CPU utilizations at run-time; (2) the task consolidation manager computes an optimized new task allocation for the cores and sends the task migration requests to the operating system. The OS then redistributes the following releases (i.e., jobs) of the periodic tasks to the cores to enforce the migration of the periodic tasks; and (3) the OS changes the affinity of the tasks to the cores accordingly. The overhead of migrating a task among cores is less than 40 μs which is sufficiently small in the majority of practical real-time systems. The detailed overhead measurement results can be found in [153].

In a real system, similar to the power management unit implemented in POWER7 [146], our control architecture can be implemented in service processor firmware to interact with the main processor and OS. Our solution can also be implemented in the OS as a periodic task with the highest priority. It is important to note that without effective integration, the processor-level task consolidation manager and the core-level utilization control loops may conflict with each other. The task consolidation manager may cause the core-level utilization control loop to be unstable, as it will change the system models used by the utilization controllers. As a result, the utilization control loops need to be configured with the proper controller parameters, according to task migration.

3.2 Core-level Utilization Control

In this section, we model, design, and analyze the core-level utilization control loop.

3.2.1 Task Model

To maximize the throughput of a multi-core system, an application assigned to run on multi-core processors typically consists of multiple tasks running in parallel; thus, we adopt a commonly used independent periodic task model (*e.g.*, in [12]). A system is comprised of m periodic tasks $\{T_i | 1 \leq i \leq m\}$ executing on n cores $\{C_i | 1 \leq i \leq n\}$ in a multi-core processor. Task T_i can be migrated among different cores. A core may host one or more tasks. Each task T_i has a *soft* deadline that is equal to its period. r_i is the inverse of the period of task T_i. A well-known approach for meeting the deadlines on a core is to ensure its CPU utilization remains below its schedulable utilization bound (e.g., Liu and Layland bound for RMS scheduling)[89]. Note that our task model can be extended to support aperiodic tasks by using the corresponding schedulable utilization bound. For example, a utilization bound has been derived for systems with aperiodic tasks in [5]. Task rate adaptation can also be used for utilization control in some real-time systems [93]. We focus on DVFS and task migration for a more general solution since the rates of many real-time tasks cannot be adapted.

Our task model has two important properties. First, while each task T_i has an *estimated* execution time c_i available at design time, a real-time task's *actual* execution time may differ from its estimation and vary at run-time due to two reasons: core frequency scaling by the DVFS and workload uncertainties. Modeling such uncertainties is important to systems operating in unpredictable environments. The estimated execution time can be an approximate estimation and is not necessarily the WCET. Second, the core frequency of each core C_i can be dynamically adjusted on a per-core basis within a range $[F_{min}, F_{max}]$. This assumption is based on the fact that more energy savings can be achieved with per-core DVFS when compared to conventional chip-wide DVFS [74] and many today's microprocessors already support per-core DVFS (e.g., AMD Independent Dynamic Core Technology). Note that our solution does not rely on WCET estimation, which is a key advantage of our solution, because WCETs are often unavailable or mis estimated in real-time embedded systems running in open execution environments. In contrast, a fundamental limitation of open-loop power optimization solutions is that they may fail the optimization goal at runtime when the actual execution times are significantly different from the WCETs used in the optimization.

3.2.2 System Modeling

We first introduce the following notation. T_s, the control period, is selected such that multiple jobs of each task may be released during a control period. The utilization control loop is invoked every T_s seconds. $u_i(k)$ is the utilization of core C_i in the k^{th} control period, i.e., the fraction of time that C_i is not idle during the time interval $[(k-1)T_s, kT_s)$. B_i is the desired utilization set point (*i.e.*, schedulable bound) on C_i. $S_i(k)$ is the set of tasks located on core C_i in the k^{th} control period. $f_i(k)$ is the normalized CPU frequency (*i.e.*, a value relative to the highest level F_{max}) of core C_i in the k^{th} control period.

Following a control-theoretic methodology, we establish a dynamic model that characterizes the relationship between the controlled variable $u_i(k)$ and the manipulated variables $S_i(k)$ and $f_i(k)$. As observed in [116], since the frequencies of real microprocessors can be scaled only within limited ranges, the execution times of computation-intensive tasks on core C_i can be approximately estimated to be proportional to C_i's relative core frequency[*]. Therefore, when core C_i runs at $f_i(k)$, the *estimated* execution time of task T_i on C_i in the k^{th} control period can be modeled

[*]In general, the execution times of tasks which have intensive memory access and I/O operations may include frequency-independent parts that do not scale proportionally with the core frequency [17]. We plan to model frequency-independent parts in our future work.

as $c_i/f_i(k)$. The *estimated* CPU utilization of core C_i can be modeled as

$$b_i(k) = \frac{\sum_{T_j \in S_i(k)} c_j r_j}{f_i(k)} \tag{3.1}$$

We then define the *estimated* utilization change of C_i, $\Delta b_i(k)$, as

$$\Delta b_i(k) = \frac{\sum_{T_j \in S_i(k+1)} c_j r_j}{f_i(k+1)} - \frac{\sum_{T_j \in S_i(k)} c_j r_j}{f_i(k)}. \tag{3.2}$$

Note $\Delta b_i(k)$ is based on the estimated execution time c_j. Since the actual execution times may differ from their estimation due to workload variations, we model the actual utilization of C_i as the following difference equation

$$u_i(k+1) = u_i(k) + g_i \Delta b_i(k) \tag{3.3}$$

where the utilization gain g_i represents the ratio between the change to the actual utilization and its estimation $\Delta b_i(k)$. For example, $g_i = 2$ means that the actual change to utilization is twice the estimated change. Also note that the exact value of g_i is *unknown* at design time due to the unpredictability of the tasks' execution times.

The system models (3.2) and (3.3) show the actual utilization determined by both the frequency and task allocation. Since $S_i(k)$ contains a discrete number of tasks, the system model introduces a significant challenge, which usually requires *hybrid model-predictive control* [94]. In a model-predictive controller, the control problem is translated to a constrained least-squares problem [93]. The hybrid model-predictive control problem is translated to a mixed integer non-linear programming problem (MINLP) and all existing MINLP solvers are not polynomial algorithms.

To address this challenge, we adopt an integrated optimization and control approach. First, we determine the task allocation based on an optimization strategy introduced in Section 3.3. The goal is to minimize the power consumption of the multi-core system. Second, a feedback controller is designed for each core to achieve the desired utilization. Based on our control architecture, the core-level utilization control loop can be designed separately from the task migration optimization strategy. As a result, model (3.3) can be simplified by having $S_i(k)$ in (3.2) as a constant S_i. This avoids designing a controller based on (3.2) to handle discrete changes of the task allocation. As a result, model (3.3) becomes

$$u_i(k+1) = u_i(k) + g_i \Delta d_i(k) \sum_{T_j \in S_i} c_j r_j \tag{3.4}$$

where $\Delta d_i(k) = 1/f_i(k+1) - 1/f_i(k)$. The model cannot be directly used to design the controller because the system gain g_i is used to model the uncertainties in task execution times and is unknown at design time. Therefore, we design the controller based on an approximate system model of (3.4) with $g_i = 1$. In a real system where the task execution times differ from their estimations, the *actual* value of g_i may not equal 1. As a result, the closed-loop system may behave differently. However, we show that a system controlled by a controller designed with $g_i = 1$ can remain stable when the variation of g_i is within a certain range. This range is established using a stability analysis of the closed-loop system by considering model variations.

3.2.3 Controller Design and Analysis

Because of our novel control architecture, the model (3.3) is simplified as the model (3.4), and we can borrow the controller design in [141]. The Z-transform of the P controller [141] is

$$C(z) = \frac{1}{\sum_{T_j \in S_i} c_j r_j}. \tag{3.5}$$

The transfer function of the closed-loop system controlled by controller (3.5) is

$$G(z) = z^{-1}. \tag{3.6}$$

Algorithm 1 rt-MBS

```
q: an index of tasks not assigned to cores
n: the number of tasks not assigned to cores
A: the current assignment
Minimum-Bin-Slack ( q )
begin
1:  for all index $i$ from $q$ to $n$ do
2:      Get $i^{th}$ tasks not assigned to cores;
3:      if $i^{th}$ tasks can be assigned to the core under $A$ then
4:          Add $i^{th}$ tasks into $A$;
5:          Minimum-Bin-Slack($i + 1$);
6:          Remove $i^{th}$ tasks from $A$;
7:          if No free space exists under the current optimal assignment then
8:              Exit;
9:          end if
10:     end if
11:     if $A$ is better than the current optimal assignment then
12:         Set $A$ the current optimal assignment;
13:     end if
14: end for
end
```

It is easy to prove that the controlled system is stable and has zero steady state errors when $g_i = 1$. When the designed P controller is used on a system with $g_i \neq 1$, the system will remain stable when $0 < g_i < 2$, which means that the actual utilization change *cannot* be twice the estimated utilization change. We have also proven that the system can achieve zero steady state error when the system is stable.

3.3 Processor-level Task Consolidation

The design goal of the task consolidation algorithm is to determine a task allocation $S_i(k)$ that can minimize power consumption $P(k)$. Task consolidation and idle core shutdown can lead to more power savings than when simply using DVFS to lower core frequencies because as feature sizes decrease below 65 nm, the leakage power consumption becomes a major contributor to the total power consumption of a processor [73][130]. For example, in 23nm processors, the leakage power consumption accounts for approximately 80% of the total power consumption. The problem for dynamic task consolidation can be transformed to a bin packing problem. Since the bin-packing problem is known to be NP-complete and so an optimal solution is not suitable to be used online in multi-core and many-core systems with many tasks, most existing work focuses on heuristics. Several suboptimal heuristics with different complexities have been proposed. In this work, we evaluate and compare several heuristics in terms of both overhead and solution quality. Note that for real-time embedded systems, run-time overhead is often a more serious concern than solution quality. High run-time overheads may impact the schedulibility of real-time tasks and cause deadline misses.

We test First-Fit, Best-Fit, and an advanced bin packing heuristic rt-MBS based on MBS (Minimum Bin Slack) [48]. To further reduce the overhead of First-Fit, we design *iFF* (Incremental First-Fit). In this section, we will compare the overheads of all heuristics theoretically. We evaluate four different heuristics presented in Section 3.3 in terms of both overhead and solution quality using realistic workloads.

First-Fit places each task, in succession, into the first core into which it fits. Best-Fit places each task, in succession, into the most nearly full core in which it fits. Incremental First-Fit has two arrays to hold the task allocation in the last control period and task allocation in the current control period, respectively. Incremental First-Fit also employ First-Fit to assign each task into a core in every control period. However, in contrast to that First-Fit assign every task into a core by calling a system call, Incremental First-Fit store the assignment of every task into a core in an array instead of calling the system call immediately, then compare the new task allocation with the task allocation in the last control period and only call the system call for the task with changed core affinity. The key observation of iFF is that the order in which tasks are packed into a core is irrelevant. What is important is the total number of core and the total utilizations of each core are the same as First-Fit. The system call overhead is up to 40 microseconds [153]. For every task, First-Fit needs to call the system call once. For a large of number tasks, the overhead may be big.

Incremental First-Fit eliminates those unnecessary system calls and provide the same solution quality as First-Fit.

MBS is bin-focused. In each step, MBS attempts to find a set of tasks (packing) that makes the core as full as possible. Building a packing for each core is implemented recursively. The detailed algorithm of applying MBS to processor-level task consolidation is shown in Algorithm 1. The algorithm is invoked repeatedly until all tasks are assigned. The procedure is invoked with $q = 1$ while the current assignment and current optimal assignment are initialized to be null sets. Note that the allocation in each step is subject to the utilization constraint, which is enforced by line 3 of Algorithm 1. The utilization constraint is checked in each step when a task is allocated to a core to guarantee the real-time executions of tasks.

We now analyze the complexity of the four heuristics. First-Fit and Best-Fit are among the simplest heuristics. MBS, in the worst case, has the same complexity as an exhaustive search. The complexity of Incremental First-Fit, First-Fit, Best-Fit, and MBS is $O(mlogm)$, $O(mlogm)$, $O(mlogm)$, and $O(m^{u+1})$, respectively; where m is the total number of tasks in the system and u is the maximum number of tasks that can be placed in one core. The overhead of Incremental First-Fit is smaller than that of First-Fit because of fewer system calls. The improved time complexity is archived by using two more arrays with space complexity of $O(m)$.

3.4 System Implementation

We first introduce the physical testbed used in our experiments. Next we introduce our simulation environment.

Our testbed is an Intel Xeon X5365 Quad Core processor with an 8MB on-die L2 cache and 1,333 MHz Front Side Bus. The processor supports four DVFS levels: 3GHz, 2.67GHz, 2.33GHz, and 2GHz. According to Intel, the processor has Core 0 and Core 1 fabricated on one die and Core 2 and Core 3 on a separate die. We must change the DVFS levels of the 2 cores on each die in order to have a real impact on the processor power consumption. Therefore, we use this processor to emulate a dual-core processor that supports a per-core DVFS. The operating system is a Fedora Core 7 with a Linux kernel 2.6.23 and a real-time-preempt kernel patch.

The default Linux kernel may migrate real-time tasks by itself, which can cause deadline misses as the core utilizations are not guaranteed by the kernel during migration. To disable task migration from the Linux kernel, we use a standard system call SCHED_SETAFFINITY [27], which is a portable approach across different platforms. The overhead of the system call for task migration among cores is less than 40 μs which is acceptable in many real-time embedded systems. The detailed overhead results are in [153].

We adopt the Mibench benchmarks [60] designed for embedded systems as our tasks. Our experiments run a medium-sized workload comprised of 10 tasks to run the Mibench benchmarks. Both cores initially have five periodic tasks with a total utilization of 0.31. The task parameters such as periods are configured according to a real real-time application [4]. The tasks on each core are scheduled by the RMS algorithm [89]. Note that our solution can also be used with other scheduling approaches, such as EDF, as long as the corresponding schedulable utilization bound is adopted. We use RMS as an example in this dissertation because RMS usually has a smaller runtime overhead in real systems. The deadline of each task T_i equals its period, $1/r_i$. The utilization set point of every core is set to its RMS schedulable utilization bound [89], i.e., $B_i = m(2^{1/m} - 1)$, where m is the number of tasks on C_i. Since the number of tasks may change according to the processor-level task consolidation, the set point can be set to 0.69 which is the limit of $B_i = m(2^{1/m} - 1)$ when $m \to \infty$. All tasks meet their deadlines if the desired utilization on every core is enforced.

We now introduce the implementation details of each component in our system architecture.

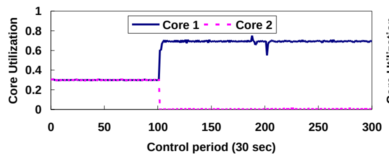
Utilization Monitor: The utilization monitor uses the /proc/stat file in Linux to estimate the core utilization in each sampling period. The /proc/stat file records the number of jiffies (usually 1ms - 10ms in Linux) when a core is in user mode, user mode with low priority (nice), system mode, and when used by the idle task, since the system starts. The utilization of each task can be calculated based on the number of jiffies consumed by the task process in each control period.

Core-level Utilization Controller: The controller is implemented as a single-thread process with the highest priority running on each core. With a control period of 30 second, the controller periodically reads the core utilization, executes the control algorithm presented in Section 3.2.3 to compute the desired core frequency, and sends the new frequency to the frequency modulator on the core.

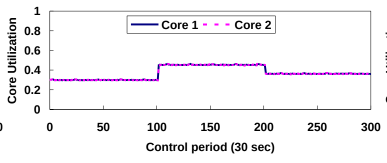
Frequency Modulator: We use Intel's Enhanced SpeedStep Technology to enforce the new CPU frequency. To the change core frequency, one needs to install the *cpufreq* package and then use the root privilege to write the new frequency level into the system file. A routine periodically checks this file and resets the core frequency accordingly. The average overhead (*i.e.*, transition latency) to change the frequency in Intel processors is approximately $100\mu s$.

Power Monitor: To measure the power consumption of the processor, an Agilent 34410A digital multimeter (DMM) is used with a Fluke i410 current probe to measure the current running through the 12V power lines that power the processor. The probe is clamped to the 12V lines and produces a voltage signal proportional to the current running through the lines with a coefficient of 1mv/A. The resultant voltage signal is then measured with the multi-meter. The accuracy of the probe is 1.5% of reading + 0.5A.

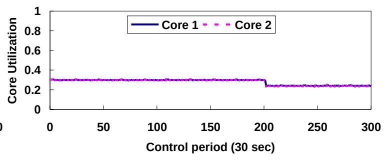
3.5 Evaluation



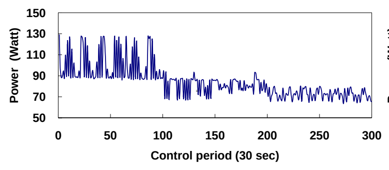
(a) Core utilization (Proposed)



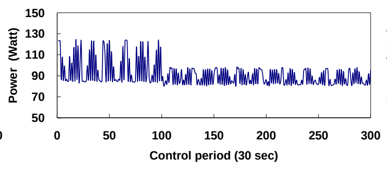
(b) Core utilization (DVFS-Only)



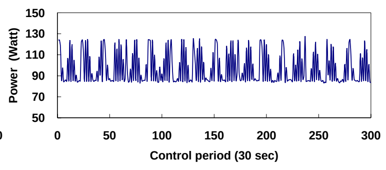
(c) Core utilization (No-Power-Management)



(d) Processor power consumption (Proposed)



(e) Processor power consumption (DVFS-Only)



(f) Processor power (No-Power-Management)

Figure 3.2: Typical runs of three solutions (Proposed, DVFS-Only, and No-Power-Management) on the hardware testbed. The solutions are activated at the 100th control period and handle a 20% execution time reduction at the 200th control period.

In this section, we first compare four heuristics in Section 3.3, then present our empirical results conducted on the hardware multi-core testbed.

Baselines

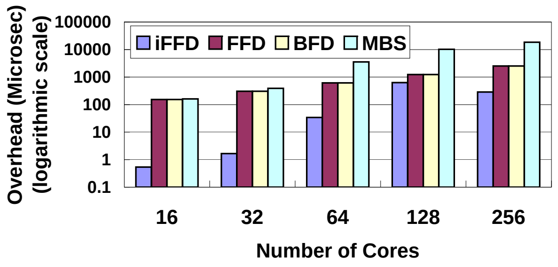
We use three baselines for comparison in this dissertation. *Dynamic core scaling* is a state-of-the-art algorithm [119], which adjusts both the core frequencies and number of active cores of a multi-core system to reduce the dynamic and leakage power consumption by task migration. The fundamental difference between Dynamic core scaling and the proposed solution is that the Dynamic core scaling makes a task migration decision based on the WCET of the task to be migrated. For systems operating in unpredictable environments, to guarantee the timeliness, the WCETs have to be conservative. The actual execution time of the task to be migrated is usually much smaller than the overestimated WCET. In contrast, the proposed solution makes a task migration decision based on the average CPU utilization, which can be easily monitored at runtime in a lightweight way. In addition, Dynamic core scaling uses a chip-wide DVFS while the proposed solution uses a per-core DVFS, which is already supported by many microprocessors. We demonstrate in Section 3.5 that the proposed solution outperforms Dynamic core scaling significantly in terms of power savings. The second baseline, *DVFS-Only*, is the frequency scaling loop proposed in [141]. It relies only on DVFS to throttle the core frequency to manage the power consumption of a core, subject to the utilization constraints without turning off any cores. DVFS-Only has a similar utilization controller design with the proposed solution, but does not perform task consolidation. *No-Power-Management* is a classical open-loop scheduling solution that partitions the tasks in a static way [89] and the frequencies of all cores in a multi-core system are fixed to the maximum frequency level. While No-Power-Management can initially guarantee the timeliness, it may fail when task execution times change at runtime and waste energy when the system is underutilized.

Comparison of Different Heuristics

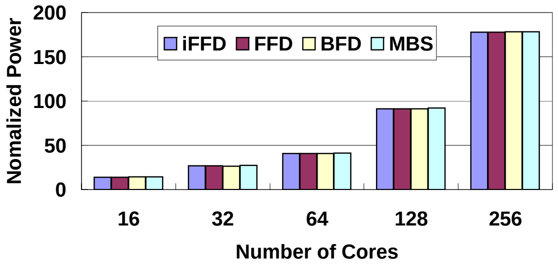
A scalability requirement for a multi-core or many-core power optimization heuristic is low run-time overhead. In this experiment, We evaluate four different heuristics presented in Section 3.3 in terms of both overhead and performance by simulations. Different workloads including 16 to 256 tasks are randomly generated to stress test all heuristics. To estimate the overhead of the heuristics, we measure the execution time of each heuristic on a 2.5-GHz Intel Core 2 Duo PC with 2-GByte RAM. To obtain high-resolution measurement, we use Windows API *QueryPerformanceCounter*. We collect the average of multiple runs. As shown in Figure 3.3a, the overhead of incremental First-Fit is smallest, while the overhead of MBS is significantly higher than the others. Figure 3.3b shows that under realistic workloads, the processor power consumption under all heuristics is very close. According to the simulations, we adopt incremental First-Fit for online power reduction in the following experiments because of its low overhead.

Empirical Results on Hardware Testbed

In this experiment, we first disable the proposed solution from the 1st to the 100th control period. As shown in Figure 3.2a, the initial utilizations of Core 1 and



(a) The overhead of four optimization heuristics

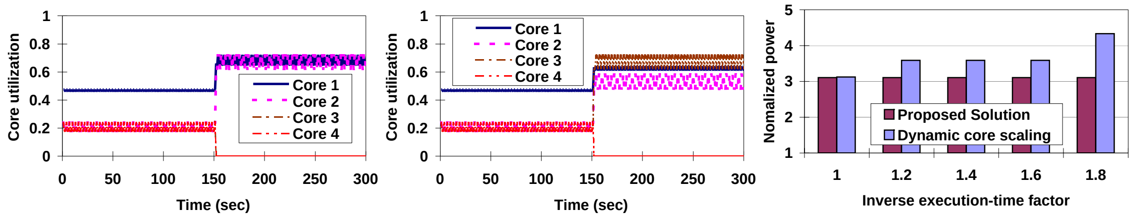


(b) Power consumptions using four optimization heuristics

Figure 3.3: Comparison among the three heuristics.

Core 2 are both 0.31. Core utilizations are lower than the RMS bound, resulting in an undesired underutilized system. We then enable the proposed solution at the 100th control period. As shown in Figure 3.2a, all tasks on Core 2 are migrated to Core 1. Core 2 becomes idle and is then turned off. As shown in Figure 3.2d, the power consumption of the CPU is consequently reduced by approximately 19%. As shown in Figure 3.2a, at the 200th control period, the execution time of all the tasks is suddenly decreased by 20%, resulting in a sharp drop of the utilization of Core 1. This decrease is implemented by reducing the number of loop iterations in the Mibench benchmarks. The proposed solution responds to the utilization drop by dynamically decreasing the core frequency of the core. Since the settling time of the utilization controller is just several control periods, the utilization converges quickly to the RMS bound again. As shown in Figure 3.2d, the power consumption of the CPU is further reduced by approximately 11%. The experiment demonstrates the effectiveness of the proposed solution with uncertain task execution times.

We then examine the power efficiency of two baselines: DVFS-Only and No-Power-Management. To make a fair comparison, we adopt the same workload and scenario used for the proposed solution. For DVFS-Only, Figure 3.2b shows that at the 100 control period the utilization of all the cores increases because DVFS-Only throttles the frequencies of both cores to the lowest levels. As a result, Figure 3.2e shows the processor power drops at the 100th control period. However, the power consumption is still much higher than that of the proposed solution. The reason is that DVFS-Only cannot consolidate tasks to reduce the leakage power consumption of the processor. At the 200th control period, even though the execution times of all the tasks are decreased by 20%, DVFS-Only can only achieve very slightly further power savings because both the cores are already at their lowest frequencies. This experiment demonstrates the necessity of task consolidation. For No-Power-Management, as shown in Figure 3.2c, at the 200th control period, the execution times of all tasks are decreased by 20%. Since No-Power-Management does not decrease the core frequencies in response to the lower workload, Figure 3.2f shows that the processor power is only slightly reduced and is much higher than that of the proposed solution. Since all the three solutions do not violate the RMS schedulable utilization bounds in their entire runs, no deadline miss is observed in this experiment for any of the solutions.



(a) Proposed solution (activated at the 150s) (b) Dynamic core scaling (activated at 150s) (c) Power consumption under different ietf

Figure 3.4: Comparison between the proposed solution and Dynamic core scaling.

Simulation Results

In this section, we first compare the proposed solution with Dynamic core scaling on a quad-core system. We then test the effectiveness of the proposed solution in many-core systems. We have also performed the evaluation of the proposed solution in heterogeneous multi-core systems. The results are not presented due to space limitations but can be found in [51].

Comparison with Dynamic Core Scaling

In this section, we compare the power efficiency of the proposed solution and Dynamic core scaling in unpredictable environments. The WCETs of tasks often have to be conservative in unpredictable environments as the actual execution time may vary across in a wide range at runtime. The majority of the time, the actual execution times can be much smaller than the pessimistic WCETs. The *inverse execution-time factor (ietf)* denotes the ratio of the estimated execution time to the actual execution time of a periodic task. The greater the *ietf* is, the more conservative the estimated execution time of a task. For Dynamic core scaling, the *ietf* can be determined by the predictability of the environment.

In the first experiment, we randomly generate a small scale task set including 5 tasks in a quad-core system. The *ietf* of the tasks is 1.5. Figure 3.4a shows that all tasks are consolidated onto two cores (Cores 1 and 2) under the proposed solution. In contrast, Figure 3.4b shows all tasks are consolidated onto *three* cores (Cores 1 to 3). The reason is that Dynamic core scaling relies on WCETs to decide whether or not it migrates a task. Because of the overestimated WCETs (*i.e.*, ietf=1.5), Dynamic core scaling may prevent task migrations. Dynamic core scaling cannot make task migration decisions based on actual execution times. Otherwise, schedulable bounds may be violated after migrations and deadline misses occur. In contrast, the proposed solution relies on the feedback of the average task CPU utilizations and so tasks can be consolidated onto fewer cores. Note that when the actual execution time of a task approaches the WCET, the proposed solution can still guarantee the timeliness by dynamically enforcing the schedulable utilization bounds. Figure 3.4b also shows that only Core 1 reaches the utilization bound under Dynamic core scaling due to its assumption of chip-wide DVFS. If the workload is not perfectly balanced, which is common in a real system, chip-wide DVFS cannot allow all cores to reach the RMS bound at the same time, resulting in undesired underutilized systems and unnecessarily more power consumption. In this experiment, after the activation at

150s, the normalized power consumption of the proposed solution is reduced from 8.01 to 3.106, while that of Dynamic core scaling is reduced from 8.01 to 3.587. Dynamic core scaling consumes about 15.5% more power than the proposed solution. The reason is that the proposed solution can consolidate tasks to reduce leakage power and utilize per-core DVFS to save more dynamic power.

We then compare the normalized power consumption of the proposed solution and Dynamic core scaling when the *ietf* varies from 1 to 1.8. Figure 3.4c shows when the ietf is 1, which means the the actual execution times are equal to the WCETs, the normalized power consumption of Dynamic core scaling is approximately the same as that of the proposed solution. The slight difference is because Dynamic core scaling does not utilize per-core DVFS. When the ietf increases from 1 to 1.2, the normalized power consumption of Dynamic core scaling increases to approximately 15.5% more than that of the proposed solution. The reason is that when the ietf is 1, both the solutions consolidate tasks onto two cores. When the task WCETs increase to 1.2 times of the actual execution times (*i.e.*, ietf=1.2), Dynamic core scaling uses three core while the proposed solution still uses only two. When the ietf increases from 1.2 to 1.6, the difference between the two solutions only changes marginally because Dynamic core scaling still utilizes three cores in this case. However, when the ietf further increases to 1.8, Dynamic core scaling begins to use all four cores. As a result, it consumes 39.6% more power than the proposed solution. Since both the proposed solution and Dynamic core scaling can enforce the CPU utilization dynamically on each core, the two solutions both achieve a zero deadline miss ratio in all runs. This experiment demonstrates that the proposed solution significantly improves the power efficiency of real-time systems in unpredictable environments.

Chapter 4

Cache Partitioning in Multi-Core Real-Time Systems

4.1 Problem Formulation

In this section, we formulate the cache-aware utilization control problem for multi-core real-time systems.

4.1.1 Task Model

A multi-core real-time system is comprised of n cores $\{C_i | 1 \leq i \leq n\}$ and m_i periodic tasks $\{T_{ij} | 1 \leq j \leq m_i\}$ executing on C_i. Each task T_{ij} has a *soft* deadline related to its period. We use partitioned scheduling to assign tasks to the cores in a multi-core processor. The tasks on each core are scheduled with rate-monotonic scheduling (RMS). Partitioning-based RMS transforms the multi-core real-time scheduling problem into the uniprocessor scheduling problem. A well-known approach to meeting task deadlines on a core is to keep the core utilization below its schedulable utilization bound (e.g., Liu and Layland bound for RMS) [89]. A more precise schedulability test (e.g., the hyperbolic bound [23]) can be used to improve schedulability. Previous studies [5] also show that the Liu and Layland bound can be replaced with the corresponding schedulable utilization bound to ensure timeliness for systems with aperiodic tasks.

Our task model has three important properties. First, while each task T_{ij} has an *estimated* execution time c_{ij} available at design time. Second, the L2 caches can be partitioned among the cores. The partition size for each core C_i may be dynamically adjusted. Third, the CPU frequency of each core C_i may be dynamically adjusted within a range $[F_{min,i}, F_{max,i}]$

4.1.2 Problem Formulation

Cache-aware power management for multi-core real-time systems can be formulated as a dynamic constrained optimization problem. We first introduce some notation. T_s, the control period, is selected so that multiple instances of each task are released during a control period. $u_i(k)$ is the utilization of core C_i in the k^{th} control period, i.e., the fraction of time that C_i is not idle during time interval $[(k-1)T_s, kT_s)$. $u_i(k)$ is calculated according to the statistics generated by the operating systems. U_i is the desired utilization set point of C_i. $p(k)$ is the power consumption of the processor and related to both the core frequencies and active L2 cache size. $E(k)$ is the energy consumption of the processor in the k^{th} control period. Since the core frequencies, active L2 cache size, and workload of the processor are all not changed during a control period, $p(k)$ can be approximated as a constant within each control period. Consequently, $E(k) = p(k)T_s$. We assume that the processor has homogeneous cores with two levels of caches and the L2 caches are shared among the cores since mainstream multi-core processors adopt this architecture. We also assume that the processor supports per-core DVFS as per-core DVFS leads to a better processor energy efficiency than a chip-wide DVFS [74]. We further assume the cache can be partitioned among tasks. The details of dynamic cache partitioning is beyond the scope of this dissertation because various ways (e.g., software or hardware) have already been designed to implement cache partitioning among tasks. Examples can be found in [86][106][76][59][31]. $s_i(k)$ is the L2 cache partition size of core C_i. $f_i(k)$

is the relative core frequency (*i.e.*, the core frequency relative to the highest level $F_{max,i}$) of core C_i.

Given a utilization set-point vector, $\mathbf{U} = [U_1 \ldots U_n]^T$, a frequency constraint $[F_{min,i}, F_{max,i}]$ for each core C_i, and the total L2 cache size S for the processor, the control goal at the k^{th} sampling point (time kT_s) is to dynamically choose the cache partition size $\{s_i(k)|1 \leq i \leq n\}$ and core frequency $\{f_i(k)|1 \leq i \leq n\}$ to minimize the difference between U_i and $u_i(k)$ for all the cores and to minimize the energy consumption $E(k)$ for the processor.

$$\min_{s_i(k)|1\leq i\leq n, f_i(k)|1\leq i\leq n} \sum_{i=1}^{n} [U_i - u_i(k)]^2 \tag{4.1}$$

$$\min_{s_i(k)|1\leq i\leq n, f_i(k)|1\leq i\leq n} E(k) \tag{4.2}$$

subject to

$$F_{min,i} \leq f_i(k) \leq F_{max,i} \qquad (1 \leq i \leq n) \tag{4.3}$$

$$\sum_{i=1}^{n} s_i(k) \leq S \tag{4.4}$$

Note that the objective (4.2) is actually equivalent to the minimization of power consumption because the power consumption during a control period can be approximated as a constant and thus $E(k) = p(k)T_s$. Constraint (4.3) guarantees that the CPU frequency of each core remains within its acceptable range. The frequency range depends on specific processors. The above formulation can be extended to add equality constraints among cores that have the same frequency (and voltage). Constraint (4.4) ensures that the summed size of all the cache partitions does not exceed the total available cache size on the processor. For each core, the optimization formulation minimizes the difference between the core utilization and corresponding set point by manipulating both partition size and core frequency while satisfying the constraints. Control goal (4.1) actually may conflict with control goal (4.2) because core frequencies throttled to the lowest levels and cache lines turned off are desired to minimize the total power consumption $p(k)$. In that case, memory accesses would be very slow because most accesses will face cache misses and non-memory access instructions would be executed at the lowest speed. Consequently, the task execution times would be too long and core utilizations might exceed set points, leading to deadline misses. Therefore, the two conflicting goals require resolution with advanced control and optimization techniques.

4.2 Cache-Aware Utilization Control

In this section, we model the cache-aware utilization control problem for energy efficiency in multi-core real-time system and present our two-level control architecture.

4.2.1 System Modeling

Following a control-theoretic methodology, we establish a dynamic model that characterizes the relationship between the controlled variable $u_i(k)$ and manipulated variables $s_i(k)$, and $f_i(k)$ in the k^{th} control period, by system identification. First, we model the relationship between $c_{ij}(k)$, the execution time of task T_{ij} running on core C_i, and the two manipulated variables, $f_i(k)$ and $s_i(k)$. According to previous research [24], $c_{ij}(k)$ normally consists of frequency-dependent and frequency-independent portions

$$c_{ij}(k) = \frac{n_{ij}}{f_i(k)} + m_{ij}(k) \tag{4.5}$$

where $\frac{n_{ij}}{f_i(k)}$ is the frequency-dependent portion and $m_{ij}(k)$ is the frequency-independent portion of T_{ij}'s execution time. The former scales with the core frequency but the latter does not because some instructions deal with memory or other I/O devices and their access speeds do not depend on core frequency. For processors whose FSB (front-side bus) speed varies with DVFS, memory accesses delay can be modeled as the frequency-dependent portion of the task execution time. We assume that the data and program of real-time tasks are loaded into main memory. Disk or I/O device accesses are not required during the execution. The assumption is valid for the majority of embedded real-time systems as the memory footprints of those applications are typically small. Intuitively, $m_{ij}(k)$ is related to the cache size reserved for T_{ij} because of the strong correlation between the cache size of an application and the number of cache misses it has. According to [31], the relationship between $m_{ij}(k)$ and $s_{ij}(k)$, the cache size allocated to T_{ij} on C_i, is modeled as

$$m_{ij}(k) = \begin{cases} A_{ij}\mathrm{s}_{ij}(k) + B_{ij} & 0 \le \mathrm{s}_{ij}(k) \le W_{ij} \\ \text{Constant} & \mathrm{s}_{ij}(k) > W_{ij} \end{cases} \tag{4.6}$$

where W_{ij} is the working set size (WSS) of task T_{ij}. A_{ij} and B_{ij} are task-specific parameters. All the parameters can be estimated using existing task profiling techniques. Example parameters for the benchmarks used in our experiments are listed in Table I in Section V. When $\mathrm{s}_{ij}(k)$ is smaller than the WSS W_{ij}, increasing the cache size of a task may lead to a reduced execution time [31]. When the allocated cache size is greater than the WSS, allocating additional cache to a task does not further decrease its cache miss rate. Although model (4.6) is an approximation of the real system, our experiments show that the linear relationship is sufficiently accurate for the benchmarks. When a workload is different from the benchmarks, it can be proved that the proposed solution still achieves the control goal if the execution time varies within a specific range.

For preemptive real-time task systems, we can establish the following relationship between the total frequency-independent execution time of all the tasks on core C_i and the total cache size $s_i(k)$ assigned to C_i

$$m_i(k) = \begin{cases} \sum\limits_j A'_{ij} s_i(k) + \sum\limits_j B_{ij} & 0 \le s_i(k) \le W_i \\ \text{Constant} & s_i(k) > W_i \end{cases} \tag{4.7}$$

where $A'_{ij} = \frac{A_{ij} s_{ij}(k)}{s_i(k)}$ and $W_i = \sum\limits_j W_{ij}$. (4.7) is derived by a sum of (4.6) across all the tasks on core C_i. We assume that each task has its own cache partition. Note that we do not need to reserve caches for every task on each core and divide $s_i(k)$ proportionally. In that case, the overhead that occurs because the cache content can be invalidated by preempting tasks is taken into consideration. It depends on the maximum number of times the task is preempted and the cache size the task is using. So, we use the WCET model from [31] and derive a model similar to (4.7).

In multi-core systems, tasks on different cores may compete and interfere with each other for shared resource (e.g., shared bus or caches) access. To avoid these interferences, we adopt the cache partitioning method proposed in multiple studies (*e.g.*, [86]). The multi-core cache architecture in [86] simplifies the WCET analysis of a real-time multi-core system. Without cache partitioning, unpredictable inter-core interferences may occur and invalidate model (4.7). Based on this architecture, a multi-core processor with shared L2 caches can be regarded as a multiprocessor system with each processor having adjustable private L2 caches. Considering (4.5), (4.6), and (4.7), we derive the following model for our system

$$b_i(k) = \frac{\sum\limits_j n_{ij} r_{ij}}{f_i(k)} + \sum_j A'_{ij} r_{ij} s_i(k) + \sum_j B_{ij} r_{ij} \tag{4.8}$$

where $b_i(k)$ is the estimated utilization of core C_i and r_{ij} is the task rate of T_{ij} running on that core. An important observation is that system model (4.8) needs to be transformed as $b_i(k)$ to be inversely related to the core frequency $f_i(k)$. From system model (4.8), the estimated change of utilization, $\Delta b_i(k)$, for core C_i is modeled as

$$\Delta b_i(k) = d_i(k) \sum_j n_{ij} r_{ij} + \Delta s_i(k) \sum_j A'_{ij} r_{ij} \tag{4.9}$$

where $d_i(k) = \frac{1}{f_i(k)} - \frac{1}{f_i(k-1)}$ and $\Delta s_i(k) = s_i(k) - s_i(k-1)$. Now $\Delta b_i(k)$ is a linear function of $d_i(k)$ and $\Delta s_i(k)$, which allows us to use $d_i(k)$ as the manipulated variable instead of using $f_i(k)$ directly. Note that $\Delta b_i(k)$ depends on the estimated values of n_{ij} and A'_{ij}. Their actual values may be different from the estimations due to workload variations. A major contribution of our work is to propose a control solution to handle this uncertainty.

The system model (4.9) represents a Multi-Input-Single-Output (MISO) system because it has two manipulated variables, $d_i(k)$ and $\Delta s_i(k)$, and one controlled variable. Two manipulated variables can provide extra flexibility for controlling both CPU-intensive and memory-bound tasks when compared with controlling the same tasks with only one manipulated variable. The additional input variable has a significant implication on the control solution design. We can achieve a certain output with an infinite number of combinations of these two inputs, but not all of them can satisfy the utilization control and power optimization goals. Therefore, we need to determine which combination to use to fulfill our goals. The details are discussed in Section 4.3.

From the system perspective, in multi-core environments that allow both DVFS and cache partitioning/resizing, relying solely on one adaptation strategy may unnecessarily reduce the system's adaptation capability. Adapting one of them can only adjust either the frequency-dependent or independent portion of the task execution time within a range, but not both. Therefore, a novel control architecture needs to be designed for utilization control and power management in multi-core real-time systems by utilizing both adaptation strategies.

4.2.2 Control Architecture

We propose a novel two-level utilization control and power management architecture. As shown in Figure 4.1, our control architecture features a core-level utilization controller and processor-level cache demand arbitrator. As described in Section 4.1, constraint (4.4) enforces that the summation of $s_i(k)$ should not exceed the total processor cache size. Therefore, if the partition size of a core is increased, the cache sizes of other cores may need to be reduced. Moreover, the utilization of a core is related to its cache partition size according to system model (4.9).

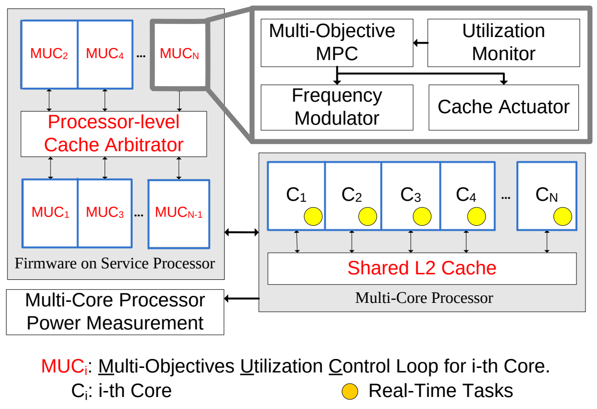


Figure 4.1: Two-level utilization control architecture.

The coupling between the cache size and core frequency for utilization control raises new design challenges. Instead of designing a single processor-level utilization controller, we adopt the two-level utilization control architecture based on the following considerations.

First, a processor-level utilization controller may not scale well in future many-core systems (*i.e.*, systems with tens or hundreds of cores), because the number of variables in the system model of the processor-level utilization controller increases proportionally with the number of cores. As a result, the computational complexity of the controller can increase significantly and thus be too expensive to control real-time systems. In addition, whenever the number of cores changes, the system model changes and the controller needs to be redesigned. Core-level controllers have better scalability because the number of controlled and manipulated variables does not increase with the number of cores. However, as a core-level controller determines its own cache partition size locally and is unaware of other core's cache demands, it can not guarantee constraint (4.4). Therefore, a processor-level cache demand arbitrator is needed to enforce the constraint by assigning a cache quota $s_{quota,i}$ to each core. The core-level local controller maintains its cache partition size below the cache quota $s_{quota,i}$ assigned by the arbitrator.

Second, as the feature size is shrinking to the nanometer scale, leakage power becomes the dominant portion of the total power consumption of the entire processor. The leakage power of a processor contains leakage power for both the cores and caches. In this dissertation, we reduce the cache leakage power by resizing L2 caches at runtime to turn off unused portions. Our solution can also be integrated with existing task migration policies to migrate real-time tasks among the cores and turn off idle cores to reduce the core leakage power. Note that task migration is complementary to our solution and that detailed integration is beyond the scope of this proposal. Task migration for power efficiency can be better supported with a core-level utilization controller than with a processor-level utilization controller. The reason is that the number of active cores may change at runtime and the system model of the processor-level MPC controller needs to be rebuilt whenever the number of the active cores changes. This may incur a large overhead to the system.

Our two-level utilization control architecture works as follows. First, the processor-level cache demand arbitrator dynamically calculates a cache quota for every core based on the real-time workloads running on them. It monitors the task arrival, termination, and migration events, periodically, to collect the cache demand of every core. The core-level utilization controller uses this cache quota to enforce the constraint (4.4). Second, each core-level controller controls the utilization of the corresponding core by scaling its frequency and resizing its cache partition. It is a MISO controller that adopts advanced MPC theory to serve this multi-objectivity: utilization control and power optimization. The core-level controller executes the following steps at the end of every control period: (1) It collects the core utilization from the utilization monitor on core C_i; (2) The controller then computes a new core frequency $f_i(k)$ and a new cache partition size $s_i(k)$, then sends the values to the frequency modulator and cache actuator on C_i, respectively; and (3) The frequency modulator and cache actuator change the core frequency and cache partition size accordingly. In a real system, similar to the power management unit implemented in

POWER7, our control architecture can be implemented in service processor firmware that manages the controlled multi-core processor.

4.3 MOMPC Controller Design

In this section, we present the formulation of the MOMPC controller and discuss the controller design in detail.

4.3.1 MOMPC Control

Based on system model (4.9), a novel MISO controller needs to be designed to enforce the utilization set points on all the cores and minimize power consumption of the processor simultaneously. Traditional MPC control theory applied in earlier studies on feedback control real-time scheduling (*e.g.*, [93]) is not suitable for the problem we formulate in Section 4.1. The reason is that traditional MPC theory can not handle multiple control goals like the two we have in our problem. To solve our control problem, we adopt a recent advance in control theory, Multi-Objective Model Predictive Control (MOMPC) [96], which is being actively studied in the control community [22]. One of the advantages of MOMPC is its capability of dealing with multi-objective MIMO control problems with constraints on the plant and actuators. This characteristic makes MOMPC suitable for our problem.

The basic idea of MOMPC control is to solve a hierarchy of optimization problems. Specifically, multiple objectives are ranked according to their priorities since they may conflict with each other and cannot be met simultaneously. In MOMPC control, the most important objective is solved first. The solution is then used to impose equality constraints when addressing the second optimization objective, and so on. Since meeting the real-time constraints is always the first priority in real-time systems, we select objective (4.1) as our primary control goal and objective (4.2) as our secondary goal. To meet the two control goals, we have a *primary* optimizer and a *secondary* optimizer. The primary optimizer is essentially a dynamic least square optimizer designed to meet the control goal (4.1), just like the optimizer in traditional MPC theory. Its control objective is to select a combination of core frequency $f_i(k)$ and cache partition size $s_i(k)$ that achieves *only* the control goal (4.1). When the system is controlled into the stable state, the secondary optimizer adjusts the core frequency $f_i(k)$ and cache size $s_i(k)$ to achieve the control goal (4.2), *i.e.*, minimizing the power consumption of the processor. To avoid conflicting with the primary optimizer, the secondary optimizer enforces an equality constraint to adjust the core frequency $f_i(k)$ and cache size $s_i(k)$, without impacting the core utilization $u_i(k)$.

4.3.2 Primary Optimizer

Following MOMPC control theory, we first design a controller for the primary optimizer to achieve the control goal (4.1). The controller employs system model (4.9) to minimize a cost function with constraints. The cost function to be minimized by the controller for core C_i is

$$\begin{aligned} V_i(k) = \textstyle\sum_{l=1}^{P} \|u_i(k+l-1|k) - ref_i(k+l-1|k)\|^2 \\ +\|\mathbf{x_i}(k|k) - \mathbf{x_i}(k-1|k)\|^2 \end{aligned} \tag{4.10}$$

subject to:

$$\begin{gathered} F_{\min,i} \leq f_i(k) \leq F_{\max,i} \\ s_i(k) \leq s_{quota,i} \end{gathered} \tag{4.11}$$

where $\mathbf{x_i}(k) = \begin{bmatrix} d_i(k) \\ \Delta s_i(k) \end{bmatrix}$. P is the prediction horizon used to predict the system behavior over P control periods, $P = 2$ in our system. $ref_i(k+l|k)$ is the reference trajectory along which the utilization vector $u_i(k+l|k)$ should change from the current utilization $u_i(k)$ to the utilization set point U_i. Note that the cache size $s_i(k)$ for C_i is bounded by $s_{quota,i}$ to ensure constraint (4.4). We can easily transform the above optimization problem into a standard constrained least-square problem that can be solved by the controller using any standard least square solver. The transformation is not presented due to space limitations, but the detailed steps can be found in [96]. Although the outputs of the primary optimizer are unique, the outputs may not be optimal in terms of energy efficiency. As explained in Section 4.2.1, multiple combinations of core frequencies and cache sizes including the outputs of the primary optimizer can satisfy the utilization set point.

4.3.3 Secondary Optimizer

The secondary optimizer uses a power model to achieve the desired control goal (4.2), *i.e.*, minimizing the power consumption of the processor. The power optimization function that we have designed for our secondary optimizer is

$$p_i(k) = M_i f_i(k)^3 + N_i s_i(k) + L_i \tag{4.12}$$

subject to

$$\begin{gathered} F_{\min,i} \leq f_i(k) \leq F_{\max,i} \\ s_i(k) \leq s_{quota,i} \end{gathered} \tag{4.13}$$

where M_i, N_i, and L_i are the power model parameters of the processor. The power consumption of the processor includes the power consumed by the cores and caches. The former has a dynamic power component $M_i f_i(k)^3$ that varies with core frequency and a leakage power component L_i, but for the latter, the dynamic power component is negligible when compared with the leakage power component [100]. Thus, the cache power consumption is approximated by $N_i s_i(k)$ which varies with the cache partition size of C_i. The power model parameters in (4.12) can be a function of processor temperature, which can significantly impact the leakage power.

The secondary optimizer finds a combination of $f_i(k)$ and $s_i(k)$ that minimizes (4.12) while satisfying the constraints of (4.13). As previously discussed, the equality constraint is imposed so that adjusting core frequency $f_i(k)$ and cache size $s_i(k)$ does not change core utilization $u_i(k)$ achieved by the primary optimizer. As both $u_i(k)$ and $p_i(k)$ are functions of $f_i(k)$ and $s_i(k)$, we can establish a relationship between them and easily impose the equality constraints. We can transform the above formulation into a standard nonlinear optimization problem with constraints and solve it using any standard solver. The detailed transformation is not presented due to page limitations. In our simulator, we implemented the secondary optimizer based on a Matlab solver. The solver can find the optimal solution with a time complexity of $O(n^3)$.

We configure the control period of the secondary optimizer to be 50 times the control period of the primary optimizer. We have proven that the configuration guarantees the stability of the proposed control solution. The detailed proof is not included due to space limitations.

4.4 Simulation Environment

Our simulation environment integrates the event-driven EUCON simulator (for real-time task scheduling) used in previous studies [93] and a multi-core cache partitioning system implemented by following the cache implementation of the cycle-accurate SESC simulator [115], which is widely used in computer architecture research. The multi-core processor simulated in our work is an Intel Xeon X5365 Quad Core processor with an 8MB on-die shared L2 cache and 1333 MHz FSB. The processor supports four DVFS levels: 3GHz, 2.67GHz, 2.33GHz, and 2GHz. All the parameters in our power and utilization models are based on the data sheet from Intel or profiling experiments conducted on the real processor. We have validated our models under different DVFS levels and cache partition sizes with the real Intel processor and original SESC simulator, respectively. The validations show that our models are sufficiently accurate (with $R^2 \geq 0.93$) for the well-known *Mibench* [60] benchmark suites designed for embedded systems. We only list the result of the first category benchmarks of MiBench suite among all the six categories because other categories are not designed to test real-time systems. [104]. Table I lists the benchmarks used in our experiments and the corresponding parameters used in model (4.6). The unit for the working set size (WSS) is the number of cache lines.

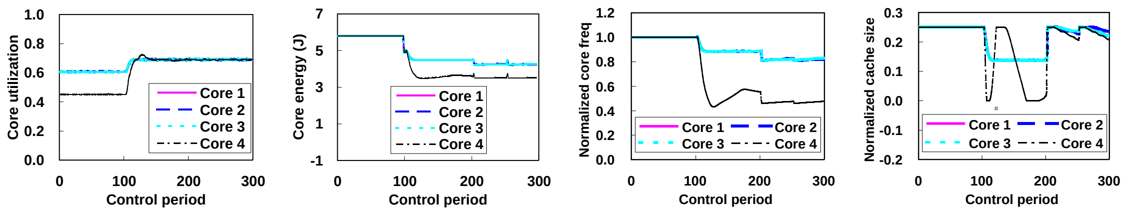
Table 4.1: System model parameters in (4.6) for typical benchmarks.

Benchmark	WSS	n_{11}	A_{11}	B_{11}	R^2
basicmath	2026	4.4e+7	-3.8e+7	3.8e+7	0.93
susan	886	4.05e+7	-9673	7.0e+6	0.99
bitcnts	445	7.64e+7	-1.7e+5	9.0e+7	0.99

The simulation environment implements a multi-core real-time system based on the simulated processor and the cache-aware power management and utilization control architecture, which includes the utilization monitors, core frequency modulators, cache partitioning/resizing actuators, and the processor-level cache arbitrator. The periodic tasks on each core are scheduled by RMS. Similar to previous studies based on the EUCON simulator, the multi-objective MPC controllers are implemented in Matlab. Specifically, the primary optimizer of an MOMPC controller is implemented based on the `lsqlin` least squares solver and the secondary optimizer is implemented based on the `fmincon` constrained nonlinear multi-variable optimizer. In each simulation, the simulator first opens a Matlab process and initializes the parameters. At the end of each control period, the simulator collects the utilization of each core from the utilization monitors, and calls the MOMPC controllers in Matlab. The MOMPC controllers compute the control inputs, $f_i(k)$ and $s_i(k)$, and return them to the simulator. The simulator calls the frequency modulators and cache partitioning/resizing actuators to enforce the control inputs. Note that the overhead of the MOMPC controllers is sufficiently small because we adopt the core-level controller design (discussed in Section 4.2). As a result, each MOMPC controller only has one controlled variable and two manipulated variables. Note also that the controllers can be implemented in service processor firmware in a real system and thus its computation and power overheads will not significantly affect the main multi-core processor. An MOMPC controller can also tolerate a considerable communication delay, as long as the delay is short when compared with the control period [96].

Cache partitioning divides a shared cache into non-overlapping partitions for independent use by real-time tasks. The benefit is that it eliminates the inter-core interferences among real-time tasks caused by the shared cache and thus leads to improved real-time performance [11], because the interferences may introduce difficulties to the estimation of WCETs of real-time tasks. It is well known that the WCET estimation for shared-cache multicores is still an open problem because interferences exist. Given a k-associative cache (not necessarily a fully-associative cache) with l cache sets, the cache can be divided based on associativity or based on cache sets. Associativity-based partitioning assigns a certain number of ways (0 to k) within each cache set to a partition while set-based partitioning assigns a certain number of sets (0 to l) to a partition. The difference of the two approaches is the partitioning granularity. In this work, we design the proposed control solution on set-based partitioning because its granularity is fine-grained ($l >> k$).

Overhead Analysis: Our simulations take into consideration both time and energy overheads of the proposed MOMPC controller, DVFS and cache partitioning. We measure the execution times and energy consumption of both the primary optimizer and the secondary optimizer of the proposed MOMPC controller by running it on the simulated multi-core processor. The time overhead of the primary optimizer is 0.8ms and its energy overhead is approximately 0.088J. The time overhead of the secondary optimizer is 2.2ms and its its energy overhead is approximately 0.242J. Although overheads of the secondary optimizer are higher than those of the primary optimizer, the secondary optimizer is only invoked every 50 control periods. The total time overhead of the proposed MOMPC controller is less than 2% of a control period. Park et al. [107] presents an accurate modeling of the time and energy overheads of DVFS techniques such as Intel's SpeedStep Technology and AMD equivalent PowerNow. The transition time is between 15.2 μs to 82.6 μs and its energy overhead is from 0.1 mJ to 0.52 mJ. Therefore, the time overhead of DVFS is less than 0.6% of the control period. To implement the cache partitioning in a chip, additional circuits have to be added which will consume additional energy compared with



(a) Core utilization (b) Core energy consumption (c) Core frequency (d) Core cache size

Figure 4.2: A typical run of the proposed cache-aware control solution. The MOMPC controllers (primary optimizers) are activated at time 100 to control utilizations and the secondary optimizers are enabled at time 200 for energy optimization.

the processors without cache partitioning support. Studies on computer architeture [86][76] have shown that the time overhead is 2% on average. The area for the circuits implementating some cache partitioning technique is only 1.5% of the total area of caches. Thus, the energy overhead of the cache partitioning is estimated to be 1.5% of the energy consumption of caches. In our simulations, we deduct all the estimated energy overheads related to the proposed control solution from the energy results.

4.5 Experimental Results

In this section, we first introduce two state-of-the-art baselines. We then evaluate our proposed control architecture using the Mibench benchmarks and compare it with the baselines.

Baselines

Our first baseline, referred to as *Dynamic repartitioning* [119], is a typical energy-efficient scheduling algorithm for real-time tasks on a multi-core processor without considering the frequency-independent component of task execution time and cache power consumption. To achieve a low power consumption, Dynamic repartitioning balances the dynamic utilization of all cores by migrating tasks among the cores. It calls a repartitioning function whenever a task is completed or a new task period starts. The function migrates a task T_m that lowers the chip-wide frequency after migration, from the core with the highest dynamic utilization, C_{max}, to the core with the lowest dynamic utilization, C_{min}. The migration process continues until the chip-wide frequency level cannot be lowered further by task migration. The key differences between Dynamic repartitioning and our solution are that 1) Dynamic repartitioning assumes the task execution time scales inversely linearly with the core frequency and 2) all the cores in a processor are assumed to have a uniform DVFS level.

The second baseline, referred to as *DVFS-Only*, is the frequency scaling loop proposed in [143]. DVFS-Only represents existing utilization control mechanisms that assume the task execution time scales only with the CPU frequency and applies DVFS for utilization control and power management.

We show that our proposed solution, which manipulates both frequency and cache size, outperforms both baselines by consuming less power consumption.

Cache-Aware Utilization Control

In this experiment, we first evaluate the performance of our MOMPC controller. We adopt two different task sets to conduct our experiments on the simulated quad-core processor. The first task set includes two periodic tasks running *basicmath* benchmarks with a total utilization of 0.6, while the second task set contains three periodic tasks running a mix of *basicmath* and *bitcnts* benchmarks with a total utilization of 0.45. The workloads for the first three cores are identical and they execute the first task set. The workload for core 4 is different and it executes the second task set. The task period of basicmath is 0.08 seconds while the task period of bitcnts is 0.16 seconds. We initially assign an even cache quota to each core. We also conduct a set of experiments to examine randomly generated workloads.

In our experiment, we activate our MOMPC controllers on all the cores at time 100 (*i.e.*, the 100th control period) and enable the secondary optimizers at time 200. Figure 4.2a shows that the utilizations of all cores are controlled accurately to their RMS bounds (*e.g.*, 0.69) after the MOMPC controllers are activated. As a result, no deadline miss is observed. Figure 4.2b shows that the energy consumption in every control period. The specific value of the control period is 0.16 seconds. The energy consumption are reduced from time 100 to 200. After the secondary optimizers are enabled on all the cores at time 200, the energy consumption is minimized: from 6J to 4J for cores 1-3 and from 6J to 3.5J for core 4. The small spikes in the energy consumption at the 200th and 250th control periods are caused by the secondary optimizer. Figures 4.2c and 4.2d detail the behavior of the MOMPC controllers by plotting the frequencies and cache partition sizes of the cores. From time 100 to 200, the MOMPC controllers, without the secondary optimizers, does not reduce the frequencies to the minimum level. As a result, the processor energy consumption is not minimized. At time 200, the secondary optimizers are enabled to achieve energy optimization by throttling the core frequencies. As a result, the cache partition sizes are increased for all the cores and overall energy consumption is reduced without affecting the core utilizations. This experiment clearly demonstrates that the MOMPC controller can achieve better energy efficiency than a traditional MPC controller that does not contain the secondary optimizer.

To test the robustness of the proposed MOMPC controller, we conduct a set of experiments with different randomly generated workloads. For each workload, the number of tasks on each core is increased from 2 to 6 (*i.e.*, 8 to 24 tasks in total). Figure 4.3a plots the average CPU utilizations of all the cores after the controllers enter the steady state. Our MOMPC controllers successfully achieve the desired utilization set points with zero steady state errors for all the workloads. Figure 3(b) shows that the MOMPC controllers achieve more energy savings than the MPC controllers.

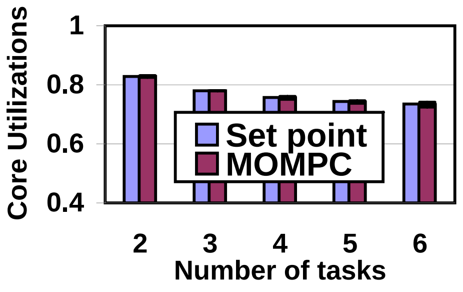
Comparison with Dynamic Repartitioning

In this experiment, we compare the proposed solution with the first baseline, Dynamic repartitioning. To have a fair comparison, we adopt the same workload used in Section 4.5. Figure 4.4a shows that after Dynamic repartitioning activates at time 150, the utilizations of all the cores increase only slightly. None of the cores achieve the desired utilization set points (*e.g.*, 0.69). The reason is that Dynamic repartitioning assumes that the execution times of tasks are inversely proportional to the core frequencies, without considering the frequency-independent execution times. In the workload we adopted, the frequency-independent execution times (about 2.83e+7 CPU cycles) comparable to the frequency-dependent execution times (about 4.4e+7 CPU cycles). As a result, Dynamic repartitioning fails to control utilizations accurately, which can lead to power inefficiency, as shown later. Another fundamental assumption of Dynamic repartitioning is chip-wide DVFS, which holds true for certain multi-core processors. However, as microelectronic technologies advance, per-core DVFS has been implemented and is expected to become the mainstream configuration. Since the workload on each core is not perfectly balanced, the cores cannot achieve their utilization set points simultaneously with chip-wide DVFS.

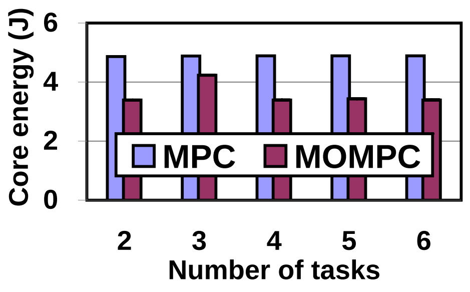
Figure 4.4b shows the energy consumption of each core. Since Dynamic repartitioning reduces the frequencies of all the cores from the highest level to the same level (with chip-wide DVFS) and does not manage the energy consumption of the shared L2 caches, the energy savings of each core is approximately identical. Even though no deadline is violated by Dynamic repartitioning, the energy consumption of each core is only slightly reduced from 6J to 5.5J, which leads to unnecessarily more energy consumption as the proposed cache-aware control can reduce energy significantly. The first reason is that Dynamic repartitioning can not control utilizations accurately. Since the proposed cache-aware control considers frequency-independent execution times, both core frequencies and cache partitions can be adjusted to achieve accurate utilization controls which translate to additional energy savings. The second reason is that Dynamic repartitioning does not take advantage of per-core DVFS, which is proven to be more energy efficient than chip-wide DVFS [74]. This experiment demonstrates that the cache-aware control solution outperforms Dynamic repartitioning in terms of energy efficiency.

Comparison with DVFS-Only

In this experiment, we compare the proposed solution with the second baseline: DVFS-Only. We activate the solutions at time 100. The workload on each core is configured to be identical and includes three periodic tasks. We simulate the

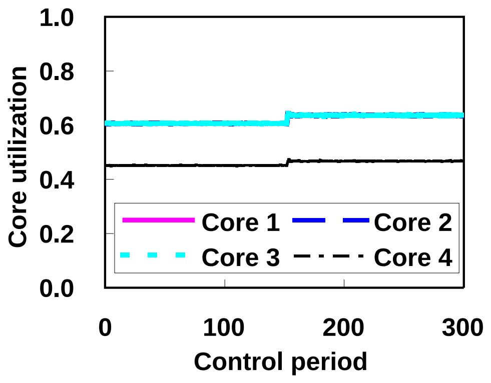


(a) Core utilization

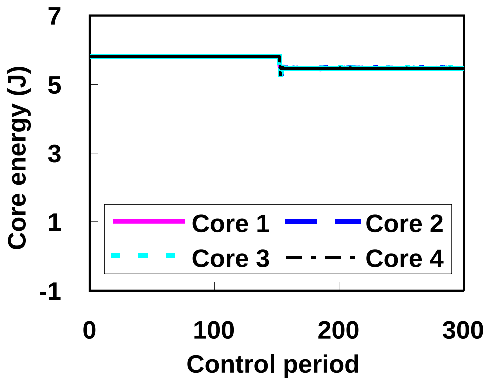


(b) Core energy consumption

Figure 4.3: The proposed cache-aware solution (i.e.,MOMPC) controls core utilization to desired set points while saving more energy than MPC.

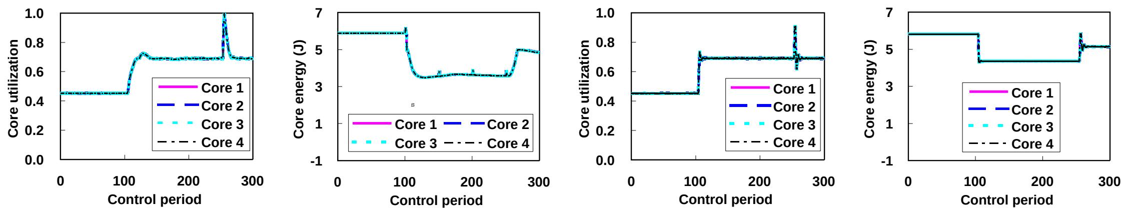


(a) Core utilization



(b) Core energy consumption

Figure 4.4: A typical run of the baseline Dynamic repartitioning (activated at time 150).



(a) Core utilization (Cache-aware) (b) Core energy (Cache-aware) (c) Core utilization (DVFS-Only) (d) Core energy (DVFS-Only)

Figure 4.5: Typical runs of cache-aware control and DVFS-Only under workload variations.

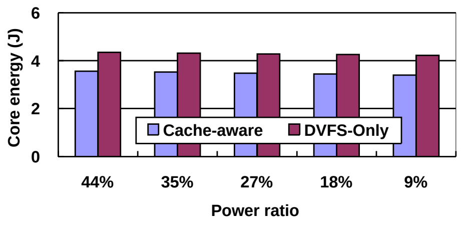


Figure 4.6: Comparison with DVFS-Only when power ratio varies (before workload increases).

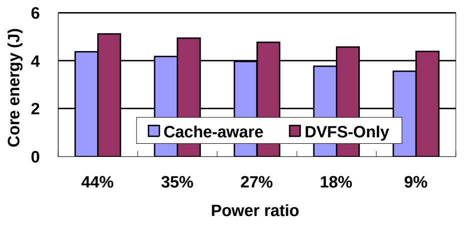


Figure 4.7: Comparison with DVFS-Only when power ratio varies (after workload increases).

typical scenario of a real-time system with uncertain execution times by increasing the frequency-dependent execution times of the tasks on all the cores by 100% at time 250. We compare the energy efficiency of the proposed solution and DVFS-Only in such a scenario.

Figure 4.5a shows that under the proposed solution, after time 100, the utilizations of all the cores are controlled to the set points (e.g., 0.69). Due to the workload variation at time 250, the utilizations increase significantly. The proposed solution successfully controls the utilizations back to the set points. The deadline miss rate is 0.5% since the utilization bound approximately at time 250 is violated, creating deadline misses. Figure 4.5b shows that before the proposed solution activates at time 100, the core energy consumption is high because the core frequencies are initially set to the highest levels and the L2 caches are all turned on. From time 100 to 250, the energy consumption is reduced significantly by the proposed solution. After time 250, both the core frequencies and cache sizes are increased due to the workload increase, resulting in an increased energy consumption.

Figure 4.5c shows a typical run of DVFS-Only in the same scenario under the same workload. Note that although DVFS-Only also assumes the execution times of tasks are inversely proportional to the core frequencies as Dynamic repartitioning, DVFS-Only can control the utilizations to the set points accurately because DVFS-Only relies on the feedback of the measured utilizations. When compared with the proposed solution, the deadline miss ratio of DVFS-Only is zero because the peak utilization of the proposed solution is 1 while the peak utilization of DVFS-Only is only 0.9. Figure 4.5d shows that from time 100 to 250, the energy consumption is approximately 4.3J, which is much higher than 3.7J, the power consumption of the proposed solution (shown in Figure 4.5b). The reason is that DVFS-Only does not turn off the caches for energy savings. Thus, the energy consumption cannot be reduced significantly by only throttling DVFS. After time 250, the energy consumption is approximately 5.1J while the power consumption of the proposed solution is about 4.3J. As the frequency-dependent portion in the execution times increases, the gap of the energy consumption of the two solutions narrows. On average, DVFS-Only consumes 20% more energy per core than the proposed solution. This experiment demonstrates that the proposed solution is more energy efficient than DVFS-Only under workload variations.

To test the impact of the parameters in the power model on energy efficiency, we define the *power ratio* of a core to be the ratio of the dynamic power consumption when the core frequency is the maximum level to the cache power consumption of the core when all the caches are turned on. We use the same scenario to increase the workload at time 250. Figures 4.6 and 4.7 show the energy consumption of the two solutions before and after time 250 (workload increase), respectively. Figure 4.6 shows that when the power ratio is lower, which means the percentage of leakage power consumption is higher in the total power consumption, the gap between the proposed solution and DVFS-Only widens because DVFS-Only can only adjust DVFS to manage power consumption. The difference between the proposed solution and DVFS-Only in Figure 4.6 is smaller than that in Figure 4.7. The reason is that when the frequency-independent execution times become relatively smaller, the advantage of the proposed cache-aware solution to dynamically resize caches for reduced leakage power becomes smaller.

Chapter 5

Power-Aware Utilization Control for Distributed RT Systems

5.1 Probelm Formulation

In this section, we formulate the new CPU utilization control problem for DRE systems.

5.1.1 Task Model

We adopt an end-to-end task model [89] implemented by many DRE applications. A system is comprised of m periodic tasks $\{T_i | 1 \leq i \leq m\}$ executing on n processors $\{P_i | 1 \leq i \leq n\}$. Task T_i is composed of a set of sub-tasks $\{T_{ij} | 1 \leq j \leq n_i\}$ which may be located on different processors. A processor may host one or more sub-tasks of a task. The release of subtasks is subject to precedence constraints, i.e., subtask $T_{ij}(1 < j \leq n_i)$ cannot be released for execution until its predecessor subtask T_{ij-1} is completed. All the subtasks of a task share the same rate. The rate of a task (and all its subtasks) can be adjusted by changing the rate of its first subtask. If a non-greedy synchronization protocol (e.g., release guard [129]) is used to enforce the precedence constraints, every subtask are released periodically without jitter.

In our task model, each task T_i has a *soft* end-to-end deadline related to its period. In an end-to-end scheduling approach [129], the deadline of an end-to-end task is divided into subdeadlines of its subtasks. Hence the problem of meeting the end-to-end deadline can be transformed to the problem of meeting the subdeadline of each subtask. A well known approach for meeting the subdeadlines on a processor is to ensure its utilization remains below its schedulable utilization bound [89].

Our task model has three important properties. First, while each subtask T_{ij} has an *estimated* execution time c_{ij} available at design time, its *actual* execution time may be different from its estimation and vary at run-time due to two reasons: CPU frequency scaling or workload uncertainties. Modeling such uncertainties is important to DRE systems operating in unpredictable environments. Second, the rate of a task T_i may be dynamically adjusted within a range $[R_{min,i}, R_{max,i}]$. This assumption is based on the fact that the task rates in many applications (e.g., digital control [97], sensor update, and multimedia [32]) can be dynamically adjusted without causing system failure. A task running at a higher rate contributes a higher value to the application at the cost of higher utilizations. Please note that our solution does *not* rely on continuous task rates. For a task with only discrete rates, its continuous rate value will be truncated to the highest discrete rate supported by the task that is below the continuous value. The utilization difference resulted from the truncation can be compensated by the frequency scaling loop. Third, the CPU frequency of each processor P_i may be dynamically adjusted within a range $[F_{min,i}, F_{max,i}]$. This assumption is based on the fact that many today's processors are DVFS-enabled. For processors that do not support DVFS, clock modulation can be used instead to change CPU frequency [82]. The frequency ranges are assumed to be continuous because a continuous value can be approximated by a series of discrete frequency levels supported by a processor, as we explain in Section 5.5.

5.1.2 Problem Formulation

Utilization control can be formulated as a dynamic constrained optimization problem. We first introduce some notation. T_s, the control period, is selected so that multiple instances of each task may be released during a control period. $u_i(k)$ is the CPU utilization of processor P_i in the k^{th} control period, i.e., the fraction of time that P_i is not idle during time interval $[(k-1)T_s, kT_s)$. B_i is the desired utilization set point on P_i. $r_j(k)$ is the invocation rate of task T_j in the $(k+1)^{th}$ control period. $f_i(k)$ is the relative CPU frequency (i.e., CPU frequency relative to the highest level $F_{max,i}$) of processor P_i in the $(k+1)^{th}$ control period.

Given a utilization set-point vector, $\mathbf{B} = [B_1 \ldots B_n]^T$, rate constraints $[R_{min,j}, R_{max,j}]$ for each task T_j, and frequency constraints $[F_{min,i}, F_{max,i}]$ for each processor P_i, the control goal at k^{th} sampling point (time kT_s) is to dynamically choose task rates $\{r_j(k)|1 \leq j \leq m\}$ and CPU frequencies $\{f_i(k)|1 \leq i \leq n\}$ to minimize the difference between B_i and $u_i(k)$ for all the processors:

$$\min_{\{r_j(k)|1\leq j\leq m, f_i(k)|1\leq i\leq n\}} \sum_{i=1}^{n} (B_i - u_i(k+1))^2 \tag{5.1}$$

subject to constraints

$$R_{min,j} \leq r_j(k) \leq R_{max,j} \qquad (1 \leq j \leq m) \tag{5.2}$$

$$F_{min,i} \leq f_i(k) \leq F_{max,i} \qquad (1 \leq i \leq n) \tag{5.3}$$

The rate constraints ensure all tasks remain within their acceptable rate ranges. The frequency constraints ensure all CPU frequencies remain within their acceptable ranges. The optimization formulation minimizes the difference between the utilization of each processor and its corresponding set point, by manipulating the rate of every task and the frequency of every processor within their constraints. The design goal is to ensure that all processors quickly converge to their utilization set points after a workload variation, whenever it is feasible under the constraints. Therefore, to guarantee end-to-end deadlines, a user only needs to specify the set point of each processor to be a value below its schedulable utilization bound. Utilization control algorithms can be used to meet all the end-to-end deadlines by enforcing the set points of all the processors in a DRE system, when feasible under the rate constraints[*].

5.2 End-to-End Utilization Control

In this section, we model the end-to-end utilization control problem and present our two-layer control architecture.

5.2.1 System Modeling

Following a control-theoretic methodology, we establish a dynamic model that characterizes the relationship between the controlled variable $\mathbf{u}(k)$ and the manipulated variables $\mathbf{\Delta r}(k)$ and $\mathbf{\Delta f}(k)$. We first model the utilization $u_i(k)$ of one processor P_i. As observed in previous research [16][116], the execution times of tasks on P_i can be approximately estimated to be a linear function of P_i's relative CPU frequency[†]. Therefore, the *estimated* execution time of task T_{jl} in the k^{th} control period can be modeled as $c_{jl}/f_i(k)$. The *estimated* CPU utilization of processor P_i can be modeled as:

$$b_i(k) = \frac{\sum_{T_{jl} \in S_i} c_{jl} r_j(k)}{f_i(k)} \tag{5.4}$$

where S_i is the set of subtasks located at processor P_i.

We then define the *estimated* utilization change of P_i, $\Delta b_i(k)$, as:

$$\Delta b_i(k) = \frac{\sum_{T_{jl} \in S_i} c_{jl} r_j(k)}{f_i(k)} - \frac{\sum_{T_{jl} \in S_i} c_{jl} r_j(k-1)}{f_i(k-1)} \tag{5.5}$$

[*]A system must apply admission control when its load exceeds the limit that can be handled within the rate and frequency constraints.

[†]In general, the execution times of some tasks may include frequency-independent parts that do not scale linearly with CPU frequency [17]. We plan to model frequency-independent parts in our future work.

Note $\Delta b_i(k)$ is based on the estimated execution time c_{jl}. Since the actual execution times may be different from their estimation due to workload variations, we model the actual utilization of P_i, $u_i(k)$, as the following difference equation.

$$u_i(k+1) = u_i(k) + g_i \Delta b_i(k) \tag{5.6}$$

where the utilization gain g_i represents the ratio between the change to the actual utilization and its estimation $\Delta b_i(k)$. For example, $g_i = 2$ means that the actual change to utilization is twice the estimated change. Note that the exact value of g_i is *unknown* due to the unpredictability of subtasks' execution times.

The system model (5.6) is nonlinear because of the definition of $\Delta b_i(k)$ in (5.5). Therefore, we need linearization to simplify the controller design for acceptable runtime overhead. There are two ways to linearize the system model. First, we may assume that all the processors always run at their highest CPU frequency and the utilizations are controlled by rate adaptation only. As a result, $f_i(k)$ becomes 1 and the system model (5.6) becomes a linear model between $\Delta b_i(k)$ and $\Delta r_j(k) = r_j(k) - r_j(k-1)$. Second, we can assume that the utilizations are controlled by frequency scaling only. As a result, $r_i(k)$ is a constant and the model becomes a linear model between $\Delta b_i(k)$ and $\Delta d_i(k) = 1/f_i(k) - 1/f_i(k-1)$.

However, in a system that allows both rate adaptation and frequency scaling, relying solely on one adaptation strategy may unnecessarily reduce the system's adaptation capability because both task rates and CPU frequencies can only be adapated within limited ranges. Therefore, a novel control architecture needs to be designed for utilizing both rate and frequency adapations to maximize the system's adaptation capability.

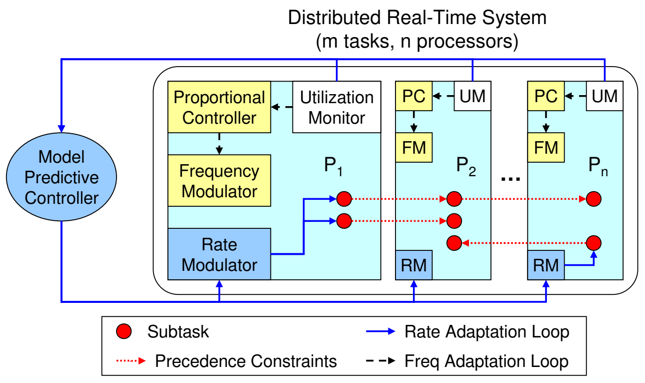


Figure 5.1: Utilization control architecture

5.2.2 Control Architecture

In this dissertation, we propose a two-layer utilization control architecture, as shown in Figure 5.1. To avoid having a nonlinear model, our control architecture features two coordinated control loops running in different control periods.

First, the cluster-level rate adaptation loop dynamically controls the utilizations of all the processors by adjusting task rates within their allowed ranges. Because the rate change of a task affects the utilizations of all the processors where the task has subtasks, this loop is a MIMO control loop, which works as follows: (1) the utilization monitor on each processor P_i sends its utilization $u_i(k)$ in the last control period to the Model Predictive Controller; (2) the controller computes a new rate $r_j(k)$ for every task T_j and sends the new rates to the rate modulators; and (3) the rate modulators change the task rates accordingly. Please note again that for a task with only discrete rates, the rate modulator will truncate its continuous rate value to the highest discrete rate supported by the task that is below the continuous value.

Second, on every processor P_i in the system, we have a local controller that controls the utilization by scaling the CPU frequency of the processor. The controller is a Single-Input-Single-Output (SISO) controller because the CPU frequency change of P_i only affects the utilization of P_i. This loop works as follows: (1) the utilization monitor on P_i sends its utilization $u_i(k)$ to the local controller; (2) the controller computes a new CPU frequency $f_i(k)$ and sends it to the frequency modulator on P_i; and (3) the frequency modulator changes the CPU frequency accordingly.

Clearly, without effective coordination, the two control loops may conflict with each other because they are controlling the same varilable, i.e., CPU utilization. To achieve the desired control function and system stability, one control loop, i.e., the primary loop, needs to be configured with a control period that is longer than the settling time of the other control loop, i.e., the secondary loop. As a result, the secondary loop can always enter its steady state within one control period of the primary control loop. The two control loops are thus decoupled and can be designed independently. The impact of the primary loop on the secondary loop can be modeled as variations in its system model, while the impact of the secondary loop on the primary loop can be treated as system noise. As long as the two control loops are stable individually, the whole system is stable.

In our design, we choose the task rate adaptation loop as the secondary control loop for two reasons. First, the secondary loop reacts faster to utilization variations. As a result, the secondary loop has the priority to increase the value of its manipulated variable(s) when the actual utiliztaion is lower than the set point, especially at the beginning of a system run. We assume that a higher task rate contributes a higher system value to the application and system value is more important than power efficientcy in our target real-time applications. Second, the secondary loop must remain stable despite its model variation caused by the primary loop. The stability of the rate adaptation loop is less sensitive based on our coordination analysis in 5.4.4.

In our control architecture, the rate adapation loop tries to achieve the desired CPU utilization set points while maximizing the task rates. When it is infeasible to control utilizations by rate adapation alone (e.g., due to rate saturation or discrete task rates), the frequency scaling loop can help to achieve the desired set points on a coarser timescale while reducing the power consumption of the processors. Since the core of each control loop is its controller, we introduce the design and analysis of the two controllers in the next two sections, respectively. The implementation details of other components are provided in Section 5.5.

5.3 Task Rate Adaptation Loop

In this section, we briefly introduce the system model and design of the rate adapation loop.

5.3.1 System Model

Based on the control architecture, we assume that the relative CPU frequency $f_i(k) = 1$ for all the processors. The case when $f_i(k) \neq 1$ is analyzed in Section 5.4.4. Hence, the estimated utilization change $\Delta b_i(k)$ in (5.5) becomes:

$$\Delta b_i(k) = \sum_{T_{jl} \in S_i} c_{jl} \Delta r_j(k) \tag{5.7}$$

where $\Delta r_j(k) = r_j(k) - r_j(k-1)$.

Based on (5.6), a DRE system with m tasks and n processors is described by the following MIMO dynamic model.

$$\mathbf{u}(k) = \mathbf{u}(k-1) + \mathbf{G}\mathbf{\Delta b}(k-1) \tag{5.8}$$

where $\mathbf{G}$ is a diagonal matrix where $g_{ii} = g_i$ $(1 \leq i \leq n)$ and $g_{ij} = 0$ $(i \neq j)$. $\mathbf{\Delta b}(k)$ is a vector including the estimated utilization change (5.7) of each processor.

5.3.2 Controller Design

In this paper, we adopt the EUCON algorithm presented in our previous work [93] for rate adapation. EUCON features an MPC controller that optimizes a *cost function* defined over P control periods in the future, called the *prediction horizon*. The control objective is to select control inputs in the following M control periods, called *control horizon*, that minimizes the followng cost function while satisfying the constraints.

$$\begin{aligned} V(k) = & \textstyle\sum_{i=1}^{P} \|\mathbf{u}(k+i|k) - \mathbf{ref}(k+i|k)\|^2 \\ & + \textstyle\sum_{i=0}^{M-1} \|\mathbf{\Delta r}(k+i|k) - \mathbf{\Delta r}(k+i-1|k)\|^2 \end{aligned} \tag{5.9}$$

where P is the prediction horizon, and M is the control horizon. The first term in the cost function represents the *tracking error*, i.e., the difference between the utilization vector $\mathbf{u}(k+i|k)$ and a reference trajectory $\mathbf{ref}(k+i|k)$ defined in [93]. By minimizing the tracking error, the closed-loop system will converge to the utilization set points if the system is stable. The second term in the cost function represents the control penalty. This control problem is subject to the rate constraints (5.2). The detailed design and analysis of EUCON are available in [93].

Although the rate adaptation loop is proved to be stable in [93], in order for the coordinated control architecture to be stable, the stability and settling time of the rate adaptation loop need to be reexamined with the impact from the frequency scaling loop. The detailed coordination analysis is presented in Section 5.4.4.

5.4 CPU Frequency Scaling Loop

In this section, we first model, design, and analyze the CPU frequency scaling loop. We then analyze the coordination between the two control loops.

5.4.1 System Model

Based on our control architecture, the frequency adapation loop can be designed separately from rate adaptation. As a result, model (5.6) can be simplified by having $r_i(k)$ in (5.5) as a constant r_i. This decouples different processors because, as discussed in Section 5.2.1, processors are coupled to each other due to the fact that the rate change of a task may affect the utilizations of all the processors where its subtasks are located. The utilization of each processor can now be modeled individually because the CPU frequency change $\Delta d_i(k) = 1/f_i(k) - 1/f_i(k-1)$ only affects the execution times of all the subtask on P_i. Specifically, the model of processor P_i is:

$$u_i(k) = u_i(k-1) + g_i \Delta d_i(k) \sum_{T_{jl} \in S_i} c_{jl} r_j \tag{5.10}$$

The model cannot be directly used to design controller because the system gain g_i is used to model the uncertainties in task execution times and thus unknown at design time. Therefore, we design the controller based on an approximate system model, which is model (5.10) with $g_i = 1$. In a real system where the task execution times are different than their estimations, the *actual* value of g_i may become different than 1. As a result, the closed-loop system may behave differently. However, in Section 5.4.3, we show that a system controlled by the controller designed with $g_i = 1$ can remain stable as long as the variation of g_i is within a certain range. This range is established using stability analysis of the closed-loop system by considering the model variations.

5.4.2 Controller Design

Following standard control theory [49], we design a Proportional (P) controller to achieve desired control performance such as stability and zero steady state error. We choose to use a P controller instead of a more sophisticated controller such as a PID (Proportional-Integral-Derivative) controller because the actuator $1/f_i(k) = \Delta d_i(k) + 1/f_i(k-1)$ already includes an integrator such that zero steady state error can be achieved without resorting to an I (Integral) part. The D (Derivative) part is not used because it may amplify the noise in utilization in unpredictable environments. The Z-domain form of our P controller is:

$$C(z) = \frac{1}{\sum_{T_{jl} \in S_i} c_{jl} r_j} \tag{5.11}$$

The transfer function of the closed-loop system controlled by controller (5.11) is:

$$G(z) = z^{-1} \tag{5.12}$$

It is easy to prove that the controlled system is stable and has zero steady state errors when $g_i = 1$. The detailed proofs can be found in a standard control textbook [49] and are skipped due to space limitations. The desired CPU frequency in the k^{th} control period is:

$$f_i(k) = \frac{f_i(k-1)\sum_{T_{jl}\in S_i} c_{jl}r_j}{(U_s - u(k))f_i(k-1) + \sum_{T_{jl}\in S_i} c_{jl}r_j} \tag{5.13}$$

5.4.3 Control Analysis for Model Variation

In this subsection, we analyze the system stability when the designed P controller is used on a system with $g_i \neq 1$. A fundamental benefit of the control-theoretic approach is that it gives us theoretical confidence for system stability, even when the controller is used in a different working condition.

The closed-loop transfer function for the real system is

$$G(z) = \frac{g_i}{z - (1 - g_i)} \tag{5.14}$$

The closed-loop system pole in (5.14) is $1 - g_i$. In order for the system to be stable, the pole must be within the unit circle. Hence, the system will remain stable as long as $0 < g_i < 2$. The result means that the actual utilization change *cannot* be twice the estimated utilization change. Since the frequency scaling loop is the outer loop of our two-layer control architecture, the difference between the actual and estimated utilization changes is mainly caused by the differences between the actual and estimated execution times. Therefore, it is preferable to use pessimistic estimation on execution times such that the controlled system can be guarnateed to be stable and the system oscillation can also be reduced. Please note that using pessimistic estimated execution times does not result in underutilization of the CPU as in systems that rely on traditional open-loop scheduling. This is because our control architecture dynamically adjusts CPU frequencies and tasks rates based on *measured* utilization rather than the estimated execution times. The downside of using more pessimistic estimation on execution times is that it leads to a smaller system gain, which may cause slower convergence to the set points. However, since it is more important to guarantee system stability in a DRE system, it is still preferable to overestimate task execution times.

We now analyze the steady state error of the controlled system when $g_i \neq 1$.

$$\lim_{z \to 1}(z-1)U(z) = \lim_{z \to 1}\left(\frac{g_i z}{z-(1-g_i)}U_s\right) = U_s \tag{5.15}$$

Equation (5.15) means that we are guaranteed to achieve the desired CPU utilization as long as the system is stable.

5.4.4 Coordination Analysis

We now analyze the coordination needed for the rate adaptation loop to work with the frequency scaling loop. A major contribution of our paper is to demonstrate the importance of coordinating different control loops.

First, we need to ensure that the stability of the rate adaptation loop will not be affected when the frequency scaling loop changes the CPU frequency and so $f_i(k) \neq 1$. Given a specific task set, the stability condition of the rate adaptation loop as a range of g_i (i.e., the ratio between the actual utilization change and the estimated change) can be established by following the steps presented in [93]. For example, the stability condition of the task set used in our experiments is that the actual change cannot be 10 times the estimated change. Accordingly, we must guarantee that the relative CPU frequency of each processor is not smaller than 0.1 because the rate adaptation controller is designed with the assumption of $f_i(k) = 1$. This constraint must be enforced in the frequency scaling loop. One of the reasons for us to choose the rate adaptation loop as the secondary loop in our control architecture is that it has a larger stability range and thus is less sensitive to the impact of the primary loop.

Second, we need to analyze the settling time of the rate adaptation loop such that we can determine the control period of the frequency scaling loop. Since settling time has not been analyzed in [93], we now outline the general process of analyzing the settling time of the rate adaptation loop when the actual utilization change is different from the estimated change, i.e., $g_i \neq 1$. First, given a specific task set, we derive the control inputs $\mathbf{\Delta r}(k)$ that minimize the cost function (5.9) based on the system model (5.8) with $g_i = 1$. The control inputs represent the control decision based on the estimated system model. Second, we derive the closed-loop system model by substituting the control inputs derived in the first step into the system model (5.8) where $g_i \neq 1$. The analysis needs to consider a composite system consisting of the dynamics of the original system and the controller. Finally, we calculate the dominant pole (i.e., the pole with the largest magnitude) of the closed-loop system. According to control theory, the dominant pole determines the system's transient response such as settling time.

Based on our analysis, the task set used in our experiments has a settling time of 5 control periods. The detailed derivation is not included due to space limitations. The control period of the rate adaptation loop is selected to be 2 seconds to include multiple instances of each task. Therefore, the control period of the frequency scaling loop is set to 20 seconds.

5.5 System Implementation

Our testbed includes 4 Linux servers, called RTES1 to RTES4, to run the end-to-end real-time tasks and a desktop machine to run the MPC controller. The 4 servers are equipped with 2.4GHz AMD Athlon 64 3800+ processors with 1GB RAM and 512KB L2 Cache. The controller machine is a Dell OptiPlex GX520 with 3.00GHz Intel Pentium D Processor and 1GB RAM. All the machines are connected by a 100Mbps internal Ethernet switch. The 4 servers run openSUSE Linux 11 with kernel 2.6.25 while the controller machine runs Windows XP.

We implement our control architecture in FC-ORB, an open-source real-time Object Request Broker (ORB) middleware system [139]. FC-ORB supports end-to-end real-time tasks based on the end-to-end scheduling framework [89]. FC-ORB implements the release guard protocol to enforce the precedence constraints among subtasks.

Our experiments run a medium-sized workload that comprises 12 end-to-end tasks (with a total of 25 subtasks). The subtasks on each processor are scheduled by the RMS algorithm [89]. Each task's end-to-end deadline is $d_i = n_i/r_i(k)$, where n_i is the number of subtasks in task T_i and $r_i(k)$ is the current rate of T_i. Each end-to-end deadline is evenly divided into subdeadlines for its subtasks. The resultant subdeadline of each subtask T_{ij} equals its period, $1/r_i(k)$. The utilization set point of every processor is set to its RMS schedulable utilization bound [89], i.e., $B_i = n_i(2^{1/n_i} - 1)$, where n_i is the number of subtasks on Pi. All (sub)tasks meet their (sub)deadlines if the desired utilization on every processor is enforced.

We now introduce the implementation details of each component in our two-layer control architecture.

Utilization Monitor: The utilization monitor uses the /proc/stat file in Linux to estimate the CPU utilization in each sampling period. The /proc/stat file records the number of jiffies (usually 10ms in Linux) when the CPU is in user mode, user mode with low priority (nice), system mode, and when used by the idle task, since the system starts. At the end of each control period, the utilization monitor reads the counters, and estimates the CPU utilization as 1 minus the number of jiffies used by the idle task in the last control period and then divided by the total number of jiffies in the same period.

MPC Controller: The controller is implemented as a single-thread process running separately on the controller machine. Each time its periodic timer fires, the controller sends utilization requests to all the 4 application servers. The incoming replies are handled asynchronously so that the controller can avoid being blocked by an overloaded application server. After the controller collects the replies from all the servers, it executes the control algorithm introduced in Section 5.3.2 to calculate the new task rates. The controller then sends the tasks' new rates to the rate modulators on the servers for enforcement. If a server does not reply in an entire control period, its utilization is treated as 100%, as the controller assumes this server is overloaded with its (sub)tasks and so cannot respond. The control period of the rate adaptation loop is 2 seconds.

Rate Modulator: A Rate Modulator is located on each processor. It receives the new rates from the controller and then resets the timer interval of the first subtask of each task whose invocation rate has been changed.

Proportional Controller: The controller is implemented as a single-thread process running on each of the 4 servers. With a control period of 20 seconds, the controller periodically reads the CPU utilization of the server, executes the control algorithm presented in Section 5.4.2 to compute the desired CPU frequency, and sends the new frequency to the frequency modulator on the server.

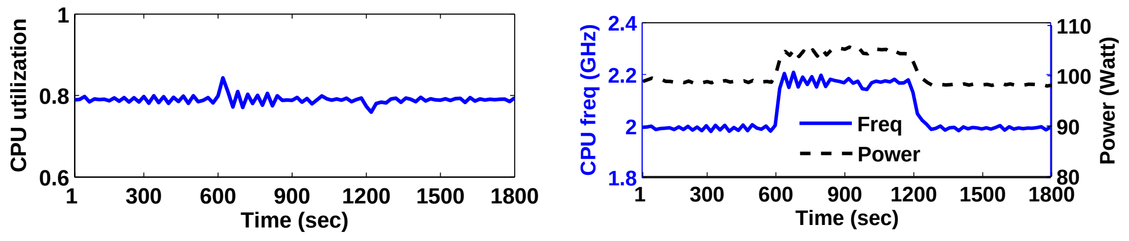
Frequency Modulator: We use AMD's Cool'n'Quiet technology [9] to enforce the new CPU frequency. AMD Athlon 64 3800+ microprocessor has 5 discrete CPU frequency levels. To change CPU frequency, one needs to install the *cpufreq* package and then use root privilege to write the new frequency level into the system file */sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed*. A routine periodically checks this file and resets the CPU frequency accordingly. The average overhead (*i.e.*, transition latency) to change frequency in AMD Athlon processors is about $100\mu s$ according to the AMD white paper report [9].

Since the new CPU frequency level periodically received from the proportional controller could be any value that is not exactly one of the five supported frequency levels. Therefore, the modulator code must locally resolve the output value of the controller to a series of supported frequency levels to approximate the desired value. For example, to approximate 2.89GHz during a control period, the modulator would output the sequence 2.67, 3, 3, 2.67, 3, 3, etc on a smaller timescale. To do this, we implement a first-order delta-sigma modulator, which is commonly used in analog-to-digital signal conversion. The detailed algorithm of the first-order delta-sigma modulator can be found in [82].

Power Monitor: The power consumption of each server is measured with a WattsUp Pro power meter by plugging the server into the power meter, which is connected to a standard 120V AC wall outlet. The WattsUp power meter has an accuracy of ±1.5% of the measured value. To access power data, the data port of each power meter is connected to a serial port of the data collection machine. The power meter samples the power data every second and then sends the reading to the data collection program through a system file */dev/ttyUSB0*.

5.6 Empirical Results

In this section, we first test the frequency scaling loop itself. We then show that the frequency scaling loop can effectively control utilizations when it is infeasible for a rate adaptation controller to do so. Finally, we demonstrate the the coordinated control solution outperforms the two single control loop solutions.

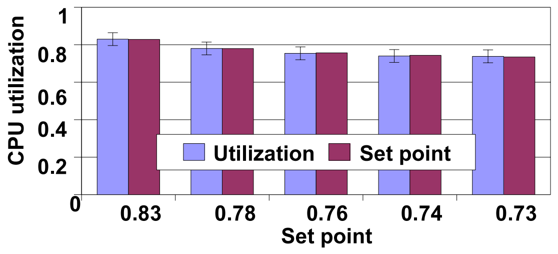


(a) CPU utilization (b) CPU frequency and power consumption

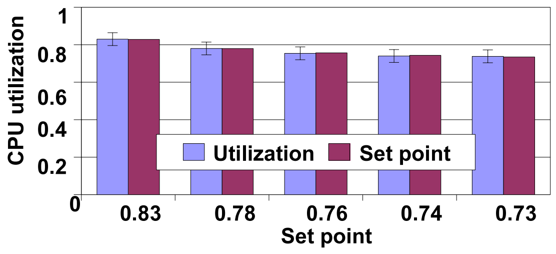
Figure 5.2: CPU utilization control by frequency scaling under a workload increase from 600s to 1200s.

Frequency Scaling Loop

In this experiment, we disable the rate adaptation loop to evaluate the performance of the frequency scaling loop on server RTES1. As a common practice in real-time systems that rely on open-loop scheduling algorithms, the workload of RTES1 is configured with carefully tuned initial task rates such that the server has a CPU utilization of 0.72, which is its RMS bound. As shown in Figure 5.2a, at time 600s, the execution times of all the tasks on RTES1 are suddenly increased by 8% to test the system capability of handling workload fluctuations. The increase makes the CPU utilization of RTES1 jump to 0.85, which is higher than the RMS bound and so may cause undesired deadline misses. Figure 5.2b shows that the frequency scaling loop responds to the utilization increase by dynamically increasing the CPU frequency of the server processor from 2.15GHz to 2.23GHz. As a result, the utilization converges to the RMS bound quickly. In contrast, An open-loop system without dynamic feedback would have its utilization stay at 0.85. At time 1200s, the execution times of workload are suddenly reduced by 7.4%, resulting in an underutilized system with a utilization lower than the RMS bound. The frequency scaling loop then responds by reducing the CPU frequency back to 2.15GHz for power savings.



(a) CPU utilization

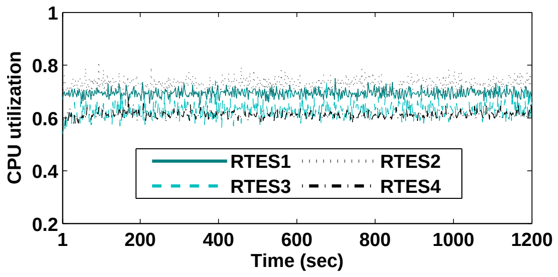


(b) Power consumption

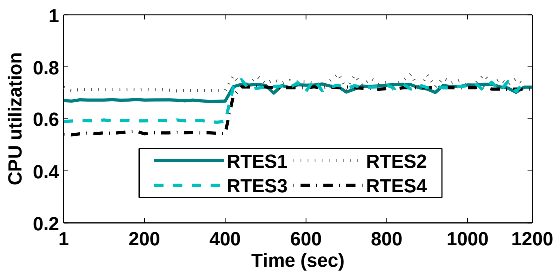
Figure 5.3: CPU utilization control by frequency scaling under different set points.

To test the robustness of the controller, we conduct a set of experiments with different utilization set points. Figure 5.3a plots the means and the standard deviations of RTES1's CPU utilization after the controller enters the steady state. We can see that the frequency scaling loop can successfully achieve the desired utilization set points. Figure 5.3b demonstrates that more power saving has been achieved when we allow the system to have a utilization closer to its RMS schedulable bound, i.e., 0.72.

Frequency Scaling vs. EUCON

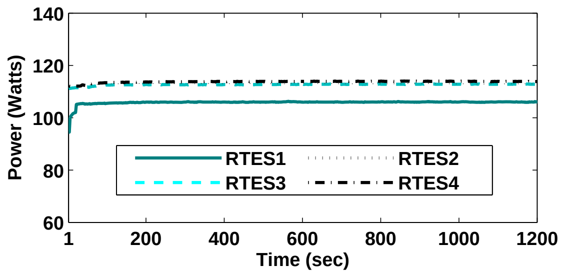


(a) EUCON

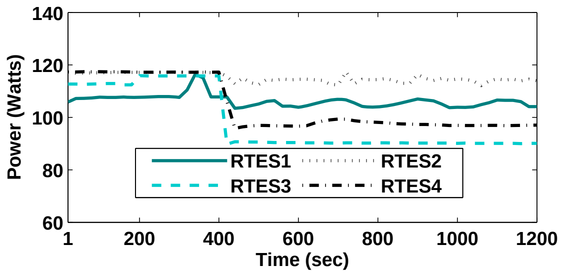


(b) Frequency scaling

Figure 5.4: Comparison of control accuracy between EUCON and the frequency scaling loop.



(a) EUCON



(b) Frequency scaling

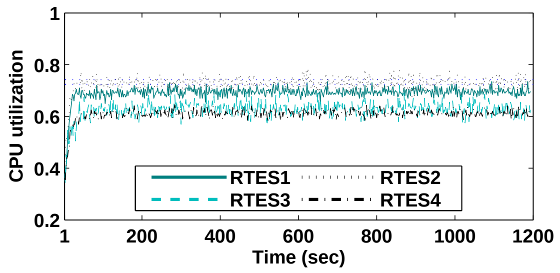
Figure 5.5: Comparison of power consumption between EUCON and the frequency scaling loop.

In this experiment, we show that rate adaptation may fail to control CPU utilizations in some cases, while the frequency scaling loop can be used for utilization control as an alternative way. We compare the frequency scaling loop with a baseline, a start-of-the-art control algorithm called EUCON [93], which relies only on the rate adaptation loop briefly introduced in Section 5.3. Figure 5.4a shows that EUCON fails to achieve the desired set points (0.74 for RTES2 and 0.72 for the other three servers) because the task rates saturate at the upper boundaries of their allowed ranges. As a result, the system is underutilized with unnecessarily high power consumption, as shown in Figure 5.5a. We then test the frequency scaling loop using the same workload with the rate adaptation loop disabled. In the experiment, to highlight the performance of the frequency scaling loop, we first let the system run in an open-loop manner (with no controller activated). Therefore, the system initially cannot achieve the desired CPU utilizations. At time 400s, we activate the frequency scaling loop. Figure 5.4b shows that the CPU utilizations quickly converge to their desired set points. As a result, all the servers achieve power savings (as shown in Figure 5.5b) while still guaranteeing the end-to-end task schedulability.

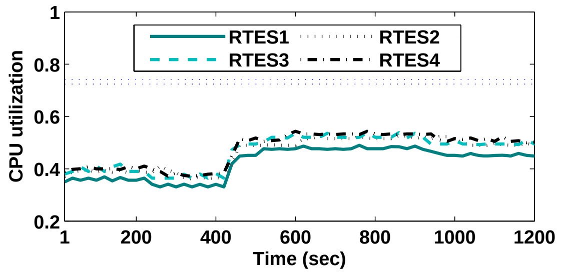
Coordinated Utilization Control

Since both task rates and CPU frequencies can only be adapted within allowed ranges, our coordinated control solution is designed to combine them based on control theory for maximized adaptation capability. In this experimenet, we run the same workload with all the tasks starting with lower initial rates. As a result, Figure 5.6a shows that the utilizations controlled by the rate adaptation loop start from values lower than those in Figure 5.4a. Similar to Figure 5.4a, the rate adaptation loop fails to achieve the desired utilization set points (dashed lines in the figure) because tasks are already running at their highest possible rates allowed by their ranges. In this case, the CPU frequencies of the processors could be lowered for power savings. We then examine the frequency scaling loop by running the same experiment in Section 5.6 with lower initial task rates. Figure 5.4b shows that the frequency scaling loop fails to achieve the desired utilizations this time because the tasks are running at lower rates. As a result, even when the processors are already running at their lowest CPU frequencies, utilizations still cannot converge to the desired set points. In this case, we could allow tasks to run at higher rates to contribute a higher value to the system.

We now evaluate our coordinated control solution. To highlight the performance of our solution, we first run the rate adaptation loop, which achieves the highest rates for all the tasks, resulting in a high system value. At time 400s, we activate the frequency scaling loop. Figure 5.7a shows that the coordinated control solution successfully achieves the desired utilization set points. In the meantime, Figure 5.7b demonstrates that servers RTES2, RTES3, RTES4 also receive considerable power savings. Therefore, the coordinated control solution can effectively control CPU utilizations to desired set points while achieving increased task rates and reduced power consumption.

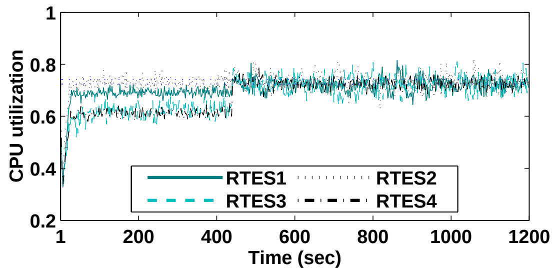


(a) Rate adaptation

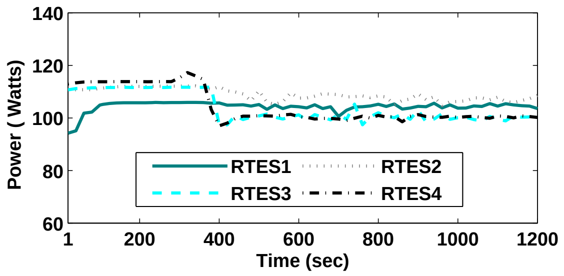


(b) Frequency scaling

Figure 5.6: Infeasible utilization control by frequency scaling or EUCON separately.



(a) CPU utilization



(b) Power consumption

Figure 5.7: Result of the coordinated utilization control solution.

Chapter 6

Temperature Control for Distributed Real-Time Systems

6.1 Coordinated Control Solution

In this section, we introduce our task model and the coordinated control architecture.

6.1.1 Task Model

We adopt an end-to-end task model [89] implemented by many DRE applications. A system is comprised of m periodic tasks $\{T_i|1 \leq i \leq m\}$ executing on n processors $\{P_i|1 \leq i \leq n\}$. Task T_i is composed of a set of sub-tasks $\{T_{ij}|1 \leq j \leq n_i\}$ which may be located on different processors. A processor may host one or more sub-tasks of a task. The release of subtasks is subject to precedence constraints, i.e., subtask $T_{ij}(1 < j \leq n_i)$ cannot be released for execution until its predecessor subtask T_{ij-1} is completed. All the subtasks of a task share the same rate. The rate of a task (and all its subtasks) can be adjusted by changing the rate of its first subtask. If a non-greedy synchronization protocol (e.g., release guard [129]) is used to enforce the precedence constraints, every subtask are released periodically without jitter.

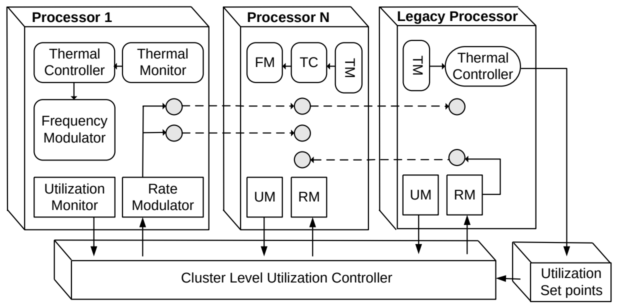


Figure 6.1: Coordinated Control Architecture

In our task model, each task T_i has a *soft* end-to-end deadline related to its period. In an end-to-end scheduling approach [129], the deadline of an end-to-end task is divided into subdeadlines of its subtasks. Hence the problem of meeting the end-to-end deadline can be transformed to the problem of meeting the subdeadline of each subtask. A well known approach for meeting the subdeadlines on a processor is to ensure its utilization remains below its schedulable utilization bound [89].

Our task model has two properties. First, while each subtask T_{ij} has an *estimated* execution time c_{ij} available at design time. Second, the rate of a task T_i may be dynamically adjusted within a range $[R_{min,i}, R_{max,i}]$. This assumption is based on the fact that the task rates in many applications (e.g., digital control, sensor update, and multimedia) can be dynamically adjusted without causing system failure. The rate ranges are determined by the applications (e.g., the limited sampling frequency of a sensor). A task running at a higher rate contributes a higher value to the application at the cost of higher utilizations.

6.1.2 Control Architecture

As shown in Figure 6.1, our coordinated solution includes a cluster-level utilization control loop and a thermal control loop on each processor. For processors with DVFS supports, the thermal controller manipulate the CPU frequency. In contrast, for legacy processors, the thermal controller manipulate the set points of the cluster-level utilization controller and set points changes cause task rates changes.

The cluster-level utilization control loop dynamically controls the utilizations of all the processors by adjusting task rates within their allowed ranges. Because the rate change of a task affects the utilizations of all the processors where the task has subtasks, this loop is a Multi-Input-Multi-Output (MIMO) control loop, which works as follows: (1) the utilization monitor on each processor P_i sends its utilization $u_i(k)$ in the last control period to the cluster-level controller; (2) the controller computes a new rate $r_j(k)$ for every task T_j and sends the new rates to the rate modulators; and (3) the rate modulators change the task rates accordingly.

On every processor P_i in the system, we have a local controller that controls the processor temperature. Two thermal controllers for two kinds of processors are both Single-Input-Single-Output (SISO) controllers because we assume the CPU frequency change or task rates change of P_i only affects the temperature of P_i. This is usually true because different processors in a DRE system may locate in different places. The DVFS based thermal loop works as follows: (1) the thermal monitor on P_i sends its temperature $t_i(k)$ to the local thermal controller; (2) the controller computes a new DVFS level $f_i(k)$ and sends it to the frequency modulator on P_i; and (3) the frequency modulator changes the processor DVFS accordingly. The rate adaptation based thermal loop works as follows: (1) the thermal monitor on P_i sends its temperature $t_i(k)$ to the local thermal controller; (2) the controller computes a new utilization set point $s_i(k)$ and sends it to cluster-level utilization controller; and (3) cluster-level utilization controller changes rates of tasks on every processor accordingly.

It is clear that without effective coordination, the thermal control loops may conflict with the utilization control loop. For example, the DVFS based thermal control loop relies on DVFS to control processor temperature. DVFS can significantly impact the execution times of the real-time tasks running in the systems and even cause the execution times to vary outside their stability ranges. As a result, the timeliness guarantees provided by the utilization control loop can be severely violated. On the other side, the CPU utilization changes made by the cluster-level utilization control loop may also impact the temperatures of multiple processors. Therefore, the coordination of control loops must be considered. For legacy processors, we decouple the rate adaptation loop and the utilization loop. For processors with DVFS supports, the coordination is designed based on robust control theory to improve control performance, *i.e.*a short settling time.

Since the core of the two control loops is a thermal controller based on DVFS, we introduce its design and analysis in the next section.

6.2 Utilization Control Loop

In this section, we briefly introduce the system model and design of the cluster-level utilization control loop.

6.2.1 System Modeling

We now establish a dynamic model that characterizes the relationship between the controlled variable $\mathbf{u}(k)$ and the manipulated variable $\mathbf{r}(k)$. We first model the utilization $u_i(k)$ of one processor P_i. The *estimated* utilization change $\Delta b_i(k)$ of P_i in the k^{th} control period can be modeled as a function of the execution times of all the subtasks on P_i and their rate changes $\Delta r_j(k) = r_j(k) - r_j(k-1)$.

$$\Delta b_i(k) = \sum_{T_{jl} \in S_i} c_{jl} \Delta r_j(k) \tag{6.1}$$

where S_i is the set of subtasks located at processor P_i.

$\Delta b_i(k)$ is based on the estimated execution time c_{jl}. Since the actual execution times may be different from their estimation due to workload variations, we model the actual utilization of P_i, $u_i(k)$, as the following difference equation.

$$u_i(k+1) = u_i(k) + g_i \Delta b_i(k) \tag{6.2}$$

where the utilization gain g_i represents the ratio between the change to the actual utilization and its estimation $\Delta b_i(k)$. For example, $g_i = 2$ means that the actual change to utilization is twice the estimated change. Note that the exact value of g_i is *unknown* at design time due to the unpredictability of subtasks' execution times.

Note that in (6.2), we assume that the relative CPU frequency of P_i is 1, which means that the processor is running at its highest CPU frequency. However, since the thermal controller on P_i may use DVFS to control the processor temperature, the relative CPU frequency can become smaller than 1 at runtime. The impact of the thermal controller on the utilization control loop is analyzed in Section 6.5. Based on (6.2), a DRE system with m tasks and n processors is described by the following MIMO dynamic model.

$$\mathbf{u}(k) = \mathbf{u}(k-1) + \mathbf{G}\mathbf{\Delta b}(k-1) \tag{6.3}$$

where $\mathbf{G}$ is a diagonal matrix where $g_{ii} = g_i$ $(1 \le i \le n)$ and $g_{ij} = 0$ $(i \ne j)$. $\mathbf{\Delta b}(k)$ is a vector including the estimated utilization change (6.1) of each processor. The relationship between the utilization and task rates is characterized as follows:

$$\mathbf{\Delta b}(k) = F\Delta r(k) \tag{6.4}$$

The subtask allocation matrix, F, is an $n \times m$-order matrix, where $f_{ij} = c_{jl}$ if subtask T_{jl} (the lth subtask of task T_j) is allocated to processor i, and $f_{ij} = 0$ if no subtask of task T_j is allocated to processor i.

6.2.2 Controller Design

In this dissertation, we adopt the EUCON algorithm presented in our previous work [93] for utilization control. EUCON features a Model Predictive Controller (MPC) that optimizes a *cost function* defined over P control periods in the future, called the *prediction horizon*. The control objective is to select control inputs in the following

M control periods, called *control horizon*, which minimizes the following cost function while satisfying the constraints.

$$\begin{aligned} V(k) = & \textstyle\sum_{i=1}^{P} \|\mathbf{u}(k+i|k) - \mathbf{ref}(k+i|k)\|^2 \\ & + \textstyle\sum_{i=0}^{M-1} \|\mathbf{\Delta r}(k+i|k) - \mathbf{\Delta r}(k+i-1|k)\|^2 \end{aligned} \tag{6.5}$$

The first term in the cost function represents the *tracking error*, i.e., the difference between the utilization vector $\mathbf{u}(k+i|k)$ and a reference trajectory $\mathbf{ref}(k+i|k)$ defined in [93]. By minimizing the tracking error, the closed-loop system will converge to the utilization set points if the system is stable. The second term in the cost function represents the control penalty. This control problem is subject to the task rate constraints . The detailed design and analysis of EUCON are available in [93].

Although the utilization control loop is proven to be stable in [93], in order for the coordinated solution to be stable, the stability and utilization control loop need to be reexamined with the impact from the thermal control loop. The coordination analysis is presented in Section 6.5.

6.3 Thermal Controller based on DVFS

In this section, we model, design, and analyze the thermal control loop based on DVFS.

6.3.1 System Model

We now model the temperature of a processor P_i. We first introduce some notation. T_s is the control period. $t_i(k)$ is the temperature of P_i in the k^{th} control period. $f_i(k)$ is the DVFS level of P_i in the k^{th} control period. $d_i(k)$ is the difference between $f_i(k)$ and $f_i(k-1)$, i.e., $d_i(k) = f_i(k) - f_i(k-1)$. $p_i(k)$ is the power consumption of P_i in the k^{th} control period. The control goal is to guarantee that $t_i(k)$ converges to the temperature set point in a finite settling time.

We use two steps to model the relationship between $t_i(k)$ and $f_i(k)$. In the first step, we analytically model the relationship between $t_i(k)$ and $p_i(k)$. In the second step, we model the relationship between $p_i(k)$ and $f_i(k)$.

First, since DVFS changes the frequency of the entire processor chip, we adopt a chip-level thermal model called resistor-capacitor model (RC-model) [121] to model the the processor temperature. To convert the thermal model in the continuous time domain to a model in the discrete time domain, the sampling rate (i.e., control period T_s) must be selected carefully to guarantee the precision of the discrete model. In this dissertation, we select the sampling rate less than the thermal time constant in the second-order circuits. Based on the model in [121], our thermal model is:

$$\Delta t_i(k) = \frac{p_i(k) \cdot T_s}{C_i} - \frac{t_i(k) \cdot T_s}{R_i \cdot C_i} \tag{6.6}$$

where $\Delta t_i(k) = t_i(k+1) - t_i(k)$. C_i and R_i are processor P_i's thermal capacitance and thermal resistance, respectively. Note that C_i and R_i are determined by the thermal characteristics of the CPU packaging of P_i and the cooling system. The temperature is related to the ambient temperature T_a. We can transform (6.6) to the following difference equation:

$$t_i(k+1) = (1 - \frac{T_s}{R_i C_i}) \cdot t_i(k) + \frac{p_i(k) \cdot T_s}{C_i} \tag{6.7}$$

In the second step, we model the relationship between processor power consumption $p_i(k)$ and $f_i(k)$. It is well-known that DVFS can allow cubic reductions in power density relative to performance loss in a processor [122]. However, a cubic power model may lead to high complexity for controller design and large runtime overhead. On the other hand, real processors usually only provide a limited DVFS range. Within the small range, previous studies [110, 136] have shown that the relationship between power and DVFS level can be approximated with a linear function.

$$p_i(k) = A_i f_i(k) + B_i \tag{6.8}$$

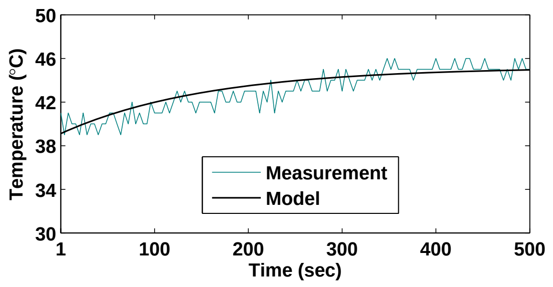


Figure 6.2: Model Prediction vs. Measurement

where A_i and B_i are generalized parameters that may vary for different processors. To determine the values of A_i and B_i, we can use a standard approach to this problem called system identification [148]. In this dissertation, we use a typical real-time system CPU computation intensive workload presented in [139] for system identification.

We now substitute (6.8) into (6.7) to establish relationship between $t_i(k)$ and $f_i(k)$. The dynamic model of the system as a difference equation is:

$$t_i(k) = (1 + \Theta_i)t_i(k-1) - \Theta_i t_i(k-2) + \Psi_i d_i(k-1) \tag{6.9}$$

where $\Theta_i = (1 - \frac{T_s}{R_i \cdot C_i})$ and $\Psi_i = \frac{T_s}{C_i} \cdot A_i$.

We then use a step-like signal to validate our system model (6.9) on our physical testbed. Figure 6.2 demonstrates that the predicted output of the our model is sufficiently close to the measured actual system output.

6.3.2 Controller Design

Following standard control theory [148], we design a Proportional (P) controller to achieve desired control performance such as stability and zero steady state error. We choose to use a P controller instead of a more sophisticated controller such as a PID (Proportional-Integral-Derivative) controller because the actuator $f_i(k) = d_i(k) + f_i(k-1)$ already includes an integrator such that zero steady state error can be achieved without resorting to an I (Integral) part. The D (Derivative) part is not used because it may amplify the noise in temperature introduced by measurement. The Z-domain form of our P controller is:

$$C(z) = \frac{1}{\Psi_i} \tag{6.10}$$

The transfer function of the closed-loop system controlled by controller (6.10) is:

$$\frac{T(z)}{D(z)} = \frac{z}{z^2 - \Theta_i z + \Theta_i} \tag{6.11}$$

It is easy to prove that the controlled system is stable and has zero steady state errors when the system model (6.9) is accurate. The detailed proofs can be found in a standard control textbook [148] and are skipped due to space limitations.

6.3.3 Control Analysis for Model Variation

In this subsection, we analyze the system stability when the system model (6.9) varies for different processors. A fundamental benefit of the control-theoretic approach is that it gives us theoretical confidence for system stability, even when the controller is used in a different working condition.

A different processor usually has different thermal resistance and capacitance R_j and C_j, where $R_j \neq R_i$ and $C_j \neq C_i$. Therefore, even though the designed P controller in (6.10) is proven to be stable on the processor used to derive the nominal system model (6.9) by system identification, the system stability when the controller is used on a different processor must be theoretically reevaluated.

We now outline the detailed steps to analyze the stability when the system model changes for different processors.

1. We first get the actual system model of a different processor by conducting automated system identification on the processor. Since the value of Θ_i is determined by the thermal resistance and capacitance R_i and C_i, Θ_i will be

different for a different processor. Therefore, the actual system model is in the following format:

$$t_i(k) = (1 + g\Theta_i)t_i(k-1) - g\Theta_i t_i(k-2) + \Psi_i d_i(k-1) \tag{6.12}$$

where $g\Theta_i$ is the actual parameters that may be different from Θ_i in the nominal model (6.9).

2. The controller function $C(z)$ presented in (6.10) represents the control decision made based on the nominal model (6.9). We then derive the closed-loop system transfer function by plugging the controller into the actual system. The closed-loop transfer function represents the system response when the controller is applied to a system whose model is different from the one used to design the controller. The closed-loop transfer function is:

$$\frac{T(z)}{D(z)} = \frac{z}{z^2 - g\Theta_i z + g\Theta_i} \tag{6.13}$$

3. Finally, we derive the stability condition of the closed-loop system (6.13). According to control theory, the closed-loop system is stable if all the poles of (6.13) locate inside the unit circle in the complex space. The poles are calculated as the roots of the denominator in (6.13), *i.e.*, the following equation:

$$z^2 - g\Theta_i z + g\Theta_i = 0 \tag{6.14}$$

The stability condition of applying the controller designed based on the nominal model (6.9) to a processor with a different system model can be stated as: if the roots of (6.14) all locate inside the unit circle in the complex space, the controlled system is stable. We have developed a script to analyze system stability automatically using numerical methods.

6.4 Thermal Controller based on Rate Adaptation

In this section, we model, design, and analyze the thermal control loop based on rate adaptation.

6.4.1 System Model

We now model the temperature of a processor P_i. We first introduce some notation in addition to notation in Section 6.3.1. T_s' is the control period. $s_i(k)$ is the utilization set point of P_i in the k^{th} control period. $ds_i(k)$ is the difference between $s_i(k)$ and $s_i(k-1)$, i.e., $ds_i(k) = s_i(k) - s_i(k-1)$.

We use two steps to model the relationship between $t_i(k)$ and $s_i(k)$. In the first step, we use the same the relationship between $t_i(k)$ and $p_i(k)$ as (6.7). In the second step, we derive the relationship between $p_i(k)$ and $s_i(k)$.

Based on [46], $p_i(k)$ and $s_i(k)$ can be modeled as

$$p_i(k) = P_{idle} + (P_{busy} - P_{idle})[2s_i(k) - s_i(k)^r] \tag{6.15}$$

Where r is a constant.

However, a nonlinear power model may lead to high complexity for nonlinear controller design, large runtime overhead and extreme difficulty of analysis of control performance. On the other hand, utilization set point range is a subset of $[0, 1]$ which is very limited. Within the small range, we use the linearization method in nonlinear systems theory to approximate with a linear function [72].

$$p_i(k) = s_{ia} + (P_{busy} - P_{idle})(2 - r{s_{ia}}^{r-1})[s(k) - s_{ia}] \tag{6.16}$$

where P_{busy}, P_{idle} and r are parameters characterizing a power model of a server that may vary for different processors. The values can be determined by real measurements. s_{ia} is chosen between interval $[0, 1]$ to make the approximation close enough to the nonlinear model.

Let $A_i = (P_{busy} - P_{idle})(2 - r{s_{ia}}^{r-1})$ and $B_i = [1 - (P_{busy} - P_{idle})(2 - r{s_{ia}}^{r-1})]s_{ia}$. (6.16) becomes the following:

$$p_i(k) = A_i s_i(k) + B_i \tag{6.17}$$

We now substitute (6.17) into (6.7) to establish relationship between $t_i(k)$ and $s_i(k)$. The dynamic model of the system as a difference equation is:

$$t_i(k) = (1 + \Theta_i)t_i(k-1) - \Theta_i t_i(k-2) + \Psi_i d s_i(k-1) \tag{6.18}$$

where $\Theta_i = (1 - \frac{T'_s}{R_i \cdot C_i})$ and $\Psi_i = \frac{T'_s}{C_i} \cdot A_i$.

6.4.2 Controller Design

Since that (6.17) has the same form as (6.8), we apply methods of Section 6.3.2 and Section 6.3.2. We design a Proportional (P) controller to achieve desired control performance such as stability. The Z-domain form of our P controller is:

$$C(z) = \frac{1}{\Psi_i} \tag{6.19}$$

To work together with the cluster-level utilization controller, the thermal controller needs to constrain it control input. Detail analysis is in Section 6.5.1.

6.4.3 Control Analysis for Model Variation

In Section 6.3.3, we already have analyzed stability under thermal model parameters variation for the thermal controller based on DVFS. The result can be extended to thermal controller based on rate adaptation. In this section, we focus on a different processor which has different A_i. Usually, different processor has different power model parameters i.e. P_{busy} and P_{idle}. Therefore, even though the designed P controller in (6.19) is proven to be stable on the processor used to derive the nominal system model (6.18), the system stability when the controller is used on a different processor must be theoretically reevaluated.

We now outline the detailed steps to analyze the stability when the system model changes for different processors.

1. Since that Bi is not in our dynamic model (6.18), the value of Θ_i is determined by the A_i, Θ_i will be different for a different processor power model. Therefore, the actual system model is in the following format:

$$t_i(k) = (1 + \Theta_i)t_i(k-1) - \Theta_i t_i(k-2) + g\Psi_i ds_i(k-1) \tag{6.20}$$

   where $g\Psi_i$ is the actual parameters that may be different from Ψ_i in the nominal model (6.18).

2. We then derive the closed-loop system transfer function by plugging the controller into the actual system. The closed-loop transfer function is:

$$\frac{T(z)}{D(z)} = \frac{gz}{z^2 - (1 + \Theta_i - g)z + \Theta_i} \tag{6.21}$$

3. Finally, we derive the stability condition of the closed-loop system (6.21). The closed-loop system is stable if all the poles of (6.21) locate inside the unit circle in the complex space. The poles are calculated as the roots of the denominator in (6.21), *i.e.*, the following equation:

$$z^2 - (1 + \Theta_i - g)z + \Theta_i = 0 \tag{6.22}$$

The stability condition of applying the controller designed based on the nominal model (6.18) to a processor with a different system model can be stated as: if the roots of (6.22) all locate inside the unit circle in the complex space, the controlled system is stable.

6.5 Coordination Analysis

6.5.1 Coordinate Thermal Controller based on Rate Adaptation

We now analyze the coordination needed for the thermal controller based on rate adaptation and a cluster-level utilization controller to work together with global stability. The analysis here and later, as well as the control architecture design in Section 6.1 and our empirical and simulation results, demonstrates the importance of coordinating different control loops, which is one of major contributions of our paper.

First, we need to ensure that the function of the cluster-level utilization controller will not be affected when the thermal control loop changes the CPU utilization set point. Given a specific task set, to guarantee timing, the set points of cluster-level utilization controller need to be configured according to the number of subtasks on each servers. The set points can be established by following formula presented in [89]. The set points derived from [89] is a upper bound. Since the lowest task rates of each tasks, a lower set points bound exists too. Accordingly, we must guarantee that the manipulated variable of the thermal controller of each processor operates in the range. This constraint must be enforced to guarantee timing and avoid control input saturation.

Second, we need to analyze the settling time of the cluster-level utilization controller in order to determine the control period of the thermal loop. Since settling time has not been analyzed in [93], we now outline the general process of analyzing the settling time of the cluster-level utilization controller. First, given a specific task set, we derive the control inputs $\Delta r(k)$ that minimize the cost function 6.5 based on the system model 6.4 with $g_i = 1$. The control inputs represent the control decision based on the estimated system model. Second, we derive the closed-loop system model by substituting the control inputs derived in the first step into the system model 6.4. The analysis needs to consider a composite system consisting of the dynamics of the original system and the controller. Finally, we calculate the dominant pole (i.e., the pole with the largest magnitude) of the closed-loop system. According to control theory, the dominant pole determines the system's transient response such as settling time. Based on our analysis, the task set used in our experiments has a settling time of 17 control periods under the cluster-level utilization control. The detailed derivation is not included due to space limitations. The control period of the rate adaptation loop is selected to be 1 seconds to include multiple instances of each task, resulting in a settling time of 17 seconds Therefore, the control period of the thermal loop is set to 20 seconds, which is longer than the settling time of the the cluster-level utilization loop.

6.5.2 Coordinate Thermal Controller based on DVFS

We now analyze the coordination needed for the utilization and thermal control loops to work together. A major contribution of this dissertation is to demonstrate the importance of a novel methodology for coordinating different control loops. Both the utilization and thermal control loops have been proven to be stable in previous sections. If both the two control loops are still stable under the impact from the other loop, the entire system is stable.

The analysis of the impact of one loop on the other loop is similar to the stability analysis of a control loop with an actual system model that is different from its nominal model. If the actual model is known, we can analyze stability by examining whether all the poles of the closed-loop system locate inside the unit circle in the complex space. However, in a real DRE system, the actual system model may vary significantly at runtime in an unpredictable way. Therefore, we adopt robust control theory to derive the stability condition for a given DRE system. The differences or errors between the actual system model and the nominal model are referred to as *uncertainty* in robust control theory. The main advantage of robust control is that uncertainty is considered explicitly in the stability analysis of a feedback control system. This characteristic makes robust control well suitable for analyzing the stability of a control loop when it is under the impact from another loop.

We have performed the above coordination analysis procedures for the DRE system deployed on our testbed and evaluated in our experiments. Our results show that both the two control loops are stable even under the impacts from each other, and thus the entire DRE system is stable. If a system is found to be unstable, workload and platform reconfigurations can be tuned to adjust the system for stability according to derived robust stability conditions.

6.6 System Implementation

In this section, we introduce our hardware testbed, simulation, workload, and the implementation details of the control loops.

6.6.1 Testbed and Workload

Our hardware testbed includes four Linux servers (RTES1 to RTES4) running end-to-end real-time tasks and a desktop machine running the cluster level utilization controller. Four servers are equipped with 2.4GHz AMD Athlon 64 3800+ processors with 1GB RAM and 512KB L2 Cache. The desktop machine is a Dell OptiPlex GX520 with 3.00GHz Intel Pentium D Processor and 1GB RAM. All machines are connected by a 100Mbps Ethernet switch. The controller machine runs Windows XP while all other servers run openSUSE 11 and the Linux kernel is 2.6.25 with real-time support.

We implement our control architecture in FC-ORB, an open-source real-time Object Request Broker (ORB) middleware system [139]. FC-ORB supports end-to-end real-time tasks based on the end-to-end scheduling framework [89]. FC-ORB implements the release guard protocol to enforce the precedence constraints among subtasks.

We will evaluate our solution on legacy hardware platforms with no DVFS support and a large scale cluster with 20 servers. Since hardware resources are not available, we use simulation instead for the evaluations. Our simulation environment is composed of an event driven simulator implemented with 2590 lines C++ code. The simulator implements the cluster-level utilization controller and both type thermal controllers.

Our hardware experiments run a medium-sized workload that comprises 12 end-to-end tasks (with a total of 25 subtasks). The subtasks on each processor are scheduled by the RMS algorithm [89]. Each task's end-to-end deadline is $d_i = n_i/r_i(k)$, where n_i is the number of subtasks in task T_i and $r_i(k)$ is the current rate of T_i. Each end-to-end deadline is evenly divided into subdeadlines for its subtasks. The resultant subdeadline of each subtask T_{ij} equals its period, $1/r_i(k)$. The initial utilization set point of every processor is set to its RMS schedulable utilization bound [89], i.e., $B_i = n_i(2^{1/n_i} - 1)$, where n_i is the number of subtasks on P_i. All (sub)tasks meet their (sub)deadlines if the desired utilization on every processor is enforced.

6.6.2 Control Components

We now introduce the implementation details of key components in the coordinated solution.

Utilization Controller: The controller is implemented as a single-thread process running separately on the desktop machine. Each time its periodic timer fires, the controller sends utilization requests to all processors in the cluster. The incoming replies are handled asynchronously so that the controller can avoid being blocked by an overloaded processor. After the controller collects replies from all processors, it executes the control algorithm introduced in Section 6.2 to compute new task rates. The controller then sends the tasks' new rates to the rate modulators on processors for enforcement. If a processor does not reply in an entire control period, its utilization is treated as 100%, as the controller assumes this processor is overloaded with its (sub)tasks and so cannot respond. The control period of the utilization loop is 4 seconds.

Rate Modulator: A Rate Modulator is located on each processor. It receives the new rates from the controller and then resets the timer interval of the first subtask of each task whose invocation rate has been changed.

Temperature Monitor: AMD processors have built-in circuits to measure the chip temperature. Two most common types of circuits are thermal diode and on-die digital thermometers. The thermal diodes are normally placed close to the maximum temperature spots (*i.e.*, hot spots) of an AMD chip [83]. The thermal values can be accessed via the Machine Specific Register (MSR) by user-mode applications. In this dissertation, we use the utility functions from the *lm-sensors* project [88], which provide a uniform user interface to monitor a wide range of processors.

Thermal Controller Based on DVFS: The controller is implemented as a single-thread process running on each processor. With a control period of 4 seconds, the controller periodically reads the temperature of the processor, executes the control algorithm presented in Section 6.3.2 to compute the desired CPU frequency, and sends the new frequency to the frequency modulator on the processor.

Since the new CPU frequency level periodically received from the proportional controller could be any value that is not exactly one of the five supported frequency levels. Therefore, the modulator code must locally resolve the output value of the controller to a series of supported frequency levels to approximate the desired value. For example, to approximate 2.89GHz during a control period, the modulator would output the sequence 2.67, 3, 3, 2.67, 3, 3, etc on a smaller timescale. To do this, we implement a first-order delta-sigma modulator, which is commonly used in analog-to-digital signal conversion. The detailed algorithm of the first-order delta-sigma modulator can be found in [82].

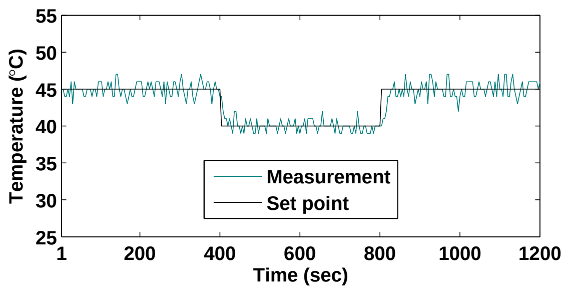


Figure 6.3: Thermal Controller

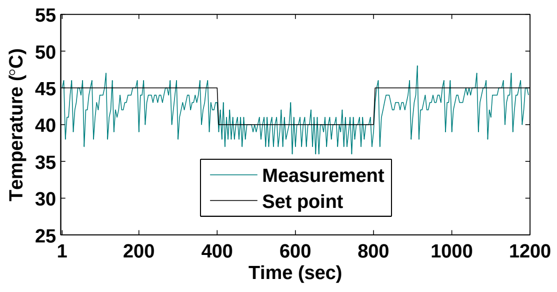


Figure 6.4: Ad Hoc

6.6.3 Simulations and Workload

We implement our control architecture in a home-grown event-driven simulator. In our simulations, we consider a Pentium 4 processor with Northwood core. The parameters of the thermal and power models are based on Intel Technical Documents [68] and [53]. For the parameters in the power model 6.17, $A_i = 38.6, B_i = 13.3$ and CPU utilization range is $[0, 1]$. For the parameters in the thermal model 6.18, $R_i = 0.467, C_i = 295.7$.

The synthetic real time applications are randomly generated using a Perl script with each task set contains 4 or 20 tasks (with a total of 12 or 60 subtasks). The controllability of the task set randomly generated is checked using algorithms in [140].

6.7 Empirical Results

In this section, we first evaluate the DVFS based thermal controller alone by comparing it with a commonly used ad hoc solution. We then test the coordinated DVFS based thermal controllers and an utilization controller in the case of thermal variations and examine the coordinated solution under task execution time variations. Finally, we test our coordinated solution in the case of a large-scale heterogeneous cluster.

We use two baselines for comparison in this dissertation. OPEN is a typical open-loop solution that configures the task rates and processor DVFS levels in a static way. While OPEN can initially achieve the desired CPU utilizations and processor temperatures, OPEN may fail when task execution times or system conditions dynamically change at runtime. Ad Hoc represents a commonly used solution to thermal control of a processor. When the current processor temperature is lower than the set point, Ad Hoc will increase the processor's DVFS level by one. When the temperature is lower than the set point, Ad Hoc sets the DVFS level to the lowest one to avoid overheating. A fundamental difference between Ad Hoc and our thermal controller is that Ad Hoc simply raises the DVFS level by one step or sets it to the lowest level, depending on whether the measured temperature is lower or higher than the set point. In contrast, our thermal controller computes a fractional DVFS level based on well-established control theory and uses the frequency modulator to approximate this output with a series of discrete DVFS levels.

Thermal Controller

In this experiment, we disable the utilization control loop to evaluate the performance of the thermal controller on RTES1. The temperature set point is initially 45°C in our experiment. Between time 400s and 800s, we reduce the set point to 40°C to emulate a thermal emergency event. As shown in Figure 6.3, under our thermal controller, the measured processor temperature converges to the desired level promptly after the set point is changed. Despite the measurement noise from

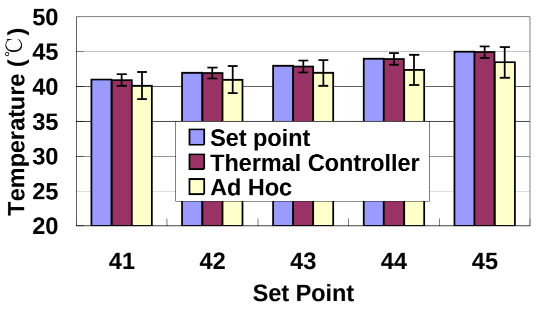
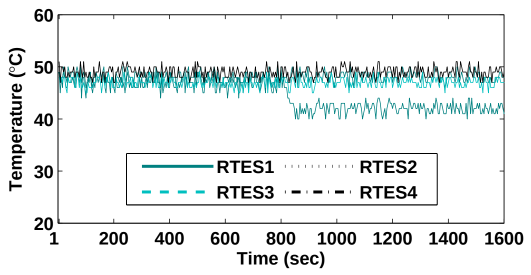


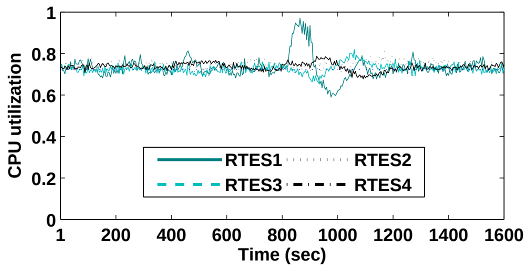
Figure 6.5: Comparison of thermal controller and Ad Hoc under different temperature set points

the processor thermometer, the thermal controller allows the temperature to stay very close to the set point by dynamically throttling the processor DVFS level. In contrast, Figure 6.4 shows that Ad Hoc causes the processor temperature to oscillate dramatically because Ad Hoc simply raises the DVFS level by one step or sets it to the lowest level. As neither of the two temperature set points (i.e., 45°C and 40°C) can be exactly achieved by the processor by staying at any of the several available DVFS levels, Ad Hoc has to continuously throttle the processor DVFS level around a set point. As a result, the average processor temperature under Ad Hoc cannot settle to the set point, leading to a steady-state error, as shown in Figure 6.4.

Figure 6.5 compares the processor temperature achieved by the thermal controller and Ad Hoc under different set points from 41°C to 45°C. The average temperatures and the standard deviations are calculated based on the measured temperature readings when the controllers enter their steady states. The thermal controller has much smaller steady-state errors and also smaller deviations compared to Ad Hoc. Note that the smaller steady-state errors can contribute to higher processor frequencies and thus better system performance (e.g., higher task rates). In addition, if the thermal controller is given an unreasonably high set point, the P controller will saturate at the highest DVFS level and thus allow the system to run at its peak performance. The experiments demonstrate that our thermal controller designed based on control theory outperforms a commonly used thermal control solution by having more accurate thermal control.

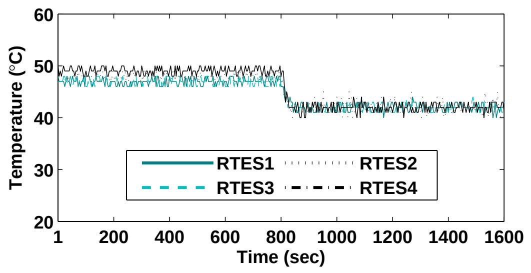


(a) Temperatures of the four processors

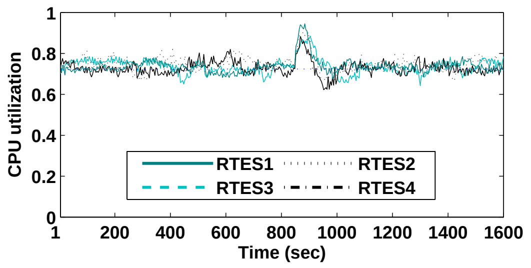


(b) CPU utilizations of the four processors

Figure 6.6: Thermal variation on a single processor (RTES1)



(a) Temperatures of the four processors



(b) CPU utilizations of the four processors

Figure 6.7: Thermal variations on all the four processors

Thermal Variations

In this experiment, we enable both the utilization controller and the DVFS based thermal controller to examine the simultaneous thermal and timeliness guarantees in two scenarios.

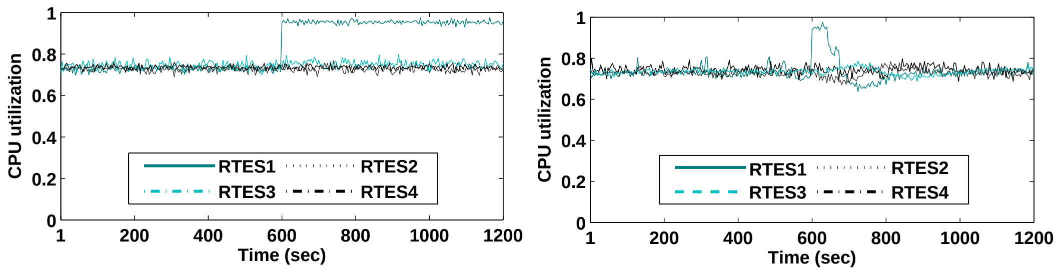
In the first scenario, the temperature set point of a single processor is changed from 47°C to 42°C at 800s to emulate a local thermal emergency event. Although the experiment in Section 6.7 has shown that our thermal controller can achieve the desired new temperature set point by conducting DVFS, the lowered CPU frequency may increase the execution times of the real-time tasks in the system and thus cause deadline misses if there is no control for task timeliness. Figure 6.6a shows that the thermal controller on RTES1 can precisely achieve the desired new temperature set point by reducing the CPU frequency of RTES1. As a result, Figure 6.6b shows that the CPU utilization of RTES1 rises to nearly 100% and so violates the schedulable utilization bound. With effetive utilization control, the coordinated control solution reduces the task rates to lower the CPU utilization of RTES1 to the desired set point. Other processors in the system also have small utilization variations because rate adaptation of a task affects all its subtasks running on multiple processors.

In the second scenario, the temperatures of all the processors are changed from 47°C to 42°C at 800s to emulate a global thermal emergency event. Figure 6.7a shows that the thermal controllers on all the processors successfully lower their processor temperatures to the new set point by reducing CPU frequencies of the processors. As a result, Figure 6.7b shows that all the processors have significant increases of CPU utilization. The coordinated control solution adjusts the rates of the end-to-end tasks in the system at the cluster level to lower the CPU utilizations of all the processors to the desired set point. The two experiments demonstrate that the coordinated control solution can provide simultaneous thermal and timeliness guarantees when the system has either a local or a global thermal emergency event.

Task Execution Time Variations

In this experiment, we examine the simultaneous thermal and timeliness guarantees when the execution times of the subtasks on RTES1 have a 25% of increase at 600s. Unpredictable execution time increases may cause the system to violate its utilization bounds, resulting in deadline misses. In addition, execution time variations may also increase system temperature and cause thermal emergency if the processor stays overloaded for a long time.

We first examine the performance of OPEN, which configures task rates and processor DVFS levels in a static way. While OPEN can initially achieve the desired utilizations and temperatures, Figure 6.8a shows the utilization of RTES1 increases to 95% at 600s and stays above the utilization bound in the rest of the run. Figure 6.9 shows that the temperature of RTES1 is consequently higher than the desired set point. Due to the lack of adaptation, OPEN may cause system malfunctions and may even reduce the lifetime of the processor. In constrast, Figure 6.8b shows that the coordinated solution can effectively control the increased CPU utilizaton by conducting rate adapation. Figure 6.9 shows that the processor temperature of RTES1 has an instantaneous increase at 620s in response to the execution time increase at 600s. However, the coordinated solution immediately reduces the temperature to the desired set point by throttling CPU frequency. This experiment demonstrates that the coordinated control solution can provide simultaneous thermal and timeliness guarantees when task execution times vary at runtime.



(a) OPEN (b) Coordinated control solution

Figure 6.8: Variations of task execution times on a single processor (RTES1)

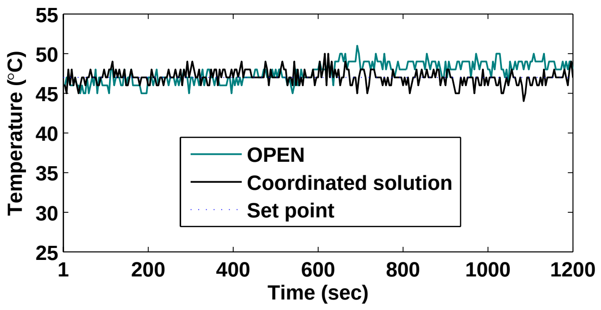


Figure 6.9: Comparison of system temperature under the coordinated control solution and OPEN

Chapter 7

Power Oversubscription in Data Centers

7.1 Background about Circuit Breaker

In a data center, groups of servers (racks) are powered from branch circuits. The branch circuits connect back to a panel box that receives power from a Power Distribution Unit (PDU). Inside the panel box there is a circuit breaker for each branch circuit. The National Electric Code (NEC) [105], used in the United States, limits the long-term power load on a circuit breaker to be 80% of the circuit breaker rating. This 80% power load represents the cap of the current power capping controllers used in industry. Therefore, a data center can, at least, safely oversubscribe the circuit breaker by 25% without causing the CB to trip, according to the above NEC rule. This can be a signficant benefit for data centers. For example, the Environmental Protection Agency (EPA) estimated that the data center power consumption has an annual increase of 9% [132]. In that case, the 25% increase in power oversubscription from power capping with an increased power cap would allow new data center construction costs, ranging in the hundreds of millions USD, to be deferred for approximately three years. However, the power cap cannot be simply raised in that way because if the power draw is not well controlled, unexpected workload variations may lead to power spikes that could trip the CB. On the other hand, if we can properly control the power draw based on the tripping characteristics of the CB, a data center can even further oversubscribe the CB without causing undesired shutdowns. Such controlled power oversubscription is an efficient way for a data center to host additional servers without significantly upgrading the power supply infrastructure. Therefore, safe power oversubscription is practical, low-risk, and financially attractive for data centers.

Generally, the majority of circuit breakers have two types of trip time behaviors which are specified in the UL489 standard. First, short-circuits (for example, over 500% of the rated load) cause the CB to trip within a few milliseconds. Second, overload conditions for a less severe current draw can trip the circuit breaker on a time scale from milliseconds to hours or even weeks, depending on the severity of the overload. Only the overload condition is relevant in this dissertation since practical uses of power oversubscription do not reach load levels sufficient to cause a short-circuit trip condition. Also, note that other devices in the power infrastructure, such as transformers and Uninterruptible Power Supplies (UPS), are also designed to tolerate overloads since fluctuations are common in power systems. Therefore, as long as the CB does not trip, power oversubscription should be safe for data centers. We discuss the impacts of power oversubscription on other devices in Section 7.3. In an overload condition (*i.e.*, an oversubscription beyond 25% above the NEC rating), the overload must be resolved before the trip time in the CB specification. For example, circuit breakers based on UL489 available from the Rockwell Automation exhibit trip times of more than 2 minutes when overloaded to 125% of the rated load (oversubscription of 56%).

Figure 7.1 shows the trip curve of the Rockwell Allen-Bradley 1489-A Industrial CB used in our experiments (at a temperature of $40°C$) [15]. Rockwell CBs are used in many data centers. Their trip curves follow the UL489 standard and are similar to Figure 7.1. This selected CB has a rated current of $I_n = 1A$. As shown in Figure 7.1, the trip curve of the CB is actually a band called the *tolerance band*. The area

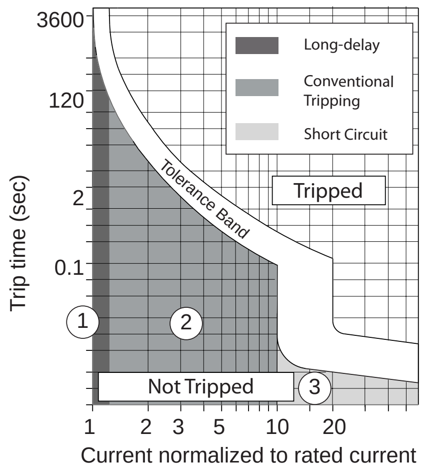


Figure 7.1: The trip curve of a typical circuit breaker.

Table 7.1: Test bed circuit breaker at 40°C

Current(A)	Measured trip time (sec)
slightly less than 1.35	¿ 7200
1.42	193
1.55	80
1.67	56
6.8	≈2
≈10	¡1

above the band is the *tripped* area, which means that the CB will trip if the duration of the CB current is longer than the specified trip time. The area below the band is the *not-tripped* area. This band represents the area where it is uncertain if the CB will trip. The lower and upper limits of the band are specified by the UL489 standard. The actual implementation is determined by the manufacturer [70]. The CB has three types of trip time behaviors that are shown as different regions on the tolerance band [15]. Region 1 is the long-delay tripping zone with the overload current as ($1I_n \leq I \leq 1.35I_n$). In this region, the CB trip time is minutes to hours to even days. Region 2 is the conventional tripping region ($1.35I_n < I \leq 10I_n$). Region 3 is the instantaneous tripping zone ($I > 10I_n$) that is designed to handle short-circuits. Table 7.1 shows that the actual measurements of CB trip time on our test bed for all three regions are consistent with the trip curve shown in Figure 7.1. While previous power capping solutions conservatively treat all the regions as the instantaneous tripping zone, a key contribution of our dissertation is to have different strategies for the different regions. As a result, we can fully utilize the long-delay tripping zone to safely boost server performance and host additional servers.

In order to fully utilize the long-delay tripping zone without tripping the CB, one has to ensure that the overload current is reduced to I_n before the trip time specified by the lower bound of the tolerance band. Based on this observation, we choose to design a power controller based on feedback control theory because recent studies (*e.g.*, [81, 120, 136]) show that control theory can provide quantitative analyses and guarantees for system stability and settling time (*i.e.*, the time for the overload current to return to I_n). A key difference between our work and existing studies is systematic analysis of the controller settling time. As shown in Figure 7.1, the allowed settling time increases whenever the overload current is reduced to a lower value. Therefore, to fully utilize the long trip time that continues to increase at each step, we propose to adopt adaptive control theory that can adjust the controller parameters based on the varying requirements of the settling time. Unlike previous power capping solutions that rely on a static power budget (*e.g.*, $0.8I_n$), our adaptive controller features a *dynamic power budget* that varies in every control period based on the overload current and its corresponding trip time. Ideally, the dynamic power budget can equal the lower bound of the tolerance band, which can be regarded as the *theoretical upper bound* of safe power oversubscription. In other words, power oversubscription is safe as long as it is lower than the lower bound of the tolerance band. A major contribution of our work is that we identify this theoretical upper bound and develop adaptive control solutions to explore a practical upper bound of safe power oversubscription. Note that the tripping behavior of the CB is also impacted by the ambient temperature. The relationship between the overload current and temperature can be modeled and handled in the proposed adaptive control framework, as discussed in detail in Section 7.2.2.

7.2 CB-Aware Adaptive Power Control

In this section, we first present the design and analysis of the proposed CB-Adaptive control solution. We then introduce a method to calibrate CB-Adaptive according to temperature fluctuations. Finally, we describe CB-Proactive to further improve performance.

7.2.1 CB-Adaptive Control

CB-Adaptive is more than just a standalone controller. It is a control methodology that adapts the parameters of existing power controllers to engineer their settling times according to the trip curves of circuit breakers. CB-Adaptive can be applied to controllers at different levels (*e.g.*, server, rack and data center) and to different control techniques (*e.g.*, proportional-integral-derivative (PID), model predictive control (MPC)), though the detailed steps to tune parameters can be different. In this dissertation, as an example, we choose a state-of-the-art server-level power controller [81] as a baseline to demonstrate the design of CB-Adaptive.

As introduced in [81], The controlled variable of the server-level power controller is the power consumption of the server in the kth control period, i.e., $p(k)$. The manipulated variable is the level of the CPU Dynamic Voltage and Frequency Scaling (DVFS), i.e., CPU frequency $f(k)$. $d(k)$ is the difference between $f(k+1)$ and $f(k)$. Specifically $d(k) = f(k+1) - f(k)$. The power model used in [81] is:

$$p(k+1) = p(k) + Ad(k) \tag{7.1}$$

where A is a parameter determined by specific server configurations and the benchmark running on the server. Based on the power model, the controller designed in [81] in the Z-domain form is:

$$C(z) = \frac{1}{A} \tag{7.2}$$

In contrast to the original power controller which simply uses the rated current of the circuit breaker as its power budget, the design goal of CB-Adaptive is to enforce a dynamic power budget that varies in every control period, based on the breaker trip curve, to guarantee that the circuit breaker does not trip if workloads vary. As a result, the server can run at its maximum performance level.

We design CB-Adaptive by adapting the controller parameter A in every control period according to the trip curve of the circuit breaker. Since the trip time is a non-linear function of the magnitude of the power overload, to reduce complexity, we use piecewise linear equations to approximate the trip curve. The Z-domain form of our adaptive controller is:

$$C(z) = \frac{1}{A^*} \tag{7.3}$$

where

$$A^* = \frac{A}{1 - \sqrt[k]{0.02}} \tag{7.4}$$

where $k = \left\lfloor \frac{settling\ time(\text{sec})}{\text{T(sec)}} \right\rfloor > 0$. *settling time* is set to the trip time of the circuit breaker when power is $p(k)$.

Example. Suppose $A = 76$ for a specific configuration of a server running LINPACK. In one control period, the measured current is $1.53A$. Since the current is greater than the rated current of our CB (1A), based on Figure 7.1, the trip time is about 80 seconds. The settling time of the proportional controller (7.2) is set to the trip time by adapting the control parameter according to (7.4). Thus $A^* = 350.38$. In the next control period, the measured current may be reduced to $1.42A$ due to DVFS throttling, the trip time becomes 190 seconds. Since the allowed settling time is now longer than before, we set $A^* = 776.89$. The key feature of CB-Adaptive is continuously adjusting the control parameter to fully utilize the allowed interval for optimized system performance.

We now consider the impacts of workload variations on the design of CB-Adaptive. In production data centers, the workload of a server may differ from the benchmark based on which we design the controller (7.3). To prevent the CB tripping when the workload varies at runtime, it is necessary to analyze the impact of the different workloads on the controller using control theory. Our analysis shows that the controller parameter A in (7.2) needs to be changed by adding a *safety margin*. We outline the main steps of the analysis as follows.

1. We test a wide range of workloads to determine the range of the parameter A in (7.2) for the typical workload, such as SPECJBB, SPECCPU 2-cores in addition to LINPACK, by conducting system identification. The dynamic model of the real system is in the following format

$$p(k+1) = p(k) + gAd(k) \tag{7.5}$$

where the *system gain* g is used to model the variation between the real system model (7.5) and the nominal model (7.1). For example, $g = 1.5$ means that the actual change to the power consumption of the server is 1.5 times the estimated change in the event of DVFS.

2. Based on the real model (7.5) that models workload variations, we derive the controller parameter of CB-Adaptive as:

$$A^{*}_{real} = \frac{gA}{1 - \sqrt[k]{0.02}}. \tag{7.6}$$

The new transfer function of the adaptive controller is

$$\mathrm{C}_{real}(z) = g\mathrm{C}(\mathrm{z}). \tag{7.7}$$

3. The key difference between (7.6) and (7.4) is g. Based on step 2, we set the safety margin as $\max\{g\}$ for the various workloads we will run on the servers. As long as we run the workloads corresponding to the range of g, the safety margin guarantees that the settling time of the adaptive controller (7.8) is less than or equal to the trip time in spite of the workload variations and the circuit breaker will not trip. In case the running workload is not corresponded to the range of g, the DVFS is decreased

quickly to prevent the circuit breaker from tripping when the power consumption of the server is higher than the power budget.

$$\mathrm{C}_{real}(z) = \max\{g\}\,\mathrm{C(z)}. \tag{7.8}$$

In addition to the settling time, we also need to check whether the adaptive controller is stable. The stability range is $0 < g < 2$, which is much wider than the variation range of g observed in our extensive experiments with various workloads. Therefore, for typical workloads such as SPECJBB, SPEC CPU2006 and LINPACK, CB-Adaptive is guaranteed to be stable.

7.2.2 Temperature-aware CB-Adaptive

In Section 7.2.1, we assume that circuit breakers operate at their normial temperature ($40°C$). However, in a production data center, servers and circuit breakers may run at different temperatures since the temperature distribution is not uniform. A typical raised-floor data center is divided into hot aisles and cold aisles to improve the data center cooling efficiency. Poor air recirculation at the ends of rows and top of racks often causes server inlet temperatures to vary widely (from $15°C$ to $45°C$) [8]. Since airflow in a data center is not ideal and the CB trip time depends on temperature, we calibrate the adaptive controller parameter as follows

$$A^*(T_{CB}) = \frac{A}{1 - \sqrt[k(T_{CB})]{0.02}} \tag{7.9}$$

To determine $k(T_{CB})$, we first calculate the rated current adjusted by temperature according to (7.10) given the measured ambient temperature of the CB. Then we calculate the normalized current with respect to the rated current specified for the measured temperature. Based on the piece-wise equations which approximate the trip curve, we calculate the trip time under the measured temperature as:

$$I_n^T = (C_1 T_{CB} + C_2)\, I_n \tag{7.10}$$

where I_n is the rated current at the nominal temperature (normally $40°C$). C_1 and C_2 are constants and specified in CB data sheets. I_n^T is the rated current adjusted by temperature at $T°C$. T_{CB} is the ambient temperature of the circuit breaker.

We assume that CB temperatures can be measured in real time. This is reasonable since deployments of sensor networks in data centers are already used in practice [147]. Section 7.4 discusses CB temperature monitoring in detail.

Example. For the circuit breaker we used in the experiments, $I_n = 1$. Suppose the ambient temperature of the CB is $10°C$. According to the CB manual [15], $C_1 = -0.004167$ and $C_2 = 1.167$. Using (7.10), $I_n^T = 1.125$. When calculating $k(T_{CB})$ based on Figure 7.1, the measured current should be normalized with respect to I_n^T instead of I_n.

7.3 Discussion

The proposed CB-aware power control solutions can have many potential applications in data center power management. In this section, we discuss hosting additional servers in a data center. First, we present our method based on proposed CB-aware power control solutions. Then, we investigate whether it is safe to apply our method in a data center.

The allowed number of servers hosted within a rack is determined by the power consumption profile of the servers. Currently, the measured peak power consumption of the servers is equal or less than the 80% of rated power capacity of the circuit breaker according to the NEC requirement. In contrast, CB-Adaptive allows hosting additional servers by configuring the measured peak power consumption of the servers beyond 80% of the rated power capacity of the circuit breaker. As shown in Figure 7.2, Fan et al. [46] present the cumulative distribution functions (CDF) of the power

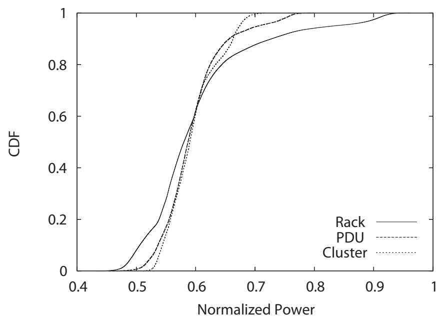


Figure 7.2: Power CDF of a real Google data center from [46].

consumption of servers running a wide range of data center workloads in a real Google data center. The highest power consumption of servers during most time is much lower than the measured peak power. Suppose we configure the allowed peak power consumption of servers as the 135% of rated power capacity of the circuit breaker. According to the CDF, during most time, the power consumption of the servers may be below 80% of rated power capacity of the circuit breaker. The time interval that violates the 80% of rated power capacity of the circuit breaker is only on a scale of minutes. Those short-term violations are allowed by NEC. Furthermore, CB-Adaptive can guarantee that the circuit breaker will not trip and boost the performance of the servers compared to the current conservative practice. Section 7.6 provides a detailed quantitative analysis.

From Figure 7.2, which shows the power load behavior for racks, PDUs, and clusters in a highly-optimized Google data center [46], we observe that among racks (40 servers), PDUs (800 servers) and clusters (5000 servers), only racks occasionally get close to 100% of the possible peak aggregate server power. Since branch circuits directly feed the rack-level, we assume that branch circuits will see similar load behavior. An important point is that load behavior varies considerably between racks (branch circuits) and this is a key reason that the PDU and cluster-level load behavior does not come close to the 100% peak power consumption possible [46]. In fact, at the cluster level, only about 70% of the peak possible server power is observed.

If a data center is attempting to utilize its power infrastructure by hosting as many servers as possible, it will likely experience overloads first at the branch circuit. For this reason, we apply the CB-Adaptive method on the circuit breaker for each branch circuit. The prior data suggests that different branch circuits will overload at different times but not together. This means that we can focus effort on controlling overloads at the branch circuit and that they will not transfer all the way to the root of the power distribution system. On average, some branch circuits will need additional overload capacity while others do not, so overloads will be rarely seen at the PDU or cluster levels. In this case, only the circuit breaker and branch circuit cable is relevant for determining the length of operation in overload.

In the unlikely case that a data center workload drives all CB-Adaptive branch circuits to operate beyond 100% capacity, overloads will be experienced by higher-levels in the facility and their overload times become relevant for consideration. In this rare case, all components of the power delivery system of a data center are relevant for determining the length of operation in overload. CB-Adaptive controllers would need to be informed by a higher-level controller to determine the overload duration time.

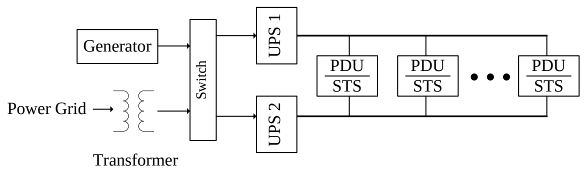


Figure 7.3: A typical power delivery system of a data center.

Figure 7.3 shows a typical power delivery system of a data center. All the components have an *overload capacity* in addition to their rated capacity. Although power overloads beyond the rated capacity in a very long term might damage a component, in many cases, tolerating short-term power overloads is necessary in practice [131]. We now summarize typical overload capacities of the components in Table 7.2. Note that all components can tolerate an overload higher than the value listed in the table, but components generally tolerate a higher overload for a shorter time interval.

Table 7.2: Overload capacity

Components	Overload capacity normalized to the rating	Time (minutes)
Static Transfer Switch	125%	60
Various cables	125%	3.5 to 110
UPS	125%	0.5
Generator	110%	60
Transformer	150%	30

Table 7.2 shows the overload capacities of all components. Static transfer switch can tolerate large over currents. The limiting factor of the overload capacity is the heat dissipation [123]. If over currents are too large, the heat generated by over currents cannot be dissipated. Cables can tolerate overloads for a short period of time [14] but overloading cables for long periods of time could damage their insulation. Generators comply with electrical standards which allow a 10% or more overload [101]. The overload capacities of transformers depend on ambient temperature, type of insulation, size of transformer and method of cooling [131]. UPS can also tolerate a short period of overload [13]. For example, certain models of data center level UPSs from APC can tolerate 125% overload for 30s. This fact implies that a single UPS basically can not tolerate much overload. However, Figure 7.3 shows that in a normal state, each UPS only runs, at most, half of its capacity for fault tolerance. If one UPS is down, the power load of the UPS shut down will be transferred to the operating UPS. In the rare case where a UPS is down, it is not desirable to perform the proposed power oversubscription solutions any longer. The servers have to run at a lower power budget.

The most important contribution of our work is to provide a technically feasible solution that allows a data center to gain the maximized return on existing investments in their power supply facilities by safely accommodating the maximum number of servers. It is important to note that our technique is not limited to circuit breakers, because it can be applied to the component with the lowest tolerance level in the power delivery system. As a result, safe power oversubscription can be achieved. More importantly, our work offers insightful discussion on the technical part of the power oversubscription problem and explores a practical upper bound for power capping, revealing that a power overload is not necessarily fatal as commonly assumed.

7.4 Implementation

In this section, we introduce our physical test bed and benchmarks, as well as the implementation details of each component in the control loop.

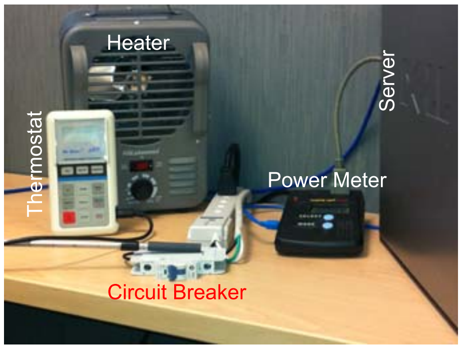


Figure 7.4: Hardware test bed.

Our test bed uses a single server to represent the load on the branch circuit. Note that CB-adaptive can also be integrated with exsiting branch circuit level or data center level power controls. Recent proposals like SHIP [138] allow for control of branch circuit power by monitoring and controlling the aggregate power of many servers. A natural place for CB-adaptive is within such a control system. Details of controlling multiple servers to realize an aggregate power are presented in the prior work [138].

Our test bed shown in Figure 7.4 consists of a Rockwell Automation circuit breaker, a heater to change the ambient temperature of the breaker, a thermostat, a power meter and a Dell OptiPlex desktop with an AMD Athlon(tm) 64 X2 Dual Core Processor 4400+ with a 2MB on-die L2 cache and 800 MHz FSB. The processor supports five DVFS levels: 2.3GHz, 2.2GHz, 2GHz, 1.8GHz, and 1GHz. The operating system is a Fedora Core 8 with a Linux kernel 2.6.23 with real-time patches. The circuit breaker model is a Rockwell Allen-Bradley 1489-A Industrial circuit Breaker with a rated current of 1A. Rockwell circuit breakers are widely used by data center operators.

We run the SPEC CPU2006 suite (V1.0) and High Performance Computing LINPACK Benchmark (HPL) (V1.0a) as our workloads. For the SPEC CPU2006, each performance measurement is the average of the four copies and is recorded as the performance ratio, i.e., the relative speed of the processor to complete each benchmark (compared to a reference Sun UltraSparc II machine at 296 MHz). The CPU2006 includes a collection of 29 benchmarks and is divided into CINT2006 and CFP2006, each of which consist of integer and floating-point benchmarks, respectively. HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic. The problem size of HPL is configured to be $10,000 \times 10,000$ and the block size is 64 in all experiments, unless otherwise noted.

The control loop consists of three components: temperature monitor and power meter (sensor), adaptive controller (controller), and CPU frequency modulator (actuator).

CB Temperature Monitor: Circuit breakers typically do not have built-in thermal sensors, however the industry is rapidly adding temperature measurement to data center products. For example, Arch Rock (Now Cisco)'s PhyNet Wireless Sensor Network is being integrated into IBM's Active Energy Manager [147]. These inexpensive and low-power sensors can easily be added to the circuit breaker panel to measure the temperature of a circuit breaker.

Power Meter: The power consumption of the server is measured with a WattsUp Pro power meter which has an accuracy of ±1.5% of the measured value. To access the power data, the data port of the power meter is connected to the USB port of the desktop. A device file is then generated for a power reading on the Linux system. The power meter samples the power data every 5 seconds and responds to requests by writing all new readings after the last request to the system file. The controller then reads the power data from the device file and conducts the control computation.

Adaptive Controller: The adaptive controller which implements CB-Adaptive or CB-Proactive runs at the highest priority (real-time priority) to guarantee fast response times. Otherwise, the controller process may be preempted by other processes with a higher priority which may cause the circuit breaker to trip due to improper control. The Linux system call *sched_setscheduler* sets both the scheduling policy and the associated parameters for the process identified by *PID*. A key advantage of CB-Adaptive and CB-Proactive is their small overheads in terms of time, space, and power consumption.

7.5 Evaluation Results

We first introduce the state-of-the-art baselines, then present our empirical results conducted on the physical test bed.

Baselines

Our first baseline is NoControl. NoControl estimates the peak power consumption of a server by measuring a high-power workload like SPECJBB over a few days. It assumes the real peak power consumption will never exceed the estimation. Although NoControl may run without any problems for weeks or months, it is risky because unexpected workloads or high input rates may drive even higher power consumption which cause the CB to trip. The second baseline, referred to as P-Control, is a state-of-the-art power provisioning algorithm widely deployed in IBM servers [81]. P-Control is briefly summarized as follows. 1) In each control period, the power meter on each server sends the server power consumption in the last sampling period to a propotional controller through its power management infrastructure [146]. 2) The proportional controller calculates the CPU frequencies in the next control period. 3) The calculated frequencies are enforced using a first-order delta-sigma modulator. The third baseline is P-Control-CB. The only difference of P-Control-CB from P-Control is that its power budget is set according to the upper limit of the long-delay region.

A fundamental difference between P-Control and our solutions is that P-Control assumes the power budget must be below the rated power of circuit breaker as soon as possible, without considering the trip curve of the circuit breaker. Moreover, P-Control adopts classical proportional control without adapting the gain of the controller.

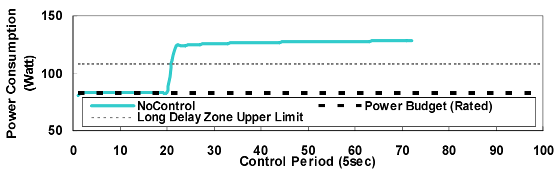
Power Capping Comparison

In this experiment set, we compare the NoControl, P-Control, P-Control-CB, CB-Adaptive, and CB-Proactive under a power emergency in which the power consumption of the server increases abruptly. To emulate the power emergency, we launch a power hungry benchmark LINPACK in the middle of the experiment. Figure 7.5a shows that with NoControl, the power consumption increases from 83W to 125W after LINPACK is launched in the 20th control period. Since the server draws a much higher current than the upper-limit of the long-delay region, the circuit breaker trips quickly, in approximately the 72th control period. Figure 7.5b shows that P-Control controls the power consumption without tripping the circuit breaker within 3 control periods to the set point which corresponds to the rated current of the circuit breaker. Similar to P-Control, Figure 7.5b also shows the P-Control-CB controls the power consumption within 3 control periods to the set point which corresponds to the upper-limit of the long-delay region. In contrast, in Figure 7.5c, it takes approximately 70 control periods for CB-Adaptive to control the power consumption to the set point. Within the time interval of the experiment, the power consumption is still higher than the set point. The reason is that CB-Adaptive changes the CPU frequencies according to the circuit breaker trip curve and the controller parameter is updated accordingly. From the 20th control period to the 90th control period, the circuit breaker runs in the conventional tripping zone. From the 90th control period on, the circuit breaker runs in the Long-delay tripping zone. Since the trip time in this zone is on the scale of days, the controller decreases the frequency slowly. Thus, the decrease of the power consumption is not visible. As shown in Figure 7.5d, CB-Proactive further increases the power consumption of the server by increasing the CPU DVFS level to the highest level proactively when the CB enters the long-delay region. After the DVFS increase, an abrupt power increase is observed at approximately the 90th control period.

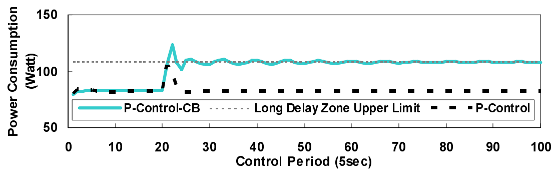
Note that P-Control-CB controls the power consumption to the set point, which is approximately 108W. Although the trip time corresponding to 108W is on the scale of days, it is not infinite. Thus, P-Control-CB can not guarantee the circuit breaker will not trip. Even within the short experiment time interval, CB-Adaptive and CB-Proactive settle to 108W. They can guarantee the circuit breaker will never trip by decreasing the power consumption according to the long-delay region. This experiment set demonstrates that CB-Adaptive and CB-Proactive can oversubscribe the circuit breaker without tripping it during a power emergency. NoControl and P-Control-CB may cause the circuit breaker to trip during a power emergency. Although P-Control will not cause the circuit breaker to trip like CB-Adaptive and CB-Proactive, as we will show in the next experiment set, the performance penalty incurred by the P-Control is large.

Performance Comparison

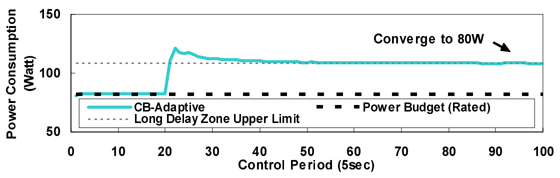
In this experiment set, we study the performance benefits of CB-Adaptive and CB-Proactive by comparing them to P-Control. Although P-Control-CB is not a safe power provisioning solution, we include it in the comparison to explain CB-Adaptive and CB-Proactive. We first test solutions running a computation-intensive benchmark LINPACK, then run all 29 benchmarks of SPEC CPU2006 to test the robustness of the solutions under a wide range of workloads.



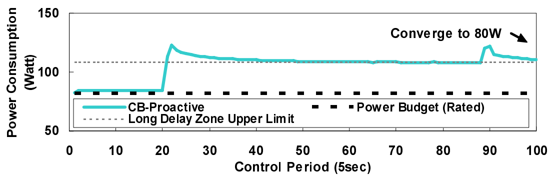
(a) NoControl (CB trips in the 72th control period)



(b) P-Control



(c) CB-Adaptive



(d) CB-Proactive

Figure 7.5: The comparison of CB-Adaptive and CB-Proactive with baselines.

Figure 7.6 compares the LINPACK and SPECJBB performance of P-Control, P-Control-CB, CB-Adaptive, and CB-Proactive. The performance of P-Control is the lowest and is only 0.85 Gflops. P-Control-CB, CB-Adaptive, and CB-Proactive outperform P-Control by 66.00%, 69.06 %, and 70.12 %, respectively. The relationship between the performance and the power consumption is approximately 0.02 Gflops per Watt. From the results, CB-Adaptive and CB-Proactive improve the performance significantly as compared to the state-of-the-art P-Control. It is demonstrated that the primary performance boost comes from the long-delay region because within the time interval of the experiment the circuit breaker runs at the upper-limit of the long-delay region, according to Figures 7.5b, 7.5c, and 7.5d. Although the performance of P-Control-CB is comparable to CB-Adaptive and CB-Proactive, as shown in Section 7.5, it may trip the circuit breaker over the long-term and not safe. For CB-Adaptive and CB-Proactive, their performance is impacted significantly by the long-delay region of a circuit breaker. For different models of circuit breakers from different manufacturers, the upper-limit of the long-delay region may vary. The larger the upper-limit of the long-delay region is, the better the performance is. The slower the decreases of the trip time with respect to the excess current in the conventional tripping zone, the better the performance is.

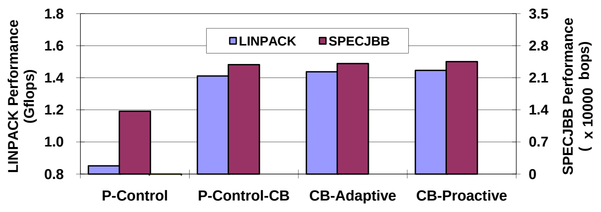


Figure 7.6: The LINKPACK and SPECJBB performance comparison.

We also test solutions by running the SPEC CPU2006 benchmark and study its performance in terms of the base rate. We only test P-Control, CB-Adaptive and CB-Proactive in this experiment because previous experiments have shown that NoControl and P-Control may trip the circuit breaker. Figures 7.7 and 7.8 compare the CPU2006 performance of P-Control and CB-Adaptive. As shown, CB-Adaptive achieves better performance than P-Control for all benchmarks since CB-Adaptive can provision the server at a higher power budget safely as compared to P-Control. The maximum performance improvement of CB-Adaptive is 49% over P-Control with the benchmark *hmmer* while the minimum improvement is 29% with the benchmark *povray*. The average improvement of CB-Adaptive is 38%. This experiment set demonstrates that CB-Adaptive and CB-Proactive can boost performance significantly during an overload condition for LINPACK and SPEC CPU2006.

Temperature Awareness

While the previous experiments are conducted at a normal room temperature, this experiment set studies the feasibility of incorporating the temperature into the design of CB-Adaptive. We use a fan heater to heat the circuit breaker to emulate non-uniform temperature within a data center. For each experiment, we monitor the temperature of the circuit breaker to be approximately stable using a regular thermostat. We first study the impact of the temperature on the circuit breaker

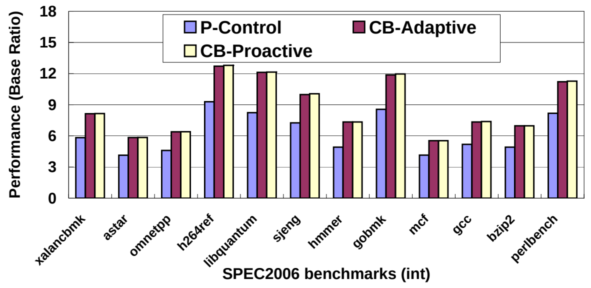


Figure 7.7: The SPEC CPU2006 int performance comparison.

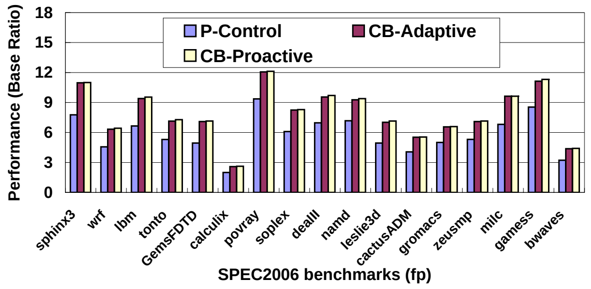


Figure 7.8: The SPEC CPU2006 fp performance comparison.

without any control. The actual rated current is not measurable. To examine the impact, we measure the trip time instead of the actual rated current by running the LINPACK benchmark on the two cores of the server. We vary the temperature from $21.7°C$ to $34.8°C$. The temperature range is a subset of the normal temperature range within a data center. We can not lower the temperature below the $21.7°C$ without a cooler due to our room temperature limit. The first four bars of Figure 7.9 show that the temperature has a significant impact on the trip time of the circuit breaker. For example, the trip time of the circuit breaker is 490 seconds at $21.7°C$ while the trip time at $34.8°C$ is only 210 seconds. For each temperature, the trip time is an average of several repeated experiments and the deviation is negligibly small. The key feature of CB-Adaptive and CB-Proactive is an adaptive controller based on the trip curve of the circuit breaker. Since the temperature impacts the circuit breaker, it will also impact CB-Adaptive and CB-Proactive.

We configure P-Control-CB, CB-Adaptive, and CB-Proactive based on the temperature of $10°C$ because the low average operating temperature in some data centers is $12°C$. Since the temperature distribution within a data center is non-uniform [8], we also test the circuit breaker at a maximum temperature of approximate $45°C$. The last three bars of Figure 7.9 show that the circuit breaker still trips even though all solutions can safely provision power at the configured temperature. Since the controllers are configured to $10°C$, they assume that the circuit breaker's rated current, as adjusted by the temperature, is 1.1 A. However, the actual rated current, adjusted by the temperature, at $45°C$ is actually only 0.9 A. Since the controllers operate according to an over-optimistic trip curve, the temperature-blinded circuit breakers still trip. Because CB-Adaptive and CB-Proactive run at a higher power budget than P-Control-CB, they trip more quickly than P-Control-CB. This experiment demonstrates that it is necessary to adopt the temperature-aware CB-Adaptive in real data center operating environments. The next experiment will demonstrate that temperature-aware CB-Adaptive can prevent the circuit breaker to trip and investigates its performance.

Impact of Temperature on Performance

In this experiment set, we study the performance of the temperature-aware CB-Proactive presented in Section 7.2.2 under different temperatures and compare the performance of P-Control, P-Control-CB, CB-Adaptive, and CB-Proactive. The temperature-aware P-Control, CB-Proactive, and CB-Proactive guarantee the safety of power provisioning under different temperatures. We present the results of LINPACK because, as shown in Section 7.5, the results of SPEC 2006 are very similar. Figure 7.10 shows that, as the temperature increases from $10°C$ to $45°C$ which is the normal temperature range of a production data center, the performance of the system decreases. The reason is that as the temperature increases, the rated current adjusted by the temperature decreases. In other word, all the solutions suffer from the lower power budget, thus performance decreases. However, Figure 7.10 shows that the performance degradation is modest even when the temperature range is wide. For P-Control, when the temperature is 45°C, the CPU DVFS is changed to its lowest level but the circuit breaker still trips, resulting in no performance reading. This experiment demonstrates that the temperature-aware CB-Proactive and CB-Proactive presented in Section 7.2.2 can successfully conduct power capping for a range of temperatures with only a modest performance degradation.

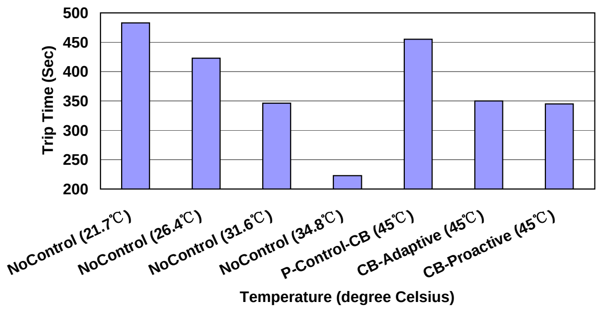


Figure 7.9: The impact of temperature on NoControl and three CB-aware solutions.

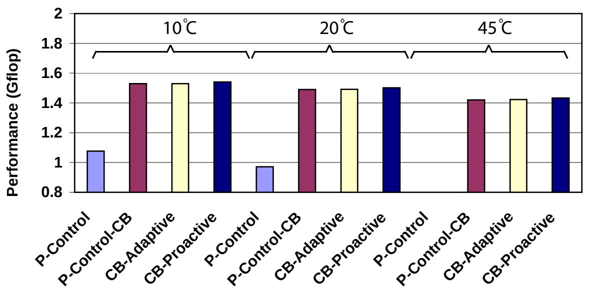


Figure 7.10: Impact of temperature on LINPACK performance.

7.6 Power Provisioning Analysis

As discussed in Section 7.3, one of the potential applications of CB-Adaptive is hosting additional servers. In this section, we quantitatively compare NoControl, P-Control, and CB-Adaptive in terms of the maximum number of servers that can be hosted within a data center. It is shown that CB-Adaptive can host many more servers than NoControl and P-Control without a performance penalty for short-term overloads.

As shown by previous experiments, NoControl may trip a circuit breaker and is not safe. In order to make NoControl safe, instead of estimating the power consumption of a server running a fixed benchmark, we assume the power consumption is the server's nameplate power value. Thus, NoControl will never trip a circuit breaker since the power consumption will never exceed its nameplate power value. For NoControl, the number of servers within a rack is the branch circuit capacity multiplied by 80%, then divided by the nameplate power consumption. Since the nameplate power consumption of a server is very conservative, the *actual peak power consumption* of the server when running most power intensive benchmarks with 100% utilization is much smaller than its name power value. For P-Control, the number of servers within a rack is the branch circuit capacity multiplied with 80%, then divided by the actual peak power consumption of the server running real data center workloads. For CB-Adaptive, the number of servers calculation is based on the branch circuit capacity multiplied by the *oversubscription ratio* (OSR).

OSR is related to the *violation interval* (i.e., the time during which the power consumption of a rack is higher than the 80% rated power capacity of the circuit breaker) and the cumulative distribution function (CDF) of a rack running real data center workloads [46]. We define OSR as:

$$OSR = \frac{0.8}{cdf^{-1}(1 - (violation\ time/24))} \tag{7.11}$$

where cdf^{-1} is the inverse function of the CDF and the unit of violation interval is 1 hour. A power CDF of a real data Google center [46] is shown in Figure 7.2. Note although Figure 7.2 is based on 6 months of measurements, since server activity correlates strongly with the hour of the day, we assume it is representative of a typical 24 hour period for our analysis.

Example. We now present an example to demonstrate the calculation of OSR. Suppose the violation interval is 3 hours which conforms to the NEC requirement [105]. Then the ratio of the violation interval to 24 hours is 12.5%. Given the percentage, we look at the y-axis of Figure 7.2 and find the point on the x-axis corresponding to the 87.5% (1-12.5%) on the y-axis. In this case, the value is 0.68. Thus the actual peak power consumption of the rack is $0.8/0.68 = 1.175$ which means that the actual peak power consumption of the rack is 111%of the rated capacity. $OSR = 1.175$. The number of servers within a rack is the branch circuit capacity multiplied with OSR, then divided by the actual peak power consumption of the server running real data center workloads.

For a data center configuration, the rated power capacity of each rack is about 2.5kW[46]. The nameplate power consumption of the hosted server is 251W. The actual power consumption of the server when running most power intensive benchmark with a 100% utilization is just 145W.

Table 7.3: Power provisioning

Items	Number of servers
NoControl (80% branch circuit capacity)	7
NoControl	9
P-Control (80% branch circuit capacity)	13
P-Control	17
CB-Adaptive $OSR = 1$	17
CB-Adaptive $OSR = 1.1$	18
CB-Adaptive $OSR = 1.175$	20

Table 7.3 shows that CB-Adaptive with $OSR = 1.175$ can host 54% more servers than the state-of-the-art P-Control (conforming to NEC) in each branch circuit and about three times as many servers using NoControl (conforming to NEC).

Chapter 8

Data Center Level Power Control

8.1 System Architecture

In this section, we present our system architecture. As shown in Figure 8.1, our system architecture features a two-level control loop which consists of an outer cooling system power controller and an inner server power controller. The two-level control loop enforces a data center power budget.

During a power emergency in which the power consumption of a data center violates its power budget, a key observation is that it is not necessary to run CRACs in a data center at their full capacity all the time. The design goal of the two-level

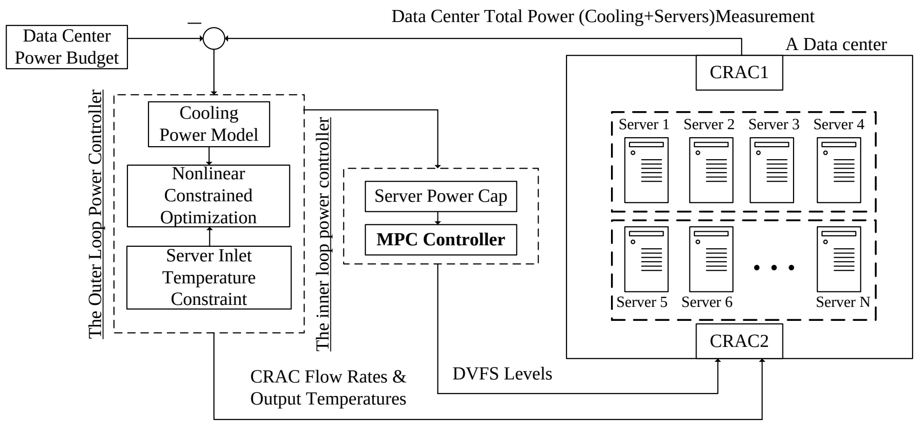

Data Center Power Budget
–
Data Center Total Power (Cooling+Servers)Measurement
A Data center
CRAC1
The Outer Loop Power Controller
Cooling Power Model
Nonlinear Constrained Optimization
Server Inlet Temperature Constraint
The inner loop power controller
Server Power Cap
MPC Controller
Server 1
Server 2
Server 3
Server 4
Server 5
Server 6
Server N
CRAC2
CRAC Flow Rates & Output Temperatures
DVFS Levels

Figure 8.1: System Architecture

control is to configure the power consumption of CRACs and servers in a balanced way which means servers run at a power budget as high as possible as long as the temperature of servers do not violate the thermal thresholds.

The outer loop power controller works as follows. According to the current power budget of data centers, it manipulats three knobs namely the power budget for servers, CRAC flow rates and output temperatures to control the power consumption of a data centers. The core of the controller is an optimization formulated in Section 8.2. Cooling system enforces calculated CRAC flow rates and output temperatures and the power cap for servers will be an input to the inner control loop.

The inner server power controller is a MPC server power controller in our previous work [138]. Its control period is smaller compared to that of the outer loop to ensure in every control period the outer loop, the inner loop has already entered into its steady state. It controls the power of the servers by scaling the CPU frequency. The controller is a Multiple-Input-Multiple-Output (MIMO) controller and every group of servers which share the power supply has its own MIMO controller since the power budget for servers can be shifted among them to make full utilization of the budget. This control loop works as follows: (1) the power monitor on each server sends the power measurement to the centralized power controller; (2) the controller collects all power measurement and computes a new set of CPU frequencies and sends them to the frequency modulator on servers; and (3) the frequency modulator then changes the server frequency using DVFS.

The proposed MPC controller is guaranteed to control a power overload to a power budget within its settling time quickly to avoid circuit breakers trip. As shown in [52], generally, the majority of circuit breakers have two types of trip time behaviors which are specified in the UL489 standard. First, short-circuits (for example, over 500% of the rated load) cause the CB to trip within a few milliseconds which [138] assumes. This assumption is pessimistic and will cause inferior performance of servers. Second, overload conditions for a less severe current draw can trip the circuit breaker on a time scale from milliseconds to hours or even weeks, depending on the severity of the overload. The proposed MPC controller can be adapted to further boost server performance which is orthogonal to the proposed solution in this work.

The two-level architecture reduces the complexity of the proposed solution while enforce thermal constraints of server inlet temperatures. For inner server power controller, the MIMO controller only a few number of state variables. As the number of state variables increases, the algorithm complexity of MIMO controller increases significantly. For the cooling system power controller, due to the thermal coupling between servers, it is undesired to reduce the number of state variables. If we have several optimizers instead of one central optimizer, the some heat transfer will be ignored and thermal constraint cannot be guaranteed.

8.2 The Outer Loop Power Controller

In this section, we present the design of the outer cooling system power controller. Our optimizer minimizes the difference between the power consumption of a data center and a data center power budget by adjusting a power cap of servers, CRAC flow rates, and CRAC output temperatures.

8.2.1 System Modeling

In order to design a controller, we must first establish the relationship between the cooling system power consumption $P_{cooling}$ and CRAC flow rates and CRAC output temperatures. A data center cooling system consists of Chiller water loop and Condensation water loop. As shown in Figure 8.2, the Chiller water loop represented by red and blue lines is used to cool down the air in the computer room. The hot air from hot aisles of racks is forced into the CRAC by blowers. Volume of air per minute is defined as *CRAC flow rate*. The hot air exchanges heat with cold water within CRAC, and then becomes cold air, finally is expelled to hot aisles. The temperature of the cold air expelled is *CRAC output temperature*. In Condensation water loop

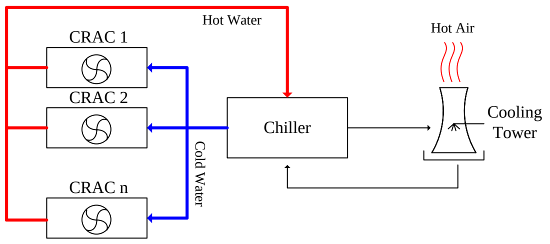


Figure 8.2: Diagram of a Data Center Cooling System

represented by black lines, hot water is circulated back to a chiller that exchanges the heat by phase change from a compressor and produces cold chilled water. The chiller expels its heat through a cooling tower to the outside environment.

Most existing work such as [8] adopts a simplified power model of a data center cooling system, the cooling power $P_{Cooling}$ is only determined by the server power using the following P_{Server} as follows.

$$P_{Cooling} = \frac{P_{Server}}{COP} \tag{8.1}$$

where COP is an thermal efficiency metric widely adopted by industries for measuring the energy efficiency of a data center.

In constrast, we derive a more accurate power model based on [66] and calculate cooling power based on server power, CRAC flow rate, and CRAC output temperature. As shown in Figure 8.2, cooling power $P_{Cooling}$ consists of three parts which are CRAC power P_{CRAC}, chiller power $P_{Chiller}$, and cooling tower power P_{tower}.

The system model is as follows.

$$P_{Cooling} = \psi\left(T_{CRACoutputTemp}, Q_i, P_{server}\right) \tag{8.2}$$

Based on the system model 8.2, the CRAC power, chiller power, and cooling tower power can be expressed as follows.

$$P_{CRAC} = \sum_{i=1}^{N_{CRAC}} \varphi_i[P_{server} + \lambda\left(P_{Cooling} + P_{server}\right)]^{\alpha} \tag{8.3}$$

$$P_{Chiller} = \lambda \left(P_{Cooling} + P_{server} \right) \tag{8.4}$$

$$P_{tower} = 0.05 \left(P_{cooling} + P_{server} \right) \tag{8.5}$$

The system model 8.2 cannot be expressed explicitly using a formula. The detailed derivation is not presented due to page limit.

8.2.2 Controller design

The goal is to minimize power consumption of a cooling system by reducing both the CRAC output temperature and CRAC output temperatures without violating thermal constraints of servers. A higher CRAC output temperature will reduce the power consumption of the entire cooling system significantly. It is predicted [29] that 4-5% of data center cooling power could be reduced for every 1°C increase in CRAC output temperature. It will impact the temperature of whole computer room. As a complementary, CRAC flow rate adjustment only impacts a certain zone within the computer room. The lower flow rate will reduce the power consumption nonlinearly.

We need to set CRAC flow rates and output temperature to optimal levels. CRAC Output Temperature adjustment needs to be coordinated with CRAC Flow Rates Throttling. Since both of them will increase the server inlet temperature, we need to increase CRAC output temperature to an optimal level which minimizes the power consumption of a data center cooling system. If CRAC output temperature is increased beyond the optimal level, there is not much room for CRAC flow rate throttle and the power consumption of the cooling system is not minimized. Thus, it is impossible to adjust CRAC flow rates and output temperature seperately.

We solve the following optimal control problem. In every control period, a dynamic optimization problem is solved to obtain the minimum cooling power consumption. The reduced amount of cooling power consumption at the *kth* control period is denoted by $\Delta P_{CRAC}(k)$. We shift $\Delta P_{CRAC}(k)$ to the servers power budget. The server power control loop will track the updated power budget by changing DVFS levels of servers which is determined by the MPC Controller.

$$\underset{\left\{\begin{array}{c} T_{CRACoutputTemp}(k) \\ Q_i(k), CAP_{server}(k) \end{array}\right\}}{Min} \{pp(k+1) - P_{date\ center}\}^2 \tag{8.6}$$

where $pp(k+1) = P_{cooling}(k+1) + CAP_{server}(k+1)$.

where CAP_{server} is power budget for servers and $CAP_{datacenter}$ is power budget for a data center. The object function 8.13 is a function of three knobs. The detailed relationship between $P_{cooling}$ and $T_{CRACoutputTemp,Q_i}$ is derived in Section 8.2.1 and is shown to be nonlinear. subject to sets of constraints:

$$Q_{i,\min} \le Q_i(k) \le Q_{i,\max}, 1 \le i \le N_{CRAC} \tag{8.7}$$

$$T_{lower} \le T_{CRACoutputTemp}(k) \le T_{upper} \tag{8.8}$$

$$P_{servers\_\min} \le CAP_{servers}(k) \le P_{servers\_\max} \tag{8.9}$$

$$\begin{gathered} f_j\left(Q_i(k), T_{CRACoutputTemp}(k)\right) \le T^j_{threshhold} \\ 1 \le j \le N_{Servers}, 1 \le i \le N_{CRAC} \end{gathered} \tag{8.10}$$

where $T^j_{threshhold}$ is the jth server inlet temperature limit. Reliability of a data center will be affected if $T^j_{threshhold}$ is set too high. ASHRAE recommends 25°C in 2005, 27°C in 2011. However, existing studies show that recommendation of ASHRAE is conservative. Rackable CloudRack C2 from SGI is designed to operate at 40°C. Servers designed to run at a higher temperature 45°C is on the way according to IBM Research[80]. Studies [133][54][109][118] also show that hardware reliability of data center equipments such as data center network, disk drive, and DRAM is uncorrelated with the temperature under a wide temperature range.

We will details the modeling of the temperature constraint in Subsection 8.2.3. It is noted that in above problem formulation, both that subjuct function and the thermal constraint 8.10 are nonlinear functions of three control knobs. We adopt fmincon in Matlab to solve the problem.

8.2.3 Server Inlet Temperature Constraint

It is challenging to establish a relationship between the server inlet temperature and CRAC flow rates and output temperature using an explicit equation. [102] adopted machine learning based modeling of temperature coupling among servers, but the model can only be presented by a large neural network. Majority data center thermal modeling work such as [112] predict server inlet temperatures given fixed CRAC flow rates and output temperatures. Quite recently, [154] derived a thermal dynamic model based on thermodynamic to predict how CRAC flow rates and output temperatures affects server inlet temperature. We derived our model based on [154]. The high-level derivation steps are as follows and detailed mathematical calculation is not presented.

1. [154] established the relationship between temperature change and current temperature and current CRAC flow rates and temperatures. First, let temperature change equal to zero, thus Eqn 8.11 of current server inlet temperature, CRAC flow rates and temperatures can be obtained.

$$\sum_{i=1}^{N_{CRAC}} A\,[T_{CRACoutputTemp} - T_{inletTemp}]Q_i + B = 0 \tag{8.11}$$

2. there are two unknown data center specific parameters A and B in Eqn 8.11. To determine them, CFD software can be used offline to obtain the server inlet temperature and corresponding $T_{CRACoutputTemp}$ and Q_i. Based on results from CFD, an linear equation array can be obtained. Using standard solver in Matlab linsolve, the unknown parameter can be obtained.

3. after some transformation, the final system model can be represented as following,

$$T_{inletTemp} = f_j\left(Q_i, T_{CRACoutputTemp}\right) \tag{8.12}$$

8.2.4 Coordination Analysis

[The control period of cooling system power optimizer is configured according to the settling time of the inner control loop.]

Whenever the power consumption of the entire data center exceeds the cap, we try to lower both the cooling and server power.We propose to formulate this problem as a mixed-integer nonlinear programming optimization problem. We believe that the proposed CRAC-aware power capping solution can lead to higher overall performance than existing solutions that cap only the server power (*e.g.*, [138][110]).

8.3 Air-side Economizer

An air-side economizer [91] brings outside air into a data center and distributes it to the servers and it bypasses some components of a cooling system presented in Figure 8.2, thus it can significantly lower the cooling system power consumption and it is a powerful knob in addition to CRAC flow rates and CRAC output temperature. However, there are different types of economizer modes. Two typical modes are air conditioner bypass via direct fresh air and chiller bypass via heat exchanger mode. For air conditioner bypass via direct fresh air, instead of being re-circulated and cooled, the exhaust hot air from the servers is simply directed outside. If the outside air is particularly cold, the economizer may mix it with the exhaust air so its temperature and humidity fall within the desired range for the equipment. The disadavantage of this mode is requiring modification of data center building shell which is not always possible and incur high costs. A chiller bypass via heat exchanger economizer mode

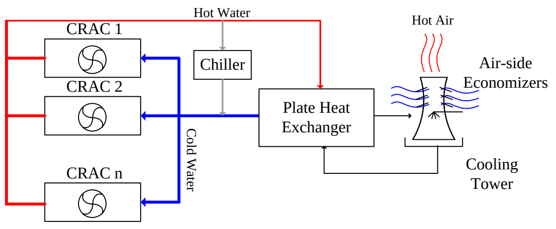


Figure 8.3: Diagram of a Data Center Cooling System with a Chiller Bypass via Heat Exchanger Economizer Mode

uses the condenser water to indirectly cool the data center chilled water when the outside air conditions are within specified set points. Pumps move the condenser water through a plate-and-frame heat exchanger to cool the chilled water used in CRAHs without mixing the two water streams as shown in Figure 8.3.

Because data centers must be cooled 24/7, 365 days per year, air-side economizers may even make sense in hot climates, where they can take advantage of cooler evening or winter air temperatures. For most regions of the United States, existing study shows the number of hours per year with ideal conditions for an air-side economizer is from 5000 to 8500 hours. For the hot region (coasts around Mexican Gulf), the hour is from 3000-5000 hours. As an Intel research shown, a 10MW facility will save 2.87 million annually.

8.3.1 Switched cooling system power controller

To fully utilize of air-side economizer, We design a switched cooling system power controller to extend the design presented in Section 8.2.2. As shown in Figure 8.4, it consists of two controllers switched based on the outside air quality such as temperatures and humidity. One is exactly the same as the one presented Section 8.2.2 while the other one is as follows.

$$\underset{\left\{\begin{array}{c} T_{CRACOutputTemp}(k) \\ Q_i(k), CAP_{server}(k) \end{array}\right\}}{Min} \{pp(k+1) - P_{date\ center}\}^2 \tag{8.13}$$

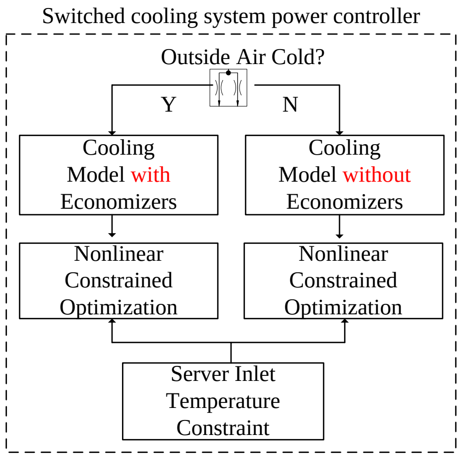

Switched cooling system power controller
Outside Air Cold?
Y
N
Cooling Model with Economizers
Cooling Model without Economizers
Nonlinear Constrained Optimization
Nonlinear Constrained Optimization
Server Inlet Temperature Constraint

Figure 8.4: Switch Control

where $pp(k+1) = P^*_{cooling}(k+1) + CAP_{server}(k+1)$.

where CAP_{server} is power budget for servers and $CAP_{datacenter}$ is power budget for a data center. The object function 8.13 is a function of three knobs. The detailed relationship between $P_{cooling}$ and $T_{CRACoutputTemp,Q_i}$ is derived in Section 8.2.1 and is shown to be nonlinear.

subject to sets of constraints:

$$Q_{i,\min} \leq Q_i(k) \leq Q_{i,\max}, 1 \leq i \leq N_{CRAC} \tag{8.14}$$

$$T_{lower} \leq T_{CRACoutputTemp}(k) \leq T_{upper} \tag{8.15}$$

$$P_{servers\_\min} \leq CAP_{servers}(k) \leq P_{servers\_\max} \tag{8.16}$$

$$\begin{gathered} f_j\left(Q_i(k), T_{CRACoutputTemp}(k)\right) \leq T^j_{threshhold} \\ 1 \leq j \leq N_{Servers}, 1 \leq i \leq N_{CRAC} \end{gathered} \tag{8.17}$$

The difference between of the two controllers are the cooling power model is different. Since we adopt a chiller bypass via heat exchanger economizer mode, the controller design presented in Section 8.2.2 is still valid and only the power model derivation in Section 8.2.1 needs modification to have a new model $P^*_{cooling}(k+1)$. If choosing air conditioner bypass via direct fresh air, the thermal modeling in Section 8.2.3 is not valid anymore.

8.4 Evaluation

We employ simulators and real-world traces to evaluate our techniques. We use AirPAK and Fluent which are computational fluid dynamics simulators by Ansys Inc to model cooling in a data center.

The data center's parameters are configured according to the previous studies [8]. The data center (40'×12'×30') consists of four rows of 1120 server blades with each row containin seven 40U racks. For ease of configuration, we group four servers into a 4U server block. As shown in Figure 8.5, the data center employs the standard configuration of alternating hot and cold aisles to facilitate air flow and to avoid mixing of hot air with cold air. Our servers are modeled after the state-of-the-art servers (e.g, IBM Systems x3650 M2 or HP Proliant DL3xx series) with power consumption of 100 W when idle, 300 W at 100% utilization, and 5 W in standby mode. Because server power varies almost linearly with server utilization [30], a 40%-utilized server consumes 100 + (300-100) × 0.4 = 180 W. Each server has a volumetric flow rate of 0.068 m^3/s. We model fan power as a function of server inlet temperature as [66]. There are four CRAC units in the center, each of which pushes chilled air at 15oC into a raised floor plenum (1.5 ft. high) at a rate of 9000 ft^3/min. The cool air enters the cold aisles (inlets of servers) and hot air exits the hot aisles (outlets of servers) to the CRAC exhaust vents.

The power budget for the entire data center is reduced by 15% from 460KW.

8.4.1 Baselines

Our first baseline, referred to as SHIP, is a control-theoretic data center level power controller proposed in a recent paper [138]. SHIP represents state-of-the-art solution that is designed, conservatively running cooling system at its maximum capacity, to use per-core DVFS to control the power of a data center. We compare our controller against SHIP to show that state-of-the-art data center power controller may fail to have accurate power control under extreme scenario and moreover lead to degraded application performance. The key component of SHIP is a rack-level control loop and briefly summarized as follows. 1) The power monitor (e.g., a power meter) measures the average value of the total power consumption of all the servers in the last control period and sends the value to the controller. 2) The controller computes the new CPU frequency level for the processors of each server, and then sends the level to the CPU frequency modulator on each server. The detailed controller design is presented in [137] 3) The CPU frequency modulator on each server changes the CPU frequency (and voltage if using DVFS) of the processors accordingly.

A fundamental difference between SHIP and our controller is that SHIP simply keeps flow rates and output temperature of all CRACs at highest levels. It only throttles DVFS levels of servers to reduce data center power consumption. In contrast, the proposed solution adopts various knobs including DVFS levels of servers, flow rates and output temperatures of CRACs, and switches cooling system modes depending on weather condition while enforce thermal constraint of server.

The second baseline, referred to as TAPO-DC, is a recently proposed thermal-aware power management solution to to reduce data center level cooling power [66]. Given a power set point, TAPO-DC uses a simple binary dynamic control method to choose CRAC output temperatures. When power consumption of servers is lower than a threshold, a high output temperature is selected and power is shifted from cooling system to servers. When power consumption of servers is higher than the threshold, a low output temperature is selected and power is shifted from servers to the cooling system. While TAPO-DC can work effectively to enforce the data center power budget and shift power between cooling system and servers to boost performance, TAPO-DC fails to achieve optimal application performance because it

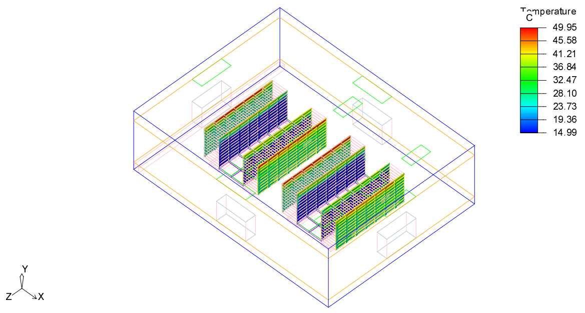


Figure 8.5: The Simulated Data Center in AirPak and Fluent

	Server Block Power	CRAC Power	Data Center Power
Baseline 1	895	38000	413800
No-CRAC-Throttle	890	40000	413800

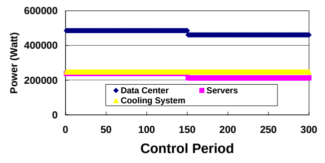
Table 8.1: Comparison of Baseline 1 and No-CRAC-Throttle

ignores the correlation between cooling system and servers, and only manipulates CRAC output temperature in a coarse granularity.

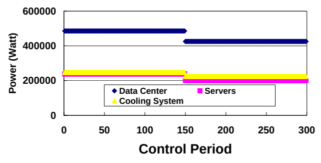
8.4.2 Power Emergency

In this experiment, we test the proposed solution and compare it with two baselines during a power emergency. As shown in Figure 8.6c, all servers in the data center initially run at the maximum power consumption and accordingly the cooling system is configured to maximum flow rates of four CRACs. The default output temperature of four CRACs is set to a typical value in production data centers which is 10 degree Celsius. At the 150th control period, the power budget is reduced by 10%. Figure 8.6c shows that the proposed solution reduce both servers and cooling system power consumption to enforce the reduced power budget for the data center. The settling time of the solution is just 1 control period. Figure 8.6f further shows it dynamically increases the output temperature of CRACs and decreases flow rates of the cooling system.

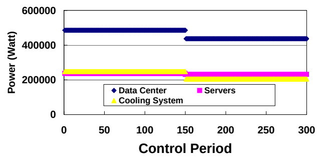
Figure 8.6a shows the performance of SHIP when the data center power budget is reduced by 5%. During the emergency, SHIP only reduces servers' power and the



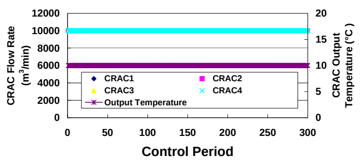
(a) Power measurement (SHIP)



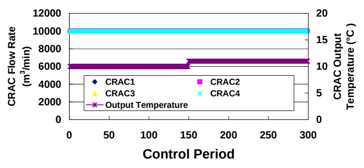
(b) Power measurement (TAPO-DC)



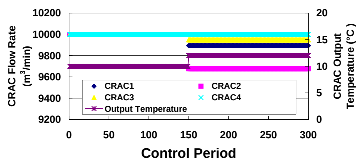
(c) Power measurement (Proposed solution)



(d) Cooling system configuration (SHIP)



(e) Cooling system configuration (TAPO-DC)



(f) Cooling system configuration (Proposed solution)

Figure 8.6: Typical runs of three solutions (Proposed solution, SHIP, and TAPO-DC).

cooling system power is not changed. Figure 8.6d shows the flow rates and output temperature remain constant. Since the servers' power has been reduced, it is a waste to conservatively remain the cooling system at its maximum capacity. Compared to the proposed solution, none power is shifted from cooling system to servers which will lead to suboptimal performance.

According to the design of TAPO-DC, if data center power reduction is below a threshold which is 11% in our case, the output temperature will be selected to be 10 degree Celsius which is the same as the default value. If the data center power budget had been reduced by 5% or 10% as above scenarios, TAPO-DC will behaves exactly as SHIP since neither flow rates nor output temperature will be reduced to shift power from the cooling system to servers. It suffers the same problem as SHIP. Figure 8.6b shows the performance of TAPO-DC when the data center power budget is reduced by 12%. Figure 8.6e reveals only the output temperature is increased from 10 degree Celsius to 11 degree Celsius while none flow rates are throttled. The reason is that TAPO-DC does not take into account the correlation between servers and cooling system and only adjusts the output temperature in a very coarse granularity.

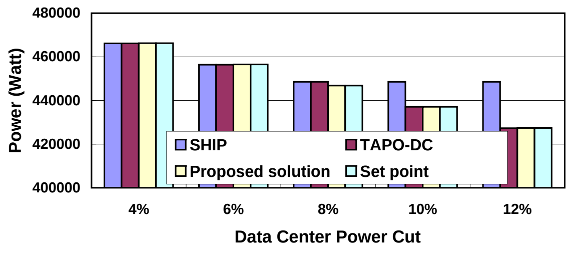


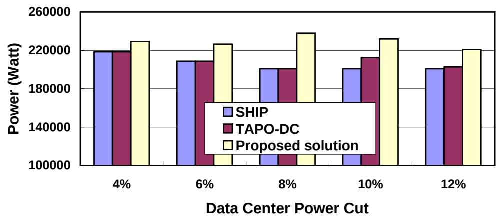
Figure 8.7: Data center power

8.4.3 Control Accuracy and Application Performance

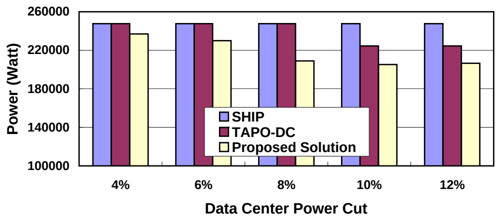
In this experiment, we reduce the data center power budget from 4% to 12% for all three solutions. The purpose is to stress test them to investigate control accuracy and compare application performance fairly.

Figure 8.7 compares the measured power consumption of the data center when three solutions enter stable states. Both the proposed solution and TAPO-DC can achieve accurate power control for a wide range of power reduction. In contrast, SHIP fails to control power accurately beyond a threshold. The reason is that cooling system power consumption accounts for a large part (around 50%) of total data center power consumption. If cooling system power consumption is not adapted during power emergency, only a small percentage reduction of data center power budget will induce a large amount of server power budget cut. Since today's servers are not energy-proportionate, the dynamic power range of servers is small. As shown, when the data center power cut is 10% and 12%, even when servers run at the lowest levels, the power consumption of servers are still higher than desired. As a result, the power cap cannot be enforced which may cause serious consequences such as power outage. The figure demonstrates an important disadvantage of state-of-the-art power control solution SHIP.

Figure 8.8a shows servers power consumption under three solutions. Since higher power consumption of servers will lead to higher application performance. Essentially, it compares the application performance. Even TAPO-DC is similar to the proposed solution in term of the total data center power consumption. The proposed



(a) Servers power



(b) Cooling system power

Figure 8.8: Comparison of three solutions under various data center power budget reductions (Proposed solution, SHIP, and TAPO-DC).

solution constantly outperforms baselines in term of application performance. The performance gain ranges from 4% to 18%. It also shows that TAPO-DC and SHIP has the same application performance when the power cut is 4%, 6%, and 8%. The reason is that TAPO-DC does not adjust output temperature at all when the data center power reduction is below a threshold and TAPO-DC behaves exactly the same as SHIP. The performance difference between TAPO-DC and the proposed solution is attributed to two main factors. TAPO-DC ignores correlations and only adjusts output temperature in a coarse granularity. Figure 8.8a shows clearly the reason of application performance. Both SHIP and TAPO-DC consumes more cooling power than the proposed solution. This experiment demonstrates that the proposed solution archives optimal application performance by efficiently utilizing the cooling power and configure the cooling and servers in a correlated way.

8.4.4 Enforcement of Thermal Constraint

In this experiment, we evaluate the temperature constraint of the proposed solution. As shown in Figure 8.9, the measured maximium temperature within the data center under the proposed solution are always below the thermal constraint. In constrast, under SHIP and TAPO-DC, the maximium temperature is far below the threshold and the servers are unnecesarily overcooled. As a results, cooling system power are wasted for overcooling.

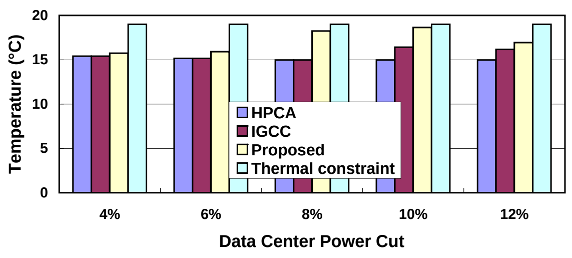


Figure 8.9: Thermal threshold guarantee

Moreover, the thermal constraint significantly impacts the application performance. A higher CRAC output temperature will lead to better application performance. The proposed solution can enforce the thermal threshold and can also minimize the difference between the maximum predicted temperature within the data center and the threshold to improve application performance.

Chapter 9

End-to-End Energy Management of Virtual Desktop Infrastructure

9.1 Energy Management with Performance Guarantee

In this section, we first present the system architecture of E-cubed (End-to-End Energy Management). We then present a model of end-to-end performance for a VDI workload. Finally, we present a formulated optimization algorithm and heuristic for large-scale deployments.

9.1.1 System Architecture

Figure 9.1 shows the E-cubed system architecture which works as follows. (1) Each user remotely controls a virtual machine called a desktop VM via a thin client (such as a smartphone or a tablet PC). In the data center, physical servers host all the desktop VMs. The desktop VMs are responsible for workload execution while the thin clients only render the display and transmit user input (from keyboards and mice) using a virtual desktop communication protocol. A monitor periodically collects the utilization of the desktop VMs and detects input events (such as key presses) to determine the number of active desktop VMs. The number of active desktop VMs changes over time due to idle periods during nights and holidays and long idle interval in office hours [114]. The number of active desktop VMs and their associated physical servers are sent to the E-cubed. (2) The core of E-cubed is a constraint nonlinear optimizer that minimizes the end-to-end energy consumption by various knobs. They include throttling CPU DVFS levels of the thin clients, throttling link rates and shutting down any idle ports on the network switches, and decreasing hard disk rotational speeds of the shared storage. Since energy consumption of servers accounts for a large portion of all energy consumption, in this work, we focus on manipulating server DVFS levels and consolidating desktop VMs. The work can be extended to integrate all knobs in the future. A key challenge is to define the performance metrics of a VDI deployment and model the relationship between the performance metrics, CPU DVFS levels and VMs consolidate ratios (a.k.a the number of VMs hosted on a single host). A performance model enforces a user-specified requirement. The detailed derivation of the performance model is in Section 9.1.2. Moreover, the optimizer takes into account hardware constraints such as making sure the CPU frequencies are adjusted according to hardware specific ranges. (3) The optimizer throttles the CPU DVFS levels of physical servers and consolidates desktop VMs by live VM migration according to the output of the optimizer.

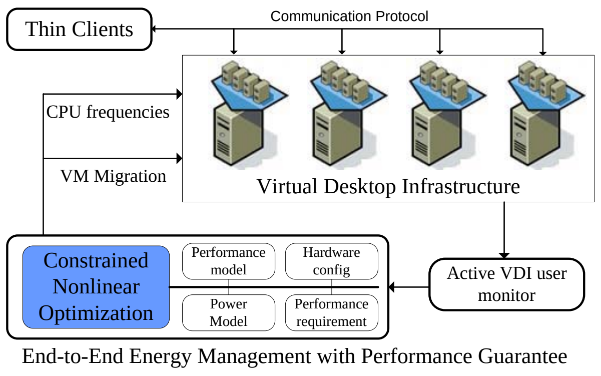


Figure 9.1: Integrated management architecture for virtual desktop infrastructure.

9.1.2 Performance Model

We first introduce notations. Q_i is the performance, f_i is chip-wide CPU frequency[*], and R_i is the number of desktop VMs hosted on the ith server. N_{host} is the total number of physical servers in a VDI deployment. N_{VM} is the total number of active desktop VMs. S_i is the status of the ith physical server. If the ith server hosts dekstop VMs, $S_i = 1$. If the server is idle and powered off, $S_i = 0$. QoS is a user-specified performance requirement, which is a constant. $f_{i,\min}, f_{i,\max}$ are minimum and maximum frequencies of the ith physical server.

End-To-End Performance

View Planner is used to generate realistic VDI workloads by emulating user operations. It performs a series of random operations of all applications of the collection. Between operations, a random sleep interval emulates user think time. Different operations emulate different users. We select a collection of applications which represent typical workloads for most VDI deployments in a production environment. The collection includes Adobe Reader, Word, Excel, Outlook, PowerPoint, Video, Internet Explorer, and 7-ZIP. For each application, users may perform various operations. For example, open a Word document, browse, edit, finally save the document.

Because VDI users expect their VMs to be ”responsive”, the first priority performance metric is response time of an operation. The end-to-end performance is defined as a high percentile response time of all operations performed by a VDI user. Usually, it is defined as the 95th percentile response times or the 98th percentile response times. Since the response time measurement has variation, we take the average to reduce its variation. Detailed analysis of the VDI performance definition is out of scope of this paper.

[*]Recent CPU models from both Intel and AMD support per-core DVFS throttling and per-tile DVFS throttling. More fine grained DVFS throttling can be utilized in future work.

Black-box Approach

In order to have an effective E-cubed design, it is necessary to model the performance of a VDI deployment, specifically, the closed-form mathematical relationship between the 95th percentile latency of all VDI user operations and the actuators described in Section 9.1.1. Since a virtual desktop infrastructure deployment is a complicated computer system, a well-established physical equation based on a queuing system is not available. To address the challenge, we select a black-box approach, namely *surface fitting*.

We establish the performance model for a single physical server off-line by running a typical VDI workload and varying CPU DVFS levels and consolidation ratios and measuring the end-to-end performance. Based on the collected data, an accurate model is derived by surface fitting the datapoints. View Planner can be used to generate realistic VDI workloads by emulating user operations. It performs a series of random operations of all applications of the collection. Between operations, a random sleep interval emulates user think time. Different operations emulate different users. CPU frequency is varied from the highest frequency to the lowest frequency, and the consolidation ratio is increased until the performance is much lower than a specified threshold. For clusters consisting of homogeneous servers, which is the most common in production, the performance model for a single physical server holds for the other identical servers. For clusters of heterogeneous servers, each type of server will need its own model.

Performance measurement shows a strong nonlinear relationship between the end-to-end performance and manipulated variables (CPU DVFS levels and consolidation ratio). Beyond a certain threshold, performance degrades significantly as DVFS level decreases and consolidation ratio increases. Some black-box techniques such as system identification [95] and linear regression [85] cannot be applied because of their assumption of linearity. Machine learning is a powerful approach [98] for deriving a complex model. However, [98] adopts an artificial neural network to represent a derived model, and an explicit mathematical formula is unavailable. The performance model for the ith server can be expressed as follows.

$$Q_i\left(f_i, R_i\right) = K_i\left(\frac{a_1}{f_i} + a_2\right)\left(b_1 {R_i}^2 + b_2 R_i + b_3\right) \tag{9.1}$$

Constant K_i denotes response time when only one desktop virtual machine runs on the ith server and its CPU runs at the maximum frequency. The term $\left(\frac{a_1}{f_i} + a_2\right)$ represents response time inflation due to CPU frequency throttling, and equals to 1 when $f_i = f_{\max}$. The term $\left(b_1 {R_i}^2 + b_2 R_i + b_3\right)$ represents response time inflation due to VM consolidation, and equals to 1 when $R_i = 1$. The coefficient of determination $\mathrm{R}^2 = 0.85$.

Figure 9.2 shows the difference of predicted performance based on the model and actual measurement. Configurations 1-8 are combinations of randomly-selected CPU DVFS levels and randomly-selected consolidation ratios. Those configurations are different from configurations used to establish the performance model as described before. Small difference shown in Figure 9.2 shows the end-to-end performance model (9.1) predicts accurately.

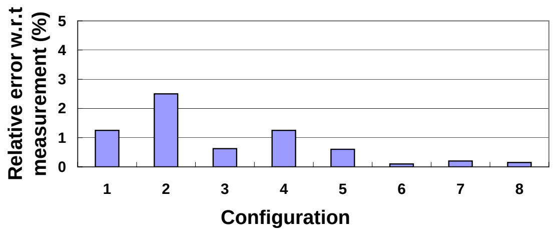


Figure 9.2: Validation of the performance model against measurement.

It is possible that the performance model derived off-line does not hold in a production environment in some scenarios. For example, hardware upgrade, physical server replacement, users running applications which are not listed in the profiled collection, or computer security compromises and resource exhaustion due to a Denial-of-service attack. However, those scenarios can be detected by comparing the predicted value based on the model and real-time performance measurement. If the difference is abnormally large for a long interval, a change of a VDI deployment has happened. For permanent changes such as hardware upgrade, recalibration of the model is necessary during maintenance time. If a new application is introduced, the model can be extended.

9.1.3 Optimization

The following optimization will be invoked when the total number of active desktop VMs N_{VM} is changed.

Optimal Algorithm

Cost function:

$$\mathrm{P}_{VDI}=\sum_{i=1}^{N_{host}} S_i \left(\alpha_i f_i^{\beta_i}+\mathrm{M}_i R_i+\mathrm{L}_i\right) \tag{9.2}$$

subject to:

$$\begin{aligned} & Q_i\left(f_i, R_i\right) \leq QoS \\ & f_{i,\min} \leq f_i \leq f_{i,\max} \\ & \sum_{i=1}^{N_{host}} R_i = N_{VM} \end{aligned} \tag{9.3}$$

where $\alpha_i, \beta_i, \mathrm{M}_i, \mathrm{L}_i$ are server-specific parameters. Those parameters can be estimated using high dimensional fitting. The detail of power modeling is similar to the performance modeling and is omitted.

In a VDI deployment, minimization of energy consumption is equivalent to minimization of power consumption. Thus, the cost function to be minimized is the power consumption. Since servers consume a majority part of the total power consumption, we only minimize the power consumption of servers and the power consumption of other components is constant.

The above optimization formulation cannot be solved using existing solvers directly. We present high-level steps of tranforming the formulation to MINLP (Mixed-Integer Nonlinear Programming). For the cost function (9.2), the relationship between S_i and R_i can be established using a signum function. $S_i = sgn\left(R_i\right)$. MINLP requires the cost function to be a continuous function while a signum function is not continuous at 0. To meet the requirement of MINLP, the signum function can be approximated using a special continuous function such as tanh. A MINLP solver will determine R_i and f_i to minimize the cost function (9.2). Several constraints on R_i and f_i exist. The CPU frequency cannot be adjusted arbitrarily and has to be within a range. VDI administrators may assign a VM to a physical server in a static way. A VM must be assigned to only one physical server and all VMs must be assigned. The details of the mathematical transformation is not presented here.

Scalable Algorithm

A key observation is that VM consolidation and idle physical server shutdown can lead to significantly more energy savings than throttling DVFS levels. Idle power consumption of a physical server accounts for more than 70% of total power consumption [21]. Thus, we design a two-step heuristic to obtain a near optimal solution to the optimization problem formulated in 9.1.3. In the first step, we minimize idle server energy by consolidating VMs and run each active server at the highest CPU DVFS level. In the second step, we further throttle DVFS levels.

The detailed huristic for a large-scale VDI deployment consisting of homogeneous servers is shown in Algorithm 2. The 2nd step of Algorithm 2 solves the following optimization problem. The optimization is a constrained nonlinear multivariable which can be solved directly using fmincon in Matlab. Although no polynomial solver exists for fmincon, the optimization is conducted on a single physical server basis rather than on a cluster basis . Thus, the the huristic reduces the algorithm

Algorithm 2 Heuristic for large-scale VDI deployments

```
begin
  1: The 1st step:
  2: Calculate the number of desktop VMs per server according to the performance
     model, and denote the number by $R_0$.
  3: Turn on $\left\lfloor \frac{N_{VM}}{R_0} \right\rfloor$ physical servers which runs at the highest DVFS level;
  4: if $Q\left(f_{\max}, R_0\right) < QoS$ then
  5:     The 2nd step: Invoke a host-level optimization algorithm
  6: end if
  7: Actuators: shut-down idle physical servers, enforce $f_i$.
end
```

complexity significantly and obtains a near optimal solution as the MINLP solver in Section 9.1.3.

$$\mathrm{P}_i = \alpha_i f_i^{\beta_i} + \mathrm{M}_i R_i + \mathrm{L}_i \tag{9.4}$$

subject to:

$$\begin{aligned} f_{i,\min} \le f_i \le f_{i,\max} \\ Q_i\left(f_i, R_i\right) \le QoS \end{aligned} \tag{9.5}$$

Where R_i is determined in the first step of Algorithm 2.

9.2 Evaluation Results

9.2.1 Implementation

In this section, we introduce our physical test bed, as well as the implementation details of each component.

Our test bed consists of two physical servers. Host1 has an AMD Opteron 6128 12-core CPU 1.9GHz with 16Gb main memory. Host2 has two AMD Opteron 254 dual-core CPU 2.8GHz with 4Gb main memory. It is connected to a shared storage

Table 9.1: System Configuration

Server Cluster:	2 AMD servers	vSphere
Fabric:	1 Gb Ethernet	vCenter Enterprise
Shared Storage:	Openfiler	View Planner

Openfiler by Ethernet. The detailed hardware configuration is presented in Table 9.1. The host1 processor supports five DVFS levels: 1.9GHz, 1.5GHz, 1.3GHz, 1GHz, and 800MHz. The virtual machine operating system is Windows 7. We run View Planner V2.0 [3] to emulate VDI user operations, which run all the typical applications in Section 9.1.2. The length of each run is approximately 4 hours.

CPU Frequency Control : we use Intel's Enhanced Intel SpeedStep Technology to enforce the new frequency. To change the CPU frequency, VMWare ESXi contains a command line tool to determine the current frequency levels, frequency islands information, and modify frequencies within allowable ranges. Some advanced servers come with power management policy built in to the BIOS to manipulate CPU frequencies. To avoid conflicts, BIOS power management policies needs to be disabled. The average overhead (i.e., transition latency) for frequency change is approximately 100 μ. The CPU frequency calculated by the optimization algorithms is continuous and physical CPUs only support discrete number of frequency levels. A delta sigma converter is implemented to approximate a continuous frequency using discrete CPU frequencies.

Performance Measurement: latencies of operations on the data center side can be measured using CPU performance counters to obtain high-resolution timing information. By default, virtual machines cannot access performance counters. For newer releases of vSphere, virtual machines can have access by adding a flag to the vmx configuration files. We develop a watermarking technique to accurately measure the latency of the image transmission from the data center side to the client side. An encoded time stamp is placed in a fixed region of the image. When a thin client receives the frame, it reads and decodes the fixed region. The thin client and data center must be synchronized using NTP (network time protocol) with a common time source. More details can be found in [124].

VM migration: VMware vMotion can migrate running a virtual machine from one host to another. Even infrequent migrations of large VMs between hosts connected by a slow link may introduce significant overhead. By leveraging multiple NICs and efficient memory propagation, the overhead can be significantly reduced. Moreover, we can add host affinity rules to the optimization problem formulated in Section 9.1.3 to avoid costly VM migrations.

Power Measurement: The power consumption of the server cluster is measured with a WattsUp Pro power meter [45] by plugging servers into the power meter and then connecting it to a standard 120-volt AC wall outlet. The WattsUp power meter has an accuracy of 1.5% of the measured value and samples power data every second. Its internal memory and can store 18 hours of power data. The USB port of the power meter is connected to a desktop and a logger tool running on the desktop configures the meter and reads power data.

9.2.2 Baseline

Our baseline is state-of-the-art power management in a virtualized environment [1][2]. It controls the utilization of CPU and main memory within a fixed range by powering on and off hosts and migrating VMs but an end-to-end performance metric for VDI deployments is not taken into consideration. Although it works well for CPU and main memory intensive workloads, it could be conservative in term of energy saving for a VDI deployment in addition to not providing an end-to-end performance guarantee. The first reason is that a high utilization of vCPU and guest memory may not necessarily lead to low end-to-end performance. Another reason is the default maximum utilization is around 70% and a fixed safe margin of 30% exists. In contrast, the proposed solution eliminates such a margin by solving an optimization problem.

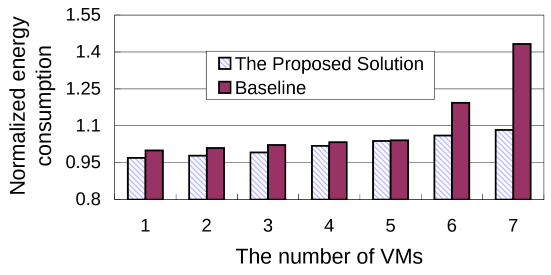
At last, the proposed solution can further extend energy efficiency of the baseline by throttling CPU DVFS levels.

9.2.3 Experimental results

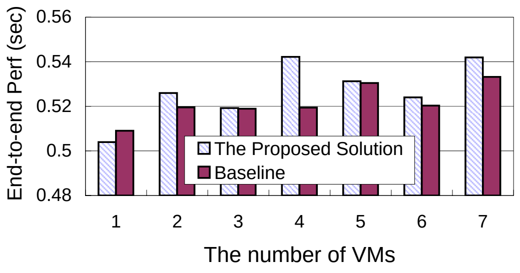
The purpose of this experiment is to compare the proposed solution against the baseline both in terms of energy efficiency and performance by varying the number of VMs (virtual machines) hosted by the hardware test bed in Section 9.2.1. This experiment demonstrates the main use case of the proposed solution in a production environment. The maximum number of VMs is determined by available free physical main memory within the cluster and represents no physical memory over-commitment scenario. During the experiment, we configure View Planner with three iterations which present a typical length of half a workday. Power consumption is measured at an interval of one second during the run and energy consumption is calculated as an integral of power with respect to time.

Figure 9.3a shows the measured energy consumption. When VDI workload is light and the number of virtual machines is from 1 to 5, the maximum energy saving is 3%. The modest saving is because when the number of VM is small, both the proposed solution and baseline utilize only one host and shut down the idle one. The proposed solution throttles DVFS levels aggressively to the lowest level and the baseline configures the highest frequency in a static way. Although the frequencies are very different, existing servers are non-power proportional and the power dynamic range of DVFS is limited. When the VDI workload is high and the number of virtual machines is larger than 5, the proposed solution outperforms the baseline by 11% and 24%. During a high workload, one host will accommodate 5 virtual machines and other virtual machines are placed on another host. The baseline consolidates based on utilization of vCPU and guest memory and will not consolidate all virtual machines until the utilization is below a certain threshold. When multiple virtual machines are powered on, utilization of vCPU and guest memory is very high due to the boot storm.

After the boot storm, the utilization does not decrease quickly, and is calculated according to a weighting algorithm to avoid frequently shutting down and turning on hosts. Thus, the baseline will power on two hosts and consolidate VMs based on slowly-changing utilization. In contrast, based on the end-to-end performance metric, the proposed solution consolidates VMs into a single host and shuts down the idle host. Figure 9.3b shows the end-to-end performance. Currently the standard requirement for VDI QoS is 1.5s. Both the proposed solution and baseline satisfy the requirement. Compared to the baseline, the proposed solution achieves significant energy savings while introducing a minimum performance penalty.



(a) Comparison of normalized energy consumption



(b) Comparison of end-to-end performance

Figure 9.3: Comparison of energy savings and performance of the proposed solution and the baseline.

Chapter 10

Conclusion

Existing power/energy-efficient scheduling algorithms focus heavily on open-loop optimization solutions. As a result, they may have degraded run-time performance in terms of power/energy efficiency and real-time guarantees when they are applied to real-time embedded systems with uncertain execution times. In this dissertation, I have presented a novel online solution that integrates core-level feedback control with a processor-level optimization strategy to minimize both the dynamic and leakage power consumption of a multi-core real-time embedded system. Our solution monitors the utilization of each CPU core in the system and dynamically responds to unpredictable execution time variations by conducting per-core DVFS. Our solution then takes advantage of the small overhead of task migration in multi-core processors with shared L2 caches to perform task consolidation on a longer timescale and shuts down unused cores for maximized power savings. Both empirical results on a hardware multi-core testbed with the Mibench benchmarks and simulation results in many-core systems show that our solution provides the desired real-time guarantees while achieving more power savings than state-of-the-art algorithms.

I have presented a two-level utilization control solution for energy efficiency in multi-core real-time systems. At the core level, our solution addresses two optimization objectives: controlling the CPU utilization of each core to its desired schedulable bound and minimizing the core power consumption by adopting per-core DVFS and dynamic L2 cache partitioning to adapt both the CPU frequency-dependent and independent portions of the task execution times of the core. Since traditional control theory cannot handle multiple optimization objectives, a novel utilization controller is designed based on advanced MOMPC theory. At the processor level, a cache demand arbitrator is proposed to coordinate the cache size demand from each core and conduct dynamic cache resizing to minimize the leakage power consumption of the shared L2 caches. The energy and time overheads of the proposed control solution are analyzed and demonstrated to be sufficiently small. Extensive experiments using the Mibench benchmarks show that our solution outperforms two state-of-the-art power management algorithms that do not consider L2 caches or per-core DVFS by having more accurate utilization control and less energy consumption.

I have presented a two-layer coordinated CPU utilization control architecture. The primary control loop uses frequency scaling to locally control the CPU utilization of each processor on a coarser timescale, while the secondary control loop adopts rate adaptation to control the utilizations of all the processors in the system at the cluster level. Both the two control loops are designed and coordinated based on well-established control theory for theoretically guaranteed control accuracy and system stability. Empirical results on a physical testbed demonstrate that our control solution outperforms EUCON, a state-of-the-art utilization control algorithm, by having more accurate control and less power consumption.

Today's DRE systems face an increasing probability of overheating and even thermal failures, due to their continuously decreasing feature size and increasing demand for computation capabilities. As a result, their temperature must be explicitly controlled for improved reliability. However, existing work provides either real-time guarantees or thermal management in an isolated manner. In this dissertation, I have presented a coordinated control solution that can provide simultaneous thermal and timeliness guarantees for heterogeneous real-time embedded systems running in unpredictable environments. The thermal control loop locally controls the temperature of each processor, while the utilization control loop provides end-to-end timeliness guarantees at the cluster level. A novel coordination analysis method based on robust control theory has been proposed to coordinate the two control loops for theoretically guaranteed global system stability. Empirical results on a physical testbed and extensive simulations demonstrate the efficacy of our control solution.

While a variety of power capping solutions have been recently proposed, a conservative assumption made by existing solutions is that peak power should never exceed the rated CB capacity. In this dissertation, I systematically study the tripping characteristics of a typical CB used in many data centers. I identify that the theoretical upper bound of safe power oversubscription is the lower bound of the tolerance band in the trip curve of the circuit breaker. I then propose two adaptive power control strategies that utilize the tripping characteristics of the CB to aggressively optimize the system performance without causing the CB to trip. Furthermore, our control schemes can also adapt to the variation of ambient temperature that is known to affect the CB tripping behaviors. Empirical results on a physical test bed show that the proposed CB-aware power control solutions achieve 29% to 49% better SPEC CPU2006 performance than a state-of-the-art baseline. The average SPEC CPU2006 performance improvement is 38%. In addition, our solutions allow a data center to host three times more servers than traditional static power provisioning schemes and 54% more servers than the current power capping practice.

Virtual desktop infrastructure is a promising virtualization technology to reduce enterprise IT expense. However, existing work cannot address the energy management issue in the context of VDI. In this paper, I first derived an explicit VDI performance model using surface fitting, then propose an optimization to aggressively reduce system energy consumption while guaranteeing performance by utilizing the performance model. Furthermore, a two-step heuristic is proposed to manage large-scale VDI deployments. Empirical results on a hardware test bed show that for high consolidation ratio scenarios the proposed solution achieves 11% - 24% better energy efficiency than a baseline widely adopted in production. It also ensures that user specified end-to-end performance requirements are met.

Bibliography

[1] Dynamic Optimization and Power Optimization in System Center 2012 . technet.microsoft.com/en-us/library/gg675109.aspx. 159

[2] VMware Distributed Power Management Concepts and Use. www.vmware.com/files/pdf/Distributed-Power-Management-vSphere.pdf. 159

[3] VMware View Planner Installation and User Guide. communities.vmware.com/docs/DOC-15578. 158

[4] Tarek F. Abdelzaher, Ella M. Atkins, and Kang G. Shin. QoS negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11):1170–1183, 2000. 35

[5] Tarek F. Abdelzaher, Vivek Sharma, and Chenyang Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers*, 53(3):334–350, 2004. 28, 43

[6] Tarek F. Abdelzaher, John Stankovic, Chenyang Lu, Ronghua Zhang, and Ying Lu. Feedback performance control in software services. *IEEE Control Systems*, 23(3), 2003. 18

[7] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *RTSS*, 2002. 18

[8] Faraz Ahmad and T. N. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. In *ASPLOS*, 2010. 23, 24, 116, 129, 136, 143

[9] AMD. *White Paper Publication 26094: BIOS and Kernel Developer's Guide for AMD Athlon 64 and AMD Opteron Processors, Revision 3.30*. Advanced Micro Devices, Inc., 2006. 79

[10] Dave Anderson, Jim Dykes, and Erik Riedel. More than an interface—SCSI vs. ATA. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, 2003. 8

[11] James Anderson, John Calandrino, and UmaMaheswari C. Devi. Real-time scheduling on multicore platforms. In *RTAS*, 2006. 21, 57

[12] James H. Anderson, John M. Calandrino, and UmaMaheswari C. Devi. Real-time scheduling on multicore platforms. In *RTAS*, 2006. 3, 19, 28

[13] APC. Experts speak on UPS output Watt, VA, and Power Factor ratings. `http://www.dcsarabia.com/whitepapers/12.pdf`, 2002. 120

[14] Ralph Atkinson and H. W. Fisher. Current rating of electrical cables. *Transactions of the American Institute of Electrical Engineers*, February 1913. 120

[15] Rockwell Automation. Rockwell Automation Inc Bulletin 1489 Circuit Breakers Selection Guide, 2010. 110, 111, 117

[16] Hakan Aydi, Pedro Mejía-Alvarez, Daniel Mossé, and Rami Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *RTSS*, 2001. 19, 68

[17] Hakan Aydin, Vinay Devadas, and Dakai Zhu. System-level energy management for periodic real-time tasks. In *RTSS*, 2006. 19, 29, 68

[18] Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *IPDPS*, 2003. 20

[19] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *FOCS*, 2004. 22

[20] Nikhil Bansal and Kirk Pruhs. Speed scaling to manage temperature. In *STACS*, 2005. 22

[21] Luiz Andre Barroso and Urs Holzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 2007. 156

[22] Alberto Bemporad and David Munoz de la Pena. Multiobjective model predictive control. *Automatica*, 2009. 5, 52

[23] Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. Rate monotonic analysis: The hyperbolic bound. *IEEE Transactions on Computers*, 2003. 43

[24] Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. Speed modulation in energy-aware real-time systems. In *ECRTS*, 2005. 5, 46

[25] Enrico Bini, Giorgio C. Buttazzo, and Marko Bertogna. The multi supply function abstraction for multiprocessors. In *RTCSA*, 2009. 20

[26] A. Block et al. Adaptive multiprocessor real-time scheduling with feedback control. In *ECRTS*, 2008. 21

[27] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition*. O'Reilly Publishers, 2005. 35

[28] Scott A. Brandt and Gary J. Nutt. Flexible soft real-time processing in middleware. In *RTSS*, 2004. 7

[29] Thomas Breen, Ed Walsh, Jeff Punch, Amip Shah, and Cullen Bash. From chip to cooling tower data center modeling: Part i influence of server inlet temperature and temperature rise across cabinet. In *ITherm*, 2010. 137

[30] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *HPCA*, 2001. 8, 22

[31] BD Bui et al. Impact of cache partitioning on multi-tasking real time embedded systems. In *RTCSA*, 2008. 21, 44, 47, 48

[32] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3), 2002. 66

[33] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Arzen. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1), 2002. 18

[34] Sudipta Chattopadhyay and Abhik Roychoudhury. Unified cache modeling for wcet analysis and layout optimizations. In *RTSS*, 2009. 19

[35] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, 2008. 21

[36] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS*, 2006. 20

[37] Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. On the minimization of the instantaneous temperature for periodic real-time tasks. In *RTAS*, 2007. 8, 19

[38] Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. On the minimization of the instantaneous temperature for periodic real-time tasks. In *RTAS*, 2007. 22

[39] Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Proactive speed scheduling for frame-based real-time tasks under thermal constraints. In *RTAS*, 2009. 22

[40] Yingming Chen, Chenyang Lu, and Xenofon Koutsoukos. Optimal discrete rate adaptation for distributed real-time systems. In *RTSS*, 2007. 6, 18

[41] James Donald and Margaret Martonosi. Power efficiency for variation-tolerant multicore processors. In *ISLPED*, 2006. 23

[42] James Donald and Margaret Martonosi. Techniques for multicore thermal management:classification and new exploration. In *ISCA*, 2006. 22

[43] Arvind Easwaran and Bjorn Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *RTSS*, 2009. 20

[44] Wolfgang Eberle, Bruno Bougard, Sofie Pollin, and Francky Catthoor. From myth to methodology: cross-layer design for energy-efficient wireless communication. In *DAC*, 2005. 24

[45] Electronic Educational Devices Inc. Watts up pro power meter. `http://www.wattsupmeters.com`. 159

[46] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007. xiii, 9, 10, 12, 21, 22, 23, 95, 117, 118, 131, 132

[47] M. Femal and V. Freeh. Safe overprovisioning: Using power limits to increase aggregate throughput. In *PACS*, 2004. 23

[48] Krzysztof Fleszar and Khalil S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 2002. 33

[49] Gene F. Franklin, J. David Powell, and Michael Workman. *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997. 74, 75

[50] Xing Fu, Khairul Kabir, and Xiaorui Wang. Cache-aware utilization control for energy efficiency in multi-core real-time systems. In *ECRTS*, 2011. 20

[51] Xing Fu and Xiaorui Wang. Utilization-controlled Task Consolidation for Power Optimization in Multi-Core Real-Time Systems, Tech Report. http://www.ece.utk.edu/∼xwang/papers/Multicore-RT.pdf, 2010. 40

[52] Xing Fu, Xiaorui Wang, and Charles Lefurgy. How much power oversubscription is safe and allowed in data centers? In *ICAC*, 2011. 12, 24, 25, 134

[53] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D. Koutsoukos, and Hongan Wang. Feedback thermal control for real-time systems. Technical Report WUCSE-2009-17, Washington University in St. Louis, 2009. 22, 103

[54] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *SIGCOMM*, 2011. 138

[55] Steve Goddard and Xin Liu. A variable rate execution model. In *ECRTS*, 2004. 7

[56] Ashvin Goel, Jonathan Walpole, and Molly Shor. Real-rate scheduling. In *RTAS*, 2004. 18

[57] Sriram Govindan et al. Statistical profiling-based techniques for effective power provisioning in data centers. In *EuroSys*, 2009. 10, 23

[58] Nan Guan et al. New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms. In *RTSS*, 2008. 19

[59] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Cache-aware scheduling and analysis for multicores. In *EMSOFT*, 2009. 21, 44

[60] Matthew R. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *WCC*, 2001. 4, 14, 35, 55

[61] Damien Hardy et al. Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches. In *RTSS*, 2009. 21

[62] D. Henriksson and T. Olsson. Maximizing the use of computational resources in multi-camera feedback control. In *RTAS*, 2004. 6, 8

[63] Jin Heo, Dan Henriksson, Xue Liu, and Tarek Abdelzaher. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *RTSS*, 2007. 22

[64] HP. Dynamic Power Capping TCO and Best Practices White Paper. http://h71028.www7.hp.com/ERC/downloads/4AA2-3107ENW.pdf, 2008. 10

[65] Huang Huang, Gang Quan, Jeffrey Fan, and Meikang Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In *DAC*, 2011. 24, 25

[66] Wei Huang, Malcolm Allen-Ware, et al. Tapo: Thermal-aware power optimization techniques for servers and data centers. In *IGCC*, 2011. 24, 136, 143, 144

[67] IBM. IBM PowerExecutive Installation and User's Guide, 2007. 10

[68] Intel. *Intel Pentium 4 Processor in the 423-pin Package Thermal-Mechanical Support Design Guide, Revision 1.0*. Intel Corporation, 2001. 103

[69] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors:methodology and empirical data. In *Micro*, 2003. 21

[70] Hemant Joshi. *Residential, Commercial and Industrial Electrical Systems: Equipment and selection*. Tata McGraw-Hill, 2008. 111

[71] Dilip D. Kandlur and Tom W. Keller. Green data centers and hot chips. In *DAC*, 2009. 24

[72] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002. 95

[73] Nam Sung Kim et al. Leakage current: Moore's law meets static power. *Computer*, December 2003. 3, 32

[74] Wonyoung Kim, Meeta Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the 14th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2008. 29, 44, 61

[75] Masaaki Kondo and Hiroshi Nakamura. Dynamic processor throttling for power efficient computations. In *PACS*, 2004. 20

[76] Kotera et al. Power-Aware Dynamic Cache Partitioning for CMPs. In *HIPEAC*, 2008. 44, 58

[77] Xenofon Koutsoukos, Radhika Tekumalla, Balachandran Natarajan, and Chenyang Lu. Hybrid supervisory utilization control of real-time systems. In *RTAS*, 2005. 6, 18

[78] Jilong Kuang, Laxmi Bhuyan, and Raymond Klefstad. Traffic-aware power optimization for network applications on multicore servers. In *DAC*, 2012. 24, 25

[79] Karthik Lakshmanan et al. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *ECRTS*, 2009. 21

[80] Charles Lefurgy, Malcolm Allen-Ware, et al. Energy efficient datacenters and systems. In *IISWC*, 2011. 138

[81] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Server-level power control. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing (ICAC)*, 2007. 23, 112, 113, 123

[82] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2), 2008. 18, 66, 80, 102

[83] Dong Li, Hung-Ching Chang, Hari K. Pyla, and Kirk W. Cameron. System-level, thermal-aware, fully-loaded process scheduling. In *IPDPS*, 2008. 102

[84] Yan Li et al. Timing analysis of concurrent programs running on shared cache multi-cores. In *RTSS*, 2009. 21

[85] Harold Lim, Aman Kansal, and Jie Liu. Power budgeting for virtualized data centers. In *USENIX*, 2011. 153

[86] Jiang Lin et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *HPCA*, 2008. 44, 48, 58

[87] Suzhen Lin and G. Manimaran. Double-loop feedback-based scheduling approach for distributed real-time systems. In *HiPC*, 2003. 18

[88] Linux hardware monitoring. http://www.lm-sensors.org, 2012. 102

[89] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000. 28, 35, 38, 43, 65, 66, 78, 85, 86, 98, 100, 101

[90] Shaobo Liu, Qing Wu, and Qinru Qiu. An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems. In *DAC*, 2009. 24

[91] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and cooling aware workload management for sustainable data centers. In *SIGMETRICS*, 2012. 140

[92] Chenyang Lu, Xiaorui Wang, and Christopher Gill. Feedback control real-time scheduling in ORB middleware. In *RTAS*, 2003. 18, 22

[93] Chenyang Lu, Xiaorui Wang, and Xenofon Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Transactions on Parallel and Distributed Systems*, 16(6), 2005. 2, 6, 18, 22, 28, 30, 52, 55, 72, 73, 76, 77, 82, 89, 90, 98

[94] Jan Lunze and Francoise Lamnabhi-Lagarrigue. *Handbook of Hybrid Systems Control: Theory, Tools, Applications*. Cambridge University Press, 2009. 30

[95] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for many-core architectures running multi-threaded applications. In *ISCA*, 2011. 153

[96] Jan M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002. 5, 52, 54, 56

[97] Pau Marti, Gerhard Fohler, Pep Fuertes, and Krithi Ramamritham. Improving quality-of-control using flexible timing constraints: metric and scheduling. In *RTSS*, 2002. 66

[98] J.F. Martinez and E. Ipek. Dynamic multicore resource management: A machine learning approach. In *Micro*, 2009. 153

[99] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: Eliminating server idle power. In *ASPLOS*, 2009. 23

[100] Ke Meng, Russ Joseph, Robert P. Dick, and Li Shang. Multi-optimization power management for chip multiprocessors. In *PACT*, 2008. 54

[101] Mitsubishi. Mitsubishi diesel generator technical specifications. `http://powercare.com.au/catalog/i32.html`, 2012. 120

[102] J. Moore, J. Chase, and P. Ranganathan. Weatherman: Automated, online, and predictive thermal mapping and management for data centers. In *ICAC*, 2006. 139

[103] Vincent Nelis and Joel Goossens. Mora: an energy-aware slack reclamation scheme for scheduling sporadic real-time tasks upon multiprocessor platforms. In *RTCSA*, 2009. 20, 21

[104] Fadia Nemer, Hugues Casse, Pascal Sainrat, Jean-Paul Bahsoun, and Marianne De Michiel. Papabench: a free real-time benchmark. In *WCET*, 2006. 4, 55

[105] NFPA. *National Fire Protection Association National Electrical Code*. NFPA, 2008. 109, 131

[106] Marco Paolieri et al. Hardware support for wcet analysis of hard real-time multicore systems. In *ISCA*, 2009. 21, 44

[107] Jaehyun Park et al. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In *ISLPED*, 2010. 57

[108] Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F. Wenisch, and Jack Underwood. Power routing: Dynamic power provisioning in the data center. In *ASPLOS*, 2010. 23

[109] E. Pinheiro, W.-D.Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *FAST*, 2007. 138

[110] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No power struggles: Coordinated multi-level power management for the data center. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008. 22, 91, 140

[111] Ragunathan Rajkumar, Lui Sha, and John P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *RTAS*, 1988. 5

[112] L. Ramos and R. Bianchini. C-oracle: Predictive thermal management for data centers. In *HPCA*, 2007. 139

[113] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey S. Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, 2006. 10

[114] Joshua Reich, Michel Goraczko, Aman Kansal, and Jitu Padhye. Sleepless in seattle no longer. In *USENIX*, 2010. 151

[115] Jose Renau et al. SESC simulator, January 2005. http://sesc.sourceforge.net. 55

[116] Saowanee Saewong and Ragunathan (Raj) Rajkumar. Practical voltage-scaling for fixed-priority RT-systems. In *RTAS*, 2003. 19, 29, 68

[117] Abhik Sarkar, Frank Mueller, Harini Ramaprasad, and Sibin Mohan. Push-assisted migration of real-time tasks in multi-core processors. In *LCTES*, 2009. 19

[118] B. Schroeder, E. Pinheiro, and W.-D.Weber. Dram errors in the wild: A large-scale field study. In *SIGMETRICS*, 2009. 138

[119] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors. *IEEE Transactions on Parallel and Distributed Systems*, November 2008. 2, 20, 21, 37, 58

[120] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *HPCA*, 2002. 23, 112

[121] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *HPCA*, 2002. 91

[122] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1), 2004. 22, 91

[123] AEG Power Solutions. AEG Static Transfer Switch Technical Specifications, 2012. 120

[124] Lawrence Spracklen, Banit Agrawal, Rishi Bidarkar, and Hari Sivaraman. Comprehensive User Experience Monitoring. *VMware Technical Journal*, 2012. 159

[125] Shekhar Srikantaiah, Mahmut Kandemir, and Qian Wang. Sharp control: Controlled shared cache management in chip multiprocessors. In *MICRO*, 2009. 23

[126] John A. Stankovic, Tian He, Tarek Abdelzaher, Mike Marley, Gang Tao, Sang Son, and Chenyang Lu. Feedback control scheduling in distributed real-time systems. In *RTSS*, 2001. 18

[127] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *OSDI*, 1999. 18

[128] Vivy Suhendra and Tulika Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *DAC*, 2008. 21

[129] Jun Sun and Jane Liu. Synchronization protocols in distributed real-time systems. In *ICDCS*, 1996. 5, 65, 66, 85, 86

[130] Xiaoyong Tang, Hai Zhou, and Prith Banerjee. Leakage power optimization with dual-vth library in high-level synthesis. In *DAC*, 2005. 32

[131] S. Tenbohlen et al. Assessment of overload capacity of power transformers by on-line monitoring systems. In *Power Engineering Society Winter Meeting*, 2001. 119, 120

[132] United States Environmental Protection Agency. Report to congress on server and data center energy efficiency, 2007. 109

[133] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *SoCC*, 2010. 138

[134] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, 23(3), 2000. 8

[135] Shengquan Wang and Riccardo Bettati. Reactive speed control in temperature-constrained real-time systems. *Real-Time Systems Journal*, 39(1-3), 2008. 8

[136] Xiaorui Wang and Ming Chen. Cluster-level feedback power control for performance optimization. In *Proceedings of the 14th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2008. 10, 91, 112

[137] Xiaorui Wang, Ming Chen, and Xing Fu. MIMO Power Control for High-Density Servers in an Enclosure. *IEEE Transactions on Parallel and Distributed Systems*, October 2010. 144

[138] Xiaorui Wang, Ming Chen, Charles Lefurgy, and Tom W. Keller. SHIP: Scalable Hierarchical Power Control for Large-Scale Data Centers. In *PACT*, 2009. 11, 12, 121, 134, 140, 143

[139] Xiaorui Wang, Yingming Chen, Chenyang Lu, and Xenofon Koutsoukos. FC-ORB: A robust distributed real-time embedded middleware with end-to-end utilization control. *Journal of Systems and Software, Special Issue on Dynamic*

Resource Management in Distributed Real-Time Systems, 80(7), 2007. 6, 18, 22, 78, 92, 100

[140] Xiaorui Wang, Yingming Chen, Chenyang Lu, and Xenofon Koutsoukos. On controllability and feasibility of utilization control in distributed real-time systems. In *ECRTS*, 2007. 6, 103

[141] Xiaorui Wang, Xing Fu, Xue Liu, and Zonghua Gu. Power-aware cpu utilization control for distributed real-time systems. In *RTAS*, 2009. 2, 31, 38

[142] Xiaorui Wang, Xing Fu, Xue Liu, and Zonghua Gu. Power-aware cpu utilization control for distributed real-time systems. In *RTAS*, 2009. 9, 22

[143] Xiaorui Wang, Xing Fu, Xue Liu, and Zonghua Gu. Power-aware CPU utilization control for distributed real-time systems. In *RTAS*, 2009. 24, 59

[144] Xiaorui Wang, Dong Jia, Chenyang Lu, and Xenofon Koutsoukos. DEUCON: Decentralized end-to-end utilization control for distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(7), 2007. 6, 18, 22

[145] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA*, 2009. 23

[146] M. Ware et al. Architecting for power management: The IBM POWER7 approach. In *HPCA*, 2008. 27, 124

[147] Business Wire. IBM to Support Arch Rock's PhyNet Wireless Sensor Network in Active Energy Manager, 2010. 117, 122

[148] Gene F. Franklin. J. David Powell. Michael L. Workman. *Digital Control of Dynamic Systems (3rd Edition)*. Prentice Hall, Upper Saddle River, New Jersey, 1997. 92, 93

[149] Ruibin Xu, Rami Melhem, and Daniel Moss. Energy-aware scheduling for streaming applications on chip multiprocessors. In *RTSS*, 2007. 19

[150] J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *RTAS*, 2008. 21

[151] Jianguo Yao, Xue Liu, Mingxuan Yuan, and Zonghua Gu. Online adaptive utilization control for real-time embedded multiprocessor systems. In *CODES+ISSS*, 2008. 18

[152] L.-T. Yeh and R. C. Chu. *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices.* ASME Press, 2002. 8

[153] Yuanfang Zhang, Christopher Gill, and Chenyang Lu. Real-time performance and middleware for multiprocessor and multicore linux platforms. In *RTCSA*, 2009. 3, 27, 33, 35

[154] Rongliang Zhou, Zhikui Wang, Cullen E. Bash, Alan McReynolds, Christopher Hoover, Rocky Shih, Niru Kumari, and Ratnesh K. Sharma. A holistic and optimal approach for data center cooling management. In *ACC*, 2011. 139

Vita

Xing Fu was born in Beijing China. He received the M.S. degree in Telecommunication and Information System in 2008, B.S. in Telecommunication Engineering in 2005, all from Beijing University of Posts and Telecommunications, China. He worked in Power-Aware Computer Systems Lab at University of Tennessee as a graduate research and teaching assistant from 2008 to 2012. He worked at VMWare a global leader in cloud infrastructure at Palo Alto, CA from 2012 to 2015 while continue to work on his dissertation.