MECHANISMS FOR COORDINATED POWER MANAGEMENT WITH APPLICATION TO COOPERATIVE DISTRIBUTED SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Ripal Nathuji

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
August 2008

MECHANISMS FOR COORDINATED POWER MANAGEMENT WITH APPLICATION TO COOPERATIVE DISTRIBUTED SYSTEMS

Approved by:

Karsten Schwan, Advisor
College of Computing
Georgia Institute of Technology

Sudha Yalamanchili, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Hsien-Hsin Sean Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology

Henry Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Gabriel Loh
College of Computing
Georgia Institute of Technology

Vijay Madisetti
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 21, 2008

To my parents for their years of support, my loving wife for being my best friend and my better half, and our daughter Elena.

ACKNOWLEDGEMENTS

There are are countless people who have been instrumental in helping me complete my Ph.D. by providing support and advice. I would like to take the opportunity to express my gratitude and appreciation for a few of them here.

First, I would like to thank my adviser, Dr. Karsten Schwan. He was always available to discuss ideas and provide suggestions for how to develop my Ph.D. research. Most, if not all, of the ideas presented in this thesis came from time spent in his office bouncing ideas around, hours which proved to be invaluable.

During my graduate years, I spent three summers with the Systems Technology Lab at Intel collaborating with the power management group. I'd like to thank some of the members of that team, including Ram Chary, Eugene Gorbatov, and Pradeep Sebestian for the great experiences, the hardware resources they always made available, and for the opportunity to learn from them while working hands on in an industry research lab.

I would also like to thank my co-advisor Dr. Sudha Yalamanchili, as well as the other members of my committee including Dr. Hsien-Hsin Sean Lee, Dr. Henry Owen, Dr. Gabriel Loh, and Dr. Vijay Madisetti. They were a source of tough questions and useful thoughts that helped improve the contents of this document.

I've been fortunate to work with, and sometimes just kill time with, some great students that have been around during the years. I thank them for the great discussions, constructive feedback, and sometimes random conversations.

Finally, I'd like to thank the members of my family who have been there for me: My parents for their support and advice throughout the years. My wife for her love and friendship that have helped me endure the ups and downs of graduate school. Last, but not least, the four legged canine members of our household, Raja, Sonya, and Ravi, for magically sensing when I needed a break from work, and always reminding me to have fun in life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Computing systems are experiencing a significant evolution triggered by the convergence of multiple technologies including multicore processor architectures, expanding I/O capabilities (e.g., storage and wireless communication), and virtualization solutions. The integration of these technologies has been driven by the need to deliver performance and functionality for applications being developed in emerging mobile and enterprise systems. These accomplishments, though, have come at the cost of increased power and thermal signatures of computing platforms. In response to the resulting power issues, power-centric policies have been deployed across all layers of the stack including platform hardware, operating systems, application middleware, and virtualization components. Effective active power management requires that these independent layers or components behave constructively to attain globally desirable benefits. Two choices are (1) to tightly integrate different policies using negotiated management decisions, and (2) to coordinate their use based on the localized policy decisions that are already part of modern computer architectures and software systems. Recognizing the realities of (2), the goal of this thesis is to identify, define, and evaluate novel system-level coordination mechanisms between diverse management components that exist across system layers. The end goal of these mechanisms, then, is to enable synergistic behaviors between management entities, across different levels of abstraction, and across different physical platforms to improve power management functionality. Contributions from this work include operating system level mechanisms that dynamically capture workload behavior thereby enabling power efficient scheduling, and system descriptor mechanisms that allow for improved workload allocation and resource management schemes. Finally, observing the strong need for coordination in managing virtualized systems due to the existence of multiple, independent system layers, a set of extensions to virtualization architectures for effectively coordinating VM management in datacenters are developed.

CHAPTER I

INTRODUCTION

Recent improvements in scalability and cost have helped fuel a surge in the size and capabilities of modern computing systems. This trend can be observed in mobile systems composed of smartphones and laptops as well as in the enterprise datacenters responsible for providing large scale services and computational power in IT infrastructures. The applications deployed on these computing fabrics have two requirements that impact the design of end platforms. First, application performance constraints demand ever increasing computational capabilities of underlying hardware components. In order to effectively meet these needs, processors rely on complex and multi-core microarchitectures and on fabrication technologies that extend smaller feature sizes, both of which increase power characteristics of the part. Second, media rich applications require new components and devices for communication and specialized processing. Incorporating these components also directly impacts the power consumption and densities of end platforms. The overall effect of these two design factors is an increased power envelope in emerging platforms, making active power management critical to their viability and success.

Active power management must go beyond the hardware aspects of computing systems to also consider the state of software with which accompanying applications are built. A common usage model in both mobile and server systems is that of a *cooperative distributed system* comprised of multiple physical platforms, where multiple software components operating on these platforms are jointly managed to maximize some global benefit. For example, in robotic systems, when a team of physically independent mobile robots cooperate to accomplish some mission, such as search and rescue, it may be desirable to minimize end to end delay for their sensor data processing tasks. Similarly, in datacenters, the goal is to provide required levels of service to each of multiple workloads while at the same time, maximizing return on investment from hardware and other physical resources (e.g., cooling and

power delivery). For power management, therefore, capabilities must exist to effectively manage the software components and the resources upon which they rely. However, the software components used by cooperative distributed application are structured as multiple layers, typically comprised of the hypervisor (or Virtual Machine Monitor – VMM), the operating systems running in virtual machines (guest VMs), middleware, and application code. As a result, effective power management must be able to exploit a wide range of associated management methods, at the VMM level including virtual machine migration, at the operating system level using techniques aware of current task loads or platform utilization, at the middleware level exploiting load balancing, and at the application level, leveraging semantic information to make potential quality/performance tradeoffs (e.g., image resolution for multimedia codes). In conclusion, then, (1) power management necessitates a careful balance of performance provided vs. the power consumed by underlying physical resources, and (2) it is directly affected by the operational behaviors of multiple system layers, each operating with different amounts of semantic information with which to make control decisions.

Active power management is enabled by new management capabilities added by hardware vendors to platforms. For example, modern processors provide dynamic voltage and frequency scaling capabilities (DVFS), P-states in ACPI terminology [41], as well as low power sleep states, C-states. Management policies utilizing these capabilities have been integrated into hardware itself [61] or into higher layers, including the operating system [43] that can directly set P- or C-states. At the same time, virtualization layers and middleware can perform resource allocation decisions that affect the power consumption of underlying components. These decisions consist of *allocation in time* where resources are time multiplexed to consuming applications, and *allocation in space* where the management policy within a layer of the system may decide how to allocate resources to workloads to improve power efficiency or thermal characteristics.

Given that each system layer, whether it be platform hardware, the VMM, the operating system, or the middleware, has different means to affect power consumption, it is clear that effective power management requires their multiple management components to cooperate.

For example, a platform manager can immediately place a hardware component into a sleep state when the virtualization layer decides to idle that component by consolidating a set of virtual machines (VMs). Conversely, a platform manager should not power down a component when the VMM is about to use it for additional load due to consolidation. Thus, to attain improvements via active power management implies the need to coordinate management actions across the multiple layers of the stack.

A key issue with achieving the benefits possible from coordinating decisions across multiple layers is that the management policies in distinct levels do not directly interact. For example, the firmware policy managing sleep states on 802.11 wireless cards runs independently of the operating system making the scheduling decisions that affect device usage. This separation can lead to reduced abilities in utilizing power management states, or even in destructive interference between policies resident in different layers, as evident from the consolidation example presented in the previous paragraph. Integration across different layers is a direct approach to solving this problem [98], but there are significant roadblocks or even disincentives concerning such integration.

First, complete integration is likely technically infeasible because of the scalability issue in optimizing multiple adaptive components [40, 95]. Specifically, since each management decision is multidimensional, an integrated solution faces the problem of scaling across a management space that grows significantly with each additional layer. Second, integration is impractical since it would require extensive cooperation between multiple hardware and software vendors, each of whom integrate power management for added value to their portion of the overall ecosystem. For example, distributed management in communication networks has been achieved by developing a well defined protocol implemented by agents/clients at each end point [25, 78, 114], but these interfaces must only be adopted by device manufacturers in a single layer of the system. Third, while there exists active work towards proposing standard interfaces for negotiating management decisions in computing systems [21, 50, 57], they are not widely adopted. Since power management involves management components across hardware and software layers such as middleware and VMMs that vary in complexity, the difficulty in defining an appropriate interface that is reasonable

across all of them makes widespread adoption prohibitively difficult.

1.1 Thesis Statement and Contributions

This dissertation investigates how to coordinate system wide decisions using lightweight mechanisms, to support the following hypothesis:
Coordination of independent, localized power management policy decisions can attain synergistic benefits while at the same time, preserve desired boundaries between component layers and maintain each layer's ability to implement and carry out its own management actions.

To support the hypothesis, we consider multiple examples of management opportunities that arise in distributed cooperative systems wherein overall power and/or energy characteristics can be improved by having components across system layers behave in a coordinated manner. Approaches, associated methods, and their implementation are evaluated experimentally using representative hardware platforms, in the mobile or in the enterprise space, depending upon the particular coordination being considered.

Specific contributions of this dissertation include the following:

- *Device windows and energy efficient scheduling.* Novel data structures, termed *device windows*, maintain power-relevant state in the OS level Energy Efficient Scheduling Infrastructure (EESI). New scheduling methods in EESI make decisions that improve power savings achieved by underlying hardware level power management policies.

- *CompatPM.* The CompatPM approach can coordinate middleware layer power management decisions with underlying power management schemes that cannot be directly controlled. Experimental evaluations use CompatPM attributes with event-based applications like those present in cooperative robotics applications.

- *Heterogeneity-Aware Load Management* (HALM). New performance and power prediction models enable Heterogeneity-Aware Load Management (HALM). HALM utilizes these models along with intelligent allocation policies for heterogeneous server systems, in order to provide reduced power consumption as well as improved power budgeting capabilities.

Table 1: System Coordination in Thesis Contributions.

	Mobile Domain		Server Domain	
	Device windows (EESI)	CompatPM attributes	HALM	VirtualPower & VPM tokens
Middleware/ Management Layer		X	X	X
Host/Guest OS	X			X
VMM				X
Hardware	X	X	X	X

- *VirtualPower.* VirtualPower is a novel architecture for power management in virtualized systems. It permits VMM layer management policies to coordinate with application specific requirements embedded into guest VMs' policies.

- *VirtualPower Management (VPM) Tokens.* VPM tokens provide a set of abstractions that can be used to employ power budgeting across virtualized applications and hardware in a manner that attempts to meet application requirements while at the same time maximizing overall measures of datacenter utility.

Table 1 identifies the layers between which management is coordinated for each of the contributions of this thesis. These contributions are presented in the order provided in the table, where the first few instances focus on specific coordinations between certain system layers or components, before we conclude with extensions targeted for virtualized systems where it is important to simultaneously coordinate between multiple layers.

1.2 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides a deeper discussion of the power consumption problem in modern computing systems as well as how these issues can be addressed across system layers. Coordination methods that apply to cooperative mobile systems are presented in Chapters 3 and 4, including the EESI and CompatPM components highlighted above. We then move on to server systems in Chapter 5, which describes the HALM workload allocation system for heterogeneous environments. Specifically addressing coordination extensions to virtualization technologies, Chapters 6 and 7

present the VirtualPower architecture and VPM token abstractions respectively. Concluding remarks and future work appear in Chapter 8.

CHAPTER II

POWER MANAGEMENT: TRENDS AND OPPORTUNITIES

This chapter presents a discussion of the underlying issues that create the need for active power management in computing systems. There are fundamental trends and characteristics of platforms and components that are driving platform power envelopes increasingly higher. Understanding the effects of these trends on both mobile and server systems is therefore critical. We review these aspects and then consider the types of management actions that can be used to address the need for power management across the layers of the stack that exist in cooperative distributed systems.

2.1 The Power Problem

Application performance has historically, and is continuing to be, a significant driving factor in computer systems. The performance obtainable on a given platform for many applications depends upon the capabilities of the system processor. This is typical, for example, with high performance or CPU bound workloads that do not perform large amounts of I/O operations. Therefore, processor designers have provided improved performance with each subsequent family of microprocessors. The performance of processors has been extended by elements such as advances in microarchitectures, (e.g., superscalar designs, multi-core chips) as well as increases in the frequencies at which these parts are clocked. These have helped the semiconductor industry sustain Moore's Law, which states that the number of transistors that can be placed on an integrated circuit increases exponentially, doubling approximately every two years [100].

Figure 1 illustrates the trend exhibited by Moore's Law with processors developed by Intel Corporation. The graph provides the transistor densities of parts spanning from the early 4004 part based upon a $10\mu m$ process, up to the newer Core 2 parts developed on a $65nm$ (and most recently a 45 nm) process. Improvements in fabrication technologies have enabled these smaller transistors and large scale integrated circuit designs. The transistors

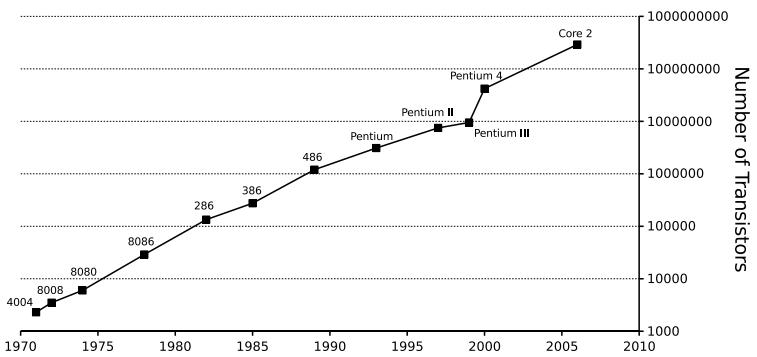


Figure 1: Transistor Densities on Intel Processors.

that compose these designs, then, consume both dynamic and leakage power where dynamic power consumption is the more significant factor, but leakage is becoming more of an issue with smaller feature sizes [46, 75]. These trends underscore the power issues created by the processor elements of modern platforms and how the increasing performance requirements of modern applications are pushing power envelopes higher. In addition, similar power effects caused by application needs for performance and I/O are also being observed across other components, such as memory, graphics cores, and communication elements. These combined effects necessitate the need for active power management across both mobile and enterprise domains, as we discuss next.

2.1.1 Mobile Trends

Power consumption is important in mobile platforms for two reasons. First, the power used by platform components is dissipated as heat. Since many mobile systems such as handhelds, cell phones, and laptops come into direct human contact, active power management can be necessary to prevent the system from becoming uncomfortably warm. This can be a significant issue with smaller form factor devices where there may be limited space for active cooling solutions like fans. The other driving factor for mobile power management is the restricted energy reserves of batteries that enable portability. The power consumed by mobile processors can quickly use up such energy capacities. Moreover, since mobile applications typically require a rich set of capabilities such as wireless communication, displays, and storage, the overall platform can be quite power hungry [72]. To illustrate the

resulting effective battery capacities, Figure 2 provides the approximate battery lifetimes of various mobile platforms.

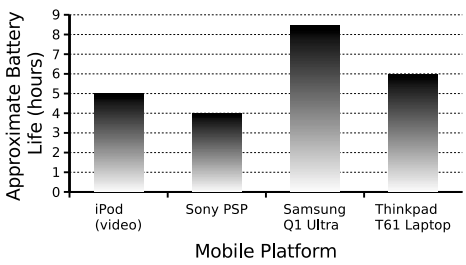


Figure 2: Battery Lifetimes of Portable Devices.

We observe from the figure that none of the devices are capable of being active for an entire twenty four hour period, and only one, the Samsung Q1 Ultra UMPC (ultra mobile PC) platform, can survive a typical eight hour work day. Given the desire for users to be able to go days without charging their mobile devices (e.g., cell phones that last for over a week in standby mode), there is considerable room for improvement. In the context of cooperative distributed mobile deployments such as a team of robots performing a search and rescue operation, the ability to extend the lifetime of the system can be critical to successful completion of the mission. The integration of active active power management techniques is necessary is necessary to address these needs.

2.1.2 Server Trends

Large scale datacenters have become increasingly prominent, both to meet the IT needs of individual companies, as well as to drive the services and content provided by the Internet. These environments consist of large numbers of server grade computing platforms, as well as the associated infrastructure for communication, power delivery/backup, and cooling. The sheer volume of platforms in these systems (now up to 100s of thousands of processors) create a significant need for power management due to the costs and limitations of power and cooling resources. The increasing densities highlighted earlier by Moore's law are contributing to this problem, as shown in Figure 3.

The figure provides theoretical rack densities, in terms of processing cores, for a typical

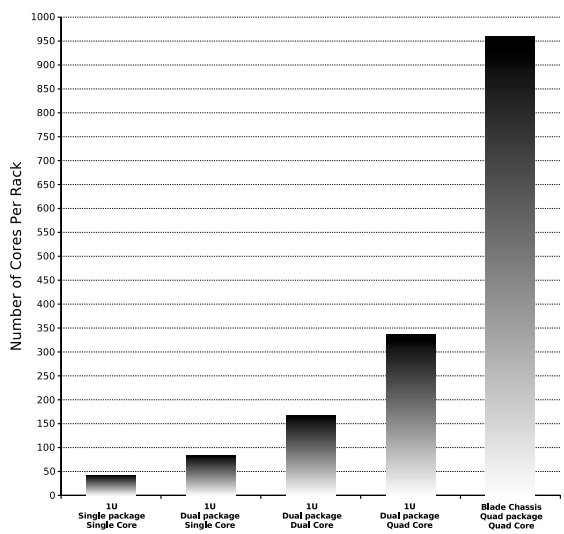


Figure 3: Datacenter Server Rack Processor Densities.

42U rack across various types of platforms. For a simple 1U server with a single package occupied by a single core chip, 42 processing cores can exist in a single rack. Using the same form factor, but extending to a dual package platform populated with quadcore chips, the density rises significantly, to 336 cores. More recently, blade based deployments developed by Supermicro will allow for 960 cores to be consolidated into a single rack using 7U chassis. Considering that a typical datacenter consists of thousands of racks in multiple aisles, hundreds of thousands of processing elements can be integrated into relatively small physical environments. Indeed, a profile of energy consumption in various datacenter deployments has shown that compute hardware alone can consume anywhere from 33% to 75% of total datacenter power [37]. The ability to provide system level solutions for power management, then, becomes critical to sustaining and growing these environments.

2.2 Methods to Improve Power Consumption

The trends in low level technologies reflected by Moore's Law, as well as the effects of the resulting power consumptions and densities in platforms that comprise distributed cooperative systems, clearly dictate the use of active power management to curb power consumption. Three specific system level capabilities exist to perform such actions. Since each is

considered in the scope of this thesis, we next discuss them briefly.

2.2.1 Hardware Management Capabilities

The need for power management has caused platform and component manufacturers to integrate power awareness into hardware. A variety of techniques are self contained in silicon and reduce energy consumption in processor caches, TLBs, or snoop logic transparently to higher level software [7, 8, 23, 24, 35]. Hardware support for multiple management states, on the other hand, can be used by firmware based policies to manage components, for example the timeout based policies used in wireless 802.11 devices and hard disks, or they can be exported for software level control via interfaces like ACPI [41]. A capability that is commonly available today for managing the dynamic power consumption of processors is dynamic voltage and frequency scaling (DVFS). DVFS exploits the relationship between power consumption and the frequency and voltage of a part, described by Equation 1 [80].

$$P \propto fV^2 \tag{1}$$

The equation highlights the impact that decreased operating points via reduced frequencies and voltages can have on power consumption. Since frequency is roughly linear to voltage [80], power can be modeled as scaling with f^3. This relationship is validated in Figure 4, using actual CPU power measurements from a PXA255 XScale processor.

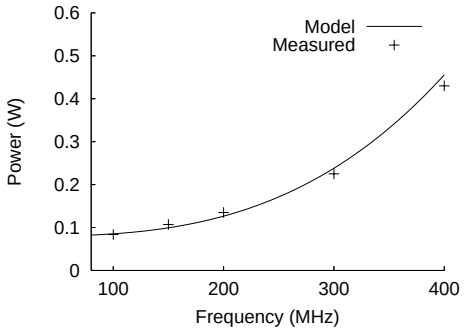


Figure 4: Hardware Manageability Effects on an XScale Processor.

The figure highlights the ability to effectively scale the power signature of the part using DVFS. Power can be reduced by up to a factor of five when active, by utilizing the various

performance (voltage/frequency) states supported by the processor. Of course, the use of these states must be balanced with the requirements of the currently running application. In addition to processor states, other components such as memory, disks, and communication devices also support (or will support) performance throttling, in order to allow the system to tune power consumption while meeting workload demands. The CompatPM and VirtualPower mechanisms developed in this thesis address how to achieve these tradeoffs at runtime, by coordinating the management components that utilize hardware supported states across system layers.

2.2.2 Resource Allocation in Time

Another method that can be used to affect power management within cooperative distributed systems is the manner in which physical resources are allocated in time. For example, the VMM layer of a virtualized system can modify the manner and time quantums used to schedule virtual resources used by applications onto physical resources, in order to increase idleness and thereby decrease the power consumption of components. Similarly, an operating system can reorder the execution of running processes in a manner that enables improved power savings for the underlying devices used by applications. In this thesis, we evaluate the use of these *resource allocation in time* techniques in multiple cooperative distributed scenarios, including both mobile and datacenter environments, by developing the approach and mechanisms used by EESI and VirtualPower.

2.2.3 Resource Allocation in Space

The final management opportunity existing in cooperative distributed systems, is the flexibility in mapping the computation or communication requirements of workloads across the physical resources available in the system. In contrast to allocation in time, which multiplexes device use across time, a *resource allocation in space* decision consists of assigning physical resources that are used concurrently. The power consumption tradeoffs between alternate mappings can be quite significant, especially in the context of heterogeneity. For example, since typical datacenters are comprised of heterogeneous physical platforms, the VMM layer can tradeoff the costs of various resources with the needs of workloads in a

Table 2: Summarized Usage of Power Management Methods.

Contribution	Hardware Management	Allocation in Time	Allocation in Space
Device windows (EESI)		X	
CompatPM attributes	X		
HALM			X
VirtualPower & VPM tokens	X	X	X

manner that optimizes for power consumption or to meet power budgets. We encounter these examples of resource allocation in space when considering datacenter level power management.

2.2.4 Power Management Methods in Thesis Contributions

Each of the coordination mechanisms proposed in this thesis utilize one or more of the power management methods outlined in this chapter. Table 2 summarizes the methods used by the components presented in the remaining chapters of this document. We begin by outlining coordination methodologies that use each of the various methods, and then progress to virtualized management schemes that can simultaneously leverage all three methods to improve power consumption.

CHAPTER III

EESI: DEVICE WINDOWS ENABLED OS SCHEDULING

Mobile devices have become a popular platform for personal, commercial, and military applications. Their portability and increasingly significant computation power allow them to be effectively utilized in these various environments. In distributed mobile systems, multiple workload instances execute simultaneously across a set of participating platforms. This is especially true when workloads may be allocated across cooperating nodes in a manner that optimizes some global objective such as battery life. Many of these software components rely upon I/O devices present on the end devices on which they run. For example, a distributed visualization workload may utilize a local wireless interface to receive data from external nodes [116] as well as a display device. At the same time, there may be local computations that are running that do not rely upon devices, but instead are completely CPU bound. This mix of workloads creates an interesting opportunity to leverage resource allocation across time to impact power consumption.

The addition of peripheral I/O devices to mobile platforms has allowed for improved capabilities, but has also increased power consumption and made effective energy management a necessary design component. To address this, many peripheral devices can autonomously reduce power consumption using timeout based policies integrated into device firmware that make use of various low power states. In this chapter, we consider developing mechanisms, termed device windows, that support a software based scheduling infrastructure for energy management that help improve the savings obtained by these device specific managers. In particular, the scheduler leverages workload mixes with varying behavior to perform coordinated scheduling decisions that behave synergistically with underlying, independent timeout based policies.

When designing software-based power management schemes, design alternatives include incorporating additional or modified programming APIs, utilizing compiler driven hints,

or devising solutions completely internal to the operating system. Solutions that require adherence to a new interface for performing device I/O provide increased information to the power management subsystem at the cost of transparency and backwards compatibility [111]. Our approach provides a solution that requires no modifications to existing APIs and is completely transparent to the programmer. In addition, we also consider how the system can utilize possible source annotations in our algorithms and infrastructure. Since these compiler-driven hints are not required, legacy binaries can easily be supported.

The Energy Efficient Scheduling Infrastructure (EESI) approach to power management relies upon a novel mechanism, device windows, to obtain runtime device usage information from dynamically monitored state as well as compiler annotation-based hints to correlate device usage to application processes [86]. This information is then provided to the process scheduler in the operating system, which attempts to order processes so as to (1) increase the burstiness of device accesses, and (2) provide increased idle periods to exploit low power states of peripheral devices that utilize timeout-based sleep mechanisms without directly interacting with the firmware policies.

3.1 Motivation: Aggregating Device Accesses

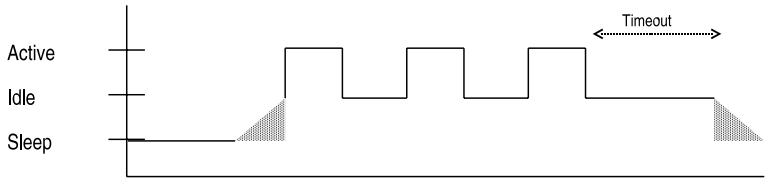
When using timeout-based mechanisms to place devices into low power modes, both the size and distribution of idle periods experienced by the device determine the amount of energy that is saved. Given a particular idle period, a device is able to transition into a low power state only if the idle period is longer than the timeout period assigned to the device. Moreover, if a device is able to transition into a low power state, the amount of time actually spent conserving energy (t_{sleep}) is equal to the length of the idle period (t_{idle}) minus the sum of the timeout period (t_{to}), the time to perform an idle-to-sleep state transition (δ_{hl}), and the time to perform a sleep-to-idle transition (δ_{lh}) for the subsequent request.[1] This relationship is provided by Equation 2. Since t_{to}, δ_{hl}, and δ_{lh} are all constant terms, this tells us that to maximize the effectiveness of a sleep period, the idle time should be maximized so as to amortize the cost of timeouts and state transitions in the most effective

[1]It should be noted that in our notation t_{sleep}, δ_{hl}, and δ_{lh} are implicitly subcomponents of t_{idle}

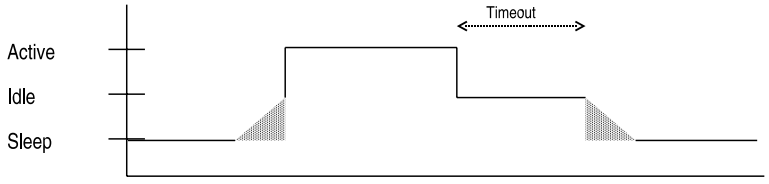
manner.

$$t_{sleep} = t_{idle} - (t_{to} + \delta_{hl} + \delta_{lh}) \tag{2}$$

In addition to the sizes of idle periods, their distributions also play a key role. Figure 5 illustrates an example of three periods of device activity. In Figure 5(a), short idle periods are distributed amongst the requests, ending with a final idle period leading to a state transition. Figure 5(b) depicts the same scenario, but with the three periods of device activity aggregated together. Though the active time of the device remains the same, it is now able to transition to sleep mode sooner. Assuming the subsequent request that "awakens" the device arrives at the same point in time for both scenarios, the bursted approach maximizes Equation 2 for the final idle period by increasing t_{idle}. Moreover, if there is a subsequent request immediately prior to the point where the device times out in the non-bursted scenario, the device experiences no sleep time at all, whereas in the bursted scenario some savings are still achieved. Therefore, bursting device accesses not only provides longer idle periods resulting in increased sleep time, but can also create periods of low power state that would otherwise not occur.



(a) Non-bursted Accesses



(b) Bursted Accesses

Figure 5: Device State with Varying Access Patterns.

To fully illustrate the energy benefits of bursting device accesses, we consider the energy consumption of a device over a given time period given by Equation 3. Regardless of

whether or not accesses are bursted, the active time is equivalent. Therefore, the first component of the equation can be ignored when comparing the energy footprint of the approaches. In other words, since ($t = t_{active} + t_{idle}$), any benefit derived from bursting accesses is dependent upon how much of t_{idle}, if any, is distributed into t_{sleep}, $n_{hl}\delta_{hl}$, and $n_{lh}\delta_{lh}$ (n_{hl} and n_{lh} denote the number of state transitions from high to low and low to high respectively). It has already been shown that access bursting increases t_{sleep}. Another, possibly significant, implication of access bursting is to reduce n_{hl} and n_{lh}. The energy benefits of this effect are illustrated by Equation 3.

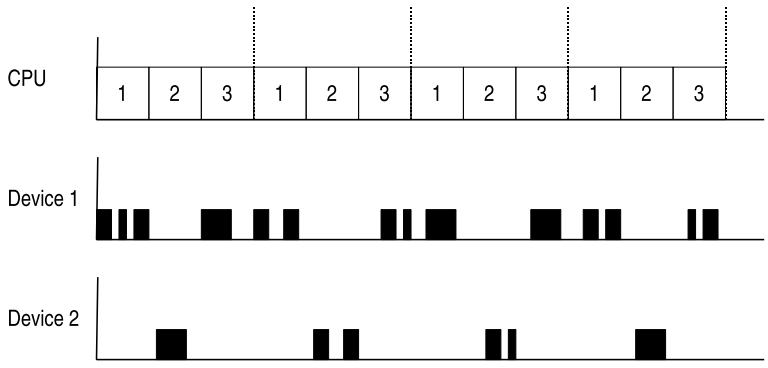
$$E(t) = P_{active}t_{active} + P_{idle}(t_{idle} - t_{sleep} - n_{hl}\delta_{hl} - n_{lh}\delta_{lh}) + P_{sleep}t_{sleep} + P_{hl}n_{hl}\delta_{hl} + P_{lh}n_{lh}\delta_{lh} \quad (3)$$

3.2 A Process Scheduling Approach

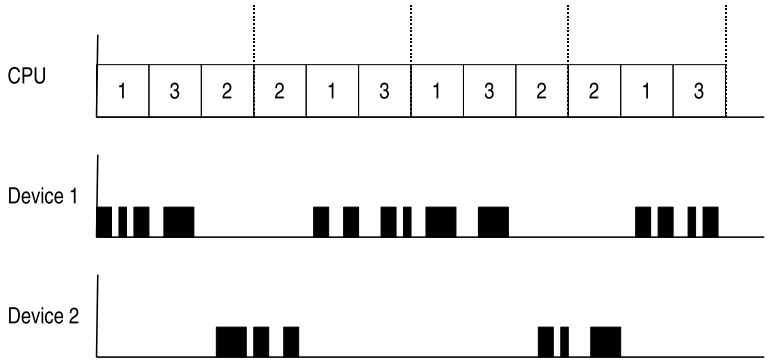
Having illustrated and discussed the benefits of bursting devices accesses, a question that arises is where in the system access aggregation should be performed. One option is to provide device bursting at the lowest possible layer, the device driver [92]. The problems of such an approach are twofold. First, bursting accesses at the driver may alter the delay assumptions made by higher level protocols. For example, bursting network packets at the driver level sometimes requires the delaying of packets. This may cause TCP to estimate varying round-trip-times, which in turn can result in changes to throughput for the connection. A second drawback to performing bursting at a low level is that it becomes difficult to cleanly integrate bursting with other power management techniques such as DFS/DVS. Instead, it would be beneficial to integrate various power management techniques at a higher level in the operating system. Therefore, with EESI, we perform device access bursting using the process scheduler.

3.2.1 Scheduling for Access Bursting

The operating system process scheduler can be utilized to perform access bursting at a coarse level. Let us assume that the scheduler is aware of the device(s) a given process will



(a) Non-bursted Scheduling



(b) Bursted Scheduling

Figure 6: Access Distributions with Varying Scheduling Approaches.

access during its next execution phase. When the current process uses up its time quantum (or relinquishes control), the scheduler attempts to choose a process that requires a similar set of devices, thereby bursting accesses to used devices and extending the idle periods of unused devices. Figure 6 illustrates an example scenario with two devices. Processes one and three access device 1, and process two accesses device 2. By comparing Figure 6(b) to Figure 6(a), we observe that by simply reordering the FIFO scheduling order to a bursted approach, the maximum idle periods experienced by both devices at least double in length. It should also be emphasized that these idle time benefits are obtained while still providing the same amount of service (*i.e.* execution time) to all processes. Indeed, processes are only "reshuffled" within the bounds of a scheduling epoch.

Implementing a scheduler that behaves as the one in Figure 6 raises a few issues. First, there is the issue of how the scheduler determines what type of device set would be optimal

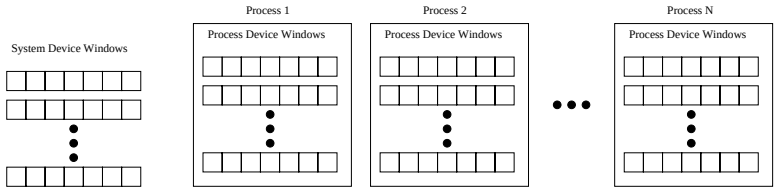


Figure 7: System Level Device Window Design.

for the next process. This set would likely be determined using some sort of system state such as recent device activity behavior. Second, there must be some type of likelihood indicator to determine what devices a process will use when scheduled. For the moment, let us assume that given a process p, the maximizing function M(p) provides a heuristic of how suited a process is to continue bursting accesses on devices and/or extend idle periods. Given M(p), Algorithm 1 provides the EESI scheduling algorithm which attempts to provide bursty access behavior.

Algorithm 1 Device Window Scheduling.

$\forall$ *Processes* $i, j \in$ *Runnable*
    Choose *Process* i s.t. $M(i) \geq M(j)$ $(\forall j \neq i)$

3.2.2 Device Windows - Data Structures for Device Usage Correlation

A remaining issue that needs to be addressed is how the scheduler obtains system-level device usage patterns and the methodology for predicting future accesses by a process. In our system, we utilize special kernel-level data structures for this purpose called device windows. Device windows are used to reflect "affinity" of device usage by the system, and by the processes. Therefore, there are two types of device windows, *system device windows* and *process device windows*. System device windows are used to collect data for determining the activity level of devices from a system perspective. Therefore, there is an individual system device window for each device. Similarly, there are process device windows associated for each task that are used to predict whether that device will be used the next time the process executes on the CPU. Figure 7 illustrates the design of our system with respect to device window state.

Semantically, device windows are used in a very simple manner. For each type of

device, there is an associated system device window value threshold and process device window value threshold. Given these values, if the value of a system device window is greater than its threshold, that device is considered active in the system, and the scheduler will attempt to burst accesses to it. Similarly, if the value of a process device window is greater than its threshold, the scheduler will predict that the process will access the device whenever it next executes on the CPU. The set of devices that are considered "active" for a process, and similarly for the system, are defined as the Active Device Set (ADS). Using the notions of values and thresholds, Table 3 outlines the parameters that allow us to define the maximizing function M(p) for our EESI scheduler in Equation 4. Additional descriptions of device windows can also be found in [83].

$$M(p) = \sum_{i=1}^{K} \lambda_i (S_{1,i} P_{1,i,p} W_{i,p} + S_{0,i} P_{0,i,p} (W_{i,max} - W_{i,p})) \tag{4}$$

Table 3: Device Window Parameter Definitions.

Parameter	Definition
$S_{0,i}$	1 if System device window value less than threshold for device i, 0 otherwise
$S_{1,i}$	1 if System device window value greater than threshold for device i, 0 otherwise
$P_{0,i,p}$	1 if Process device window value less than threshold for process p and device i, 0 otherwise
$P_{1,i,p}$	1 if Process device window value greater than threshold for process p and device i, 0 otherwise
$W_{i,p}$	Process window device value for process p device i
$W_{i,max}$	Maximum window device value for device i
λ_i	Weight value for device i

3.3 EESI System Design

We next describe the realization of the EESI system in a variety of scenarios. First, we consider the context of utilizing only dynamic usage state wherein devices windows are updated based upon application behavior monitored at runtime. Within this context we develop an EESI scheduler based upon the default Linux 2.4 scheduler, as well as a real time scheduling algorithm. Second, we describe how the EESI architecture can combine dynamic

updates with device usage hints placed into application source code by an annotator.

3.3.1 EESI with Dynamic State

A dynamic scheme is dependent upon the creation and use of history information to perform predictions of device usage behavior. Figure 8 illustrates how device windows are incorporated into the scheduling mechanics of a reactive system. Whenever a process is removed from the CPU, run-time monitoring state of device usage is used to update system device windows and the device windows of the particular process. To describe the manner in which these windows are updated, we first present how the abstraction of device windows is implemented.

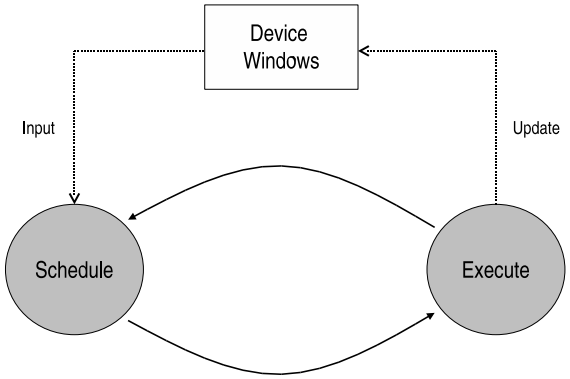


Figure 8: Dynamic Device Window Update Approach.

Both system and process device windows are treated as statically sized sliding windows. Each time a process is removed from the processor, each of its device windows is shifted to the right. For all devices that were used during the execution period of the process, the 'last' (leftmost) bit is set in the respective process device window. The system device windows are updated in a similar fashion each time a process is removed from the processor.

Given this update mechanism, the value of device windows can be interpreted in one of two ways. Device windows can be compared by the binary values of their bits. This approach exponentially reduces the significance of a bit whenever the window is shifted. We use this method of calculating values for process device windows. For this type of window the maximum value is $(2^l\text{-}1)$. Whenever the age of a particular bit is not significantly relevant, another possible function is to count the number of 'set' bits in the device window. This

Table 4: DWCS Precedence Rules.

Earliest deadline first (EDF)
Equal deadlines, then order tightest window-constraint first
Equal deadlines and zero window-constraints, then order highest window-denominator first
Equal deadlines and equal non-zero window-constraints, then order lowest window-numerator first
All other cases: first-come first-serve

is the approach we use to interpret system device window values since the algorithm only requires knowledge of whether a device has been active in the recent past and not when in the past it was used. In this case the maximum value of a window is equal to the number of bits associated with it, or its length l.

3.3.2 Real-Time EESI with Dynamic State

To demonstrate the flexibility of the device window enabled EESI system, we also develop a design which builds upon a real-time scheduling algorithm. DWCS (Dynamic Window-Constrained Scheduler) [112, 113] is a hard real-time scheduling algorithm. With DWCS, each process is assigned a period T, a service time C, and a window-constraint $\frac{x}{y}$. In each period, DWCS attempts to service a process for C time units, and it guarantees that it will service a process $y - x$ periods in a window of y periods if the CPU utilization is less than or equal to 100%. Note that if a window constraint is not specified, the DWCS scheduler behaves the same as an EDF scheduler, where the period T_i of a process i is used to determine deadlines. Table 4 lists the precedence rules used by DWCS to determine what process to schedule next [113].

The DWCS scheduler employs a utilization-based admission control mechanism. In particular, the utilization of the system at any given time is given by Equation 5. Given this description of the DWCS algorithm, the resulting problem is how to combine its benefits of performance guarantees with the EESI approach. We discuss this problem by illustrating the types of functions the EESI scheduler should be able to perform, and the mechanisms that may be used to implement such functionality.

$$U = \sum (1 - \frac{x_i}{y_i}) * \frac{C_i}{T_i} \tag{5}$$

As previously described, device window scheduling seeks to reorder process executions so as to burst the idle or the busy periods of a device. In the context of the DWCS real-time scheduler, this translates into possibly scheduling a non-EDF task (including sporadic or best-effort tasks) before the EDF process, or allowing the CPU to be idle when it would otherwise execute the EDF process (*e.g* by scheduling the *idle task* in Linux). In either of these situations, the modified behavior must not violate the original guarantees provided by the DWCS scheduling parameters (C,T,$\frac{x}{y}$).

As an example. let us assume we have three processes p_1, p_2,and p_3 with deadlines $D_1 < D_2 < D_3$. Let us further assume that there is only one device under consideration, and it belongs in the ADS of p_1 and p_2, but not in the system ADS. Given this scenario, our device window scheduler would execute process p_3 ahead of p_1 and p_2 in order to attempt to extend the idle period of the device. This behavior conflicts with the scheduling precedence of DWCS, however. Moreover, the process schedule can be reordered in this manner only if it does not result in any DWCS constraint violations. For simplicity, let us assume all three processes are scheduled as EDF (*i.e.* there are no window constraints). Then, the execution of p_1 and p_2 can only be delayed if $(D_2 - t) \geq (C_2 + C_1 + C_3)$ and $(D_1 - t) \geq (C_1 + C_3)$, where t is the current time of the system.

To generalize this argument, let p_{dwcs} be the process that should be scheduled according to the DWCS scheduler, and p_{dw} be the process selected by the device window scheduling algorithm. If the two processes are the same, there is no conflict. Otherwise, p_{dw} can only be safely scheduled if the following holds for all p_i with $D_i < D_{dw}$:

EDF Process Delay Constraint: *If $D_i < D_j$, the EDF process p_j can be scheduled before process p_i if, and only if, the slack time of process p_i is greater than or equal to the sum of C_j and the service time C_k of all processes such that $D_k < D_i$.*

Evaluating this constraint can create substantial overhead at each scheduling point in order to determine if p_{dw} can be scheduled. Indeed, this overhead remains even when the system utilization is high and tasks cannot be reordered, or idle periods introduced.

Table 5: Modified DWCS Precedence Rules.

Earliest deadline first (EDF)
Equal deadlines, then order by increasing value of device window function $M(p)$
All other cases: first-come first-serve

Therefore, utilizing this constraint directly is a poor solution. Instead, we propose a solution that introduces minimum complexity and gracefully handles increased system load:

$$U_{slack\_task} = 1 - \sum (1 - \frac{x_i}{y_i}) * \frac{C_i}{T_i} \tag{6}$$

The solution consists of defining a *slack task* in the system that is given the utilization in Equation 6. This task is a pseudo task that is included in the state of our real-time device window scheduler. The *slack task* is dynamically assigned a service time C_{slack_task} and period T_{slack_task}, so that its utilization is equal to U_{slack_task}. The deadline and service time of this pseudo task is updated along with other tasks. At each scheduling decision, the scheduler compares the service constraints of the *slack task* with the process selected by the DWCS scheduler. In particular, we check to see if $D_{slack\_task} \leq D_{dwcs}$. If this condition holds, our hybrid scheduler can select another task, namely p_{dw}, ahead of the EDF task, or keep the CPU idle, without violating DWCS constraints. The alternative process p_{dw} is selected using the device window algorithm given in Algorithm 2. If p_{dw} is not a real-time task, the scheduler denotes the *slack task* as having been serviced for that time unit. In this manner, the scheduler differentiates when it consumes system slack by executing a sporadic task, or keeping the CPU idle, and when it merely reorganizes the execution order of real-time tasks, in which case the slack utilization given in Equation 6 is not consumed.

In order to construct our real-time EESI scheduler, we must also determine how window-constraints should be handled. As Equation 6 describes, the utilization allocated to the *slack task* includes the utilization attempted by DWCS, but it does not guarantee to provide that to processes. The idea is that for many programs that specify non-zero window-constraints, such as multimedia applications, the application-level QoS is not significantly hampered if the task receives its minimum specified utilization given in Equation 5. In this case,

the system QoS may be maximized by placing power considerations first. Therefore, in addition to allowing the real-time device window scheduler to select a task based on the device window scheduling algorithm when the *slack task* is EDF, we also allow the scheduler to optimize for power if the EDF process has room in its current window-constraint (*i.e.* it has not missed x deadlines in the current window of y periods). Since we now handle window-constraints in this manner, we redefine the DWCS scheduling constraints to those shown in Table 5. Given these mechanisms, we next present the overall real-time device window scheduling algorithm.

Algorithm 2 Real-Time Device Window Scheduling.

p_{dwcs}=$DWCS\_Scheduler(run\_queue)$
p_{dw}=p_i s.t. $M(i) \geq M(j)$ $(\forall j \neq i)$

```
if ((slack_task.deadline ≤ p_dwcs.deadline) OR
    (room in p_dwcs.current_window)) then
  if ((system.ADS == NULL) AND
      (p_dw.ADS != NULL)) then
    slack_task.time_serviced++
    CHOOSE idle_task
  else
    if p_dw is a non-RT process then
      slack_task.time_serviced++
    end if
    CHOOSE p_dw
  end if
else
  CHOOSE p_dwcs
end if
```

Our final real-time device window scheduling algorithm is presented in Algorithm 2. As previously discussed, we first determine the tasks that would be scheduled by the DWCS and basic device window scheduler independently, where now the DWCS scheduler utilizes the precedence rules given in Table 5. Then, if the *slack task* is the EDF process, or p_{dwcs} has suitable window constraints, we proceed to possibly modify the DWCS schedule since we're guaranteed not to violate any guarantees made by the DWCS scheduler. Otherwise, we select the DWCS task. Note, that as the system utilization approaches 100%, the ability of our scheduler to perform power optimizations is reduced since the period of the

slack task must be extended to reduce its utilization. This corresponds to the fact that as utilization increases, there is a reduced ability to delay tasks without violating their real-time constraints.

3.3.3 Use of Source Annotations with EESI

Our EESI process scheduler utilizes device window information to determine scheduling priority. In this section, we investigate using annotations as an additional update mechanism for process device windows.

Annotations convey the following information to the kernel:

- An identifier used to uniquely identify a particular annotation within a process
- An identifier denoting the I/O resource that the annotation pertains to

Annotations are automatically inserted into the source code by an annotator, which can either be integrated into the compiler, or used as a precompilation tool. Annotations are of two types–ones that precede a resource usage and ones that follow it. An annotation describes the resource that will be used next in the former case, and the resource that was just used in the latter.

Our annotator design stems from the observation that applications need to access devices through system calls, which are then managed in kernel space. System calls are typically called through wrapper functions that prepare the arguments to the actual call. For example, the `fread()` function, which is part of the `stdio` library, ultimately calls the `read()` system call. The annotator looks for such library calls, that ultimately result in system calls which may use devices, and inserts annotations both before and after each call occurring in the source code.

We have implemented the annotator as a Perl script that takes source files in C as input. It inserts annotations according to the rules discussed to produce an annotated version of the code as the output. The modified source is then compiled in the usual manner to produce an executable binary. The overall process is illustrated in the Figure 9. The annotator has a list of standard function calls that make system calls and the devices corresponding to

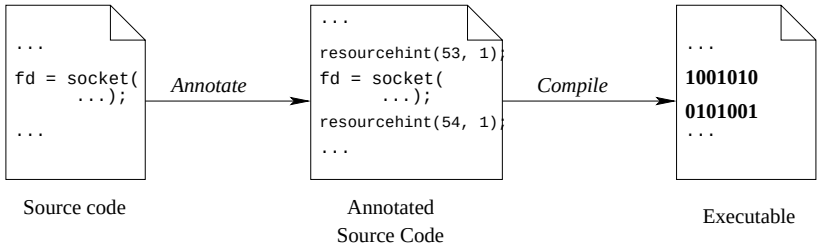


Figure 9: Source Code Annotation Approach.

the system calls. For instance, the `socket()` function is associated with a network device. The annotator compares the source code with the list of functions, line-by-line, to detect any occurrences. If there is a match, the annotator adds the annotation in the form of a `resourcehint()` call (which is defined as `void resourcehint(int, int);`). This function call takes in a unique annotation identifier as the first argument (implemented as a simple counter in the annotator) and an integer denoting the device corresponding to the system call under analysis as the second.

The decision to keep our annotation scheme simple has been intentional. In addition to minimizing implementation overheads, a simple approach permits one to automatically insert annotations directly even into the executable. Since the unique annotation identifiers merely serve to establish application context in kernel space, the same effect can be achieved by examining the stack during a system call. Such an approach can be used for applications where a lack of source code availability prevents recompilation.

The `resourcehint()` call interfaces with the kernel as a system call and provides the annotation and resource identifiers. A direct-mapped *annotation cache* is also maintained in the kernel and is updated during every `resourcehint()` call. As discussed before, annotations can signal an upcoming device call, or the completion of the call. We term the former an 'entering' annotation and the latter a 'leaving' annotation. Thus, an entering annotation signals the beginning of potential device usage, and a leaving annotation signals its end. The time elapsed between a leaving annotation and the next entering annotation that uses the same device indicates how frequently a process accesses a device (we do not keep track of the time between an entering annotation and a leaving one as it is not useful for our requirements). This information is collected by storing the current value of `jiffies`

during every leaving annotation and comparing it to the current value of `jiffies` during the next entering annotation that uses the same device. Existing estimates of inter-access times for a given device and process are updated using an exponentially weighted moving averaging (with $\alpha = 1/2$) as shown in Equation 7. These estimates are also maintained in the annotation cache.

$$t_{estimate} = \alpha t_{estimate} + (1 - \alpha) t_{observed} \tag{7}$$

Algorithm 3 Device Window Updates with Annotation Hints.

During each $resourcehint(uid, dev)$ call,
  $\forall$ *devices* d s.t. $d \neq dev$,
    if $next\_use(d) > threshold(d)$ then set $W_{d,p} = 0$
In case of a leaving $resourcehint(uid, dev)$ call,
  if $next\_use(dev) > threshold(dev)$ then
    set $W_{dev,p} = 0$
In case of an entering $resourcehint(uid, dev)$ call,
  set $W_{dev,p} = 2^l - 1$

Using all of this information, in a static scheme, process device windows can be updated whenever information is obtained from annotations. The resulting update algorithm is shown in Algorithm 3. In the algorithm, p denotes the current process and l denotes the size of the device window in bits. The process device window of a device associated with the calling `resourcehint` is possibly set to zero or maximum based on whether it is a leaving or entering call. An entering call indicates that the device will be used shortly, so we preemptively maximize the process device window value. Similarly, we would need to set the process device window value to zero on a leaving call, since the device is not likely to be used. However, if the device will be used after a short while, zeroing the device window value preemptively will be detrimental. Hence we selectively set it to zero by comparing the expected next use of the device (based on history) against a threshold time value (dependent on the device). For other devices, a similar check is performed to decide if the device window should be set to zero. In our implementation, we use a constant threshold value of two clock ticks (`jiffies`) for all devices.

Though Algorithm 3 provides a means of using static hints alone, our goal is to combine

the use of annotation hints with dynamic updates. The introduction of compile-time hints allows process device windows to more rapidly adapt to changes in device usage patterns. In other words, when a process stops using a device, with the dynamic algorithm, the device window takes a non-negligible time to reduce to zero value since any set bits must be shifted out. On the other hand, annotations can provide enough information to immediately clear a window. Similar behavior is observed when a process starts to use a device. However, a static method fails when a system call selectively uses a device. This scenario occurs, for instance, when an application writes to the disk. The actual write doesn't occur until the dirty buffers are flushed by the kernel. A scheme that relies on annotations alone to set device window values will incorrectly predict device usage.

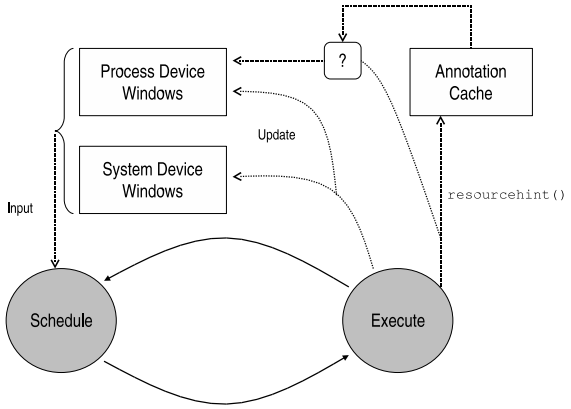


Figure 10: EESI Device Window Update System.

To avoid these 'false positives', we allow an entering annotation to assign the maximum value to the device window, but also check if the device is subsequently used. If it turns out that the device is not actually used (this check is done at the corresponding leaving annotation), a flag in the annotation cache is set so that the next time the annotation is encountered, the maximum value is not assigned to the device window. In this case, the device window is set using dynamic device usage information only. A leaving annotation, on the other hand, doesn't present such a problem since it is expected that a device will not be used outside the system call.[2] In other words, we combine the two approaches by

[2]Buffering can sometimes lead to such a scenario, in which case the dynamic update can set the appropriate device window value

allowing both mechanisms to update process device windows simultaneously. If it is found during execution time that a particular entering annotation is ineffective, it is subsequently ignored and the system relies solely on dynamic updates to predict device usage in that area of the application. Figure 10 presents the design of this collaborative update mechanism used with an EESI system that can utilize both dynamic and static hints.

3.4 Experimental Evaluation

3.4.1 Intel Sitsang Evaluation Platform

The Intel Sitsang evaluation platform shown in Figure 11 is designed around the PXA255 processor. The PXA255 integrates the Intel XScale microarchitecture with various controllers and peripherals including a memory controller, universal serial bus support, and a DMA controller. Additional features include slots for CompactFlash and Secure Digital memory cards. We utilize the CompactFlash slot for our 802.11 network card (Linksys WCF12), and a USB port for a flash storage solution.

Figure 11: The Intel Sitsang Platform.

We choose to utilize the Sitsang platform in our experiments for various reasons. First, the board is representative of future 'high end' mobile and embedded platforms such as cellular phones. Such platforms are intended to replace what are now multiple devices carried by end users like PDAs, cell phones, or calculators into single, multi-function devices

able to carry out a wide variety of tasks. The XScale processor is also an energy efficient core with dynamic frequency and voltage switching capabilities. The Sitsang combines all of these attributes with a PDA-like form factor.

To test the ability of EESI to provide energy benefits for timeout based systems, we attach a wireless network card and a flash device to the Sitsang. For the wireless card, we enable the normal 802.11 sleep mode. The Sitsang is also designed to provide the ability to toggle power to various subsystems. Since the flash device is powered from the USB bus, we use this feature to implement a software based timeout mechanism. This mechanism powers down the flash device whenever it is idle for 100ms. Future embedded platforms will likely be designed for aggressive power management of subsystems similar to this approach. In some experiments, a USB disk was also utilized as an additional peripheral device.

3.4.2 Power Measurement Methodology

The power measurement methodology used to evaluate the device windows based EESI architecture is based upon a test circuit designed with the SCD10PUR current sensor. The SCD10PUR measures currents from 0A to 10A with linear voltage scaling from 2.5V to 4.5V respectively. If we label the output from the SCD10PUR as V_t and the measured current as I Equation 8 provides the relationship between the two.

$$I = 5(V_t - 2.5) \tag{8}$$

In order to measure the small voltage variances with lab test equipment, we must amplify the voltage shifts in the output of the SCD10PUR part. Therefore, we utilize a test circuit design shown in Figure 12.

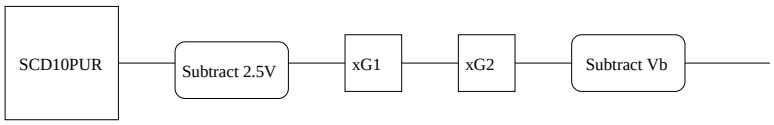


Figure 12: SCD10PUR Based Measurement Circuit Design.

Using this overall design, the relationship between the output of the test circuit, V_t, and the bias current V_b is provided by Equation 9.

$$V_o = (V_t - 2.5)G_1G_2 - V_b \tag{9}$$

Finally, the actual circuit implementation of this design used in our experiments is illustrated in Figure 13.

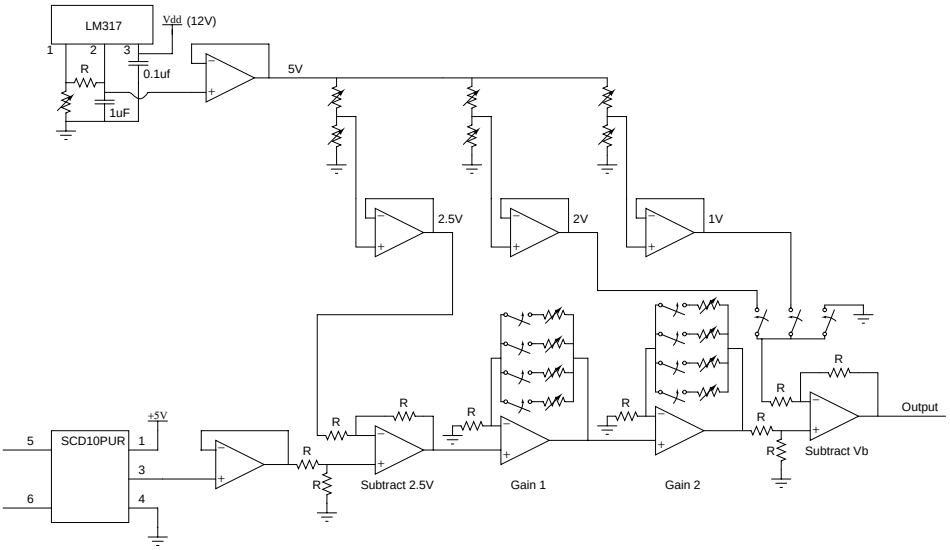


Figure 13: SCD10PUR Based Measurement Circuit Implementation.

3.4.3 EESI Results with Dynamic State

3.4.3.1 Default Linux Scheduler Implementation

The ability to save energy through process scheduling strongly depends on the mix of application processes running on the mobile device. The experiments conducted consist of applications that would reasonably be executed on a mobile device in a cooperative distributed system. Table 6 lists the applications being used. Our experimental evaluation attempts to reflect reasonable platform usage by creating different scenarios, each comprised

Table 6: EESI Experimental Applications with Default Linux Scheduler.

Application	Description	Devices Used
A	Background tasks	None
B	Data computation and transmission	802.11
C	Periodic data update	802.11
D	Audio Streaming	802.11
E	Image Streaming	802.11
F	Audio Streaming w/Save	802.11 & Disk

Table 7: EESI Experimental Scenarios with Default Linux Scheduler.

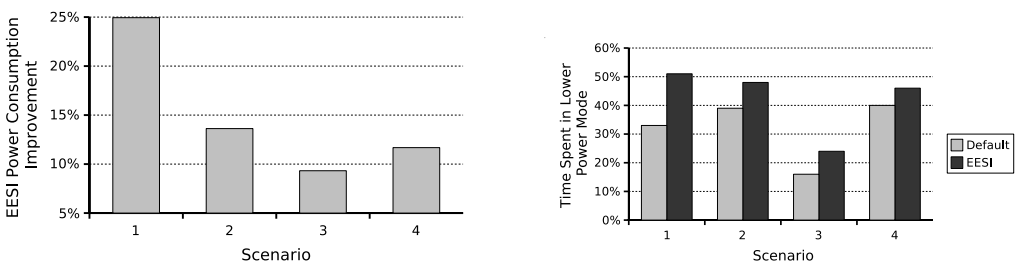
Scenario	Applications Used
1	A,B,C
2	A,B,C,D
3	A,B,C,E
4	A,B,C,F

of multiple applications. All of these scenarios include sporadic background computational tasks (application *A*). Application *B* is a simple application that performs computations and then transmits data over the network. This application mimics the behavior of many programs executed on mobile platforms, including those that manipulate data obtained from sensors or context, and then send it back to a data sink. Application *C* imitates the type of updates a cellular phone or PDA may send over the network to a back end server. The remaining applications are used to capture multimedia program behavior.

Given the list of applications, Table 7 depicts the particular experimental scenarios evaluated. We present the experimental results, consisting of power consumption data from the system 802.11 device, with these mixes in Figure 14. First, it is apparent that EESI significantly impacts the average power consumption of the wireless device, providing savings from about 10%-25%. We also see substantial improvements in the time spent in low power mode, 6%-18%.

3.4.3.2 DWCS Scheduler Implementation

In evaluating our DWCS EESI implementation, we utilize a similar set of applications as with the default Linux scenario, but define the real-time constraints shown in Table 8. The devices used for each workload are the same as in Table 6.



(a) Power Benefits

(b) Low Power State Usage

Figure 14: Experimental Results with Default Linux Scheduler.

Table 8: EESI Experimental Applications with DWCS Scheduler.

Application	Description	Period (T)	Service Time (C)
A	Background task	200 ms	20 ms
B	Data computation and transmission	200 ms	20 ms
C	Periodic data update	200 ms	20 ms
D	Image Streaming	200 ms	40 ms
E	Audio Streaming w/Save	200 ms	40 ms

Similarly to the previous results, we define a set of experimental mix scenarios as defined in Table 9.

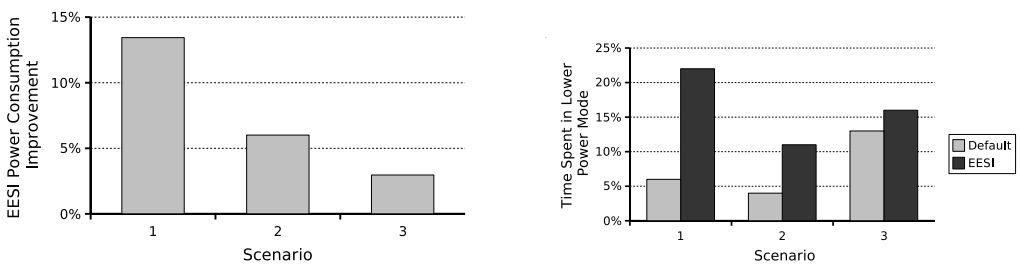
Table 9: EESI Experimental Scenarios with DWCS Scheduler.

Scenario	Applications Used
1	A,B,C
2	A,B,C,D
3	A,B,C,E

Again, the power results from the platform 802.11 device are provided in Figure 15, where all applications have a zero loss tolerance. We observe several facts. First, we can see that EESI can still significantly impact the average power consumption of the wireless device, even in the context of real-time constraints. Savings from 3%-14%, can be observed. We also see improvements in the time spent in low power mode, with improvements varying from 3%-16%.

3.4.4 EESI Results with Static Hints

In our final EESI evaluation results, again, all test scenarios include background computational tasks that do not require any device I/O. For communication-bound processes, we



(a) Power Benefits

(b) Low Power State Usage

Figure 15: Experimental Results with DWCS Scheduler.

use four different applications. The first is a synthetic application designed to mimic periodic network applications. For many application domains, such as multimedia or robotic software where local sensor data is manipulated and transmitted, there is periodic communication activity where each transmission is preceded by some computation. To generally analyze this application class, we execute a synthetic application which periodically computes and transmits data. Scenarios 1, 2, and 3 include two instances of this application with increasing device usage rates (relatively 1x, 5x, and 50x). We also use actual network application scenarios. These applications include a JPEG compression application which processes PPM images and transmits the compressed results. We also develop a scanner application that imitates the transmission of data resulting from label scanning, a scenario common in inventory or processing applications. Finally, we use an audio playback application, Dreamplayer, to stream WAV files over the network. Our 4th experimental scenario includes the JPEG and scanner applications, and scenario 5 utilizes the JPEG application along with streaming audio.

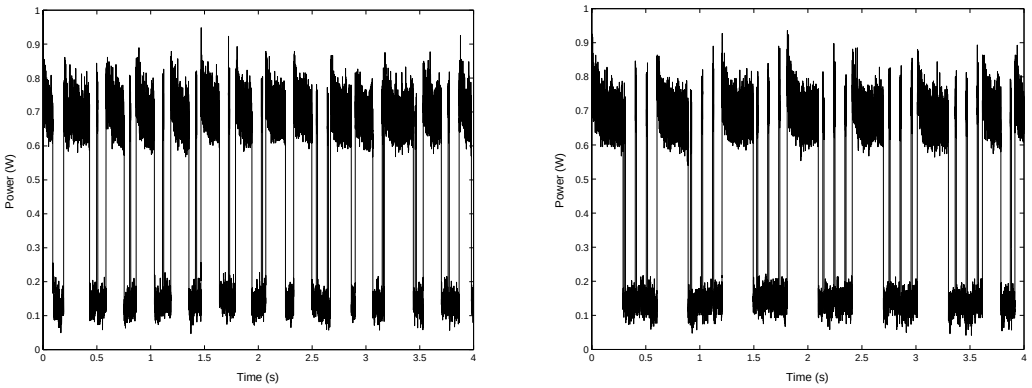
To test EESI on yet another I/O device, and again with the use of multiple devices, we include the use of flash storage. Scenario 6 consists of audio playback from a file stored on the flash device. Scenario 7 extends this mix by including JPEG compression and transmission, where the PPM image files are stored locally on flash and the resulting JPEGs are transmitted over the wireless link. These scenarios are summarized in Table 10.

To evaluate the EESI system, we perform power measurements of both 802.11 and flash I/O devices during the various execution scenarios. For scenarios (1)-(5), we measure the power consumed by the wireless card, and for the last two scenarios, we monitor the power

Table 10: EESI Experimental Applications with Source Annotation Hints.

Scenario	Applications Used
1	Periodic network usage applications (1x)
2	Periodic network usage applications (5x)
3	Periodic network usage applications (50x)
4	JPEG compression and data scanning
5	JPEG compression and audio streaming
6	Audio streaming from USB flash
7	Audio streaming and JPEG compression with data on USB flash

usage of the USB flash storage device. To illustrate the ability of EESI to burst accesses and provide extended idle periods, we provide the power signature of the network card during the execution of scenario three in Figure 16. The figure clearly shows that EESI is able to perform access bursting and simultaneously provide longer sleep times. The isolated spikes during sleep periods are due to normal client transmissions to the access point at 100ms intervals.

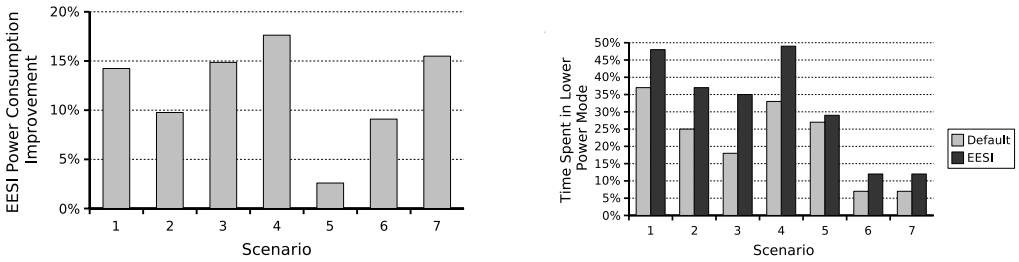


(a) Default Scheduler (b) EESI

Figure 16: Network (802.11) Power Signatures with EESI.

Figure 17 provides experimental results for various scenarios. We see significant savings, up to 18%, in average power. Since processes are allotted the same amount of service with EESI as they are in the default Linux system, this directly relates to energy savings. Similarly, benefits of up to 17% in time spent in low power mode are obtained. We observe that for scenario five the benefits are not as significant as the others. This is due to the fact that the audio streaming application acts as a data sink as opposed to a data source.

Since a read access to a network device is an asynchronous operation, local scheduling changes are not as impactful in modifying actual device activity periods. Overall, though, our experimental results illustrate that our EESI system can provide significant I/O energy benefits for an embedded platform.



(a) Power Benefits

(b) Low Power State Usage

Figure 17: Experimental Results with Source Annotation Hints.

3.5 Related Work

The need for power management in mobile systems has lead to a variety of proposed solutions and architectures. Many of them concentrate on the issue of managing I/O and peripheral devices. A substantial amount of the research for utilizing device idle periods is concerned with communication devices. The benefits of a separate low power channel to determine when to turn off these devices has been demonstrated [101]. Other work has considered modifications to the the 802.11 protocol at the client and base station to develop a collaborative approach to putting a wireless device to sleep [54]. An adaptive approach to managing sleep states based on the secondary effects of powering down wireless devices on higher layer protocols such as TCP has been shown to be beneficial as well [53]. As opposed to these efforts, the EESI system is designed to scale to multiple devices.

When considering multiple devices, the problem of putting devices to sleep becomes more complicated. Given a predetermined task schedule and device usage list, an algorithm to determine a schedule for device sleep/active states can be developed [106]. Given perfect knowledge of device usage information, an algorithm that schedules tasks so as to maximize device idle periods can be used [66]. Online usage information can also be used to directly change device states as well [67, 69]. Similarly, applications themselves can specify access

information to allow the system to effectively power down devices [68, 111]. At a lower level, the system can have more sophisticated policies than just timeout based mechanisms for device management, such as adaptive timeout [63] or stochastic models [70]. All of these approaches require additional information from workloads, or have OS layer policies directly manage devices. With EESI, we maintain existing power management independence between OS layer decisions (e.g. scheduling), and integrated device management policies.

An accounting-based approach to provide system lifetime guarantees in mobile devices can be used to extend battery life [120]. By making power a first class system resource, the approach strictly allocates power to processes, and can use various scheduling algorithms [119]. While providing lifetime guarantees, this approach does not provide any performance or user-level guarantees such as task deadlines or throughput. In [10], the author proposes energy-aware scheduling techniques using energy accounting based upon hardware performance counters [44, 47]. The EESI infrastructure allows for energy efficient scheduling, while maintaining the performance semantics of the existing quantum based or real-time scheduler. The approach leverages an "affinity" aware scheduling approach inspired by cache affinity aware scheduling for performance [103]. We have shown that by utilizing the device windows mechanism to encapsulate affinity, EESI can significantly extend idle and active periods, allowing for improved power consumption on end devices.

3.6 Summary

The proliferation of I/O devices into mobile platforms has created an immediate need for effective management of their power consumption characteristics. Device manufacturers have responded by integrating simple, but effective, timeout based management policies into devices in order to transparently provide energy savings while alleviating system software from directly managing device states. In this chapter we have highlighted the type of interactions that dictate the coordination of OS scheduling decisions to improve device power consumption. The address this need, we describe and present the EESI design based upon our device windows mechanism for enabling intelligent scheduling decisions. Experimental evaluations highlight improvements of up to 25% with an EESI enabled system.

CHAPTER IV

POWER EFFICIENT MOBILE WORKLOADS WITH COMPATPM

Technological advancements in the areas of computational and communication resources have led to the deployment of increasingly sophisticated cooperative mobile systems. Since many of the applications running across these systems utilize middleware solutions to instantiate necessary software components [99, 32], there is considerable flexibility in how to distribute the associated computations, to improve performance or QoS, to gain fault tolerance, or, the focus of this thesis, to reduce energy consumption. As a concrete example, cooperative multimedia applications can be found in autonomous mobile robots performing critical search and rescue missions. In this environment, robots use sensors like cameras, laser range scanners, GPS, microphones, and infrared cameras as inputs for navigation algorithms and situational awareness. The resulting multimedia data flows must be processed for use by specific subsystems, typically coordinated by a system manager in the middleware that controls the deployment and configuration of the associated software components. Control actions are driven by global metrics like end to end performance [76], energy usage, and/or application lifetime.

Interestingly, for modern robotic systems, the energy used by distributed multimedia tasks can substantially contribute to total power consumption. This is because the high performance components and devices on these systems create elevated platform-level power signatures. Since mobile robots have a limited energy supply, efficient energy management translates into prolonged lifetimes, longer deployments, and ultimately, more successful missions. Further, the computing platforms used in these systems enable energy efficiency with integrated components (e.g., processors and memory) that provide multiple power management states. Memory may support a doze or low power state that can be utilized during idle periods [33, 58]. For processors, during periods of inactivity, sleep states or ACPI supported C-states [41] can be entered to minimize power consumption [5, 22]. In

addition, multiple performance states in the form of discrete voltage/frequency operating points are provided for execution time power management. Each operating point represents a processor that is able to run applications with some specific speed and consumes a certain amount of energy over time, with slower processors typically consuming less power than faster ones.

As motivated earlier, power management of resources is typically performed at multiple layers, e.g., local platform hardware, local platform software via the operating system, and global management decisions by a system manager. Indeed, for certain components, control naturally belongs in one layer over another. For example, in order to obtain memory power savings during active periods, the time granularity of management is much finer than the operating system can attain. Therefore, the management of this resource must be performed in the platform itself, integrated into hardware via chipsets or memory controllers. On the other hand, the operating system can handle mechanisms like processor sleep modes whenever it schedules the idle thread on some processor core. Utilizing dynamic voltage and frequency scaling (DVFS) is interesting as its control can feasibly be assigned to any of the three layers. Since frequency scaling of processors directly affects the end performance of applications, in a distributed environment where global time constraints are not known locally at each node, the control of processor operating points naturally belongs in the system management component of the architecture.

In this chapter, we consider the additional gains possible when allowing the system middleware to specify a processor operating point at which to execute a multimedia workload deployed onto a robot. We begin by performing measurements on our Sitsang platform described in Chapter 3, and analyze the energy tradeoffs of frequency scaling on periodic multimedia tasks without the use of other power management mechanisms. There are two important conclusions from these results:

1. Energy savings attained with DVFS on the processor need not directly translate to platform-level savings.

2. Even when the lowest operating point can be used, from a platform-level perspective,

it may actually consume more energy than a higher frequency.

The above results can be attributed to the tradeoffs between the active and idle power signatures of mobile platforms, each of whose contributions to periodic energy vary as frequency is scaled. Or, in other words, platform idle power is not compatible with frequency scaling. This notion of compatibility can also be applied to underlying power management mechanisms below the system manager layer. In particular, there are those power management schemes whose achievable savings are not affected by frequency scaling assignments by the system manager and can be called "compatible" with DVFS, and others whose effects vary significantly depending on processor modes and are "incompatible". It is this idea that we utilize to address the counterintuitive trends provided in our initial results. Towards this end, we propose *CompatPM*, a coordination approach to power management for cooperative distributed multimedia workloads. CompatPM exports attributes of platforms, workloads, and underlying power management schemes to the middleware based system manager which then properly assigns performance points given the QoS constraint of the distributed system and application. As part of our contributions, we utilize our experimental evaluation to define this set of attributes and find that the attributes defined for power management schemes vary depending on our aforementioned compatibility classification.

4.1 Energy Efficient Deployment of Multimedia Tasks

4.1.1 Multimedia Workloads in Cooperative Distributed Systems

A basic assumption underlying this research is that it is possible to exploit the cooperative nature of distributed multimedia applications, subject to some set of joint operating constraints. Cooperation is well-recognized for applications like distributed gaming and surveillance/sensing, but it is also present in mobile robotics. An example in a search and rescue scenario is one in which robots have to preserve the energy of all team members in order to sustain a longer application lifetime. One way to achieve such longevity is to equalize loads across different team members and/or to remove load from an energy-poor team member, thereby sharing the team's combined energy and computing resources.

The multimedia workloads of robotic systems subject to runtime management may be

divided into two groups: (1) those responsible for providing situational awareness for human operators and (2) those in which robots consume and process media data. Workloads in (1) are similar to traditional multimedia workloads, since a human consumes the data. For example, the robot may use a microphone along with a camera to relay information from a victim to a far-away human operator. For this task, the robot would have to perform encoding and transmission of the input data. *The audio and image encoding and decoding algorithms we use in our experimental evaluation are representative of such situational awareness workloads.* Workloads in (2) use some combination of cameras, GPS, laser range scanners, sonar, bump sensors, and odometers for robot navigation. These sensor data flows must be processed by perceptual algorithms, either locally or on another robot in a cooperative team, to be useful for navigation. These flows may also have different requirements and desired guarantees than those consumed by human operators. As an example, in some of our related work [89], we considered a robot to robot data flow for blob finding analysis of images, where energy was reduced by offloading image processing. Such offloading relies on image encoding and decoding to minimize communication energy consumption. *The JPEG workloads used in our experiments are typical of multimedia data flows for distributed image processing.*

Abstractly, the multimedia workloads used in our research are represented as "application task chains" of computation and associated communication tasks. Tasks are units of functionality (e.g., JPEG encoding, edge detection) as well as units of allocation. That is, any task may be allocated anywhere in the mobile distributed system, where task chains are connected via communication tasks operating across a wireless network. Task chains are described by: (1) the desired rate of the chain (e.g., the obstacle avoidance chain should be executed at 4Hz),(2) the computational latency of each task in the chain (e.g., JPEG encoding takes 4 ms), and (3) the data-size communicated along the chain (e.g., the raw image is 4 MB). The goal of CompatPM based coordination is to increase power efficiency at nodes executing such multimedia workloads by effectively exploiting local power saving mechanisms.

4.1.2 Power Management Architecture

Multimedia workloads in distributed systems are particularly interesting to study since increases in execution time for each media "frame" incur zero performance degradation as long as some desired frame rate is met. Or, in other words, for multimedia workloads, there is the notion of slack that allows for flexibility in execution performance while maintaining application performance. In the context of DVFS, this means that multiple processor frequencies on a platform can achieve the same quality of service. Therefore, we can choose the processor frequency that gives the best energy performance.

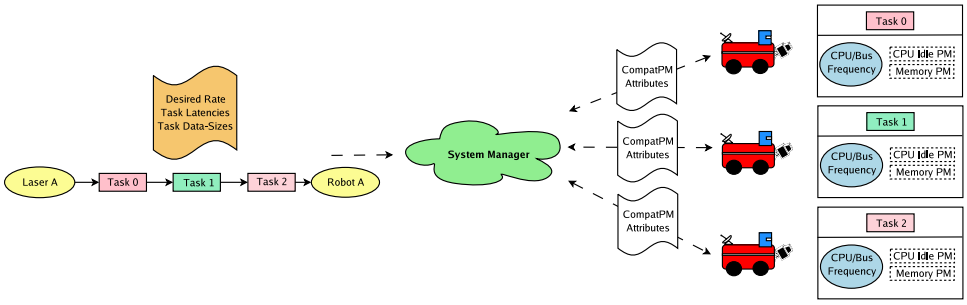


Figure 18: System Power Management with CompatPM.

Our power management architecture assumes that task chains are known to and manipulated by system managers integrated into middleware, as shown in Figure 18. Such managers are both necessary and useful. They are useful because they can exploit their global knowledge to perform better allocations and select appropriate operating points of tasks on participating platforms. They are necessary, because allocation and selection decisions are not just based on task workloads, but also on end to end application-level constraints, such as meeting some required maximum delay for gaining some perceptual insight from a set of raw sensor data readings. In past work, a common approach has been to compute the minimum processor frequencies needed along the chain to meet required end to end delay, thereby exploiting the slack available in distributed multimedia or real-time applications. Unfortunately, this can result in suboptimal system-level energy consumption. In addition, there is the issue of coordination required to ensure constructive interactions between the execution speed chosen and the savings achieved by the underlying power management mechanisms integrated into the mobile platform. The specific resources considered in our

analysis are memory management support integrated into platform hardware and the use of processor sleep states by the operating system during idle periods. Toward this end, we identify CompatPM attributes that must be utilized by the software management layer. The overall power management design is summarized in Figure 18.

4.1.3 Evaluation Infrastructure

Experimental results are again based on measurements obtained from representative embedded hardware, using Intel's Sitsang-400 evaluation platform designed around the PXA255 processor. The PXA255 supports multiple frequency states. The operating points not only vary CPU frequency, but also the frequency to the internal PXA bus, thereby affecting latency to memory and I/O devices. These points, along with the associated core voltages, are summarized in Table 11. For example, the maximum operating point consists of the core clocked at 400MHz, a 200MHz bus, and a 1.3 volt supply to the core logic. Our experimental workloads consist of benchmarks from the Mediabench suite that exemplify the two types of multimedia workloads for robotic systems discussed in Section 4.1.1. In particular, we use both the encoders and decoders for `adpcm`, `g721`, `gsm`, and `jpeg`. Benchmark execution time and power consumption are monitored for every available operating point.

Table 11: PXA255 Operating Points.

Core (MHz)	Bus (MHz)	Voltage (V_{core})
400	200	1.3V
400	100	1.3V
300	100	1.1V
300	50	1.1V
200	100	1.0V
200	50	1.0V
150	50	1.0V
100	50	1.0V

Since the Sitsang platform is designed to allow for power measurements of various subcomponents, in addition to system power, we also measure the power consumed by the CPU core (not including I/O). All power measurements included in this chapter are performed using a Tektronix TDS5104B oscilloscope, Tektronix TCP202 current probes, and Tektronix P6139A voltage probes. The oscilloscope's on-board advanced math routines automatically

generate power data from the measured current and voltage values.

4.2 Energy Analysis of Multimedia Applications

When managing the power consumption of computational platform components, the use of frequency reduction must be balanced with the resulting performance degradation. The metrics energy-delay (ED) or energy-delay^2(ED^2) are commonly used to quantify the tradeoff between two different operating frequencies for best effort execution systems. In periodic real-time systems, this tradeoff can be formulated in a more precise manner, since QoS is maintained as long as the execution deadline is met. Specifically, if the execution time of a task at frequency f_i is less than or equal to the period (deadline) T, then f_i is a plausible frequency for execution.

For the remainder of this section, we assume that all of the operating modes supported by the PXA255 are plausible for the execution of the assigned task. In particular, we assume the deadline/periodicity for the multimedia component of a deployed application chain to be the execution time at the lowest frequency. The result of this assumption is that the system manager has the greatest freedom possible in choosing operating points and that accordingly, we can understand clearly all associated tradeoffs in platform level energy usage, and better evaluate the benefits of CompatPM based coordination.

Table 12: Sitsang Platform Power Consumption Overview.

	Average Power Consumption (mW) per Operating Point (Core/Bus)							
State	400/200	400/100	300/100	300/50	200/100	200/50	150/50	100/50
Idle	1339	1281	1255	1223	1229	1213	1213	1213
Active	2351	2230	1985	1858	1798	1714	1666	1587

Table 13: Sitsang CPU Power Consumption Overview.

	Average Power Consumption (mW) per Operating Point (Core/Bus)							
State	400/200	400/100	300/100	300/50	200/100	200/50	150/50	100/50
Idle	126	90	63	44	44	33	33	32
Active	449	400	235	206	140	126	106	82

An overview of the Sitsang platform power characteristics when active and idle at various operating points is provided in Tables 12 and 13. As expected, the power consumption of both the CPU and system decrease with reduced operating points. Table 14 summarizes

the execution energy for each benchmark at 100MHz. In the entirety of this chapter, energy results for the CPU and system are normalized to these values.

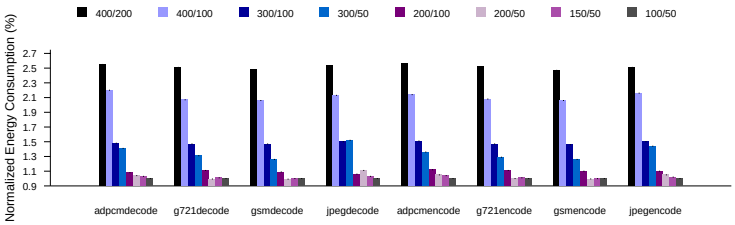
Table 14: Benchmark Execution Energies at 100MHz.

Benchmark	CPU (mJ)	System (mJ)
adpcmdecode	8.90	176.72
g721decode	338.10	6373.43
gsmdecode	87.78	1667.20
jpegdecode	8.30	171.26
adpcmencode	10.11	199.26
g721encode	355.55	6747.54
gsmencode	158.18	2819.88
jpegencode	24.65	477.24

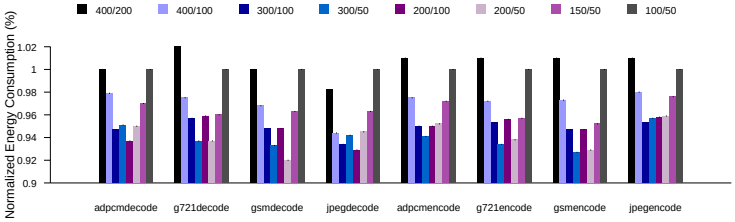
We proceed by first analyzing our representative platform from both a CPU and system energy perspective as frequency and voltage are varied. We then move on to how the existence of other power management mechanisms in the platform can affect the system-level energy trends, and thereby the frequency selection. In order to do so, we propose a novel classification scheme for these other mechanisms: those that are voltage/frequency incompatible (VFI) and those that are compatible with voltage/frequency scaling (VFC). We evaluate the effects of each of these schemes both independently and jointly by utilizing processor sleep state management (a VFI scheme) and memory power management (a VFC scheme).

Frequency Scaling and Energy Tradeoffs. Due to the periodic nature of multimedia workloads, the metric for energy consumption can be formalized using the notion of cycle energy. The *cycle energy*, E_{cycle}, consists of the sum of the execution energy E_{exec} and the idle time energy E_{idle} for a given task. We assume that the deadline of each task is its execution time at 100MHz in order to calculate cycle energy values.

Figure 19 provides the CPU and system-level results for cycle energy. We observe a clear trend in CPU cycle energy, where consumption generally decreases with lower performance points. These results support the intuition used in various real-time and multimedia schedulers that attempt to minimize frequency whenever it is possible to do so without violating performance constraints. Indeed, we observe CPU energy benefits of up to 60% when



(a) CPU Energy Results



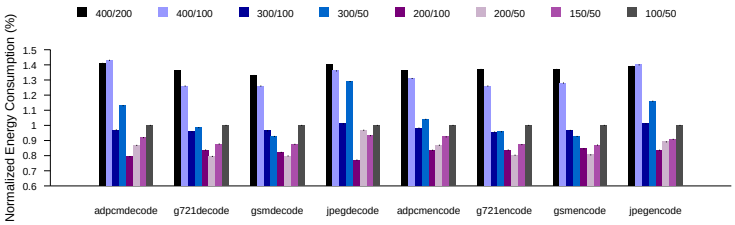
(b) System Energy Results

Figure 19: 'Cycle Energy' Consumption of Benchmarks.

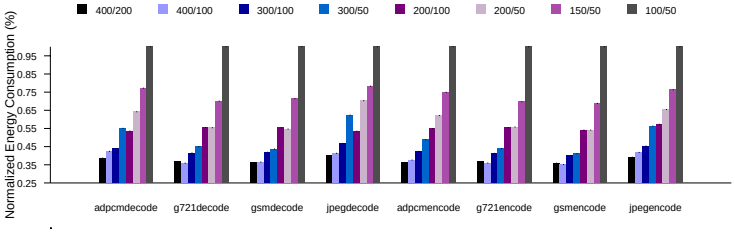
executing at the lowest frequency as opposed to the highest. Unfortunately, in practice, this intuition is only partially correct.

Figure 19(b) provides system-level 'cycle energy' data. These trends clearly challenge the common assumption made when performing DVFS. An important result in this figure is that the optimal operating point is never the lowest frequency. Indeed, the optimal frequency provides benefits of up to 8% compared to the lowest frequency. This result contradicts the assumptions made in prior work that reducing processor frequency is beneficial to system energy consumption, and it highlights the importance of carefully considering platform tradeoffs for frequency scaling rather than using a simple heuristic approach that attempts to minimize frequency. We also observe interesting trends with bus frequencies. For example, the system-level cycle energies of the `adpcmdecode` and `jpegdecode` benchmarks are reduced when executing at 300/100 versus 300/50, while the reverse is true for `g721decode`. These results illustrate that the relative benefits of operating points are workload dependent for actual systems and applications.

An interesting trend in platform design is the push towards highly efficient idle modes. In both mobile and high performance domains, manufacturers are providing increasing support



(a) CPU Energy Results



(b) System Energy Results

Figure 20: Execution Energy Consumption of Benchmarks.

towards minimizing the power consumption of components when the system is not actively executing applications [49]. We consider the effect of this trend on our results by assuming a best case scenario of zero idle power consumption, or, in other words, considering only the execution energy of our applications. In Figure 20(a), we observe the energy consumed by the CPU during execution. We see that execution energy generally decreases with core frequency (as expected), but this holds only until the 150MHz operating point. This behavior is due to the fact that while performance continues to degrade in proportion to frequency, the power benefits decrease as shown in Table 13. The reason is that the core voltage is the same for the lower frequencies, thereby removing any voltage scaling benefits[1]. The shift in trends from Figure 19(a) to Figure 20(a) is due to the decrease in the E_{idle} component of E_{cycle} when executing at higher frequencies. With regards to system execution energy trends, it can be seen that execution energy is minimized at the maximum core frequency (the particular bus frequency depending on the benchmark). These results support an approach that executes programs quickly in order to optimize idle periods.

[1]The PXA255 electrical specifications prescribe the operating points used, along with the 1V minimum requirement.

Of course, since our idealistic assumption of zero idle energy does not hold for current platforms, the optimal operating point will fall somewhere in between the minimum and the maximum. We therefore identify CompatPM attributes that allow for the dynamic assessment of tradeoffs between these operating points.

Interaction of DVFS and Incompatible Schemes. DVFS inhibits the use of VFI mechanisms, for instance, due to a reduction in the slack time t_{idle}. In such cases, energy savings achieved by DVFS must be traded off with the inability to obtain savings from any VFI power management mechanisms that can operate only during idle periods. As an example, we consider the use of utilizing CPU sleep state into our measurements. For the PXA255, the sleep mode consumes approximately 45μA, a relatively negligible amount of power. We utilize this value to project the cycle energy savings when combining the use of processor sleep state with DVFS.

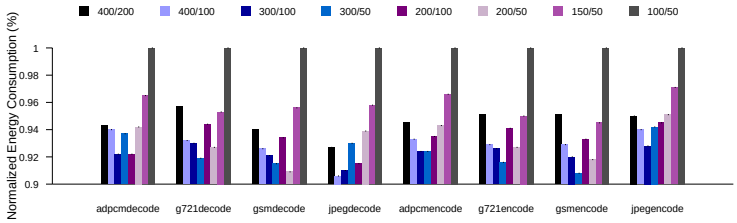


Figure 21: Energy Characteristics of DVFS with CPU Sleep Management.

Figure 21 provides the normalized system-level energy savings when using both DVFS and CPU sleep power management. Comparing it to Figure 19(b), we see that the benefits of higher operating points increase quite dramatically in relation to lower frequencies. This is consistent with the results in Figure 20(b), where we observe that the more efficiently idle power is managed, the more the optimal frequency trends shift towards high-end operating points. We conclude that when performing frequency scaling in conjunction with VFI power management mechanisms which provide for more efficient idle periods, the cycle energy trends move towards the execution energy trends of a platform. It is clear from these results that the middleware layer manager must be made aware of these types of underlying power managers.

Interaction of DVFS and Compatible Schemes. While certain types of power management mechanisms do not coexist well with DVFS, other schemes work synergistically with processor management. These VFC power management mechanisms can reduce the energy expended during execution periods by performing active management. Therefore, the E_{exec} portion of cycle energy decreases. Moreover, a VFC mechanism not only reduces energy during execution time, but can also be used during idle periods. Indeed, VFC power management mechanisms simply shift savings from idle to execution periods when frequency is scaled down creating independence from DVFS. Some inefficiencies exist in this displacement, however, due to the additional overheads of transition costs that are incurred during periods of activity. At the same time, since reducing execution speed increases the inter-access times to VFC managed components, these inefficiencies decrease as frequency is reduced. In our results, we include both execution and idle time benefits of VFC mechanisms.

Previous work has shown that memory power management can be utilized simultaneously with frequency scaling [27]. Therefore, we use this power management scheme as our VFC example. Our unique contribution here is that we determine the resulting trends using actual execution data, thereby providing to future researchers realistic data points for a representative platform. In particular, using XScale performance counters, we obtain execution traces of memory usage with our workloads at each operating point. These traces are then used to estimate the power savings that can be incurred with the two Samsung K4S561632D parts incorporated into the Sitsang platform.

In our calculations, we define memory idle power P_{idle} to be 165mW and the sleep power P_{sleep} as 40mW. These values constitute the power required by both memory chips. We also define t_{access}, the time it takes to retrieve the 32 bytes required on a cache miss, to be 100ns. Finally, the transition cost t_{trans} of utilizing the low power state, which is incurred on the transition back to active, is set to 10ns. These timing values are similar to those assumed in previous studies [27]. By combining our runtime power measurements, execution traces, and memory attributes, we can estimate the additional savings that could be achieved with memory power management during both execution and idle periods. These results are

presented in Figure 22.

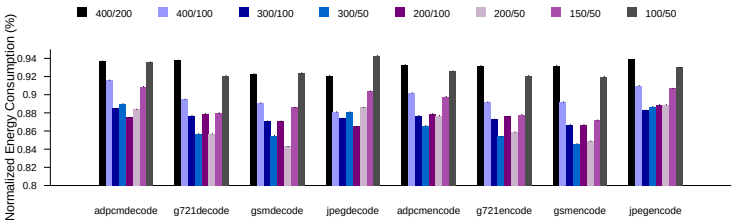


Figure 22: Energy Characteristics of DVFS with Memory Management.

We observe from the figure that memory power management can provide significant power savings on the system across all frequencies, compared to the default platform. Indeed, the benefits achieved are up to 16% in system energy when compared to executing at 100MHz with no memory management. The general trends between operating points, however, are similar to Figure 19(b). This result illustrates the relative independence of VFC power mechanisms. Since similar energy savings are achieved regardless of frequency, this class of power managers can exist transparently to the system management layer and frequency assignment decisions.

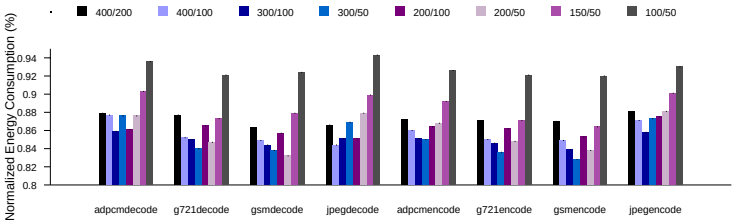


Figure 23: Energy Characteristics of Complete System Power Management.

Combining VFI and VFC Methods. We next obtain results when both of our power management examples are incorporated with DVFS. In Figure 23 we observe that the trends are more similar to the VFI results in Figure 21 than the VFC results, and that the optimal operating point shifts towards higher frequencies. This tells us that the trade-off trends between operating points when using DVFS continue to be influenced by VFI power management mechanisms. Moreover, even when coupled with VFI mechanisms, the independence of VFC managers remains intact.

4.3 Evaluation of CompatPM

The previous section presented an in depth study of the system-level power tradeoffs when using different operating points for a processor, as well as the interactions between utilizing this power management support with other underlying mechanisms. The goal of CompatPM is to take these types of tradeoffs and interactions into account in order to assign efficient operating points at runtime. It is clear that utilizing a profile-based approach where the manager is provided all of these tradeoffs explicitly does not scale. Therefore, we begin by defining a limited set of attributes that should be exported to the system manager as per Figure 18, and then determine the effectiveness of decisions based upon these inputs.

4.3.1 CompatPM Attributes

Platform Hardware Characteristics In terms of platform hardware, the system manager must be made aware of the inherent power tradeoffs of the system independent of any other power management. In our evaluation, we use a simple approach wherein average active/idle power values per operating mode for the platform are specified in the CompatPM attributes. For improved accuracy, a more sophisticated specification could be used. An example is a parameterized power model that is a function of workload characteristics such as memory access rate.

Platform Power Management Characteristics Our experimental results have highlighted the need for the system manager to understand the underlying power management methods supported by a robot's computational platform. Moreover, the results have shown the benefits of our classification scheme for these power managers into VFI and VFC mechanisms, as the CompatPM based management can essentially ignore the existence of the latter type of schemes. Since savings from VFI mechanisms can only be obtained during idle periods, in its CompatPM attributes list, a node should specify the average power savings achievable when idle for each VFI mechanism it supports.

Platform Workload Characteristics Two workload specific items must be exported to the management layer. First, the latency at the lowest operating point of a platform should

be specified. Second, in our measurements, we observed that given a processor frequency, the particular bus frequency that is most energy efficient is workload specific. Therefore, we require workload specific memory access behavior in order to differentiate what bus frequency to utilize. For our system, the memory access rate (MAR), or the number of memory accesses per instruction, is provided for each benchmark. Since memory access behavior is independent of frequency, this attribute does not need to be specified on a per operating point basis.

4.3.2 Results

Given the four CompatPM attributes: (1) active/idle hardware power characteristics, (2) average power savings for VFI schemes, (3) workload specific execution time at the lowest frequency, and (4) memory access rate information for applications, the management layer can attempt to determine an operating point for a workload that will maximize system-level power efficiency to include the possible benefits of underlying power management schemes on a platform. This decision process is driven by estimating the possible energy savings compared to running at the lowest operating point which meets latency requirements, and then choosing the mode that maximizes savings. In order to perform this estimation, the manager must determine what fraction of the period will be spent actively executing workloads, and what portion will be idle.

We propose to use a model driven approach to estimate the active time for execution in various operating modes using limited information, e.g. given the execution time, t_{min} at the lowest performance state provided by a platform in its CompatPM attributes. Instead of benchmarking each workload at every possible frequency, for scalability reasons, we hope to use a small number of features to predict the performance at different operating points. In our evaluation, we analyzed different learning algorithms [115] and different feature sets for predicting the workload latencies at different operating points. Using data from workloads at various frequencies, we found a simple linear regression model to be the best predictor. This model can be extended to take into account bus frequency and MAR information as well, but for our workloads and platform we found the simpler model to be sufficiently

accurate for our needs. Our final model is provided in Equation 10 as a function of t_{min} and the processor core frequency f_{core} under consideration.

$$t_x = 99.88 * \frac{t_{min}}{f_{core}} + 0.01 \tag{10}$$

Table 15: Workload MAR and Bus Frequency Correlation.

Benchmark	MAR	Optimal Frequency (MHz)	Other Frequencies (MHz)
adpcmdecode	0.0235	100	50
g721decode	0.0010	50	100
gsmdecode	0.0022	50	100
jpegdecode	0.0315	100	200
adpcmencode	0.0111	50	100
g721encode	0.0008	50	100
gsmencode	0.0008	50	100
jpegencode	0.0150	100	50

Once the manager has determined the division of a period into its active and idle times, it can use the power related attributes to estimate power savings and thereby the optimal frequency point. As a last step of the decision process, we propose a heuristic that allows the system to determine an appropriate bus frequency given a core frequency. Table 15 provides the MAR values of each of our workloads, along with the optimal bus frequency, and any other possible bus frequencies for the core frequency of the measured optimal operating point. We see that, generally, workloads with relatively high MAR (greater than 1%) benefit from a higher bus frequency, and vice versa. This behavior makes sense since for workloads with high memory access rates the increased bus frequency provides performance improvements that can improve power efficiency, whereas the increased power of a higher bus frequency is not worthwhile for workloads with low memory usage. Therefore, we use this correlation between MAR and efficient bus frequencies to decide the bus operating point to utilize for an application.

Given the described decision process based upon CompatPM attributes, we evaluate the resulting chosen operating point for a workload. The mode is compared to the optimal found by doing a full system analysis as in Section 4.2, and the mode that would be chosen according to traditional constraints, i.e. the lowest possible operating point which meets

performance requirements. Figure 24 presents these results.

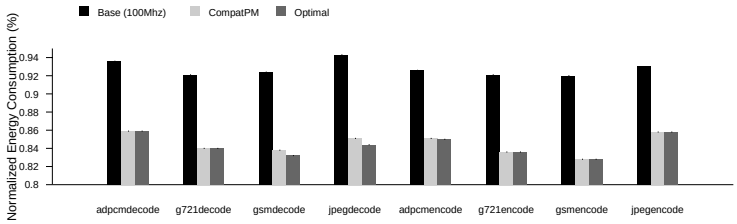


Figure 24: Comparison of CompatPM vs. Optimal and Minimum Frequency.

We can see clearly from the results that the CompatPM enabled system is often able to accurately determine the optimal operating point for each workload. Even when the decision is imperfect due to inaccuracies introduced by limited information and modeling, the power efficiency of the CompatPM decision is very close to the optimal. These results show that by utilizing our in depth measurements and analysis, we have successfully isolated a set of CompatPM attributes that can be easily provided to the management layer of a distributed system to create an effective, scalable, and coordinated power management approach.

4.4 Related Work

Currently, energy used for processing and communication in mobile robots is small compared to the energy expended on physical movement. However, Mei et al. [74], in a study of the Pioneer DX-3 mobile robot, found motion to be only 12.1%–44.6% of the energy used depending on speed, and they found the embedded computer accounted for 33.3% –65.3% of the energy. Because of this the authors claim frequency scaling is an important mechanism for energy savings in robot systems. Energy for computation will increase as robots become more autonomous and use more powerful computer platforms such as multicore systems, increasing the importance of management towards sustaining missions of multi-robot teams.

Recent processor architectures like the Intel XScale support dynamic frequency and voltage scaling (DVFS) to meet the needs of mobile systems. Since the dynamic power consumption of a CPU is proportional to the product of frequency and voltage squared [61], DVFS can be effective in reducing power consumption during program execution [110] while meeting performance requirements [73]. Adhering to application-level requirements such as task

deadlines while performing frequency scaling and necessary adjustments to process schedules can also be performed [107, 107]. A design framework for exploring power/performance trade-offs when developing hard real-time systems has been proposed [17], as well as an offline scheduler coupled with online slack reclaiming for enhancing the benefits of DVFS [6]. Other past research exploits the ability to obtain energy savings by performing application-level adaptations [91] for multimedia applications. Processor energy savings can also be achieved with a compiler/OS collaborative system for frequency scaling [1]. Other research aggressively pursues reduced frequencies within the constraints of application deadlines using memory access information [102].

In all of the DVFS approaches reviewed above, the effect of frequency scaling on the CPU energy signature is considered independently of the rest of the system. Fan *et al.* [27] begin to move away from this assumption by investigating the synergy between DVFS and power-aware memory systems. Miyoshi *et al.* [77] question the underlying assumption of DVFS, that frequency should be reduced whenever possible, by isolating poor performance points created by efficient idle modes. Similarly, the optimality of lower frequency points when considering system sleep modes has also been studied [42], as well as the tradeoff of overheads such as processor leakage power and extended resource standby times [45, 46]. As part of this research, we have experimentally evaluated similar tradeoffs in the context of multimedia workloads.

4.5 Summary

Mobile middleware solutions provide functionality to deploy and manage distributed workloads across cooperative systems [99]. As part of the resource management capabilities of these layers, they are aware of the end-to-end quality of service constraints of application chains that are distributed amongst participating nodes, for example robots in a search and rescue scenario. This makes the system manager integrated into middleware well qualified to balance the performance of application components achieved on nodes with energy costs by controlling underlying processor frequencies and voltages. As the experimental results in this chapter illustrate, it is critical to coordinate such decisions not only using

real-time performance constraints, but also the characteristics of underlying platforms and integrated power management solutions therein. Our results illustrate how utilizing CompatPM to perform this coordination can achieve energy benefits of nearly 10% compared to previously used heuristics of choosing minimal frequencies within application deadlines.

CHAPTER V

HETEROGENEITY AWARE DATACENTER POWER MANAGEMENT

Power management has become a critical component of modern computing systems, pervading both mobile and enterprise environments. As highlighted in Chapter 2, limitations in battery capacities and demands for longer device lifetimes have motivated research for mobile and embedded systems [31, 99, 118, 119]. In datacenters, power consumption has become a significant issue, stimulating a variety of research for server systems [11]. Increased performance requirements in datacenters have resulted in elevated densities enabled via consolidation and reduced server form factors. This has in turn created challenges in provisioning the necessary power and cooling capacities. For example, current datacenters allocate nearly 60 Amps per rack, a limit that is likely to become prohibitive for future high density rack configurations such as blade servers, even if the accompanying cooling issues can be solved [97]. In addition, a 30,000 square feet datacenter with a power consumption of 10MW requires a cooling system which costs $2-$5 million [79]. In such a system, the cost of running the air conditioning equipment alone can reach $4-$8 million a year [97]. Coupled with the elevated electricity costs from increasingly high performance servers, these effects can substantially affect the operating costs of a datacenter. These trends in power/cooling delivery and cost highlight the need for support in datacenters for power and thermal management.

Previous work on server management has focused on managing heat during thermal events [79] or utilizing platform power management support, such as processor frequency scaling, for power budgeting [29, 60, 97]. In this chapter, we consider the problem of managing datacenters from a different perspective. We consider how to allocate workloads to heterogeneous platforms intelligently to (i) improve datacenter power-efficiency while preserving/satisfying workload performance requirements, and (ii) meet datacenter-level power

budgets with minimal impact on workload performance. Typically, datacenters statically allocate sets of platforms to applications based upon peak load characteristics to maintain isolation and to provide performance guarantees. With the continuing growth in capabilities of virtualization solutions (e.g., Xen [9] and VMWare [105]), the necessity of such offline provisioning is removed. Indeed, by allowing for flexible and dynamic migration of workloads across physical resources [19], the use of virtualization in future datacenters enables a new avenue of management and optimization. Our approach begins to leverage some of these capabilities to enhance power efficiency, by taking advantage of the ability to assign virtualized applications to varying sets of underlying hardware platforms, based upon performance needs as well as power consumption constraints.

Throughout their lifetimes, datacenters continually upgrade servers due to failures, capacity increases, and migrations to new form factors [39]. Over time, this leads to datacenters comprised of a range of heterogeneous platforms with different technologies, power, performance and thermal characteristics, and power management capabilities. When assigning platforms to application workloads in these heterogeneous environments, power efficiency can vary significantly based on the particular allocation. For example, by assigning a memory bound workload to a platform that performs dynamic voltage and frequency scaling (DVFS), run-time power consumption can be reduced with minimal impact to performance [61]. To obtain this power-friendly behavior in datacenters, we propose the use of our heterogeneity-aware load management (HALM) architecture in this chapter.

Allocating power and cooling is another significant challenge in the modern datacenter. In addition to transient power delivery and cooling issues, the need for power budgeting arises due to provisioning of these resources. Traditionally, power and cooling have been allocated based on nameplate rating of the system power supply or its maximum output power. However, a fully utilized server with typical configuration will see its electrical load between 60% - 75% of the name plate rating on most enterprise workloads. Therefore, providing power and cooling capacity based on these worst case assumptions results in either over allocation of power and cooling capacity or underutilization of server rack space leading to increased capital costs and underutilized datacenters. Allocating power and cooling

capacity based on the average workload behavior within a server and across a datacenter allows significantly increased densities but requires dynamic protection mechanisms that can limit power of servers when demand temporarily exceeds available capacity. These mechanisms have been recently proposed in the literature and explored in the industry [28]. While very effective in limiting power and protecting the infrastructure, they may result in nontrivial degradation of peak performance especially when the power constraint is too prohibitive. Therefore, in this chapter we also show how HALM can lessen the impact of datacenter power budgeting and increase its overall performance.

Intelligent mapping of applications to underlying platforms is dependent upon the availability of relevant information about workloads and hardware resources. As part of HALM, we extend the use of *workload* and *platform descriptors* for this purpose, which are then used by a *predictor* component that estimates the achievable performance and power savings across the different platforms in the datacenter. These predictions are finally used by an *allocation layer* to map workloads to a specific type of platform. This overall infrastructure is evaluated using datacenter configurations consisting of variations upon four distinct platforms. In summary, the contributions that enable coordinated management with our HALM system include:

1. A platform heterogeneity-aware power management infrastructure that improves datacenter power efficiency under workload performance constraints and limited datacenter power budgets;

2. An allocation infrastructure that uses workload and platform descriptors to perform mappings of hardware to virtualized workloads; and

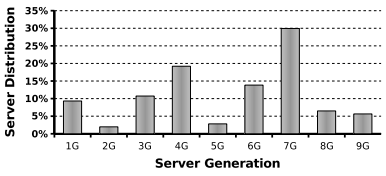
3. An intelligent load shedding policy to dynamically meet transient changes in power consumption limits.

Evaluations of our system performed on state-of-the art platforms, including Intel® Core™microarchitecture based hardware, demonstrate the benefits of exploiting platform heterogeneity for power management.

5.1 Motivation

5.1.1 Datacenter Composition and Exploiting Heterogeneity

Datacenter deployments are inherently heterogeneous. Upgrade cycles and replacement of failed components and systems contribute to this heterogeneity. In addition, new processor and memory architectures appear every few years, and reliability requirements are becoming ever more stringent. The effect of these trends is reflected by a recent survey of datacenter managers that found that 90% of the facilities are expected to upgrade their compute and storage infrastructure in the next two years. Figure 25(a) shows a distribution of different systems in a representative enterprise datacenter. As the figure shows, the datacenter contains nine different generations of systems that have either (1) different processor architectures, cores and frequencies, (2) varying memory capacity and interconnect speeds, or (3) different I/O capabilities. While all systems support the same software stack, they have very different and often asymmetric performance and power characteristics.



(a) Datacenter Composition

	System A		System B	
	W1	W2	W1	W2
CPU Power	90W	40W	90W	20W
System Power	160W	120W	160W	120W
PSU Efficiency	86%	70%	87%	80%
Total Power	**291W**	**229W**	**287W**	**175W**

(b) Heterogeneity Management Example

Figure 25: Datacenter Heterogeneity and Management Benefits.

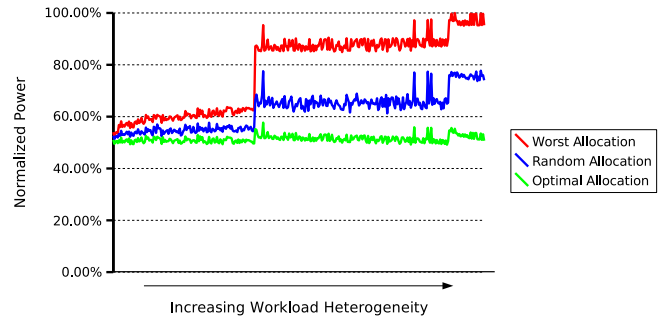
Traditionally, the non-uniformity of systems in a datacenter has been characterized by different levels of performance and power consumption. However, recently, another dimension has been added to this heterogeneity, because server platforms are beginning to offer rich thermal and power management capabilities. Processors support DVFS and aggressive sleep states to conserve CPU power. New memory power management implementations allow different DRAM devices to go to lower power states when inactive, and enable bandwidth throttling for thermal protection. Server power supplies exhibit different conversion efficiencies under different loads, directly impacting the overall power efficiency of the system. Finally, since power efficiency has become an important thrust in enterprise systems,

we expect component and platform vendors to continue introducing new power and thermal management capabilities into their products, including I/O and system buses, chipsets, and network and disk interfaces, making future platforms even more heterogeneous.

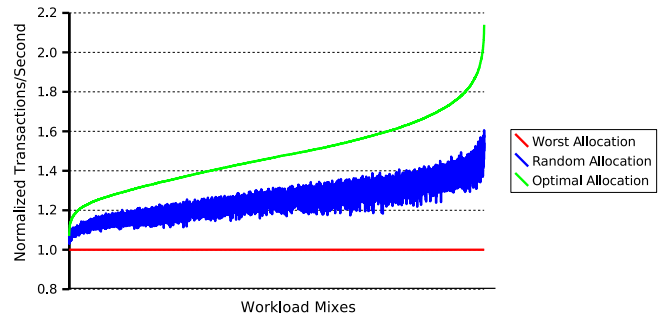
Previous work has proposed different approaches for energy-efficient workload allocation in clusters, but none have accounted for system level power management and thermal characteristics. Therefore, the workload allocations proposed by previous approaches will yield less than ideal results since they are completely unaware of power and thermal management effects on system performance and power consumption. To illustrate this phenomenon, we experimentally compare two dual processor systems, A and B, running two different workloads, as shown in Table 25(b). The differences between the two systems are in the power supply unit (PSU) and processor power management capabilities. System A has a less efficient power supply at light load and has processors with limited power management support. System B, on the other hand, has a high efficiency power supply across all loads and processors that support a rich set of power management capabilities. We measure power consumption on these platforms using two different synthetic workloads: one with full utilization ($W1$) and one with a very low level of utilization ($W2$) on both systems. $W1$ consumes about the same amount of power on both platforms. However, allocating the low-utilization $W2$ to system A leads to very power inefficient execution. Since A does not support power management and has low PSU efficiency at light load, its total system power is more than 50W higher than that of system B. Thus, while both systems meet the performance demand of both workloads, heterogeneity-aware resource allocation can decrease total power by more than 10%, translating into millions of dollars in savings for large datacenters. As this example shows, a full knowledge of system power and supported power management features is required to efficiently allocate workloads. Our HALM system is designed to provide such functionality.

5.1.2 Benefits of Heterogeneity-Aware Management

To further motivate the need and benefits of heterogeneity-aware management in datacenters, we perform two opportunity studies. The first study considers the possible benefits



(a) Allocation Opportunity



(b) Budgeting Opportunity

Figure 26: Opportunity Analysis of Heterogeneity-aware Management.

of allocating workloads by matching system capabilities and workload execution characteristics to reduce a datacenter's power profile while also meeting workload performance demands. We analyze an example of running a set of workloads in a datacenter configuration with four unique types of platforms described later in the chapter, each with different power/performance characteristics. The set of workloads includes ten computational benchmarks (`swim`, `bzip2`, `mesa`, `gcc`, `mcf`, `art`, `applu`, `vortex`, `sixtrack`, and `lucas` from SPEC CPU2000) and one transaction-oriented workload (SPECjbb2005). We generate all subsets of four from these eleven benchmarks and compare three allocation policies for each of the subsets in Figure 26(a). The 'worst case' allocation distributes the benchmarks across platforms to maximize power consumption, 'random' allocates workloads to platforms randomly, and 'optimal' distributes the workloads to minimize power consumption. For each workload, we allocate as many systems of a given type as necessary to meet workload throughput requirements. Subsets that have benchmarks with more homogeneous behavior, *i.e.* similar processor and memory usage behavior, appear on the left side of the graph,

while subsets with more heterogeneous benchmarks appear on the right. As can be seen from the figure, subsets of workloads with more asymmetric behavior can substantially benefit from heterogeneity-aware resource allocation. Averaging across all subsets, the optimal policy can reduce total power by 18% when compared to random allocation and by 34% over worst-case allocation, without compromising workload performance.

The second opportunity study considers how the aggregate throughput of a set of workloads varies within a given power budget based upon allocations. In particular, we assume that we have one of each of our four unique platforms and again generate subsets of four workloads from a set of SPEC CPU2000 benchmarks. For each subset, we calculate the minimum, average, and best case throughput across all permutations of possible allocations of the four workloads onto the four platforms. Figure 26(b) provides the results, where each scenario is normalized by the minimum throughput value to provide fair comparisons. We find that on average, the best case allocation provides a 23% improvement in performance over the random allocation, and a 48% improvement compared to the worst-case. These results highlight the relationship between allocation decisions and performance when a power budget must be imposed.

Summarizing, HALM addresses the power benefits of heterogeneity-aware allocation for two cases: (1) when there is no power budget and (2) when such a budget must be imposed temporarily due to power delivery or cooling constraints or as part of a power provisioning strategy [28].

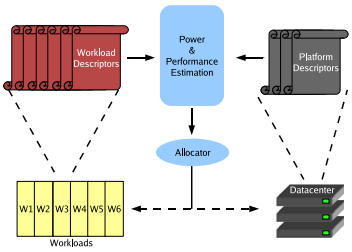
5.2 Scalable Enterprise and Datacenter Management

Our previous discussions have motivated the need to augment the behavior of datacenters to improve manageability, by leveraging the heterogeneity of platform capabilities. HALM extends this support with its heterogeneity-aware workload allocation infrastructure that utilizes the flexibility of rapidly developing virtualization technologies. Virtualization attempts to provide capabilities and abstractions that significantly impact the landscape of enterprise management. For example, there is active work to ensure performance isolation benefits, where it will be possible to run multiple virtual machines (VMs) within a

given physical platform without interference among applications [51]. Currently VMs can coexist on a platform with negligible performance interference as long as resources are not overcommitted. Approaches that allow for resource reservation and dynamic resource ballooning, though, can aid in providing isolation even when systems are overprovisioned. Secondly, by encapsulating application state within well defined virtual machines, migration of workloads among resources can be performed easily and efficiently. A more powerful contribution of virtualization, however, is the ability to combine multiple resources across physical boundaries to create virtual platforms for applications, providing a *scalable enterprise* environment. HALM assumes this flexible and powerful virtualization support.

The usage pattern of datacenters is becoming increasingly service-oriented, where applications and workloads may be submitted dynamically by subscribers/clients. When managing these types of applications, certain management actions, such as allocation decisions, happen at a coarse granularity, with finer adjustments being made at runtime to address transient issues such as reduced power budgets due to power delivery or cooling issues. One can imagine how such a datacenter might be managed with the typically used assignment approaches. At some infrequent interval the pool of applications and service level agreements (SLAs) which specify their required performance, in metrics such as throughput or response time, are compiled. Applications are then assigned to platforms using a simple load balancing scheme based upon utilization or queue lengths, possibly even accounting for differences in the performance of the systems [121], so that SLAs are met. When load must be reduced to address power budgeting requirements, load might be shed from workloads in a similarly random or round robin fashion. This approach clearly leaves room for improvement, since it does not consider power or platform differences in any way. HALM addresses this weakness by performing heterogeneity aware allocations as well as intelligent load shedding.

The HALM architecture can be organized into three major components: (1) platform/workload descriptors, (2) a power/performance predictor, and (3) an allocator, as shown in Figure 27(a). We use platform and workload descriptors to provide our workload allocator with the differences amongst workloads and platforms. These descriptor inputs are



(a) System Components

```
Component: Processor
   Architecture:    Netburst
   No_Cores:        4
   Frequency:       3.2 GHz
   p-state_support: Yes

Component: Memory
   Cache_size:  2MB
   Memory_type: DDR2
   Memory_size: 8GB
.
.
.
```

```
Attribute: MPI
   val1: <Memory.Cache_size=2MB>
   val2: <Memory.Cache_size=4MB>

Attribute: CPI_CORE
   val1: <Processor.Architecture=
          NetBurst>
   val1: <Processor.Architecture=
          Core Microarchitecture>
.
.
.
```

(b) Platform and Workload Descriptor Examples

Figure 27: HALM Architecture.

utilized by the predictor to determine: (1) the relative performance of workloads on different types of platforms, and (2) the power savings achievable from platform power management mechanisms. Coupled with coarse platform power consumption information (obtained via online power monitoring) (3) the allocator, performs the assignments of workloads to the available resources.

The purpose of platform descriptors is to convey information regarding the hardware and power management capabilities of a machine. A platform descriptor is made up of multiple modules, representing different system components, as shown in Figure 27(b). Each module specifies the type of component to which it refers, such as processor, memory subsystem, or power supply. Within each of these modules, then, various component parameters are defined. For example, a module describing the processor component may have attributes like its microarchitecture family, frequency, and available management support. Workload descriptors are also structured in modules, headed with attribute declarations. Within each module, a list of values for that attribute is provided. As workload attributes often vary with the platforms on which it executes, our descriptor design allows multiple attribute definitions, where each definition is predicated with component parameter values that correlate back to platform descriptors. Figure 27(b) illustrates the structure of the resulting workload descriptor. We further explain the meaning of the MPI (memory accesses per instruction) and CPI_{CORE} (core cycles per instruction) attributes in subsequent sections.

Platform descriptor information can be provided in a variety of ways. It can be made readily available using platform support such as ACPI [41], and possibly also with some

administrative input. To provide the required workload descriptors, we profile workloads on a minimal set of *orthogonal platforms*, with mutually exclusive component types. We then use an analytical prediction approach to project workload characteristics on all available platforms. As we discuss in Section 5.4, this approach provides accurate predictions that scale with increased amounts of heterogeneity.

5.3 Methodology

5.3.1 Platform Hardware

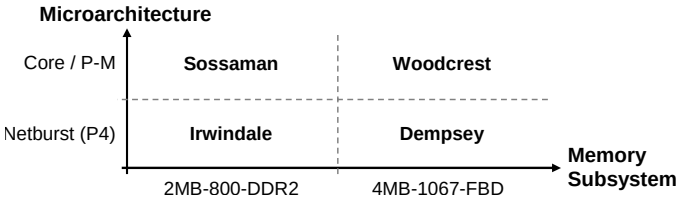
Our hardware setup consists of four types of rack mounted server platforms summarized in Table 28(a), where LLC denotes last-level cache size. All four types of platforms contain standard components and typical configurations that entered production cycles. In our experiments Linux was installed on all systems for measurement of various attributes (e.g. CPI, MPI, etc) as well as performance. We validated that the performance results matched those with Xen using a subset of workloads and platforms, but performed the majority of our experiments in a non-virtualized environment to have better access to performance counters used to measure other workload attributes.

The platform names are based on their processor code name in this chapter. All four platforms are dual-processor systems. Woodcrest, Sossaman, and Dempsey are CMP dual-core processors, and Irwindale is a 2-way SMT processor supporting Hyper-Threading Technology. All platforms have 8GB of memory. Woodcrest and Dempsey support Fully Buffered DIMM (FBD) memory with a 533MHz DDR2 bus, while Sossaman and Irwindale support unregistered DDR2 400MHz memory. Woodcrest and Dempsey have dual FSB architectures with two branches to memory and two channels per branch.

All four types of systems are heterogeneous in a sense that each has a unique combination of processor architecture and memory subsystem. If we assume that Intel Core microarchitecture/Pentium® M and NetBurst constitute two types of processors and LLC-4MB/FSB-1066/FBD-533 and LLC-2MB/FSB-800/DDR2-400 constitute two types of memory, all four platforms can be mapped as having unique processor/memory architecture combinations. Note that all four platforms also have vastly different power and performance

	Woodcrest	Sossaman	Dempsey	Irwindale
Processor	3GHz/4MB LLC Core architecture	2GHz/2MB LLC Pentium® M	3.7GHz/4MB LLC NetBurst/P4	3.8GHz/2MB LLC NetBurst/P4
FSB	1067 MHz Dual FSB	800 MHz	1067 MHz Dual FSB	800 MHz
Chipset	Blackford	Lindenhurst	Blackford	Lindenhurst
Memory	DDR2-533 FBD	DDR2-400	DDR2-533 FBD	DDR2-400

(a) Platform Characteristics



(b) Heterogeneity Quadrants

Figure 28: Heterogeneous Experimental Platforms.

characteristics. For example, the Intel Core microarchitecture is superior to NetBurst both in terms of performance and power efficiency. FBD based memory, on the other hand, provides higher throughput in our systems at the expense of elevated power consumption due to increased DDR2 bus speed and the power requirements of the Advanced Memory Buffer (AMB) on the buffered DIMMs. The four platforms occupy separate quadrants of a heterogeneity space with dimensions of microarchitecture heterogeneity and memory subsystem heterogeneity, as shown in Figure 28(b). We refer to this initial level of heterogeneity as *"across-platform heterogeneity"*. However, in addition to this, all these server platforms also support chip-level DVFS. This leads to a second degree of heterogeneity, where one type of platform can have instances in a datacenter that are configured to operate at different frequencies. We refer to this as *"within-platform heterogeneity"*. As process variations increasingly result in the *binning* of produced chips into different operating points, this within-platform heterogeneity becomes an inherent property of the general datacenter landscape. Finally, many of these platforms may incorporate some processor dynamic power management (DPM) techniques that adaptively alter platform behavior at runtime. This creates a third source of heterogeneity, *"DPM-capability heterogeneity"*, where platforms with built-in DPM hooks exhibit different power/performance characteristics from the ones with no DPM capabilities. In Table 16, we show how these three levels of heterogeneity quickly escalate the number of distinct platform configurations in a datacenter scenario.

	Across-Platforms				Within-Platform	DPM-Capability		Heterogeneous Configuration
	Microarchitecture		Memory		Frequency [GHz]	Enabled	Disabled	
	Core	Netburst	FBDIMM	DDR-2				
Woodcrest	X		X		3.0	x		1
							x	2
					2.6	x		3
							x	4
					2.3	x		5
							x	6
					2.0	x		7
							x	8
Sossaman	X			X	2.0	x		9
							x	10
					1.6	x		11
							x	12
					1.3	x		13
							x	14
					1.0	x		15
							x	16
Dempsey		X	X		3.7	x		17
							x	18
					3.2	x		19
							x	20
Irwindale		X		X	3.8	x		21
							x	22
					3.2	x		23
							x	24
					2.8	x		25
							x	26

Table 16: Levels of Heterogeneity in Experimental Platforms.

All experimental power measurements are performed using the Extech 380801 power analyzer. The power is measured at the wall and represents total AC power consumption of the entire system. The power numbers presented in this chapter are obtained by averaging the instantaneous system power consumption over the entire run of each workload. Our assumption is that infrastructure support for monitoring power consumption will be utilized to obtain this type of workload specific power characteristics online, instead of parameterized models. For example, all power supplies, which adhere to the latest power supply monitoring interface (PSMI) specification, support out-of-band current/voltage sampling allowing for per platform A/C power monitoring reflected by our actual power measurements.

5.3.2 Application Model

When power managing computing environments, improvements can be attained with a variety of approaches. In this work, we consider two scenarios. The first assumes a lack of budgeting constraints, concentrating on a workload allocation that reduces power consumption while maintaining baseline application performance. In other words, we maximize the performance per watt, while holding performance constant. The second addresses power budgeting by performing load shedding to reduce power consumption while minimizing performance impact to workloads. We consider application performance in terms of throughput, or the rate at which transaction operations are performed. Therefore, it is not the

execution time of each transaction that defines performance, but the rate at which multiple transactions can be sustained. This type of model is representative of applications such as transaction based web services or payroll systems.

The goal of HALM is to evaluate the power-efficiency tradeoffs of assigning a workload to a variety of platforms. Since the performance capabilities of each platform are different, the execution time to perform a single operable unit, or atomic transaction, varies across them. As previously mentioned, virtualization technologies can help to extend the physical resources dedicated to applications when necessary to maintain performance by increasing the number of platforms used to execute transactions. In particular, transactions can be distributed amongst nodes until the desired throughput is reached. This approach is summarized in Figure 29.

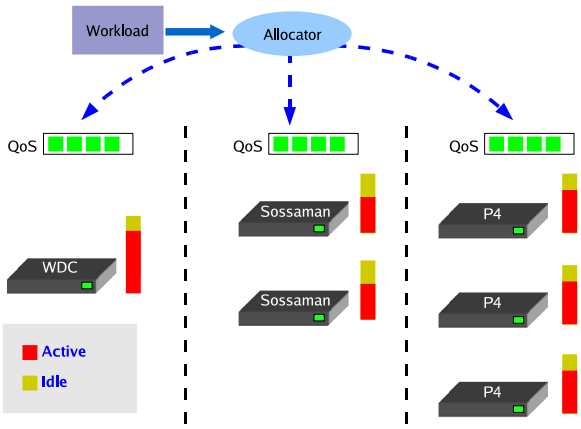


Figure 29: HALM Allocation and Performance Model.

For our analysis, we consider applications that mimic the high performance computational applications common to datacenter environments, and also heavily exercise the power hungry components of server platforms, the processor and memory. Two aspects of these workloads are captured in our experimental analysis. First, these workloads are inherently transactional, such as the previous financial payroll example or the processing of risk analysis models across different inputs common to investment banking. Second, with the ability to incorporate large amounts of memory into platforms at relatively low costs, these applications often execute mostly from memory, with little or no I/O being performed. Though I/O such as network use can play a significant role in multi-tier enterprise applications, we

leave consideration of such characteristics to future work. To realize our application model, while also providing deterministic and repeatable behavior for our experimentation, we utilize benchmarks from the SPEC CPU2000 suite as representative examples of transaction instances. SPEC benchmarks allow for the isolation of processor and memory components, while also generating different memory loads. Indeed, many SPEC benchmarks exhibit significant measured memory bandwidth of 5-8 GB/sec on our systems. In order to provide an unbiased workload set, we include all SPEC benchmarks in our experiments. For each application, we specify an SLA in terms of required transaction processing rate, equal to the throughput achievable on the Woodcrest platform.

5.4 Workload Behavior Estimation

The power/performance predictor component of our HALM framework can be implemented in multiple ways. For example, one can profile a set of microbenchmarks on all platform configurations and develop statistical mapping functions across these configurations. However, as the platform types and heterogeneity increase, the overhead of such approaches can be prohibitive. Instead, we develop a predictor that relies on the architectural platform properties and adjusts its predictions based on the heterogeneity specifications. We refer to this model as the *"Blocking Factor (BF) Model"*. The BF model simply decomposes execution cycles into *CPU cycles* and *memory cycles*. CPU cycles represent the execution with a perfect last-level cache (LLC), while memory cycles capture the finite cache effects. This model is similar to the *"overlap model"* described by Chou *et al.* [18]. With the BF model, the CPI (cycles per instruction) of a workload can be represented as in Equation 11. Here CPI_{CORE} represents the CPI with a perfect LLC. This term is independent from the underlying memory subsystem. CPI_{MEM} accounts for the additional cycles spent for memory accesses with a finite-sized cache.

$$CPI = CPI_{CORE} + CPI_{MEM} \tag{11}$$

The CPI_{MEM} term can be expanded into architecture and workload specific characteristics. Based on this, the CPI of a platform at a specific frequency f_1 can be expressed

as in Equation 12. Here, MPI is the memory accesses per instruction. This is dependent on the workload and the LLC size. L is the average memory latency, which depends on the memory subsystem specifications and BF is the *blocking factor* that accounts for the overlapping concurrent execution during memory accesses, which is a characteristic of the workload.

$$CPI(f_1) = CPI_{CORE}(f_1) + MPI \cdot L(f_1) \cdot BF(f_1) \tag{12}$$

To estimate how the CPI of an application changes with frequency, we need to consider how the independent parameters vary. CPI_{CORE} is independent of frequency, as cycles spent with perfect LLC do not change with frequency. MPI is a feature of the workload and does not change significantly with frequency. The actual memory latency is constant in time and does not scale with CPU frequency. Therefore, the cycle memory latency should scale with frequency. Finally, for the BF parameter we experimentally compared prediction accuracies using both a constant BF and one that varies with frequency. Both approaches perform comparably in terms of CPI prediction accuracy (with 1.2% average prediction errors). Therefore, to simplify our workload descriptors we assume BF is constant across frequencies. Based on these observations, the CPI of a platform at a different frequency f_2 can be expressed as in Equation 13.

$$CPI(f_2) = CPI_{CORE}(f_1) + MPI \cdot L(f_1) \cdot (f_2/f_1) \cdot BF(f_1) \tag{13}$$

With this interpretation of the BF model, by simply knowing the CPI_{CORE}, MPI and BF of a workload at a specific frequency, we can predict its behavior on all other instances of within-platform heterogeneity for a platform. However, the more interesting application of the BF model is for the across-platform heterogeneity. Here the natural decoupling of the microarchitectural and memory subsystem differences in the BF model enables us to estimate application performance on a platform lying on a different corner of the memory and microarchitecture heterogeneity space. Among our four experimental platforms, two of these can be chosen as *"orthogonal platforms"*, which span the two opposite corners of the across-platform heterogeneity. For our experiments, Sossaman and Dempsey platforms

satisfy this condition, as they have mutually exclusive microarchitectural and memory properties. Then, the characteristics of the remaining two platforms can be composed from the subcomponents of the two orthogonal platforms. For example a Woodcrest platform can be approximated as the composition of the microarchitectural features of the Sossaman platform and the memory subsystem of the Dempsey platform. Conversely, the Irwindale platform can be considered as a composition of Sossaman-like memory and Dempsey-like microarchitecture properties. Note that although LLC features are architectural features, these are considered as part of the memory subsystem as their effect pertains to the memory CPI. With this division of platforms into *"orthogonal"* and *"derived"* platforms, we can simply gather workload characteristics on single instances of the orthogonal platforms, and project application behavior on all other platform configurations in the data center.

Considering across- and within-platform heterogeneity, by determining the CPI_{CORE}, MPI and BF for a workload on two specific frequency settings of Sossaman and Dempsey platforms, we can predict the workload behavior on all—total of 13—configurations of all the platforms. To determine the workload behavior on another instance of the orthogonal platforms, we simply use the within-platform prediction method described above. To predict the behavior on a derived platform, we use the corresponding CPI_{CORE} and CPI_{MEM} components from the orthogonal platforms with the memory latency L of the derived platform. For example, Equation 14 shows how the CPI for a Woodcrest platform at frequency f_1 can be predicted from Sossaman and Dempsey descriptors.

$$CPI(W@f_1) = CPI_{CORE}(S) + MPI(D) \cdot L(W@f_1) \cdot BF(D) \tag{14}$$

Here, $CPI(W@f_1)$ is the CPI of Woodcrest at frequency f_1, $CPI_{CORE}(S)$ is the perfect LLC CPI of Sossaman, $MPI(D)$ is the memory accesses per instruction for Dempsey, $L(W@f_1)$ is the memory latency of Woodcrest at frequency f_1, and $BF(D)$ is the blocking factor observed from Dempsey. With this approach, we can provide reasonably accurate predictions of workload behavior on different platforms, without actually accessing the derived platforms.

Next we present the achieved accuracies for all the experimented platform configurations. We compare the predicted workload performance to the actual workload performance acquired by performance counters. First, Figures 30 and 31 show the actual and predicted execution times for our choice of orthogonal platforms, Sossaman and Dempsey respectively, for all available frequency settings. In these examples, the BF model uses the within-platform estimations to derive the execution times at all frequencies. As the figures show, the BF model can very accurately account for within-platform heterogeneity. The average prediction error for this case is around 2%.

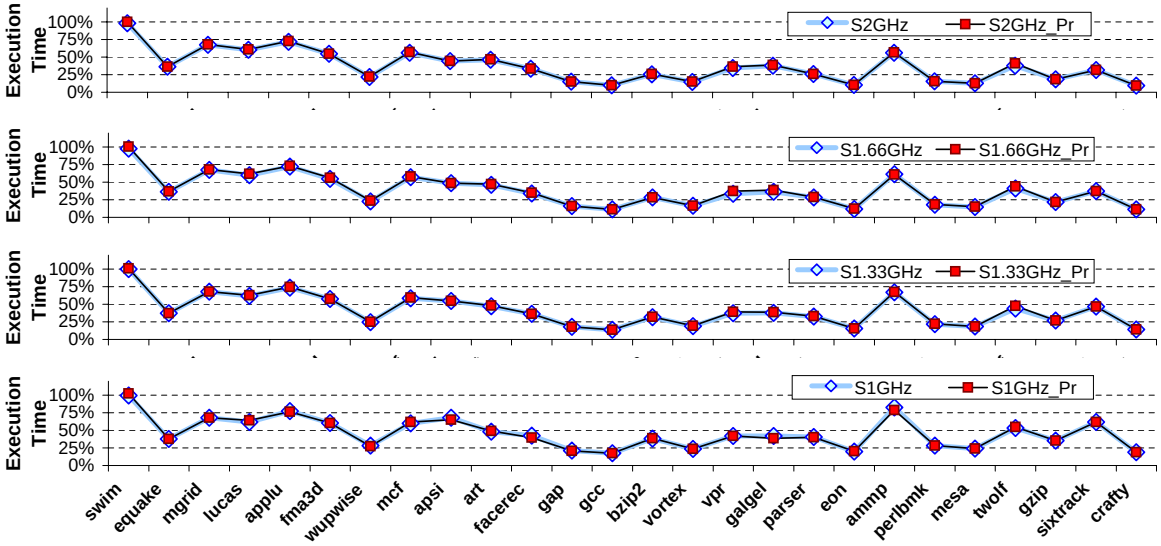


Figure 30: Prediction Results for Sossaman Platform.

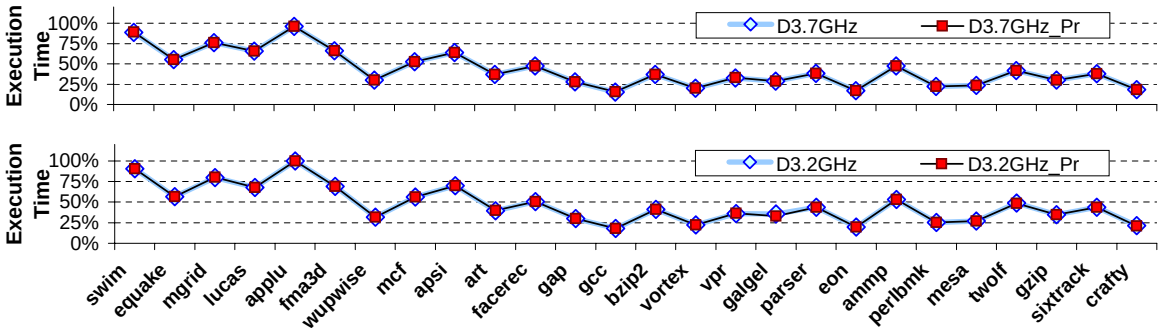


Figure 31: Prediction Results for Dempsey Platform.

Figures 32 and 33 show the actual and predicted execution times for the derived platforms, Woodcrest and Irwindale respectively, for all available frequency settings. Interestingly, the predictions track actual execution times very well also in this case even though the

BF model does not rely on any actual measured application behavior information on these platforms. By only using the independent parameters observed on Sossaman and Dempsey, the BF model can produce accurate projections of application behavior on Woodcrest and Irwindale, simply by leveraging the architectural and memory system similarities between the derived and orthogonal platforms. For these derived platforms, the average prediction error is around 20% (with 63% of the applications having less than 20% errors).

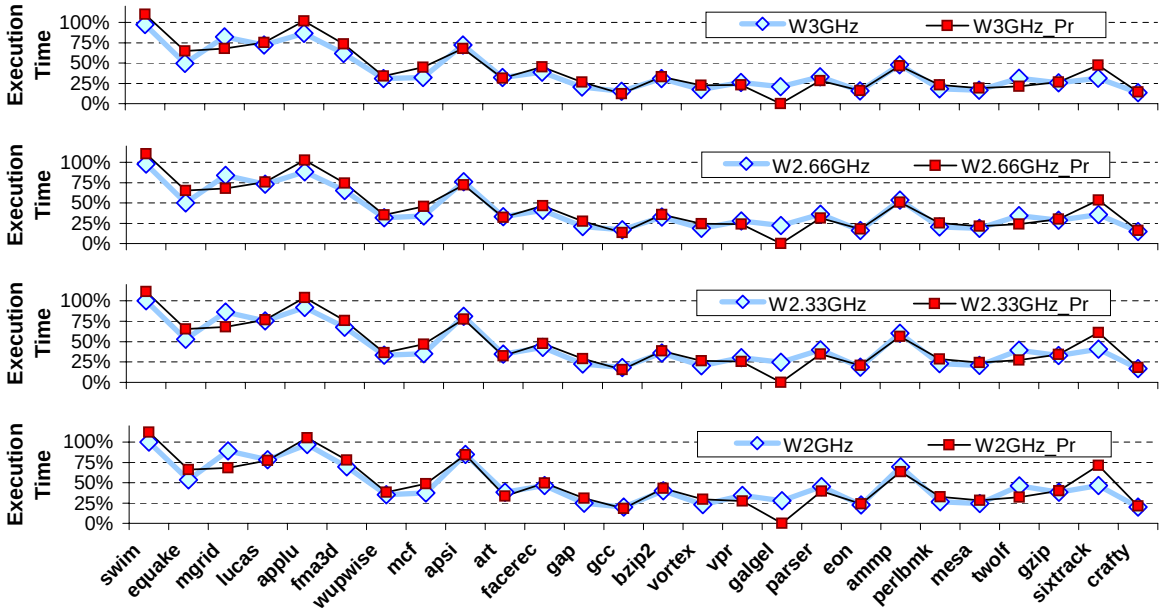


Figure 32: Prediction Results for Woodcrest Platform.

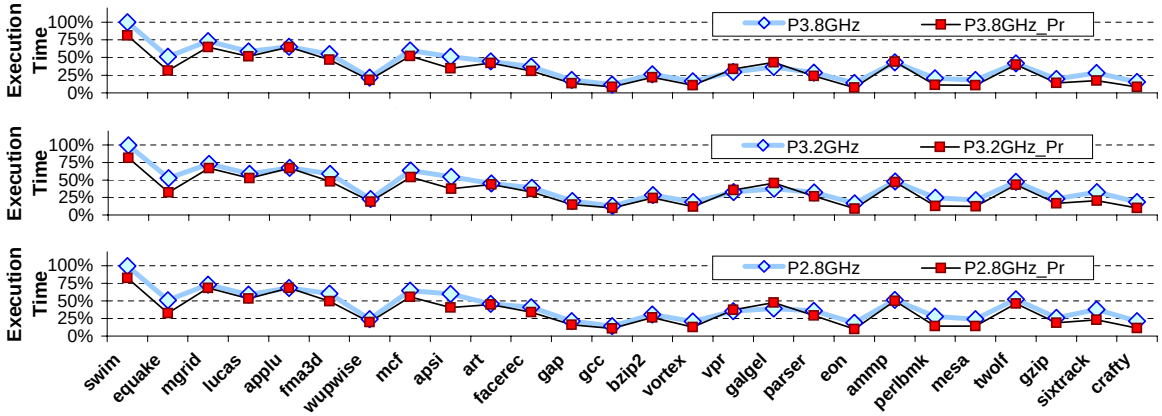


Figure 33: Prediction Results for Irwindale Platform.

These results show that our prediction method with the BF model can produce reasonable approximations to workload behavior under within- and across-platform heterogeneity.

Especially with the within-platform predictions, we achieve close to actual performance behavior. Although the across-platform prediction errors are relatively higher, the figures show that they successfully preserve the performance relations across benchmarks, and more importantly across platforms. In the following sections, we show that these estimates achieve sufficient accuracy to represent workload behavior and lead us to achieve close to optimal allocation with our heterogeneity-aware allocator.

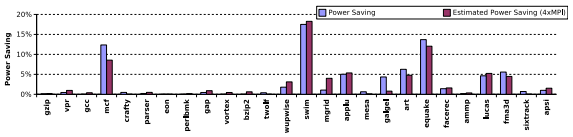


Figure 34: Power Saving Predictions for DPM Enabled Platforms.

The final heterogeneity type supported by our predictor is the DPM-capability heterogeneity. For this, we consider a platform that enables DVFS during memory bound execution regions of an application. We implement this functionality as part of OS power management, based on prior work [43]. To incorporate DPM awareness, we extend the predictor component to estimate the potential power savings that can be attained when executing a workload on a DPM enabled platform. Experimental results show that there is a strong correlation between the MPI of a workload and its power saving potential. Therefore, we utilize the MPI attribute in the workload descriptors to predict the power saving potentials of workloads on DPM enabled platforms. Figure 34 shows that our MPI based prediction approach effectively captures the power saving potentials of different workloads and successfully differentiates applications that can benefit significantly from being allocated to a DPM enabled machine. As we describe in Section 5.5.1, we use this predictor to choose workloads that should be assigned to the DPM enabled platforms.

5.5 Management Policies

5.5.1 HALM Allocation Policy

After processing the workload and platform descriptors, and utilizing our BF model for performance prediction, the next step is to perform allocation of resources to a set of applications in a datacenter. Evaluations are based on a greedy policy for allocating workloads. In particular, with each application i, we associate a cost metric for executing on each platform type k. Workloads are then ordered into a scheduling queue based on their maximum cost metric across all platform types. The allocator then performs application assignment based on this queue, where applications with higher worst-case costs have priority. The platform type chosen for an application is a function of this cost metric across the available platforms as well as the estimated DPM benefits. As a cost metric for our policy, we define $N_{i,k}$, the number of platforms of type k required to execute a workload i, $N_{i,k}$. This value is clearly a function of both the performance capabilities of the platform and the SLA requirement of the workload. $N_{i,k}$ can be analytically defined, given the transaction based application model utilized in our work. For each application i, the service level agreement (SLA) specifies that X_i transactions should be performed every Y_i time units. If $t_{i,k}$ is the execution time of a transaction of application i on platform k, the resulting number of platforms required to achieve the SLA can be expressed with Equation 15.

$$N_{i,k} = \lceil \frac{X_i}{\lfloor \frac{Y_i}{t_{i,k}} \rfloor} \rceil \tag{15}$$

The $t_{i,k}$ values are provided by the performance predictor. It should be noted that there is a discretization in $N_{i,k}$, which is due to the fact that individual atomic transactions cannot be parallelized across multiple platforms. $N_{i,k}$ is therefore better able to handle errors due to the inherent discretization performed, making it a strong choice as a cost metric (other possible metrics are discussed and defined in our paper [81]). Given the use of $N_{i,k}$ as our cost metric, our allocation approach first determines the platform types of which (1) there are enough available systems to allocate the workload and (2) the cost metric is minimized. We then use DPM savings to determine whether a more power efficient platform alternative

should be used between those with the same cost value. In other words, if there are multiple platform types for which an application has the same $N_{i,k}$ value, we utilize a DPM specific threshold to decide whether or not it should be scheduled to a DPM enabled platform type. As we demonstrate in the following section, this threshold based approach can be effective in identifying workloads that can take advantage of DPM capabilities.

5.5.2 HALM Power Budgeting Policy

In order to address transient power delivery or cooling issues, it is sometimes necessary to temporarily reduce power consumption in a datacenter. To provide this mechanism, we develop a load shedding policy based upon an existing workload allocation scheme. The goal of the policy is to reduce the amount of resources provided to applications in order to meet a power budget while still allowing all workloads to make some progress. In other words, application performance may be degraded compared to prescribed SLAs, but all applications achieve some fraction of their SLA.

Our power budgeting policy is, again, a greedy approach. For all applications with resources that can be shed, *i.e.*, applications that utilize more than one platform, we define a power-efficiency metric as the throughput per Watt that is being provided by each resource. Afterwards the resources with minimal power efficiency are shed until the power budget is met. As our experimental results demonstrate, this simple metric allows for improved performance when power budgeting must be performed, as well as better fairness across workloads in terms of performance degradation experienced.

5.6 Experimental Evaluation

5.6.1 Increasing Power Efficiency

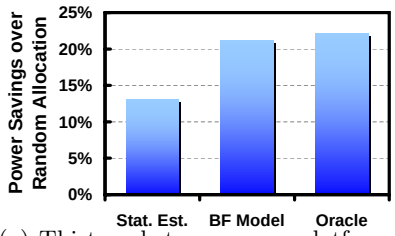
In order to evaluate our heterogeneity-aware allocation approach, we perform power and performance measurements of our SPEC based representative transactional workloads across each type of platform. To scale these results to the number of platforms present in datacenters, this measured data is extrapolated analytically using a datacenter allocation simulator which combines real power and performance data, prediction output, and allocation policy

definitions to calculate power efficiency in various datacenter configurations. In the simulator, we provide the output of the predictor as input to the allocation policy. We always assume that the platforms which are profiled are the 2GHz Sossaman platform and the 3.7GHz Dempsey system. Since we assume the workload attributes are profiled accurately on these systems, for fairness we also assume that for these two platforms performance data is obtained via profiling as well and is therefore known perfectly. We then consider three different scenarios: (1) all other platform performance information is known perfectly (oracle) (2) our BF model is used to predict performance for the remainder of platforms as described in Section 5.4 (BF model) (3) incorporating a simple statistical regression approach (Stat. Est.). For this regression method, we profile a subset of applications across all platforms to obtain linear performance prediction models parameterized by variables that can be obtained by profiling a workload on the 2GHz Sossaman and 3.7GHz Dempsey systems (CPI, MPI, etc.). The regression models can then be used to predict performance of any application. The baseline allocation scheme we compare against is a random one, since it closely estimates the common round-robin or utilization based approach.

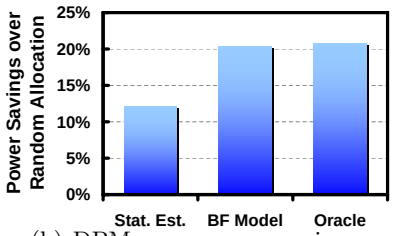
The efficiency improvements achievable in a datacenter are also dependent upon the system's current mix of applications. To obtain our results, we randomly pick applications and allocate them using the random approach until no more workloads can be scheduled. Using the resulting set of workloads, we then evaluate power consumption when using our prediction and allocation policies, and compare them against the random allocation result. This is repeated a hundred times for each of our data points.

We first look at the benefits achieved with HALM in datacenter configurations with across-platform and within-platform heterogeneity but no DPM support. In particular, we include the four base platforms, Woodcrest, Sossaman, Dempsey, and Irwindale, as well as the frequency variations of the platforms. We create datacenter configurations with equal numbers of each type of system. Trends are consistent across various datacenter sizes, so for brevity, we include here only results with 1000 platforms of each type. The resulting datacenter has 13 types of platforms, and power consumptions vary with allocation as shown in Figure 35(a). The first interesting observation is that platform heterogeneity allows us

to achieve improved benefits over a simple random approach. Indeed, we see improvements of 22% with perfect knowledge and 21% using our BF based prediction compared to a random allocation policy. We also observe a significant difference between the statistical and analytical prediction schemes. The regression approach is unable to scale in terms of accuracy with increased heterogeneity, whereas the BF approach achieves close to optimal power savings.



(a) Thirteen heterogeneous platforms



(b) DPM-aware power savings.

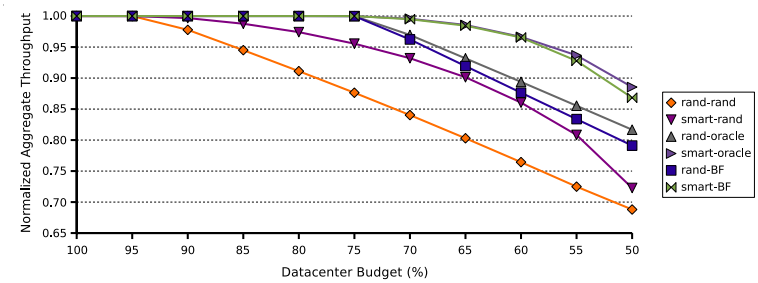
Figure 35: HALM Power Improvements.

In order to evaluate how well our allocator can exploit DPM support, we extend the thirteen platform type configuration with an additional Woodcrest 3GHz platform which provides DPM support. We again find that our BF prediction method can provide improved savings over the statistical approach as shown in Figure 35(b). To more closely determine our ability to exploit DPM mechanisms, we also evaluate the power consumption of the thousand DPM-enabled platforms (all of which are active). We find that our BF model based allocation is able to improve the power efficiency of these platforms by 3.3%. This illustrates the potential of HALM to provide additional benefits when platforms vary in the power management they support.

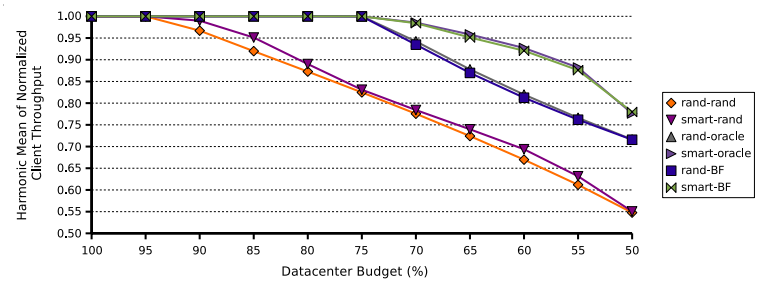
5.6.2 Maximizing Performance Under Power Budgets

As a second benefit of HALM, we evaluate the ability to perform load shedding when power budgeting must be performed. In particular, as previously described, our goal is to maximize performance obtained in the datacenter while observing power budgets. For the purposes of this chapter, we consider transient power budgeting where workloads are not (re-)migrated, but instead, based upon an existing allocation, resources are temporarily withheld from applications and placed into low power states. For comparison, we consider three such initial

allocations from the previous section, one based upon a random allocation, one based upon perfect oracle performance information, and finally an allocation based upon prediction with our BF approach. For each allocation, we consider two load shedding policies, one which randomly selects an application and sheds resources (*i.e.* a single compute node) if possible, and our greedy "smart" policy described in Section 5.5.2.



(a) Throughput Results



(b) Client Fairness

Figure 36: HALM Power Budgeting Results.

Figure 36(a) provides the aggregate performance across all workloads in the datacenter for different power budgets. The performance results are normalized with respect to the maximum achievable performance at the maximum unconstrained power budget (100%). The figure shows how the performance of different allocation policies decrease with decreasing power budgets. The range of applied power budgets goes only down to 50% as there is no possible allocation that meets the budget below this point. The six scenarios considered are random shedding based upon a random allocation (rand-rand), our load shedding policy based upon a random allocation (smart-rand), similarly the two shedding approaches based upon oracle based allocation (rand-oracle and smart-oracle), and finally, shedding based

upon a BF prediction based allocation (rand-BF and smart-BF). We see multiple interesting trends. First, the intrinsic benefits of the heterogeneity-aware allocations towards budgeting are apparent in the figure by the fact that performance does not begin to reduce until lower power budgets compared to a random allocation scheme. We also see that given a particular allocation, our shedding policy provides benefits in performance across the set of power budgets. Moreover, again, we find that our BF prediction model behaves very close to an oracle based allocation when our load shedding policy is used. Overall, we see benefits of up to 18% in performance degradation compared to a random load shedding policy based upon a random allocation. Figure 36(b) evaluates the performance degradations for the six approaches by also taking the fairness of load shedding into account. Here, we show the aggregate datacenter performance as the harmonic mean of the individual workload throughputs, which is a commonly used metric for evaluating fairness [71]. We see from the figure that a random allocation always exhibits poor performance regardless of the load shedding policy used. A heterogeneity-aware allocation, on the other hand, provides improved fairness, particularly when combined with our load shedding policy.

5.7 Related Work

HALM builds upon existing work and extends the state of the art in power management research. A variety of mechanisms exist to provide power and thermal management support within a single platform. There are proposed mechanisms for the management of thermal issues in modern processor designs [13, 64], focusing on single platform characteristics as opposed to datacenter level management achieved with HALM. Processor frequency and voltage scaling based upon memory access behavior has been shown to successfully provide power savings with minimal impact to applications. Resulting solutions include hardware based approaches [61] and OS-level techniques, which set processor modes based on predicted application behavior [43]. HALM is designed to perform load management that is aware of such underlying power management occurring in server platforms. Power budgeting of SMP systems with a performance loss minimization objective has also been implemented via CPU throttling [52]. Other budgeting solutions extend platform support for fine grain

server power limiting [60]. The power budgeting achieved with HALM is based upon the use of resource allocation to reduce power consumption across multiple systems, as opposed to throttling performance of individual components.

Power prediction tools have been utilized in previous research as well. Various simulators for estimating power consumption of processors have been developed to aid in microarchitecture design [14, 20]. Other methods focus on using exposed hardware counters for prediction [59, 62], or estimate power consumption of peripheral devices alone [15]. The prediction methods used with HALM are geared to estimate platform level power consumption, and also to scale across heterogeneous platform hardware.

At the datacenter level, incorporating temperature-awareness into workload placement has been proposed by Moore *et al.* [79], along with emulation environments for studies of thermal implications of power management [38]. HALM can use these thermal aware strategies to perform power budgeting based upon datacenter temperature characteristics. Chase *et al.* discuss how to reduce power consumption in datacenters by turning servers on and off based on demand [16]. Utilizing this type of cluster reconfiguration in conjunction with DVFS [26] and the use of spare servers [96] has been investigated as well. As opposed to these approaches, HALM attempts to reduce power consumption by intelligently managing workloads across heterogeneous servers. Enforcing power budgets within datacenters by allocating power in a non-uniform manner across nodes has been shown to be an effective management technique [29]. Techniques for enforcing power budgets at blade enclosure granularities have also been discussed [97]. HALM budgets aggregate power consumption via resource allocation without assigning per server power budgets as with these previous approaches.

Heterogeneity has been considered to some degree in prior work, including the evaluation of heterogeneous multi-core architectures with different core complexities [56]. HALM considers platform level heterogeneity, as opposed to processor asymmetry. In cluster environments, a scheduling approach for power control has been proposed for processors with varying fixed frequencies and voltages [34]. HALM supports heterogeneity across additional dimensions, such as power management capabilities and memory. A power efficient

web server with intelligent request distribution in heterogeneous clusters is another example which considers leveraging heterogeneity in enterprise systems [39]. HALM goes beyond these methods, by considering not just the differences in platforms' performance capabilities, but also in their power management capabilities.

5.8 Summary

The integration of power management solutions into large scale enterprise environments has become critical due to overwhelming power and cooling overheads. In this chapter, we illustrate the possible benefits of exploiting a natural artifact of these systems, platform heterogeneity, to enable improved power efficiency. We propose our HALM architecture which utilizes sets of system descriptors as well as prediction models to enable policies for reducing power consumption as well as employing effective load shedding to meet power budgets. Our evaluation of the system exposes improvements of up to 21% in power consumption compared to a non-aware management scheme, and up to 18% in performance attained in power constrained scenarios.

CHAPTER VI

VIRTUALPOWER: POWER MANAGEMENT IN VIRTUALIZED SYSTEMS

In Chapter 5 we highlighted the need for power management in modern datacenters, and evaluated the benefits of heterogeneity awareness. An aspect of future datacenter environments not directly addressed, however, is the proliferation of virtualization techniques into the enterprise domain. Specifically, the growing capabilities of hardware support for virtualization [87] and of software solutions like Xen [9] or VMware [105] have made it easier to flexibly use datacenter resources for applications [4]. The resulting interplay of power management and virtualization may be formulated more precisely as follows:

1. 'Soft' and 'Hard' power scaling – Virtualization creates new possibilities for scaling the allocation of physical resources to guest virtual machines (VMs), both using 'soft' techniques that exploit a hypervisor's ability to limit hardware usage by guests, and 'hard' techniques that leverage underlying hardware support such as processor frequency scaling. *To be effective, virtualization layer power management must exploit both of these methods.*

2. Independence and coordination – Guest VMs are designed to execute their own, independent OS- and/or application-specific methods for power management. For example, the Linux operating system allows for dynamic voltage and frequency scaling (DVFS) of the processor, where governing policies can be loaded into the kernel or executed in userspace daemons. Application specific policies that can address real-time workloads [90, 93] or meet application requirements with minimum power consumption [30] can then be integrated. *From these facts, it is clear that a necessary element of power management is the need to coordinate between VM-level solutions and global goals that concern platform-, rack-, and then datacenter-level power consumption.*

3. Flexibility in management – Modern datacenters that utilize virtualization often house multiple generations of equipment with different attributes and management capabilities [82], and deploy a variety of applications that have distinct requirements expressed by Service Level Agreements (SLAs). These traits dictate the use of diverse dynamic management policies. *Therefore, to effectively address virtualized environments it is important to give administrators the flexibility to provide their own power management policies.*

The *VirtualPower* approach to power management presented in this chapter can exploit both hardware power scaling and software-based methods for controlling the power consumption of underlying platforms. Its power management actions take into account guest VMs' independent power management policies, and can coordinate across multiple such policies to attain desired global objectives. Effective for both fully and para-virtualized guests, VirtualPower's abstractions and methods may be summarized as follows. First, to permit guest VMs to run their own, independent power management methods, VirtualPower exports a rich set of virtualized (i.e., 'soft') power states, termed VirtualPower Management states – *VPM states*. Guest VM-level power management policies, then, observe and act upon these states when carrying out their management actions. Second, coordination is based on the fact that independent VM-level management policies change power states via privileged actions that, for example, modify model-specific registers (MSRs) in hardware. VirtualPower can leverage the fact that these events are trapped by the hypervisor to create an abstraction of *VPM channels*. These channels deliver guest VM power management actions as a set of 'soft' state updates that can be used for application-aware management at the virtualization layer. Third, VirtualPower enables flexibility by encoding the actual power management actions carried out by the infrastructure as *VPM rules*, provided by hardware vendors and/or system administrators. VPM rules treat the 'soft' VPM state changes conveyed by VPM channels as 'hints' for assigning actual *shadow VPM states* for guest VM resources, a control relationship we term *state-based guidance*. Finally, VPM rules are based on a rich set of underlying *VPM mechanisms* that provide a uniform basis for implementing management methods across heterogeneous hardware platforms.

The implementation of VirtualPower with the Xen hypervisor significantly extends the infrastructure's power management capabilities. On this basis, our contributions include: (1) an experimental study of power management and its implications in virtualized systems, (2) the use of VPM channels and states to obtain application-specific power/performance tradeoffs for representative enterprise applications, (3) the definition and evaluation of multiple management actuators in the form of our VPM mechanisms, and (4) the evaluation of VirtualPower with sets of rules that jointly implement a tiered policy-driven approach to online power management. Experimental evaluations are based on micro-benchmarks, the RUBiS enterprise application, and transactional workloads. Using VPM rules that implement representative power management policies, measured results show improvements of up to 31% in the active power consumption of underlying computing platforms. They also demonstrate VirtualPower's abilities to perform QoS-aware power throttling and to exploit the consolidation capabilities inherent in modern multicore platforms. Moreover, when using the tiered VPM rule components for managing VMs across heterogeneous platforms, we reduce power consumption up to 17% by exploiting power management heterogeneity and up to 34% by performing runtime consolidation onto more power-efficient hardware. Finally, VirtualPower enables future work on management policies and mechanisms that can extend across multiple devices and/or separately manageable platform components like memory, can take into account additional properties like thermal events [38], and can address the multiple machines, racks, and enclosures found in modern datacenters.

6.1 Managing Virtualized Systems

This section discusses the impact of virtualization on the power management landscape of enterprise systems. Of particular interest are barriers to utilizing guest VM- or application specific policies for enabling workload-aware management in these environments. We identify such hurdles and describe how VirtualPower is designed to overcome them.

6.1.1 Scalable Enterprise and Virtualized Resources

As previously summarized in Chapter 5, desired benefits from virtualization include improved fault isolation and independence for guest VMs [9], performance isolation [51], and

the ease of seamless migration of VMs across different physical machines [19]. Jointly, these permit usage models in which VMs freely and dynamically share pools of platform and datacenter resources, as also considered in recent research on autonomic management [3, 36] and as indicative of next generation enterprise management environments. The goal, of course, is the delivery of flexible and scalable enterprise infrastructures, as illustrated in Figure 37.

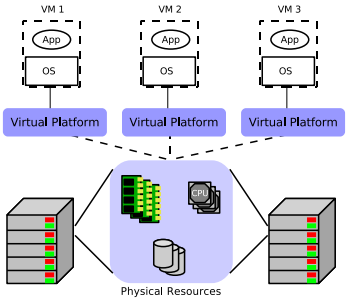


Figure 37: Scalable Enterprise Infrastructure.

Power management directly threatens the independence and performance isolation properties of virtualization technologies. First, how can guest VMs realize the goals of their application-specific power management policies when there is a disassociation between the virtual resources they are managing and the physical resources actually being used? Here, a specific issue is that guest machines may have inconsistent views of the management capabilities of the underlying resources they are trying to manage, particularly in the presence of VM migration. Second, in datacenter settings, inconsistencies are aggravated by the need to frequently update platforms, to replace them to deal with failures, or to add new platforms for capacity increases. A natural outcome of such changes is increased heterogeneity, with respect to platform power properties, performance characteristics, and/or management capabilities. Isolation and independence, however, demand that VMs' power management policies not be altered to deal with such changes. *The VirtualPower approach maintains the isolation and independence properties of virtualization systems by exporting to VM-level management policies 'soft' VPM states, wherein guest machines have a consistent view of hardware management capabilities regardless of the underlying set of physical resources.* The

implementation of the approach, then, performs state-based guidance to map changes to these soft states to changes in underlying hardware according to associated VPM rules, as described further in Section 6.2.

6.1.2 Leveraging Guest VM Policies

The increasingly apparent need for power management has led to the implementation of system- and application-level policies that carefully balance hardware performance states to attain application-specific notions of quality-of-service (QoS). An example of such a policy is the *ondemand* governor integrated into the Linux kernel. This policy scales the frequency of the processor based upon the CPU utilization observed by the operating system and is effective for workloads with variations in processor utilization, such as web servers. Other management policies include those specific to real-time systems, where processors' DVFS capabilities are used to reduce power consumption while still maintaining application deadlines [90, 93]. Since VM-level management policies can meet application-specific QoS constraints, it is highly desirable to use them to manage the power/performance tradeoffs of modern computing platforms. Giving these policies direct access to hardware power management facilities, however, negates the notion of virtualized hardware resources. Moreover, direct access would threaten desired performance isolation properties, as evidenced on thermally sensitive multicore chips where a VM increasing its frequency may heat up its core in a manner that compromises another core's ability to run at a speed sufficient for meetings its VM's desired QoS. Worse, such actions might be malicious, as with a VM compromised by a power virus.

VirtualPower responds to these challenges with an approach that takes advantage of the power management policies built into guest VMs, but interprets them as 'hints' rather than executable commands. These hints are acquired by exploiting the ACPI [41] interface for power management provided by modern platforms. When guest VMs attempt to perform privileged operations on this interface, current hypervisors ignore them, but VirtualPower intercepts them and maps them to VPM channels. These channels, in turn, provide them as useful hints to VirtualPower's VPM rules. VPM rules may then use the 'soft' state requests

that make up these hints as inputs to localized power management and/or to global policies that coordinate across multiple guest VMs and multiple machines. *The combination of VPM states and channels provides for coordinated power management while maintaining VM independence.*

6.1.3 Limitations of Hardware Management

An issue with utilizing hardware support for power management in virtualized systems is that resources within a management domain may be shared by multiple guest VMs. For example, memory DIMMs that are accessible via the same bus and/or share the same voltage plane may contain memory that is allocated to many guests. This means that this component cannot be managed unless all guest machines desire reduced memory bandwidth via bus scaling, or are currently not utilizing the DIMMs such that they can be powered down. Similar limitations exist for disks that contain multiple partitions, each of which may be allocated to different guests. Any one partition can be power-managed by placing the disk into a low power state only if all partitions can be placed into that state.

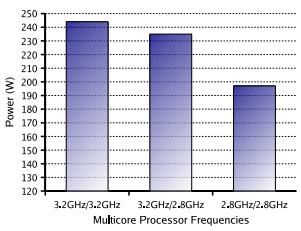


Figure 38: DVFS Limitations on a Dual Core Chip.

An approach to overcoming limitations for power managing shared resources is to use time domain multiplexing, which sets components to the hardware states that reflect the power management criteria of the currently executing VM. Unfortunately, this approach has two significant drawbacks. First, it can only be used if the transition time of resources between management states is much smaller than the scheduling time granularity used by the hypervisor. Second, even if this were the case, this approach is not promising in the context of multicore platforms. Here, the intent to concurrently execute a large number of guest VMs on available cores significantly reduces the ability to utilize time

domain multiplexing for reasons that include the following. Consider the use of DVFS to manage the processor component of multicore platforms. Since the dynamic power consumption of a core is proportional to the product of frequency and voltage squared, in order to obtain significant energy or power savings, voltage scaling must be performed in conjunction with frequency scaling. In multicore packages, however, even when the frequency of cores can be scaled independently, there is only one voltage rail into the package from the motherboard. Therefore, the voltage is constrained by the highest frequency of any core in the package. As shown in Figure 38, this restriction can significantly limit the system-level benefits of hardware power management, where power is only reduced by 4% when scaling frequency on only one processor, but is reduced by 19% when both are scaled at the same time. VirtualPower overcomes hardware limitations derived from resource sharing and limited opportunities for voltage scaling by providing mechanisms for power management that go beyond mere reliance on hardware operating modes. These mechanisms include consolidation techniques and runtime changes to hypervisor-level scheduling. *The outcome is a rich set of VPM mechanisms for VPM rules that exploit both 'soft' and hard power scaling techniques.*

6.2 VirtualPower Architecture

In keeping with common practice, VirtualPower does not require modifications to guest operating systems. Further, while implemented with the Xen [9] virtualization framework, the VirtualPower architecture and abstractions are designed for the wide range of virtualization solutions currently deployed or under development. Therefore, most of its functionality resides as a controller in the driver domain, termed Domain Zero (`Dom0`) in current systems, with only small changes required to the underlying virtual machine monitor or hypervisor. Specifically, VirtualPower's monitoring functionality is integrated into the hypervisor, whereas its higher level mechanisms and policies (i.e., VPM mechanisms and rules) reside in `Dom0`. As a result, policies have easy access to CPU and device power states, and they can take advantage of `Dom0`'s control plane functionality for creating guest virtual machines, migrating them, ... *etc.*

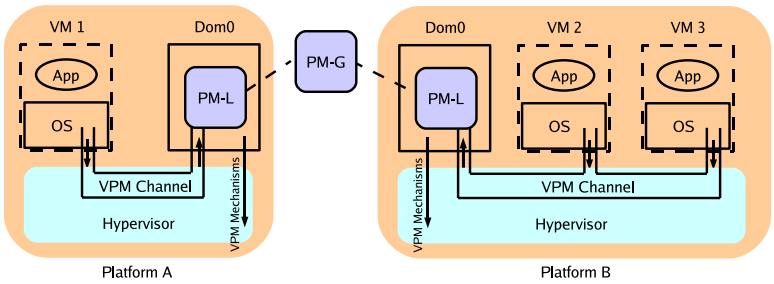


Figure 39: VirtualPower Management Architecture.

Figure 39 illustrates the VirtualPower architecture. Each physical platform runs a hypervisor and associated `Dom0`. Guest VMs perform power management based upon the 'soft' VPM states exported to them. VPM channels capture guest VM power management requests, via updates to these 'soft' states, in the hypervisor. This information is then passed to power management software components, composed of sets of VPM rules running in `Dom0`, which finally, use VPM mechanisms to actually carry out management decisions. As described further in Section 6.5, our current VPM rules implement a tiered policy approach: local PM-L policies perform actions corresponding to resources on local platforms, and a higher-level PM-G policy component uses rules responsible for coordinating global decisions (e.g., VM migration). We next describe in more detail the VPM states, channels, and mechanism components that permit VPM rules to perform coordinated power management.

6.2.1 VPM States

It is important to leverage the substantial investments and knowledge about online power management embedded in guest operating systems. Technically, this implies the need to provide to the policies run by guest VMs what appears to be a richly manageable set of underlying resources, even if the hardware platform does not actually provide the different power states being exported. In heterogeneous hardware environments, for instance, a guest machine with sophisticated power management policies may initially be deployed on hardware that does not support power management and later be migrated to manageable components. VirtualPower is designed to permit guest VM policies to run in both systems,

by exporting to them 'soft' VPM states and then using appropriate VPM rules and VPM mechanisms to realize them. These VPM states need not reflect the actual operating points (i.e., Px states [41]) supported by underlying hardware, but instead, they consist of a set of performance states made available by VirtualPower for use by VMs' application-specific management policies. This is implemented by simply defining these 'soft' states in the relevant portions of the ACPI tables created for guest VMs. In addition, for management flexibility, VM-level changes to 'soft' VPM states are disassociated from the assignment of *shadow VPM states* to actual resources by VPM rules.

In exporting VPM states, a limitation is imposed by the fact that the ACPI interface specification defines a maximum of sixteen Px states for a device, thereby limiting the number of possible soft states available for each manageable component. We do not consider this an issue because, in general, most devices, such as processors, only support a small number of performance states. Therefore, even in heterogeneous environments where one must define a set of VPM states that allows VMs to provide useful hints across a range of machines, there is sufficient ability to define a rich set of VPM states that express the performance scaling capabilities of multiple physical platforms.

6.2.2 VPM Channels

VPM channels are the abstraction via which guest VMs provide input on power management to VPM rules, as illustrated in Figure 39. Since hardware power management is privileged, whenever guests attempt to execute such operations, they are trapped by the hypervisor, whereupon VirtualPower re-packages them as VPM events made available to the VPM rules in `Dom0`. In particular, the information encapsulated by a VPM event includes a timestamp, the 'soft' state set by the guest VM, and information regarding the actual performance state being provided to the guest when the request was made. The implementation of VPM event information retrieval by VPM rules is an asynchronous one, where VPM event information is temporarily stored in the hypervisor using a cyclic buffer from which VPM rules may retrieve it. The timing and frequency of retrievals depend upon whether the control algorithm being used requires polling or event-based retrieval techniques.

The implementation of VPM channels uses hypercalls and Xen event channels. VirtualPower defines its own hypercall interface which can be used to perform a variety of actions for VPM rules. The interface defines a `VPM_POLL` operation that allows VPM rules in `Dom0` to query for VPM events corresponding to a particular guest VM. VPM rules can use this operation to periodically retrieve updates about the state changes desired by guest VMs. In addition, VirtualPower provides a Xen event channel that may be used if VPM rules desire immediate notification of desired state changes by guest VMs. This is useful when timely responses to VM behavior are necessary for balancing power management with performance requirements.

6.2.3 VPM Mechanisms

VPM mechanisms are the base level support for online power management by VPM rules. They provide uniform ways of dealing with the diversity of underlying platform power management options. The mechanisms supported by VirtualPower include hardware scaling, soft scaling, and consolidation.

Hardware Scaling Hardware scaling capabilities vary across different platform and device architectures. Moreover, whether or not such scaling is possible or effective depends on VM-level resource sharing (e.g., VMs running across multiple cores in multicore platforms). The VPM mechanism supporting hardware scaling is straightforward, permitting VPM rules to set the hardware states to be used during the execution of a particular guest VM via a `VPM_SET_PSTATE` operation on the VirtualPower hypercall interface. Of course, the actual rules that decide upon such state changes can be non-trivial. For example, policy rules must determine whether or not a set of hardware performance states for VMs can create conflicts due to resource sharing. In the particular case of processor DVFS management, such conflicts are handled by hardware that ensures that the voltage provided to the chip is sufficient for the core running at the highest frequency.

Soft Scaling Hardware scaling is not always possible, or it may provide only small power benefits, as shown in our earlier example of utilizing DVFS on multicore chips. In response,

we introduce the notion of 'soft' resource scaling, which uses resource scheduling to emulate the performance loss of the hardware scaling action that would otherwise be performed. For processor management, this entails modifying the hypervisor's scheduling attributes for a VM to emulate the VM's desired performance mode. For instance, if a VM has requested scaling a core to half of its previous performance state, then with soft scaling, the hypervisor scheduler may reduce the guest's maximum time slice in a period by half. The necessary adjustments to scheduling parameters are performed using a `VPM_SET_SOFT` operation with the VirtualPower hypercall interface.

An obvious issue with soft scaling is that the performance degradation attained in this fashion may not accurately represent what the application would have observed from actual hardware scaling. For example, a guest VM may request a reduced processor frequency, because it expects to observe little or no performance degradation for its current memory-bound application workload. If soft scaling proportionally changes CPU allocation in response to such requests, the application would experience notable and unexpected levels of performance loss. Our implementation of soft scaling, therefore, is carried out by VPM rules that operate with feedback loops based upon state-based guidance in which such degradation is observed, indirectly, by VM requests for increased performance states. This results in policies that dynamically tune to what extent, if any, soft scaling is applied to a physical resource.

Soft scaling can result in substantial power benefits, for two reasons. First, there is the idle power management of resources. Components such as processors are achieving increasingly substantive savings when idle compared to active, even when active power management cannot provide improvements. Thus, soft scaling is most effective when it results in processors being idle for some time (i.e., no VMs currently running on any of their threads). Second, additional savings may be derived from appropriately managing soft scaling across multiple resources, by coordinating their concurrent use [83] and/or by use of consolidation.

Consolidation When soft scaling multiple VMs for multiple resources, such as cores on a multicore chip, it becomes possible to share resources, perhaps to totally idle one core while fully using another one. An obvious first benefit from such resource consolidation is the substantial power improvements derived from placing offloaded resources into their idle or suspend states. An interesting second benefit is derived from resource heterogeneity. In particular, in datacenter environments, there likely exist physical resources that are more power efficient for certain workloads or that are more amenable to online management. By combining soft scaling with VM re-mapping or migration (i.e., consolidation), therefore, it is possible to map multiple 'compatible' instances of virtual resources to appropriately efficient physical resources. This dynamic migration capability is implemented using existing control interfaces in `Dom0`.

6.3 Evaluation Methodology

6.3.1 Experimental Setup

Experimental evaluations of the VirtualPower management infrastructure use standard multicore server hardware. Our testbed consists of multiple dual core Pentium 4 machines, which are identical in terms of their hardware capabilities and components. These machines have processors based upon the Intel Netburst microarchitecture, 3GB of memory, gigabit network cards, and 80GB hard drives. In terms of manageability, they support two physical operating modes for their processor cores: 3.2GHz and 2.8GHz.

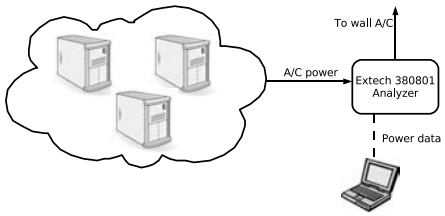


Figure 40: VirtualPower Power Measurement Setup.

Power data is obtained using an Extech 380801 power analyzer, which allows for out of band measurements via a laptop, thereby avoiding undesirable measurement effects on the system under test. Power is measured 'at the wall', in order to include the AC power

consumption of the entire system. We obtain power traces at the maximum sampling rate of 2Hz using the Extech device, then use these traces to calculate offline the power consumption characteristics seen during experiments. Figure 40 summarizes our power measurement configuration.

6.3.2 Guest Applications and Policies

Experiments are conducted with representative enterprise workloads, using the RUBiS tiered web service application and using transactional loads. The distributed components of these applications are instantiated in their own virtual machines, with each VM mapped to a single virtual processor. The different workload characteristics of these codes, along with the varying VM-level policies which drive their management, present interesting challenges to VirtualPower's management policies, as explained next.

Transactional Workloads Many backend services deployed in enterprise systems and datacenters are of a transactional nature. For example, as part of an ongoing collaboration with our corporate partner Delta Air Lines, we have obtained workload traces from a back-end subsystem responsible for tracking operational revenue earned from the services offered by the company [2]. The backend operates by continually receiving, queuing, processing, and then, emitting revenue-relevant events, with current incoming event rates at the level of thousands per hour. Two interesting elements of these event traces are (1) that event queuing delays often outweigh actual event processing times and (2) that arrival rates vary significantly over the course of a day. For example, a large batch arrival of events submitted every day at 4 AM EST creates queue lengths as high as 4000. From (1), it is clear that for these types of applications, changes in the execution times of events due to power management may have negligible effect on overall processing time, due to queuing delays. From (2), it is evident that there exist possibilities for management to exploit changes in application behavior across larger time scales.

For transactional applications like these, the goal of backend servers is to process requests at rates that are sufficiently high to meet certain application-specific SLAs. For the actual traces received from our corporate partner, these SLAs are dictated by revenue

reporting requirements. More generally, different transactional applications will have different performance requirements, and in addition, such requirements may vary over time (e.g., due to changes in reporting requirements or when dynamic pricing methods are used). In all such cases, the principal performance metric for a transactional service is whether it can run its transactions at some currently desired minimum processing rate. For power management, this means that management policies can exploit the slack available between minimum desired rate and the current rate being observed. Accordingly, we define a simple VM-level power management policy that keeps track of 'performance slack' while also maintaining a desired transaction execution rate. When slack is large, the policy requests reduced processor frequencies, and if slack becomes negative, it scales up the underlying processor's performance state. Experiments show that this simple control policy proves to effectively enforce a defined application-specific transaction processing rate. For purposes of experimentation, we use the well-known SPEC CPU2000 suite to create code modules that carry out computationally expensive transactions. These code modules are driven by varying transaction rates and rate requirements. Future steps in this work will consider transactional computations using the Nutch open source search engine [88].

Tiered Web Service Workloads (RUBiS) RUBiS is a well-known tiered web service benchmark, implementing some of the basic functionality of an auction website, including selling, browsing, and bidding. We use a PHP-based two node configuration of RUBiS where there is a web server front end (Apache) connected to a database backend (MySQL). Load is provided using a third client machine. All of these nodes are connected via a gigabit Ethernet switch. Web service applications typically exhibit variations in processor utilization based on current request arrival rates from clients and on current request behaviors. This characteristic can be exploited for power management via reactive policies that attempt to scale the processor to an appropriate performance state based on current load statistics. Since VM-level policies like Linux's *ondemand* governor are designed to exploit this fact, this is the application-specific policy used in our experiments with the RUBiS VMs.

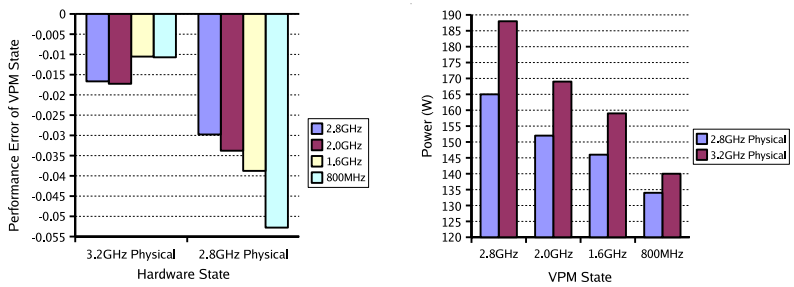
Nutch: Semantically Rich QoS Metrics The third and final workload is a transactional web service application based upon the Nutch open source search engine [88]. In particular, we modify the behavior of Nutch in order to resemble aspects of actual deployed service workloads from one of our corporate partners, Worldspan. Worldspan provides services to online ticket reservation companies. This includes receiving ticket search requests and responding within some period of time with a set of results corresponding to search criteria. In such a scenario, requests from different clients may be associated with varying SLA requirements, including the quality of results in responses, a semantically rich metric which is impossible to measure from outside of the application. We implement this quality model into Nuch for our experiments. In particular, we define requests to require a varying number of search responses in order to be completed, resulting in more (or less) work by the search engine. The responses achieved compared to the amount required are used to calculate the resulting Quality of Information (QoI) of the transaction. We integrate an application specific management policy into the Nutch VM that manages the workload in a manner that maintains a specified QoI level while minimizing power consumption.

6.4 Viability of Soft Scaling

As described in Section 6.2.1, VPM states act to disassociate the management actions of virtual machine policies from the manageability support of underlying physical resources. Our Pentium 4 based processors support two operating frequencies of 3.2GHz and 2.8GHz. With the use of VPM mechanisms, however, we can perform power management actions at the virtualization layer beyond just these two hardware performance states, resulting in the export of five different 'soft' Px states (frequencies) to guest VMs. In particular, guest VMs see frequency capabilities of 3.2GHz, 2.8 GHz, 2.0 GHz, 1.6GHz, and 800MHz. When guest VMs attempt to set these states, it is up to VPM rules to map these requests to actual changes (or not) of the shadow states they maintain. Changes applied to shadow states trigger changes in the Xen scheduler's scheduling attributes and/or in underlying processor power states.

Experimental measurements of soft scaling presented use the Xen hypervisor's earliest

deadline first (SEDF) scheduler, which allows for dynamic tuning of the amount of processing time allotted to a VM's virtual CPU (`vcpu`). In addition, based on the SEDF parameters, the scheduler can be set to be work-conserving, or to limit CPU time in a non-work-conserving fashion. We take advantage of its non-work-conserving mode in order to use the scheduler as a control actuator via the VPM soft scaling mechanism. For example, consider the case of a VM executing with the physical processor at 3.2GHz and using work-conserving scheduling parameters. Based on VM power management request information, VPM rules wish to set the shadow VPM state of the VM to 1.6GHz to reduce the performance of the application by 50%. One method for doing this is to use the VirtualPower hypercall interface to schedule the `vcpu` with half its original slice size and with the non-work-conserving setting. Another method is to use hardware scaling in conjunction with soft scaling. In this example, a VPM rule can specify a reduced physical frequency f_l and simultaneously soft scale the resource to $\frac{1}{2} * \frac{3.2GHz}{f_l}$ of its original amount.



(a) Performance Error (b) Scaling Power Effects

Figure 41: VirtualPower Soft Scaling Characteristics.

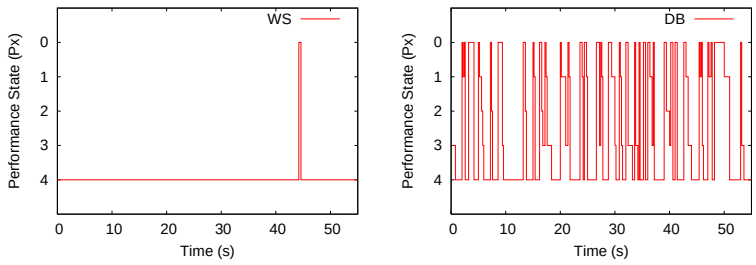
In Figures 41(a) and 41(b), we compare VPM rules that use 3.2GHz versus 2.8GHz as physical frequencies for realizing the different VPM states less than 3.2GHz. In these measurements, rules define VPM states so as to attain linear performance reductions based on frequency for computation-bound workloads. In other words, the 1.6GHz VPM state should provide half the performance of the 3.2GHz state. Figure 41(a) shows how actual measured performance degradations are within a few percent of desired goals, thereby demonstrating the viability of the approach. Errors are larger for the 2.8GHz physical frequency, but the implementation is designed to forego potential (slight) improvements in power to ensure

'negative' errors, that is, to attain performance slightly better than would be expected from a strictly linear degradation. In Figure 41(b), we observe the power tradeoffs between using only soft scaling versus using soft and hard scaling simultaneously, for a single instance of the computation-bound transactional application. From this figure, the power benefits of soft scaling with either physical frequency are apparent, with increased power reductions of up to 48W when the processor is running at 3.2GHz. Summarizing, these experiments establish (1) the viability of soft scaling and perhaps more importantly, (2) the importance of simultaneously using soft and hard scaling to create the rich set of VPM states desired by guest VMs and their applications.

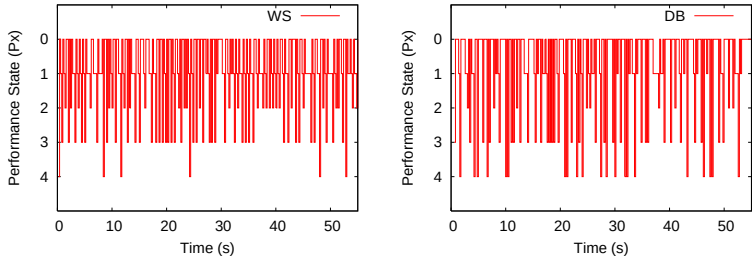
Table 17: VPM State Definitions.

	Hardware	Soft (Scheduler)	
VPM State	CPU	Work-conserving	Slice
3.2GHz	3.2GHz	Yes	-
2.8GHz	2.8GHz	Yes	-
	3.2GHz	No	88ms
2.0GHz	2.8GHz	No	71ms
	3.2GHz	No	63ms
1.6GHz	2.8GHz	No	57ms
	3.2GHz	No	50ms
800MHz	2.8GHz	No	29ms
	3.2GHz	No	25ms

Table 17 lists the parameters used by VPM rules to define VPM states. Each state can be implemented using only soft scaling or with a combined hardware and software scaling approach. Experimentally, we find that the 'slice' definitions for the 3.2GHz mode and for the hardware-scaled version of the 2.8GHz mode are irrelevant because of their use of work-conserving scheduling. For the remaining states, the slices correspond to periods of 100ms. A tradeoff when using both soft and hard scaling is that reduced hardware frequencies imply larger allocation slices for VMs, as found in Table 17. As seen later in our evaluations, the interesting and somewhat counterintuitive outcome of this fact is that for minimizing power consumption across multiple machines, it is sometimes necessary to implement VPM states with higher hardware frequencies so that multiple `vcpus` can be mapped to a single machine (i.e., to permit consolidation for exploiting low idle/sleep power



(a) No Active Power Management



(b) Active Power Management

Figure 42: VPM Channel Feedback for State-Based Guidance with VirtualPower.

or power-efficient hardware). For example, if two VMs are being set to the 1.6GHz VPM state, using soft scaling alone at a hardware frequency of 3.2GHz, they each require 50% of a CPU allowing for consolidation, whereas if hardware scaling is used, the requirement is 57% per VM. Since VirtualPower allows VPM rules to define VPM states based upon hard and soft scaling mechanisms as they see fit, such dynamic, coordinated management is easily realized with our approach.

6.5 Policy Based Coordination

The VirtualPower infrastructure can be used to implement complex management policies. We demonstrate this fact with hierarchical power management policies [29]. These are comprised of local policies (PM-L) running on each physical system and driven by local guest VM and platform parameters, coordinated by global policies (PM-G) with knowledge about rack- or blade-level characteristics and requirements. Specifically, PM-L policies obtain VMs' desired power states via VPM channels and use state-based guidance to determine suitable local shadow VPM states. The higher level PM-G policies use the 'consolidation' VPM mechanism based on information about the shadow VPM states assigned to VMs by

PM-L policies and the platforms on which they run (e.g., rack or blade enclosures). One objective of these experiments is to evaluate VirtualPower's ability to run diverse rules with different and varying goals while exercising the multiple VPM mechanisms presented to these rules. Goals sought by the VPM rules policies demonstrated include power minimization and power throttling/limiting, the latter designed to enforce limits on blade- or rack-level power draws, perhaps triggered by thermal events. Another objective is to demonstrate, experimentally, the importance of coordination: (1) across the different levels of abstraction at which VM-level vs. virtualization layer policies run, and (2) across policies running on different platforms.

6.5.1 PM-L Policies: Platform Management

The VirtualPower approach establishes an indirect feedback mechanism that permits power management at the virtualization level to take into account the desires of VM specific management policies using state-based guidance. The existence of this desirable feedback loop is established with traces of power management requests from the web server (WS) and database (DB) VMs for our RUBiS workload in Figure 42. Here, we denote VMs' requests in terms of desired Px states, where P0 is 3.2GHz, P1 is 2.8GHz, P2 is 2.0GHz,... *etc.* Figure 42(a) shows the requests made by the two VMs without active VirtualPower management. That is, VPM rules are disabled and do not perform any hard or soft resource scaling. When active power management is enabled and PM-L policies are run, a dramatic change is seen in the request patterns of both VMs in Figure 42(b). This demonstrates the interplay between active management in the virtualization layer with the internal policies run by guest VMs. In other words, the Figure visualizes performance feedback data communicated from VMs to the virtualization layer power management. We next evaluate how this indirect feedback loop can be used for effective, coordinated platform- and system-level power management.

A simple virtualization-level policy is one that attempts to minimize power consumption while maintaining the desired performance of an application, the latter expressed via the VM's power management requests. We term this policy PM-L$_{min}$. An alternative useful

local policy is one that throttles power consumption for certain periods of time, perhaps due to thermal events or transient power delivery issues [13, 16]. PM-L$_{throttle}$ is a power throttling policy that attempts to maintain average power consumption at some desired level. Finally, to address changes in request behavior experienced for the transactional loads explained earlier, a third useful local policy, termed PM-L$_{plan}$, is one that trades off potential short term reductions in power usage with longer term gains enabled by planning based on monitoring historical trends in power management request behavior. We next experiment with these policies.

Minimizing Power in the Presence of QoS Constraints A common objective for power management policies is to minimize execution time power consumption while maintaining the quality of service (QoS) required by applications. As outlined in Section 6.2, VirtualPower attains this goal by observing and reacting to guest VMs' management requests. Using combined hard and soft scaling to export a set of five virtual frequency states to guest VMs, the experimental results shown here demonstrate that the PM-L$_{min}$ VPM rule successfully assigns suitable shadow VPM states to meet a VM's QoS goals while also reducing its power consumption. Power management behavior with this policy is depicted in Figure 43, where the straightforward 'mapping' version of the PM-L$_{min}$ policy reacts to a change request by a VM by simply making the appropriate change to some corresponding shadow state.

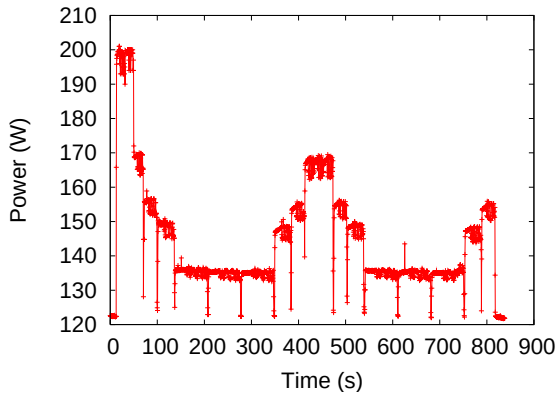
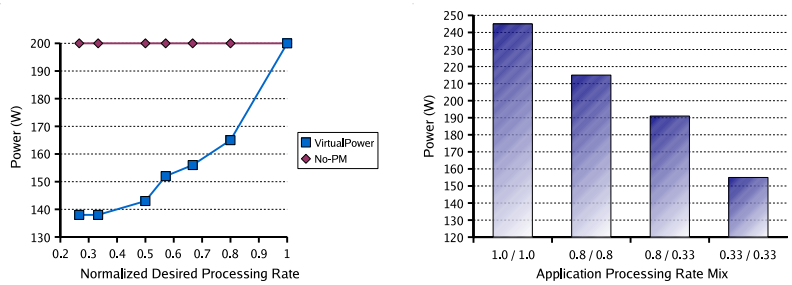


Figure 43: Power Trace of Transactional Application.

The power trace in Figure 43 measured for the transactional application is obtained from an experiment in which the VPM state for the application is initialized to its maximum (or 3.2GHz). The VM-level policy determines whether or not a scaling request should be made between transaction executions. These decision points can be seen in the figure where the power consumption drops for a very small period of time. We observe that over time, the VM issues several frequency reduction requests since there is slack between its performance and the necessary computation power to maintain its required processing rate. When such slack is depleted, the system scales the computational resources back up to an appropriate performance level, in response to subsequent VM-issued management requests. Via this feedback loop, the PM-L policy is able to provide desired levels of performance for the transactional VM across all experiments run in this series.



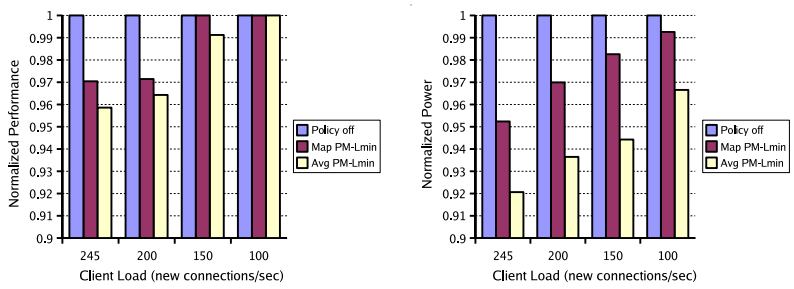
(a) Single Transaction VM (b) Two Transaction VMs

Figure 44: Transactional Application Power Results.

An analysis of the power reduction capabilities of the 'mapping' PM-L$_{min}$ policy used with transactional applications is shown in Figure 44. The effectiveness of this policy is measured with varying rate requirements, where for presentation purposes, the processing rate values are normalized by the rate that requires the highest (i.e., 3.2GHz) power state on our Pentium 4 machine. Figure 44(a) depicts the measured average power consumption of a single instance of the transactional VM. Also shown are comparative results attained with a non-VirtualPower-enabled system, where the guest VM's performance state is always set to the maximum. These measurements establish that our approach can capture and then effectively use VM-specific power/performance tradeoffs via its VPM channels and

mechanisms. In fact, with VirtualPower, we see power benefits of up to 31% for transactional VMs that require reduced performance (i.e., they execute transactions with low QoS requirements). The diminishing returns seen with decreased desired rate/throughput are because the soft scaling VPM mechanism reduces active power consumption linearly, but is limited by the idle power consumption of the platform.

Figure 44(b) depicts measurements with two backend transactional VMs that both run on the same dual core platform. The purpose is to understand whether and to what extent simple policies like 'mapping' PM-L$_{min}$ can reduce power consumption in the presence of multiple VMs with possibly different individual performance requirements. With VM-level performance requirements expressed as 'rate mixes' in the figure, it is apparent that with performance slack, the mapping PM-L policy again successfully reduces execution time power consumption. Moreover, the policy recognizes and successfully exploits the differences in VM-level performance needs.

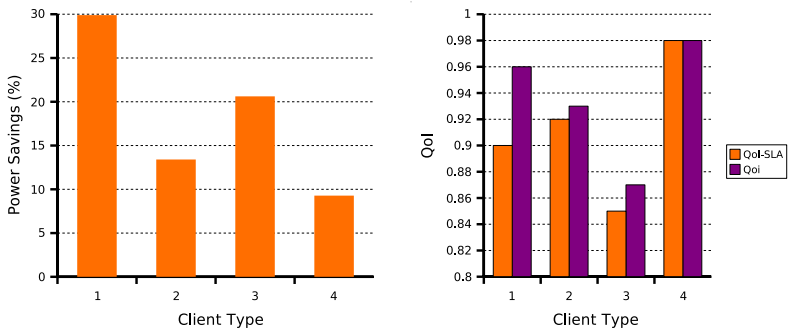


(a) Performance (b) Power Consumption

Figure 45: RUBiS Experimental Results.

Realistic web service applications have behaviors that are more complex than those of the transactional codes we have evaluated so far. To experiment with these, we use the tiered RUBiS application, running its web server (WS) and database (DB) virtual machines on separate physical machines, with a PM-L policy executing in each `Dom0`. Again using the 'mapping' PM-L$_{min}$ policy, we measure the resulting performance, in terms of requests served per second, and power consumption for various client loads in Figure 45 (each load class normalized by comparative measurement of the "policy off" case). Interestingly, while this policy is able to limit total performance impact to within 5% across all loads, its power

savings aren't appreciably better than the performance degradation being experienced. This is because the *ondemand* governor used by the RUBiS VMs is extremely reactive, as shown in Figure 42. A simple 'mapping' PM-L policy that carries out each of these requests can be inefficient, because it is unable to capitalize on periods of low utilization. A better policy for applications like these is an averaging "Avg" PM-L_{min} policy, which smooths the VPM state request pattern observed from the RUBiS VMs. Our implementation of this policy polls for VM power management requests received on VPM channels, once per second from `Dom0`. If a VM has made any requests, an averaging function is used along with the current VPM state of the VM to determine a new suitable state. Otherwise, if the shadow VPM state of the VM does not match the last received request, the VPM state is shifted towards the last request by one performance state. Figure 45 shows that this policy provides similar performance characteristics to the mapping approach, but reduces power consumption further, by up to 4%.



(a) Power Savings (b) SLA Results

Figure 46: Managing Workloads with QoI Performance Metric.

As a final power minimization example, we consider our Nutch based application which is driven by the semantically rich QoI performance metric. Figure 46 provides results with this application VM. We run the workload with various client types that have different QoI and latency requirements. We see from Figure 46(b) that across all the client loads the system is successfully able to maintain the QoI SLA (the latency requirement was also held across all loads). This is a very powerful result, as at the virtualization layer the system is completely unaware of the nature of the QoI performance metric, and cannot track it

by any means. It is simply by using the state based guidance approach as feedback that the VirtualPower system is able to maintain the SLA, while achieving savings of up to 30% as shown in Figure 46(a). Overall, we conclude that our VirtualPower infrastructure allows power minimization policies to effectively reduce power within application specific performance requirements for all of our experimental workloads.

Power Throttling with VPM Mechanisms In modern datacenters, there is a need for policies that can limit the average power consumption of a system, perhaps in response to some temporary event like a reduction in cooling or power delivery capabilities, or in order to adhere to a global power provisioning strategy [28]. Ongoing industry research is developing system support for power consumption monitoring that can be used as feedback for such policies [60, 97]. Unfortunately, current platforms like our dual core Pentium 4 systems have limited performance states for this type of management. Soft scaling corrects this deficiency, and it makes it possible to construct policies like PM-L$_{throttle}$, which we describe and evaluate next.

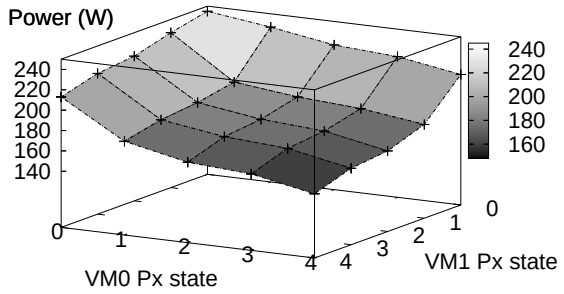
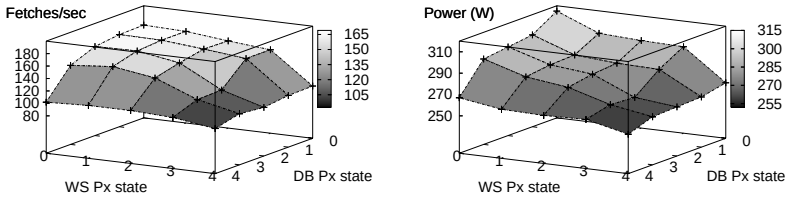


Figure 47: Transactional Workload Management.

Consider a single dual core platform executing two transactional VMs. Each of these VMs can be assigned a static Px state independently, by setting its shadow VPM state in the hypervisor. Figure 47 provides the associated power consumption trends. An interesting observation from this figure is that there is a significant power reduction when both VMs are set to VPM states of P1 or less, compared to either of them running at P0. Particularly,

the power consumption of both VMs at P1 is 8% less than with one VM at P0 and the other at P4. This is because for our platform, voltage scaling can only be performed on the dual core package if both cores are scaled down. A more general observation from this figure is that soft scaling substantially improves power limiting capabilities: hardware scaling alone provides up to 20% throttling capability, while soft scaling extends to up to 40%. *These results again highlight the need to utilize both hard and soft scaling in conjunction for effective management.*



(a) Performance (b) Power Consumption

Figure 48: RUBiS Management.

Next, we again consider the RUBiS setup with two machines in Figure 48. If the power of both machines is limited simultaneously such as in a rack or enclosure setting, from the figure, we see that with RUBiS, power savings scale with performance degradation with the first few VPM states (P0-P2). Within this range, an up to 9% performance degradation can be traded for a 10% power reduction. If PM-L$_{throttle}$ needs to limit power beyond this, then performance drops by as much as 45% for an up to 20% reduction in power. This nonlinearity exists because for distributed applications like RUBiS, their end-to-end performance metrics can be significantly affected by local performance degradations. These experiments demonstrate, for a single application, the effects of coordinated, multi-machine power management.

Our final measurements in this set of experiments go beyond coordinating across multiple machines to also demonstrate the importance of coordination across multiple VM policies, specifically, in order to attain some global goal, such as limiting a server's total power consumption to about 180W. Consider two guests VM0 and VM1 with different

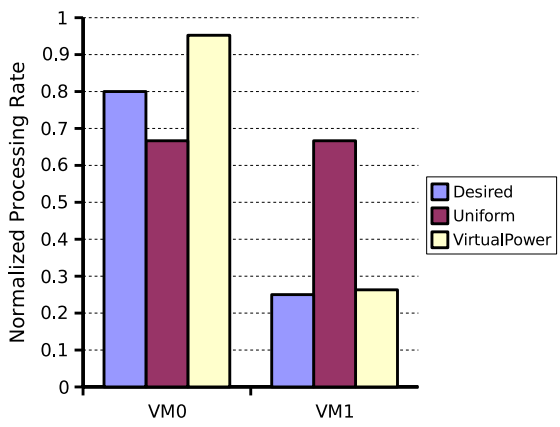


Figure 49: Non-uniform VPM State Allocation.

power/performance tradeoffs, expressed by the commands they issue to their respective VPM channels. A "fair", application unaware approach to throttling the machine would divide the power budget uniformly and execute both VMs at the P2 VPM state. On the other hand, if VM0 needs to process transactions at a normalized rate of 0.8 while VM1 requires only a rate of 0.25, then when executing at P2, VM0 will try to upscale its frequency, since its performance is too low to maintain its desired transaction rate. At the same time, VM1 will attempt to down-scale its processor. With the workload-awareness enabled by VirtualPower, a coordination policy can exploit information about VM desires to determine that the first VM can run at P1, while the second can run at P4. Although both assignments observe the 180W power limit, Figure 49 shows how the non-uniform VirtualPower allocation provides adequate performance for both VMs, while a uniform allocation overprovisions VM1 and causes VM0 to underperform.

VirtualPower Enabled Power Planning As a final example of a PM-L policy that can benefit from the VirtualPower system, we briefly consider how power management requests observed from VMs can be used to create power management plans based on predicted behavior. For example, an observation about the traces from Delta Air Line's enterprise subsystem is that a large batch of requests arrives for processing at the same time every day. By tracking power management requests associated with these events, a PM-L_{plan}

policy can make more appropriate decisions than other policies. To highlight how this type of policy is enabled by the VPM architecture, we next describe a PM-L$_{plan}$ that attempts to throttle power consumption to 180W on a machine over some time period.

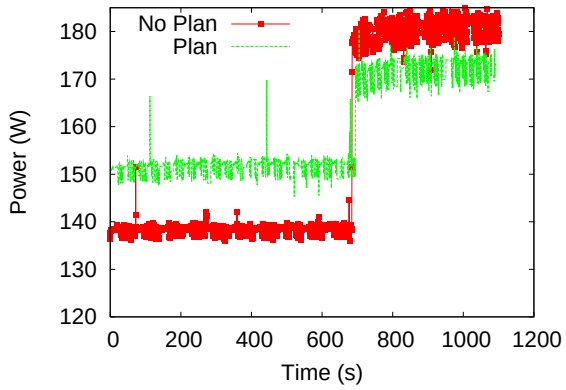


Figure 50: Power Tradeoffs of a Planning Policy.

Consider the 'mapping' PM-L$_{min}$ policy explained earlier. This policy is 'aggressive' in that each VM is treated independently and scaled down as much as possible. A global planning policy, however, may wish to limit scaledown in order to complete all existing transactions prior to the large batch arrival, particularly when multiple VMs are involved. Figure 50 depicts measurements of an illustrative example in which a large batch job arrives after 700 seconds. The "no plan" approach illustrates what happens if the minimal VPM state corresponding to each VM's currently desired QoS is assigned to each machine. With this approach, initial power consumption is low, just under 140W. In order to provide adequate performance when the second VM receives requests, power exceeds the desired throttling point. Therefore, the policy would have to reduce the VPM state of one or both VMs, thereby violating their performance requirements. On the other hand, using past VM request behavior, a planning policy can predict the future high demand experienced by the second VM, and preemptively provide additional performance to the first VM, so that it may complete its transactions. In Figure 50, the "plan" results show that the throttling point is maintained using this approach. This illustration underlines that infrastructures like VirtualPower must be able to run diverse and rich policies, including those that require past observation and/or use predictive techniques.

6.5.2 PM-G Policies: Global Coordination

Coordination across multiple local policies has already been recognized as an important element of any management infrastructure. A specifically interesting case for power management is that global policies can exploit the ever-increasing efficiency in idle, standby, and sleep power sought and attained by hardware vendors, by using VirtualPower's consolidation mechanism. It is a well-known concept, of course, that consolidation can be used to minimize the number of currently active resources so that standby and sleep states can be used. The new capability provided by VirtualPower is that aggressive soft scheduling based on observed VM-level management requests can be used to substantially improve consolidation-based power management. In addition, we show that a PM-G policy that combines consolidation with migration can perform power efficient allocations in heterogeneous environments.

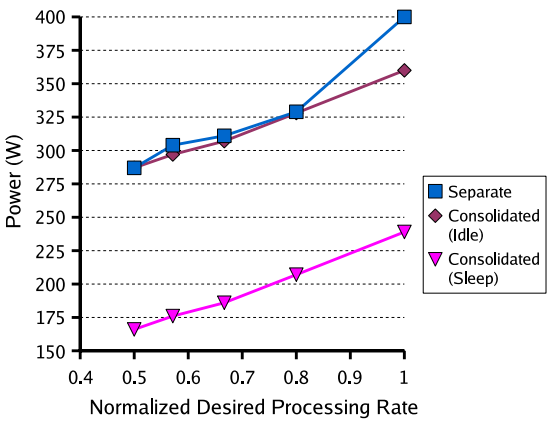


Figure 51: Consolidation with Sleep States.

Soft Scale-Enabled Consolidation To understand the benefits of consolidation, particularly in multicore machines, consider Figure 51. The figure provides power measurements of two Pentium 4 machines when there are two transactional VMs, each with the same desired processing rate. Both machines run the mapping PM-L$_{min}$ policy described earlier. The figure compares the measured power consumption of the two machines when the VMs are on separate physical machines vs. being consolidated onto one multicore machine vs.

the projected power consumption when there is consolidation and when the idle platform is shut down or placed into a low power sleep/standby state (which we assume consumes close to zero power). We see from the figure that at high processing rates (i.e., high VPM states), there is an up to 10% power improvement derived from consolidation. This is due to the power characteristics of the multicore chips where when one core is being highly utilized, from a global power perspective, it is better to use the second core on the same chip. At lower request rates, however, we see that the only significant power reduction stems from the act of placing the unused resources, in this case a machine, into a low power/off state.

A VirtualPower-enabled system can aggressively exploit the power characteristics of multicore chips. Specifically, by monitoring lower level PM-L policies, a PM-G policy can determine whether and when soft scaled VMs should be consolidated. In our implementation, PM-L policies spawn communication threads that listen for information requests from PM-G policies. Upon a request, PM-L policies respond with a list of guest domains currently executing and the shadow VPM states used to run them. The PM-G policy can then act based upon its knowledge about the use of shadow VPM states on some set of machines. Given the definition of our VPM states, of particular interest are VMs that run at states P3 and P4 for long durations. An illustrative example is one in which four transactional VMs with normalized desired processing rates of 0.4 are deployed onto two physical machines. Table 18 provides the power consumption of these machines under various configurations.

Table 18: Managing Two Machines with Four VMs.

Configuration	Power (W)
Two VMs per machine (no PM)	490W
Two VMs per machine (PM-L$_{min}$)	318W
Consolidated with 1 idle machine	362W
Consolidated with 1 sleep/off machine	242W

The scenario illustrates several interesting facts. First, when consolidating all four VMs onto one machine, these machines are set to run at P3, or the 1.6GHz VPM state. The implementation of this state reduces the sizes of VM slices by setting the physical frequency of the two cores to 3.2GHz (rather than the lower 2.8GHz). This implementation of the VPM state consumes more power, but it also allows for improved power consumption

from a global perspective (i.e., due to consolidation and the associated efficient use of sleep power). Indeed, we see from Table 18 that power consumption increases under consolidation, compared to two active machines managed by PM-L$_{min}$ policies, until the idle machine is placed into a sleep mode which provides a 24% reduction in power consumption across both machines.

Exploiting Heterogeneity in Power Management Upgrade cycles and cost considerations are causing increased levels of platform heterogeneity in datacenter environments. Even within a platform and processor generation, there may be differences between components. In particular, due to fabrication issues, it is becoming necessary to bin processors into different groups based on their frequency and management capabilities. Therefore, our next experimental scenario considers the heterogeneity case in which there are some machines that support hardware scaling, while others do not. In our testbed, this simply involves treating some of our Pentium machines as non-scalable. Figure 52 shows how this fact can affect the power consumption of one of our transactional VMs. With varying processing rates, we see from the figure that the benefits of hardware scaling support can be significant, up to 8%, especially for application VMs that can be executed at reduced VPM states.

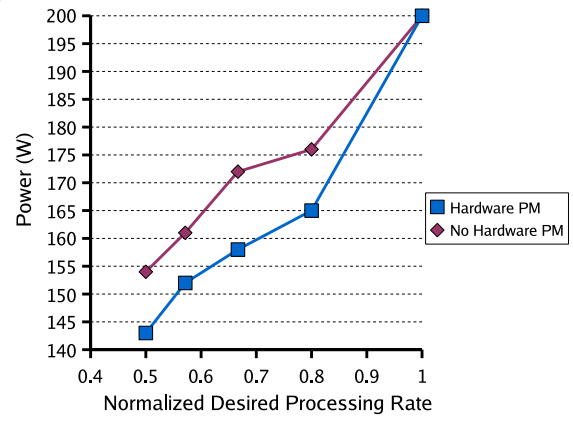


Figure 52: Power Management Heterogeneity.

VM consolidation and migration are important methods for dealing with PM heterogeneity. To illustrate, consider three transactional VMs with normalized performance requirements of 0.4 running on a single machine. On a platform with hardware scaling and a PM-L$_{min}$ policy, these workloads consume 180W, but they consume 218W on a platform without hardware management support. This constitutes a 17% reduction in power. It is clear therefore, that in the context of VirtualPower-enabled consolidation, PM-G policies can and should be designed to intelligently leverage hardware manageable components whenever permitted by VMs' performance requirements.

Resource Heterogeneity Exploitation with VPM States The final VirtualPower results concern aggressive consolidation to exploit heterogeneity in the power and performance tradeoffs of datacenter platforms. In particular, we include in our experimental setup a dual core platform using Intel's new Core microarchitecture, and we enable live VM migration as one of the VPM mechanisms offered to VPM rules. In this setup and for the transactional VMs used in our work, migration time is measured to be, on average, 8.5 seconds when the machine is executing two VMs, and 4.4 seconds when there is only one VM.

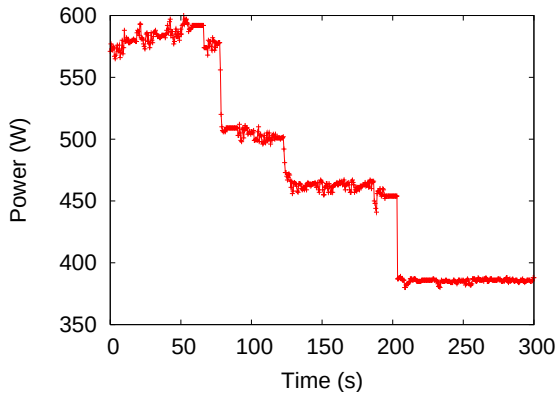


Figure 53: Consolidation with Heterogeneity.

The experiment deploys four transactional VMs with normalized rate requirements of 1.0, two on each dual core Pentium 4 machine. All machines run a PM-L$_{min}$ policy, and there is a PM-G policy whose goal is to maximize usage of the Core microarchitecture-based

system since it is quite power efficient and also provides significant performance. In fact, it consumes around 140W when running both cores as compared to the 240W used by the Pentium based machines. Moreover, it can execute transactions at roughly 2.4 times the rate of the older machines.

The PM-G policy begins by offloading two VMs from one of the P4 platforms and onto the Core microarchitecture-based platform. Due to the improved performance experienced by the VMs after migration, they request reduced performance states. The ability to obtain these hints transparently across these heterogeneous platforms highlights the benefits of our VPM state abstraction for VM policies. Over time, the PM-G policy detects that the PM-L_{min} policy running on the Core microarchitecture machine has reduced allocations to the existing VMs, freeing up room for additional consolidation. As shown by the experiment's power trace in Figure 53, this causes a chain of migrations. Here, the Pentium machines are not powered down or placed into sleep states, but instead, continue to consume idle power. Nonetheless, we observe power savings of almost 200W, or 34%, by allowing for extensive use of the more power-efficient resources when all four VMs are finally migrated. This experimental scenario highlights the ability of the VirtualPower infrastructure to support complex policy strategies that leverage heterogeneity, as well as the ability of its VPM channels and VPM states to provide seamless coordination across systems and migration activities.

6.6 Related Work

Managing power and thermal issues has been addressed extensively for single platforms, particularly their processor components. Brooks and Martinosi propose mechanisms to enforce thermal thresholds on the processor [13]. For memory-intensive workloads, dynamic voltage and frequency scaling (DVFS) during memory-bound phases of execution has been shown to provide power savings with minimal performance impact. Solutions based on this premise include hardware-based methods [61] and OS-level techniques that set processor modes based on predicted application behavior [43]. Power budgeting of SMP systems

with a performance loss minimization objective has also been implemented via CPU throttling [52]. VirtualPower leverages the investments in application-specific power management represented by these approaches by encouraging their use, rather than replacing them, in virtualized systems.

In high performance settings, energy savings can be obtained for parallel programs during their communication periods using hardware frequency scaling capabilities in clusters [65]. For server systems, Chase *et al.* discuss how to reduce power consumption in datacenters by turning servers on and off based on demand [16]. Identifying the need for disk power management in datacenters, support for energy-efficient disk arrays has been extended [122]. Incorporating temperature awareness into workload placement in datacenters was proposed by Moore *et al.* [79], along with emulation environments for investigating the thermal implications of power management [38]. Researchers have also investigated using such cluster-level reconfiguration in conjunction with DVFS [26], or by using spare servers [96] to improve the thermal and power properties of enterprise systems. Other methods have enforced power budgets by allocating power in a non-uniform manner across nodes [29], at the granularity of blade enclosures [97], or even amongst virtual machines using energy accounting capabilities [104]. The experimental evaluations performed in this chapter show that global management policies like those above are easily realized with appropriate VPM rules based on VirtualPower's rich set of power control mechanisms.

In datacenter environments, an important attribute of online power management is its ability to deal with hardware heterogeneity. This is evident from prior work on management extensions for heterogeneous multicore architectures [55, 56] and in cluster environments, the latter proposing a scheduling approach for power control of processors with varying fixed frequencies and voltages [34]. In enterprise systems, intelligent methods for request distribution have leveraged hardware heterogeneity to curtail power usage [39]. The ability to exploit heterogeneity in platform hardware and underlying power management capabilities for datacenters has also been investigated [82]. Experiments presented in this chapter demonstrate the importance of support for heterogeneity in the VirtualPower infrastructure and provide further evidence of the utility of heterogeneity awareness in power management

techniques.

6.7 Summary

Virtualization solutions are becoming increasingly prevalent in modern datacenters due to the intrinsic manageability and consolidation benefits they can provide. It is therefore necessary to provision such systems to allow for effective power management polices and methods to be deployed. We propose a set of coordination extensions as part of our VirtualPower architecture that allow the virtualization layer policies to perform coordinated management decisions based upon application specific policies integrated into guest VMs. Our evaluation of VirtualPower shows that the combination of our VPM states, channel, and mechanisms provide for an effective and flexible means of managing virtualized systems. Results include power savings of up to 34% across multiple platforms, as well as the ability to manage workloads with sophisticated performance metrics such as QoI.

CHAPTER VII

VIRTUAL MACHINE-AWARE POWER BUDGETING WITH VPM TOKENS

As we have previously discussed, power budgeting is important for several reasons. Transient environmental issues may raise the need for managing thermal events [79]. The ability to strictly limit the power consumption of certain physical resources is a key element of response scenarios. The ability to budget resources can also be used to provision datacenters more effectively [28]. By ensuring power budgets are met in sets of machines during transient peaks of usage, additional resources can be brought online within a given power envelope. As a result, there has been substantial previous work on power budgeting, exploiting hardware management capabilities like processor frequency scaling [29, 97], or employing platform-level controllers for fine grain power limiting [60]. In this chapter, we extend this state of the art by providing system-level support for power budgeting. In particular, we introduce mechanisms to enforce power budgets across a distributed set of physical resources, virtual machines, and the applications using them. This support is comprised of a set of managers that coordinate their actions via *Virtual Power Management (VPM) tokens*. Before explaining this in more detail, we first describe the technical issues managers and their coordinated actions must address:

1. Differential treatment of VMs – power budgeting limits activity on physical resources. In virtualized systems, however, such resources may be shared by multiple application VMs. *Since enforcement of power budgets may reduce performance capabilities causing VMs to experience varying levels of degradation and associated impact to applications' service level agreements (SLAs), it must be possible to perform differential treatment of VMs to maximize datacenter level utility [109].*

2. VM-aware management – VMs encapsulate distinct applications or application components, and therefore exhibit different power and performance tradeoffs. For many

workloads, depending upon load characteristics, a reduced power allocation may result in minimal application level performance degradation. *VM awareness, that is, the runtime determination of the power-performance tradeoffs experienced by VMs, can substantially improve overall performance under power budgets.*

3. Budget compensations – applications with varying performance demands may sometimes use less than an alloted amount of power, but require more than their limits in the future. *Management layer abstractions should allow for such applications to be "compensated" to allow for fair budgeting across time.*

4. Resource heterogeneity – datacenter resources are naturally heterogeneous in terms of performance, power, and management capabilities [82]. Therefore, the relationship between discrete power allotments and performance reduction on a given platform varies. *Datacenter level power budgeting policies should be able to perform control decisions in a manner that is conscious of platform heterogeneity.*

This chapter describes extensions to the VirtualPower system that directly address the four technical issues raised above. These extensions are comprised of a set of system managers performing management tasks coordinated via *VPM tokens*. Managers include budget managers, platform managers, and a utility manager. *Budget managers* are charged with maintaining some aggregate power budget on a set of physical platforms. This is accomplished by using VPM tokens to communicate constraints to underlying *platform managers* that observe budgets by locally managing hardware performance states and allocations of physical resources to VMs. The *utility manager* is responsible for determining datacenter utility tradeoffs of different VMs based on static and dynamic criteria, and for communicating them to the other managers so that they may perform management decisions that minimize negative utility impacts of power budgeting. *VPM tokens* are used for all such communications, where *budget tokens* are used for transactions between budget managers and platform managers, *VM tokens* are used by the utility manager to convey VM tradeoffs to the other two system managers, and *compensation tokens* are used by both budget and

platform managers to accommodate workloads with varying demands.

7.1 Motivation

The need for power budgeting in datacenters arises for multiple reasons. First, there could be power delivery or cooling events that dictate the temporary reduction of power consumption in some physical set of resources. For example, downtime to cooling chillers or computer room air conditioning (CRAC) units can create an emergency situation where power consumption must be lowered on some set of machines or racks in order to adhere to reduced cooling capabilities. Another catalyst for power budgeting can be the use of power provisioning strategies that allow for increased resources to be brought online within cooling and power envelopes so long as power budgets are enforced when machines consume peak load [28]. This type of provisioning strategy can be particularly useful in the context of heterogeneous resources, where the benefits of bringing online additional platforms with enhanced or workload-specific capabilities outweighs the overheads implied by maintaining power budgets.

In general, power budgeting consists of a simple control relationship between a manager and a set of underlying physical resources, typically managed at the granularity of platforms. The job of the manager is to ensure that the aggregate power consumption of the platforms does not exceed some specified power budget, or that it adheres to some average over time. The actual constraints maintained by the manager are, in turn, a function of the events or provisioning strategies in play in the datacenter. The goal of our research is to extend the capabilities of this control relationship to better address the needs of virtualized enterprise systems.

A particular issue for virtualized datacenter systems is that they typically simultaneously execute a wide range of application VMs, each with different performance requirements. For example, there may be low priority batch jobs that need to be completed, but have minimal performance requirements because they have no deadlines. Other jobs, such as processing payroll transactions, may have soft deadline constraints. Server applications like web services may have varying constraints based upon load characteristics or the priorities

of incoming requests. The ability to simultaneously manage these different workload VMs is a key requirement of datacenter power budgeting solutions. In particular, we next discuss four technical issues that arise in these scenarios in more detail.

7.1.1 VM-Centric Budgeting

Existing and proposed solutions for power budgeting distribute power allocations amongst physical resources by dividing them according to some static distribution criteria (e.g., equally), or dynamically based upon utilization [97]. The problem with such an approach is that in virtualized environments, particularly in the context of VM migrations, there is no clear mapping between budget allocations to physical resources and the VMs that utilize the managed resources. Since budgeting of physical resources directly impacts the performance of associated VMs, there is a need to incorporate awareness of virtual machines when dividing power budgets amongst resources. Toward this end, we propose the use of *proportional allocation* methods, wherein power budgets are distributed based upon the relative utility of VMs that run on platforms. Namely, since power budgets can result in performance degradations relative to application SLAs, a proportional allocation determines the relative, not the absolute, performance degradations that should be observed between VMs.

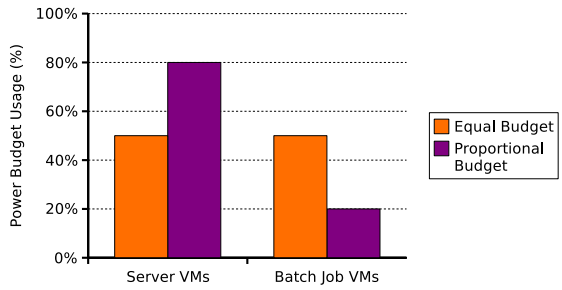


Figure 54: Proportional Budget Allocation.

As an illustrative example, consider two servers that are completely utilized by VMs and are being budgeted by a manager. One machine is mapped to VMs running critical server applications, while the other is running guest VMs that execute low priority batch jobs. Without knowledge of the tradeoffs of the VMs running on the two systems, an unaware

budgeting manager would simply split an allocation equally amongst the two machines as shown in Figure 54. By being aware of the utility of the different VMs, however, a VM-centric budgeter can determine a proportional allocation of power between the two machines. In our example, such a budgeter determines a proportional allocation of 4:1, resulting in the more intelligent non-uniform power allocation shown in Figure 54. The outcome is that a VM-centric budgeter can distribute power allocations in a manner that minimizes impact to datacenter utility by favoring applications whose SLAs are impacted more significantly by reduced budgets.

7.1.2 Application-aware Management

The enforcement of power budgets on platforms can require performance scaling to actively running VMs. Similar to the question of budget allocation amongst platforms, this scenario raises the issue of how to distribute performance scaling across candidate VMs. One criteria for determining allocations is to again use proportional allocations, based on utility information, to divide budgets amongst VMs. In addition, however, we must consider the case where certain VMs may experience minimal performance impact, in terms of SLA, from performance scaling actions, possibly during certain phases of load or execution. This type of information can be obtained online, using the VPM channel and VPM state abstractions provided by the VirtualPower system.

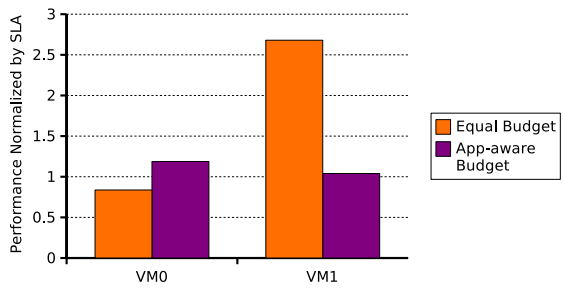


Figure 55: Application-aware VM Budgeting.

The type of application-aware tradeoffs we consider are illustrated by the experimental example in Figure 55. In this scenario, there are two VMs, VM0 and VM1, running on a system. Both VMs have the same importance, or utility. Therefore, the proportional

allocation between them is equal. We see from Figure 55 that VM1 is overprovisioned with the equal proportional allocation. Using VirtualPower's existing mechanisms, the system can dynamically determine that VM1 can currently be scaled down further without violating its SLA. Or, in other words, there is *budget slack* with VM1 that can be used towards VM0. With this application-awareness, then, the platform can recognize that an unequal scaling between VMs can maintain the power budget, while still providing adequate performance to both VMs, as shown in Figure 55.

7.1.3 Compensating Budgeted Applications

Application-aware management allows system managers to shift budgets between VMs or platforms when there is "budget slack". An interesting issue, then, is how to fairly allocate such slack amongst VMs, especially when some VMs create slack by explicitly requesting reduced power allocations (e.g., VMs that use the Linux on-demand governor). For such VMs, it would be beneficial to provide some ability to reclaim "lost" power allocation, by giving them priority when other applications provide slack in the future and when their need for power exceeds their proportional budget allocation. This idea of compensation is similar to that proposed in prior scheduling solutions [108], but we perform compensation in units of power as opposed to time.

For example, as part of an ongoing collaboration with one of our corporate partners, Delta, we have obtained workload traces from a backend subsystem responsible for tracking operational revenue earned from the services offered by the company [2]. The backend operates by continually receiving, queuing, processing, and then, emitting revenue related events, with current incoming event rates at the level of thousands per hour. An interesting observation from these traces is that arrival rates vary significantly over the course of a day, including a large number of events submitted every day at 4 AM EST. With workloads such as these, there are clearly periods when the VM can be budgeted more aggressively, such as when there is less pressure on queues or when lower priority events are being received. There are other periods where additional budget would be useful due to an increased influx of events or high priority requests. Figure 56 provides an illustrative example of the type

of compensation we envision for such workloads.

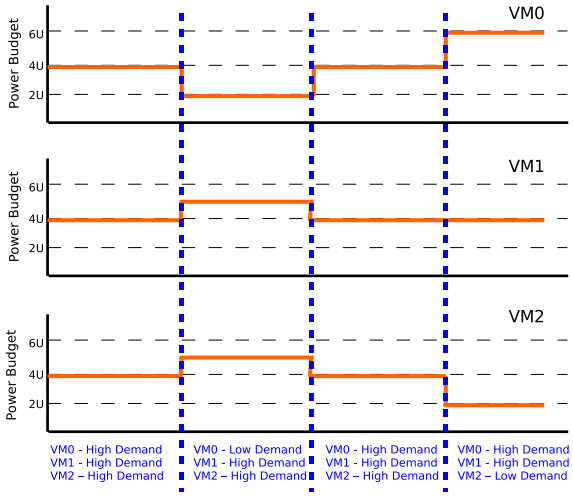


Figure 56: VM Budget Compensation.

In the figure, there are three VMs being managed under a budget of twelve units of power. The workloads are of equal importance, so the proportionally allocated budget is initially divided as four units of power each. The guests exhibit behavior akin to the example above, where there are low demand periods when performance, hence allocations, can be reduced while meeting SLAs, and high demand periods when the proportionally allocated budget is insufficient to meet SLAs. In our example, the workloads either need two units of power, or six units of power depending on the phase of execution. At some point, VM0 hits a phase where it can consume less power. This frees up two units of power, which are allocated evenly amongst the other two VMs that are still in a high demand phase. Eventually, VM0 needs its power back, returning allocations to the initial distribution. When VM2 subsequently hits a low demand phase, instead of allocating the "slack" evenly amongst VM0 and VM1, VM0 is given the entire budget slack as compensation for its earlier underuse of power. This type of behavior provides long term fairness in such scenarios, and our system and approach strive to attain it.

7.1.4 Budgeting Heterogeneous Resources

A natural artifact of modern datacenters is platform heterogeneity. Due to upgrade cycles and continuing incorporation of newly evolving hardware, systems in datacenters often vary in terms of power characteristics and even power management capabilities [82]. This type of heterogeneity will likely increase as multicore platforms become more prevalent and possibly include heterogeneous processing cores [56]. Platform heterogeneity has a direct impact on budgeting managers, especially when performing VM-centric budgeting decisions. In particular, different platforms have varying power efficiencies, or performance capabilities per Watt. Moreover, efficiencies may vary among different workloads based upon dynamic power management (DPM) support provided by platform hardware [82].

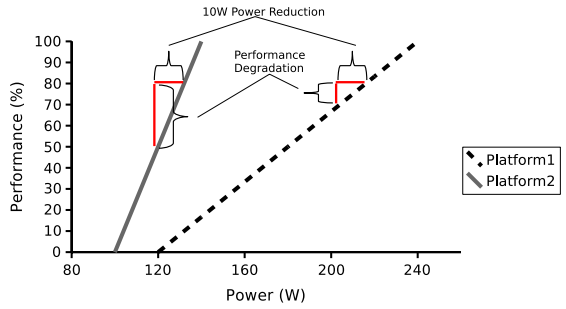


Figure 57: Budgeting Heterogeneous Platforms.

Typical budgeting policies manage power in units of Watts amongst machines. To highlight the effect of this when budgeting heterogeneous systems, we consider an example scenario with two platforms Platform1 and Platform2. For simplicity, we consider a case where performance scales linearly with power for the sets of VMs running on the two machines, as shown in Figure 57 (data based upon actual measurements). We see from the figure that the first system consumes between 120W and 240W from idle to fully utilized, while Platform2 ranges between 100W and 140W. If both systems are providing the same performance to their VMs, but current budget requirements require a 20W reduction in power consumption between the two machines, it is clear that imposing a 10W reduction in power consumption on both machines would result in a significantly higher performance loss on Platform2 as opposed to Platform1. If the VMs that run on the two platforms dictate

a proportionally equal allocation, in terms of performance, across the two systems, this reduction results in a poor outcome. Even if reductions are requested in terms of percentage of current power usage instead of power values to account for different power levels, it would not address varying or nonlinear relationships between power and performance that can result from power supply characteristics or workload specific DPM savings [82]. It is clear, then, that policies that budget heterogeneous resources must use an alternate unit of exchange. The VPM token abstraction and associated currency exchange models provide such an alternative to budgeting policies.

7.2 Power Budgeting Architecture

This section describes our power budgeting architecture for virtualized systems [85]. To provide power budgeting capabilities with the benefits highlighted in Section 7.1, we extend our VirtualPower solution [84] with additional components. In particular, we use a currency abstraction, termed VPM tokens, to coordinate a set of system managers by communicating system requirements between them. Figure 58 provides a high-level overview of how these proposed management components interact in our infrastructure.

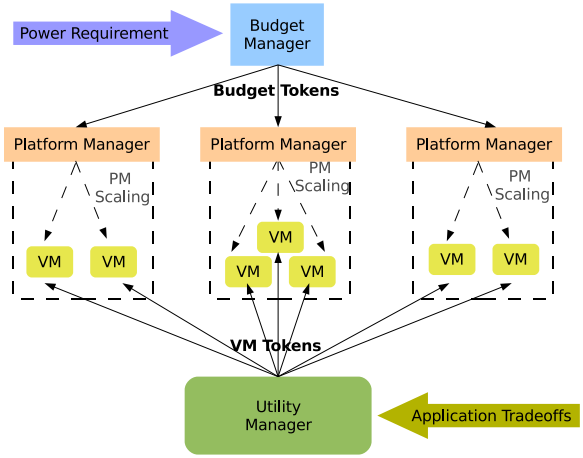


Figure 58: Power Budgeting Architecture.

The system managers include budget managers, platform managers, and a utility manager. There is a budget manager associated with each set of systems that must be budgeted

(e.g., a blade chassis or server rack). Therefore, in the general case, there could be multiple budget managers acting upon disjoint sets of resources in a datacenter. For the other two managers, there is a platform manager associated with each physical system, and we currently use a single utility manager for the entire datacenter.

7.2.1 System Managers

System managers are responsible for decision making about power budgeting. In particular, we define a set of managers that coordinate via VPM tokens to enforce power budgets while considering both VM level performance tradeoffs and effects on datacenter level utility.

Budget Manager The budget manager is charged with accepting a power requirement and imposing it upon underlying platforms. In general, depending upon whether a budget is being imposed for power delivery or cooling reasons, such a manager may need to ensure that peak power delivery does not exceed power requirement, or that average power is kept under the budget with small but limited violations [97]. Our system and its relevant abstractions are agnostic towards the particular budgeting constraint, but for the sake of discussion, we refer to a manager limiting average power consumption across nodes. Any such manager can be easily converted to a peak power manager using a margin factor (*i.e.*, maintain average power equal to budget minus a spike factor), or possibly a statistical means of correlating maximum power consumption based upon an average consumption rate of workloads [97]. In either case, budget managers must decide how to convert an input power criteria into allocations to underlying platform managers, taking into account the tradeoffs between VMs running across systems and the heterogeneity characteristics of hardware. This includes utilizing our notions of proportional allocations, as well as budget slack and compensation techniques in order to manage power budgets while maximizing datacenter utility.

Platform Manager Platform managers provide the base level means of acting upon power budgets in our system. In particular, platform managers determine whether current system configurations, including hardware and soft scaling of resources utilize by VMs,

are appropriate for the power allocation provided by the budget manager. As previously mentioned, our power budgeting solution builds upon our VirtualPower management infrastructure. Towards this end, the platform manager functionality is combined with the PM-L, or local power management policy, entity in the VirtualPower architecture. Therefore, the platform manager is guided not only by power budgets, but also by application specific power/performance tradeoffs communicated using VirtualPower's VPM state and VPM channel abstractions. The task of the platform manager, then, is to determine proportional allocations based upon datacenter utility tradeoffs, while coordinating such allocation decisions using VM power management "hints". In addition, it must realize required compensations, as described in Section 7.1. The precise mechanics of how platform budgets are mapped to scaling of VMs using hardware and soft scaling are discussed in more detail in Section 7.5.

Utility Manager The role of the utility manager is to provide the other system managers with information regarding the relative priorities of VMs deployed in the datacenter when power budgeting must be performed. In other words, the utility manager should communicate what the datacenter level tradeoffs are when power reductions to VMs cause SLA degradations. These tradeoffs could be based upon some static conditions, e.g., VMs that run web services may be relatively three times more important than batch jobs. More interestingly, they may be based on dynamic aspects such as the importance of certain functionality in critical thermal situations. For example, if cooling capacities are drastically reduced, it may be more important to allow database applications to quickly bring the state of systems to a consistent state so that applications can be shutdown cleanly. In addition, there may be trust or security reasons to tradeoff power budget allocations amongst VMs. For example, if other management mechanisms detect that a VM is compromised and behaving poorly (such as a power virus), the utility manager can dictate that other VMs should get prioritized when power budgeting must be performed. As we will describe, VPM tokens are the abstraction used to convey this information.

7.2.2 VPM Tokens

VPM tokens are a unifying abstraction proposed as part of our work. They allow for coordination in control across platforms and VMs, by providing a means to express constraints between system managers. Indeed, they act as a means of currency for management, similar to the use of ticket currencies for scheduling in prior work [108]. Tokens are defined as $< assignee, value >$ tuples, where the assignee may be a platform or VM, depending upon the type of token. In particular, we define three types of VPM tokens for power budgeting, termed budget tokens, VM tokens, and compensation tokens. These are described in more detail next.

Budget Tokens Budget tokens are used to express power allocations from the budget manager to underlying platform managers. One of the goals of our system is to alleviate the budget manager from directly dealing with heterogeneity nuances when provisioning tokens. Therefore, budget token values are not the actual power allocation assigned to platforms, but instead, a value between zero and hundred that has a platform specific value in units of power. In particular, the token values correspond to scaling platform performance from some minimal capabilities to maximum performance. The benefit of using these "normalized" currencies for budget token values is that it allows the budget manager to easily set tradeoffs across heterogeneous systems in a proportional fashion based upon VM utility tradeoffs without having to directly consider heterogeneity aspects. As described in Section 7.3, this also requires the use of token conversion models to map between budget token values and platform specific power numbers.

VM Tokens VM tokens are used by the utility manager to express tradeoffs for budget prioritization between application VMs across the datacenter. In particular, they are provided to budget and platform managers to allow them to determine appropriate proportional allocation schemes. Since tradeoffs between VMs are relative, the values of these tokens are not of particular significance. Instead, the other managers utilize the ratio of token values assigned to VMs when making decisions. Therefore, the utility manager operates

within the constraint that the sum of all VM tokens dispersed is equal to some constant. This allows both budget and platform managers to use the relative values of VM tokens associated with managed VMs to determine proportional allocations.

Compensation Tokens Compensation tokens are used to earmark allowances to either VMs or platforms when they consume less power than their respective allocations, so that they have priority towards using "budget slack" in the future. They are used in a manner such that at some management interval, for each unit of allocation unused, the value of a compensation token assigned to a platform or VM is increased. As we will describe, when there is budget slack available, its allocation is prioritized based upon compensation tokens, which are then reduced in values as the slack is used up.

7.3 Token Currency Exchange

Our budget tokens are used to allow managers to make control decisions using a normalized unit of "currency" thereby alleviating them from having to consider underlying platform heterogeneity aspects. In order to extend this ability, then, budget token currency conversions must be performed by the budget and platform managers that utilize them. We describe these two conversion techniques in this section.

7.3.1 Budget Tokens and Power

The budget manager must take a power value and convert it to a set of budget token values between one and one hundred that designate system performance levels which fit within the budget. The set of tokens must additionally adhere to some proportional allocation scheme that specifies the performance and utility tradeoffs between VMs that run on the systems, as illustrated in Section 7.1.1. To simultaneously address both constraints, we formalize the problem by defining for each platform a token-to-power conversion function $F_i(T)$ that gives a platform's power consumption value as function of a token value. In particular, the performance provided by a platform should scale with the token value, and $F_i(T)$ provides the related power value. This relationship between performance and power need not be linear, and it may even vary based upon workloads or use of hardware management capabilities.

Therefore, we assume that this function is tuned online, possibly even utilizing runtime power monitoring and prediction techniques to determine tradeoffs [82]. Equation 16, then, describes the relationship between a budget being imposed by a manager and the set of N token values for platforms.

$$P_{Budget} \geq \sum_{i=0}^{N-1} P_i = \sum_{i=0}^{N-1} F_i(T_i) \tag{16}$$

Given the budget formulation, the proportional allocation problem can be described by defining T_i to be less than or equal to $\lambda_i T_C$, where λ_i is a *proportional coefficient* whose value is a function of VM tokens. The T_C value is a control token value that is varied by a decision algorithm. As we describe in Section 7.5, by defining a constraint that all λ_i be greater than one, and that T_C must be between zero and 100, a budget manager can quickly determine a set of budget token values that meet a power budget and provide a proportional allocation.

7.3.2 Budget Tokens and VM Management

Once the budget manager has determined an appropriate budget token for a platform, the platform manager is responsible for taking the token value and translating it into constraints for managing local VMs. Similar to the budget manager, the platform manager also has two constraints to which it must adhere. The first is that it must observe the performance level specified by the token value. In doing so, the platform manager must use soft scaling of resources and exploit the underlying hardware based performance states. In addition, the platform manager must divide the prescribed performance amongst VMs in a proportional manner. To achieve these goals, we define a budget token conversion approach for platform managers based upon resource specific management.

Figure 59 provides a high level overview of the token conversion approach used by platform managers. The first step of the conversion is to take the budget token, and utilize a platform performance model to determine a per resource budget which provides the system-level performance dictated. As an example, this could involve determining a utilization limit for the processor component. The platform manager must then ensure,

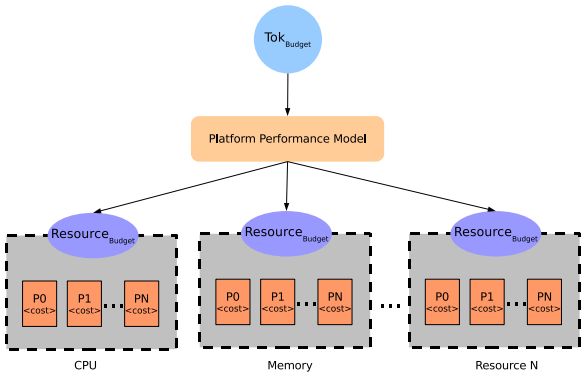


Figure 59: Token Conversion on Platform Managers.

using soft scaling capabilities, that the combined processor utilization across all VMs does not exceed the limit. Given the use of hardware power management, however, complicates this resource limit. Clearly, a reduced performance state on a processor, via a smaller core frequency, allows for a larger utilization while consuming the same power as a higher state. Depending upon the workloads that are being managed, this tradeoff may be worthwhile. Therefore, we enable the platform manager to consider the tradeoffs between different states while maintaining the power budget implied with the budget token, by associating a cost with each possible performance state (or a combination of performance states in the context of resources such as multicore chips). Initial resource budgets are based upon the use of the highest P-state [41], P0, making its cost '1'. Reduced performance states have lower costs, allowing them to "purchase" extended use within the same resource budget. By using these costs, then, the platform manager can convert a budget token into a utilization budget for each resource at each possible performance state. It then simply selects a performance state for each resource, and determines how to proportionally allocate the resource budget at that state amongst VMs using our soft scaling mechanism. This process again utilizes our proportional coefficients λ_i, as described in Section 7.5.

7.4 Experimental Methodology

7.4.1 Platforms and Power Measurements

Our evaluation testbed consists of modern multicore platforms. In particular, for our purposes, we focus on meeting power budgets by managing processors using frequency and

voltage scaling capabilities as well as soft scaling by strictly allocating CPU time to VMs. Our setup contains multiple dual core Pentium 4 machines based upon the Intel Netburst microarchitecture, as well as a dual core platform based upon the new Intel Core microarchitecture to provide heterogeneity. The token-to-power conversion functions for both of these types of systems, using the experimental workloads we describe next, are linear as shown earlier in Figure 57, where Platform1 trends are based upon our Pentium 4 system, and Platform2 results are from our Core platform. The Pentium 4 curve is based upon its highest frequency, or P-state, but to incorporate frequency scaling management, we utilize all P-states supported, which are 3.2GHz and 2.8GHz.

While our experimental results focus on application performance and datacenter utility under power budgets, we utilize actual power measurements to drive the system. Power data for our systems under various workloads are attained with an Extech 380801 power analyzer, which allows for out of band measurements via a laptop, thereby avoiding undesirable measurement effects on the system under test. Power is measured 'at the wall', in order to include the AC power consumption of the entire system. We measure power at the maximum sampling rate of 2Hz using the Extech device to ensure that average power budget constraints are met in our experiments. Our system managers also require the ability to monitor power consumption of platforms in order to detect budget slack. While this capability is increasingly found on modern systems [60, 97], our hardware does not support it. To overcome this limitation, we utilize actual power measurements to create tuned power models as a function of system statistics such as CPU utilization. We then deploy software daemons on each machine that monitor these statistics across VMs as well as the control domain `Dom0` in Xen, and we use the tuned power models to provide system managers with power information.

7.4.2 Experimental Applications

In order to evaluate VPM token based budgeting in realistic scenarios, experiments are conducted with multiple types of workloads. The goals are to effectively manage workloads with different performance tradeoffs, as well as load characteristics. Therefore, we define

three types of workloads based on actual applications and describe how they are realized.

Batch Workloads Certain enterprise batch jobs have few or no performance requirements. That is, they need to be completed, but have no critical deadlines or time frames for completion. Examples of such batch jobs include logging or tracing tools that analyze monitored data as well as the processing of risk analysis models across different inputs common to investment banking. For these types of applications, the results are useful sometime in the future, but are not immediately required. Therefore, these types of applications have very loose SLA requirements. Or, in other words, scaling resources to these types of workloads has nominal impact on application quality of service (QoS) as long as some minimal level of performance is met. We implement this class of workloads using benchmarks from the SPEC CPU2000 suite as code modules to provide load.

Transactional Workloads Many backend services deployed in enterprise systems and datacenters are of a transactional nature. For example, payroll backend systems that process transactions, or our earlier example of the revenue pipeline system in Section 7.1.3. These backend servers must process requests at rates that are sufficiently high to meet application-specific SLAs. SLAs associated with the revenue pipeline are dictated by revenue reporting requirements. In general, different transactional applications will have different performance requirements, and all such requirements may vary over time. The principal performance metric for these types of transactional services is whether they can process transactions at some currently desired minimum processing rate. Therefore, the performance degradation experienced by these workloads is based upon if, and to what extent, processing rates drop below the SLA specified value. For purposes of experimentation, we again use the well-known SPEC CPU2000 suite to create code modules that carry out computationally expensive transactions, and monitor the rate that they can be executed.

Web Service Workloads The third and final workload is a web service application based upon the Nutch open source search engine [88]. In particular, we modify the behavior of Nutch in order to resemble aspects of actual deployed service workloads from one of our

corporate partners, Worldspan. Worldspan provides services to online ticket reservation companies. This includes receiving ticket search requests and responding within some period of time with a set of results corresponding to search criteria. In such a scenario, requests from different clients may be associated with varying SLA requirements, including the quality of results in responses. We incorporate both of these aspects, a differentiated service model and a response quality metric, into Nutch. In particular, we define requests to require a varying number of search responses in order to be completed, resulting in more (or less) work by the search engine. The types of requests include a light request and a heavy request where the heavy request needs twice as many search responses to be found for completion. We define two client classes where the first uses only light requests, while the other uses an even distribution of both light and heavy requests. In addition, the client classes have SLAs based upon an average quality of information (QoI) metric. We define the QoI to be the ratio of search responses found within some processing time constraint and the total responses required for the request type. A QoI below 80% is defined to provide zero QoS. The SLA definitions, then, for the two classes are an average QoI of 90% for the "light" client, and 95% for the "heavy" client. Performance degradation is then defined as the amount of QoI degradation between 80% and the SLA for a client class.

7.5 Implementation and Evaluation

We evaluate VPM token based power budgeting with an implementation using the Xen hypervisor. It should be noted, however, that our architecture and abstractions are designed for the wide range of virtualization solutions currently deployed or under development. The majority of the components are implemented in Xen's driver domain, termed Domain Zero (`Dom0`), with some changes made to the underlying hypervisor for VirtualPower's VPM channel and VPM states [84]. In this section, we provide details regarding the implementation of our system managers, and then move on to experimental results.

7.5.1 Manager Implementations

System managers are implemented as user space daemons running on test machines. The budget and utility managers run on a desktop machine outside of the dual core system

testbed. Platform managers are implemented in `Dom0` of the dual core machines.

Budget Manager The budget manager communicates with the platform and utility managers in order to calculate and distribute appropriate budget tokens to systems that are being budgeted. First, the budget manager queries all underlying platform managers for the names of VMs that are currently active. These names are used to uniquely identify workloads across a datacenter. The budget manager then queries the utility manager for the current VM token values for all the VMs found. These values, along with knowledge of where VMs are running, allow the budget manager to calculate a per platform aggregate utility value, U_i. In our implementation, this is done by simply summing the VM token values associated with a platform. Finally, these are used to calculate the set of proportional coefficients λ_i by finding the platform j with the smallest aggregate utility value, and then setting λ_i to $\frac{U_i}{U_j}$. This method of defining proportional coefficients ensures that the constraint of all $\lambda_i \geq 1$ is met.

Algorithm 4 Proportional Token Allocation.

```
1: T_C=1
2: while ((∑_{i=0}^{N-1} F_i(T_i) ≤ P_Budget) AND (T_C ≤ 100)) do
3:    for all Platforms i do
4:       T_i = T_C λ_i
5:       if T_i > 100 then
6:          T_i=100
7:       end if
8:    end for
9:    T_C++
10: end while
```

Given the set of λ_i values, the budget manager must calculate the budget tokens to assign. Algorithm 4 presents the algorithm used in our current implementation. The goal of the algorithm is to find a set of token values that prioritize power allocation according to utility based upon the λ_i values, which in turn are calculated as proportional aggregate utilities. We note that the token-to-power conversion functions are what enable the actual power throttling for the budget manager using this algorithm. As previously mentioned, we assume that these functions are based on power and performance models that are tuned

online using previously proposed methods including our own prior work [82]. For the purposes for our evaluation in this chapter, we hard code functions that are tuned using offline experiments with our platforms and workloads.

The remaining aspect of the budget manager implementation is its use of compensation tokens. In particular, every time quantum, which we define to be one second, the budget manager compares the current power consumption allocated to each platform based upon its budget token and $F_i(T)$ function against its measured power usage. Since the $F_i(T)$ function is tuned to provide the power consumption at a particular platform performance level, a platform manager will always use at most the budgeted power, but may utilize less power if it determines that reduced system resources can meet workload performance constraints. Upon detecting this "budget slack", the budget manager performs two tasks. First, it increments compensation tokens for platforms that underuse power by $(F_i(T) - P_i)$. Next, the budget slack is distributed among nodes that are currently using their budgeted amount. In particular, if any such systems have compensation tokens with nonzero values, the slack is distributed among them in a proportional manner based upon the values of compensation tokens, decrementing those compensation tokens by the amount received. Otherwise, it is distributed among the systems based upon their proportional coefficient values.

Platform Manager In our implementation, platform managers meet power budgets by managing system processors. This is done by utilizing the soft scaling VPM mechanism implemented with the Xen hypervisor's earliest deadline first (SEDF) scheduler, which allows for dynamic tuning of the amount of processing time allotted to a VM's virtual CPU (`vcpu`). In addition, where applicable, the platform manager utilizes DVFS capabilities. When DVFS is used, we set both cores of our dual core systems to the same frequency.

The platform manager implementation is composed of two parts. The first determines, based upon an assigned budget token, the frequency to utilize, and the maximum allowable aggregate utilization at that performance state. The second part of the platform manager, then, determines how the aggregate utilization should be divided amongst VMs using a

proportional allocation approach.

The budget token value given to a platform describes budget constraints in terms of performance level, not power. This performance level is relative to the highest performance states for systems with power manageable resources. For example, in our implementation a budget token with a value of one hundred means the system is free to utilize both processors at full utilization at their maximum P-states. A smaller token value basically implies a limit on the combined utilization across both cores (e.g., a value of fifty implies combined utilization of 100%). This tradeoff explains why our experimentally obtained token-to-power functions are linear in nature, since with our platforms power scales linearly with utilization. In order to determine what frequency to use on a DVFS capable platform (*i.e.*, our Pentium 4 systems), the platform manager estimates the maximum utilization that can be used at a reduced performance state while still maintaining the power limit imposed by the budget token value. We do this using a simple power model for translation, which we envision would be derived by the platform manager using power monitoring capabilities. In our implementation, the manager chooses the lower frequency between 3.2GHz and 2.8GHz if the utilization budget at 2.8GHz is less than the maximum that could possibly be used by the number of active VMs in a power unconstrained case.

The utilization budget determined by the platform manager is divided amongst VMs using the same algorithm as the budget manager uses to determine proportional allocations, where now the constraint is a maximum utilization instead of power, and instead of budget tokens it is a utilization allotment. Similarly, compensation to VMs is performed in the same manner as with the budget manager. In this case, however, compensation tokens are incremented, and decremented, based upon utilization slack as opposed to power slack.

Utility Manager The utility manager assigns a VPM token to each application VM in the datacenter. These VPM tokens are then used by system managers to determine budget allocations. For the purposes of this work, since we focus on the power budget management aspects of our system, we do not implement a dynamic utility determination system. Instead, a simple daemon exports an interface that can be used by other system

managers to query the current VM token value of guest VMs. The token values themselves are hard coded, and are assigned so that their sum is one hundred. As future work, we plan to integrate an actual utility management system based upon solutions being actively developed [109].

7.5.2 Experimental Results

The experimental evaluations presented in this section highlight the capabilities of VPM token based power budgeting. In particular, we demonstrate the benefits of improved data-center utility and VM performance with respect to workload specific SLAs, when budgeting is required. Two metrics are used in our results to measure these characteristics. The first, $Perf_i$, denotes the normalized performance experienced by VM i relative to its SLA (value between zero and one). For batch workloads, this value is one as long as some progress is being made, otherwise zero. For the transaction workloads, this value is one if the required processing rate is met, otherwise it is the ratio of observed processing rate and the SLA specified rate. Finally, for our web service Nutch application, $Perf_i$ is defined as one if the SLA is met for the client load experienced, otherwise it is $(1 - \frac{(QoI_{SLA} - QoI_{exp})}{QoI_{SLA} - 80\%})$, where QoI_{exp} and QoI_{SLA} are the experienced and SLA levels of QoI for the client class load, and 80% is defined to be minimal QoI, as described in Section 7.4.2.

The first set of results considers three systems, including two P4 platforms and our Core based system. We deploy two batch VMs onto one P4 machine, a batch VM and a web service VM (e.g., Nutch) on another, and two transactional VMs with different SLA constraints onto the Core machine, where one requires transactions to be processed at twice the rate compared to the other. The web service is loaded with "heavy" client requests as defined in Section 7.4.2. The utility manager defines the VM tokens for these workloads to be 5 for each of the batch VMs, 45 for the Nutch workload, and 20 for each of the transactional workloads. We compare the performance of our virtualization-aware power budgeting system against a simple 'unaware' budgeting system. In particular, we define a power-unconstrained budget, P_{unc}, to be equal to the sum of the maximum power consumption of systems being managed. Whenever a power limit, P_{Budget}, below

the unconstrained value must be enforced, the unaware budget manager simply requests power reductions on each of the N systems by $\frac{(P_{unc}-P_{Budget})}{N}$. If a platform manager cannot meet the request (e.g., it is already at idle power), the unaware manager reduces power allotments to remaining systems equally to compensate. The unaware platform manager similarly divides derived resource budgets uniformly amongst VMs.

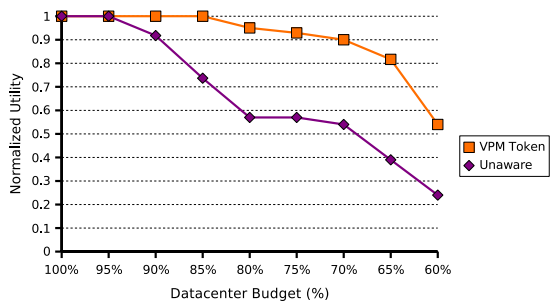


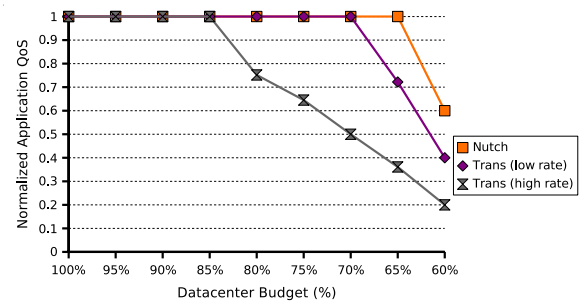
Figure 60: Datacenter Utility Across Budgets.

Figure 60 provides the results of the two budgeting approaches across power budgets as a percentage of P_{unc}. The normalized utility is calculated per Equation 17, where Tok_{VM} is the VM token value assigned by the utility manager to workload i and the sum of all VM tokens is equal to one hundred.

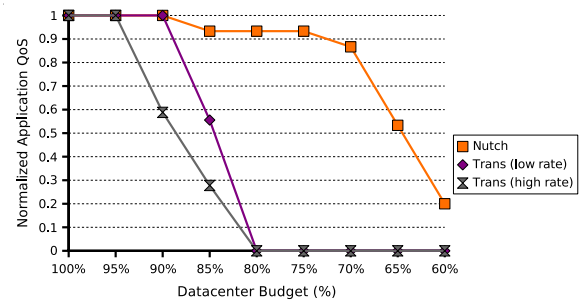
$$Utility = \sum_{i=0}^{W-1} \frac{Perf_i * Tok_{VM_i}}{100} \tag{17}$$

We observe from Figure 60 that VPM token based management is able to improve datacenter utility degradation by up to 43% when budgets must be enforced. Moreover, it is able to limit utility degradation within 10% when budget is reduced by as much as 30%. In comparison, an unaware budgeting approach incurs a 10% penalty when budget is reduced by 10%. These results emphasize the datacenter level utility benefits of VM-aware management enabled with VPM tokens. Figure 61 provides further detail into the management behavior that provides these trends.

Figure 61 summarizes the application QoS metric $Perf_i$ for the non batch workloads in our experimental scenario across the power budgets. We exclude the batch workload results since for both management approaches, the QoS of batch jobs is not impacted



(a) VPM Token



(b) Unaware Management

Figure 61: Application QoS Across Budgets.

across budgets. We observe in Figure 61(b) that the transaction workloads are dramatically affected when performing unaware management. This can be attributed to the fact that the unaware management scheme does not account for platform heterogeneity and therefore, reduces allocations to underlying platforms uniformly when distributing power budgets. Since the Core based platform on which these VMs run has a smaller power range between idle and full performance, without heterogeneity awareness, the budgeter reduces power to the point where the Core platform is forced to minimal performance, which causes utility for the associated VMs to drop to zero. In contrast, we see in Figure 61(a) that the performance impact on the transaction VMs is reduced much more gracefully and not as severely with VPM tokens. In addition, we see that by proportionally allocating budgets across systems based upon the VM tokens of executing VMs, the VPM budgeter provides preferential treatment to the transaction and Nutch VMs by first reducing allocation to the system running batch workloads aggressively to meet power budgets, as opposed to equally reducing power allotments. This allows the system to minimally impact the QoS of the

VMs that critically impact datacenter utility. An interesting result from the figure is that the QoS of the high rate transaction VM is impacted while the low rate is not, despite the fact that they have the same utility. This implies that there could be "performance slack" in the low rate transaction workload that can be used to alleviate some degradation to the high rate VM. This type of application-aware budget tradeoff can be determined by using our VPM channels along with workload policies, as considered next.

Application-specific feedback via VPM channels can be used to improve the utility of applications when budgets must be enforced, particularly when VMs have the same VM token values but different performance SLAs, as in our previous example. To study this effect, consider two transaction VMs running on a P4 platform. Again, one VM has a low transaction rate requirement, while the other must process rates four times as fast. We install in these VMs policies that monitor the slack available between minimum desired transaction processing rate and the current rate being observed. When slack is large, the policy requests reduced processor frequencies, and if slack becomes negative, it scales up the underlying processor's performance state. The VM policies act upon a set of virtualized processor performance states, or VPM states, consisting of 3.2GHz, 2.8GHz, 2.0GHz, 1.6GHz, and 800MHz.

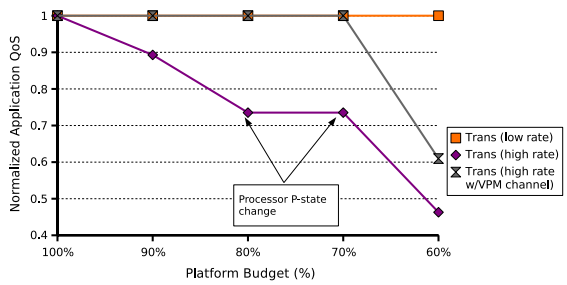


Figure 62: Budgeting Benefits of Application-awareness.

Figure 62 provides QoS results for the high rate transaction VM, with and without the use of application hints from VPM channels, as well as the QoS of the low rate transaction VM whose performance is not impacted in either case. We see that by being aware of the power/performance tradeoffs of the low rate VM via its requests of VPM states, the platform manager can reduce allocations to it and provide them to the high rate VM, thereby

improving its QoS by up to 26%. We also observe from the case of unaware management of the high rate VM the use of P-state support to maintain QoS while reducing budgets from 80% to 70%. This highlights the opportunities available to platform managers when they can dynamically utilize different hardware performance states to meet power budgets, using our token currency exchange functionality.

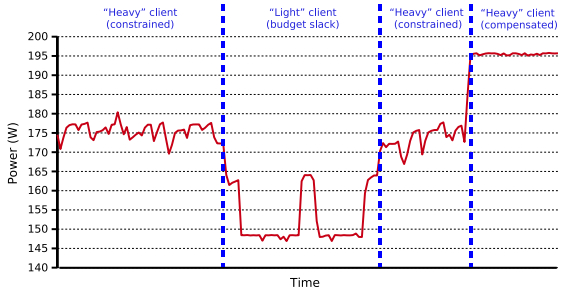


Figure 63: Budgeting Across Load Variations.

As a final experimental result, consider the use of compensation tokens in the context of the Nutch web service workload. In particular, based upon our observations of the Delta revenue pipeline behavior, we vary the load on the application between the "heavy" and "light" client models. We also employ an application-aware policy in the Nutch VM that requests processor states based upon slack between monitored QoI and SLA specifications of the current load class. Figure 63 illustrates the power behavior of the platform on which the Nutch VM runs. Initially, the VM is loaded with heavy client requests and power constrained due to global power budgeting. This results in a QoS reduction of 24%. The client load is then shifted to light client requests, which causes the VM to request reduced power allocations that still allow it to meet the SLA of the load resulting in budget slack that causes an increase in the compensation token value of the VM as described earlier. Subsequently, the load resumes to be driven by the heavy class, again resulting in the 24% QoS degradation due to power constraints. When budget slack is later created by other managed platforms, though, the slack is allocated preferentially to the Nutch VM due to its compensation token value, allowing it to meet its SLA even with the heavy load class. Without the use of compensation tokens the Nutch VM would instead get allocated

less power preventing it from achieving its SLA and instead experiencing degradation of approximately 12%. This illustrative experimental example highlights the ability of VPM token based management to balance changes in application behavior in time with power budgeting.

7.6 Related Work

Significant amounts of work have addressed the critical need for power management technologies on emerging platforms including mechanisms to enforce thermal thresholds on the processor [13]. Increasing support for dynamic voltage and frequency scaling (DVFS) on modern processors can be leveraged during memory bound phases of execution to provide power savings with minimal performance impact [43, 61]. Our VPM token-based management methods can be used when these types of power saving techniques alone cannot meet system power constraints.

Power management solutions have already been extended to distributed server environments [12]. In high performance settings, energy savings have been shown obtainable during communication periods for parallel programs, using hardware frequency scaling capabilities [65]. Datacenter level power consumption can also be reduced by turning servers on and off based on demand [16]. Utilizing this type of cluster reconfiguration in conjunction with DVFS [26], or by using spare servers [96] for power management has been investigated as well. Thermal management solutions in distributed settings have considered temperature-aware workload placement [79] as well as emulation tools for investigating the thermal implications of power management [38]. Our approach also considers managing distributed sets of physical platforms, but is focused on the ability to meet power constraints.

Previous work on power budgeting includes platform level controllers that perform fine grain throttling to meet power limits [60] as well as budgeting solutions with a performance loss minimization objective for SMP systems [52]. Considering multiple systems, methods that enforce power budgets using non-uniform allocations across nodes in datacenters [29] or across blades in a chassis [97] have been proposed. Energy accounting capabilities that allow for accurate division of platform power budgets between virtual machines have been

developed, as well [104]. By extending the original VirtualPower system with power budget management capabilities for distributed platforms, we aim to combine non-uniform power allocation techniques between physical platforms with VM-level power budgeting actions, to provide a system that can maximize datacenter utility while maintaining specified power constraints.

7.7 Summary

Power budgeting is a necessary ability in the landscape of datacenter management. In this chapter we extend our VirtualPower system to enable VM-aware power budgeting in enterprise systems. We extend a set of coordination mechanisms called VPM tokens that can be used to balance power, performance, and utility tradeoffs amongst various managers distributed in the environment. Our experimental evaluation of VPM tokens highlight benefits of up to 43% in overall utility while enforcing power budgets, and QoS improvements of up to 26% by coupling VM-budgeting with our earlier VPM state and channel mechanisms. Overall the results in this chapter underscore the benefits of VM-aware power budgeting, and the value of the contributions of our approach.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

The increasing power envelopes of emerging platforms have made power consumption a critical problem in modern computing systems. Power management components designed to address this need often exist in distinct levels of the overall system, and have no direct interaction. Since these power management entities can influence each other, it is critical to coordinate them to ensure synergistic behavior across the system. In this thesis we have proposed and evaluated multiple coordination mechanisms that make use of three categories of manageability available in cooperative distributed systems.

First, hardware management support that provides the ability to tune power efficiency of platform components such as processors (e.g. dynamic voltage and frequency scaling). Second we consider a technique we call allocation in time, where mechanisms can perform time multiplexing of consumers onto resources by modifying scheduling behavior within operating system or VMM resource managers. Finally, there is the ability to perform allocation in space, where modifying the mapping of various resources onto resources can improve power efficiency.

Specific contributions from this study include:

1. A device windows based EESI scheduling architecture to improve benefits of timeout driven management policies integrated into I/O devices

2. CompatPM attributes for coordinating middleware level processor frequency management of underlying mobile platforms

3. A heterogeneity aware load allocation system, HALM, for managing server environments

4. VirtualPower, a coordination architecture for application aware management of virtualized systems

5. VPM token extensions for virtual machine aware power budgeting

Experimental evaluations of these mechanisms support the claim in this dissertation that coordination can enable synergistic behavior for improved power characteristics. Moreover, the results show that such coordination can be attained without having to resort to integrated solutions, or well defined interfaces or protocols that must be adhered to across the layers of the system.

Future work that can build upon the results contained in this thesis include leveraging hardware capabilities in future platforms, extending coordination functionality into virtualization solutions, and exploring facility level management of datacenters that includes coordinating with cooling systems. In the context of next generation platforms, there are two emerging capabilities that could further beneficially impact the management landscape: 3D architectures and reconfigurable logic (FPGAs). Advances in fabrication techniques are creating the ability to build 3D architectures that allow for vertical stacking of micro-architectural circuits [94]. Researchers have already considered possible benefits that can be attained with this technique including stacking DRAM on top of processing cores in order to provide a closer memory source that avoids the use of slow, power hungry buses to go off package. These capabilities can also possibly be used to tightly couple reconfigurable logic to processing cores [117]. With this opportunity, it becomes interesting to investigate how the dynamic creation of workload-specific heterogeneous computing elements using reconfigurable logic could benefit system power, performance, and efficiency. These methods could be integrated into power management components in next generation virtualization systems such as VirtualPower, or even at the application middleware layer.

An interesting aspect of the VirtualPower approach is that it highlights the ability to obtain benefits by disassociating the virtual management states of system components from their actual physical states. This idea can be carried forward more aggressively, by considering the use of enhanced virtual device models in virtualized systems. The idea here is to provide a richer set of virtual devices than underlying physical device capabilities. For example, a secure virtual network device can be provided to a guest operating system by enhancing an unencrypted link layer with encryption methods executed on an accelerator

(e.g., using the encryption device already present on IXP network processors). Here, the discrepancies between the virtual and physical devices can be met using software, hardware, or both, in order to implement the behavior expected by the virtual device. This capability can allow applications in virtual machines to explicitly describe the type of I/O and compute resources required in a more flexible fashion by requesting a particular virtual device model. Moreover, policies can dynamically map the virtual device model onto physical resources in a manner that can improve power and thermal characteristics.

At the highest end of the spectrum, improved management capabilities are being integrated into datacenter cooling systems [48]. Researchers are actively pursuing techniques such as heterogeneous airflow supported by adjustable flaps in the plenum underneath server racks, as well as integrating active cooling solutions, such as liquid cooling, into racks and even platforms. These cooling technologies can dramatically benefit thermal management in server systems. The problem that is created, then, is to balance the dynamic power costs of these various tunable cooling solutions with the power management of compute resources being performed by technologies like VirtualPower. This problem, along with each of the avenues of future work outlined here, build on the themes developed in this thesis, to coordinate management across multiple layers of a distributed system.

REFERENCES

[1] AbouGhazaleh, N., Mosse, D., Childers, B., Melhem, R., and Craven, M., "Collaborative operating system and compiler power management for real-time applications," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2003.

[2] Agarwala, S., Alegre, F., Schwan, K., and Mehalingham, J., "E2eprof: Automated end-to-end performance management for enterprise systems," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2007.

[3] Almeida, J., Ardagna, D., and Trubian, M., "Resource management in the autonomic service-oriented architecture," in *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2006.

[4] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2.

[5] Amur, H., Nathuji, R., Ghosh, M., Schwan, K., and Lee, H.-H., "Idlepower: Application-aware management of processor idle states," in *Proceedings of the Workshop on Managed Many-Core Systems (MMCS)*, June 2008.

[6] Aydin, H., Melhem, R., Mosse, D., and Mejia-Alvarez, P., "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, 2004.

[7] Ballapuram, C., Puttaswamy, K., Loh, G., and Lee, H.-H., "Entropy-based low power data tlb design," in *Proceedings of the ACM/IEEE Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, October 2006.

[8] Ballapuram, C., Sharif, A., and Lee, H.-H., "Exploiting access semantics and program behavior to reduce snoop power in chip multiprocessors," in *Proceedings of the 13th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.

[9] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., "Xen and the art of virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

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

[11] Bianchini, R. and Rajamony, R., "Power and energy management for server systems," *IEEE Computer*, vol. 37, no. 11, 2004.

[12] Bohrer, P., Elnozahy, E., Keller, T., Kistler, M., and Rawson, F., "Energy-induced process migration." United States Patent 6985952, January 2006.

[13] Brooks, D. and Martonosi, M., "Dynamic thermal management for high-performance microprocessors," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA)*, January 2001.

[14] Brooks, D., Tiwari, V., and Martonosi, M., "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 83–94, June 2000.

[15] Celebican, O., Rosing, T., and Mooney, V., "Energy estimation of peripheral devices in embedded systems," in *Proceedings of the Great Lakes Symposium on VLSI (GLVLSI)*, April 2004.

[16] Chase, J., Anderson, D., Thakar, P., Vahdat, A., and Doyle, R., "Managing energy and server resources in hosting centers," in *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, October 2001.

[17] Chou, P., Liu, J., Li, D., and Bagherzadeh, N., "Impacct:methodology and tools for power-aware embedded systems," *Kluwer Design Automation of Embedded Systems*, October 2002.

[18] Chou, Y., Fahs, B., and Abraham, S., "Microarchitecture optimizations for exploiting memory-level parallelism," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2004.

[19] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A., "Live migration of virtual machines," in *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[20] Contreras, G., Martonosi, M., Peng, J., Ju, R., and Lueh, G.-Y., "Xtrem: A power simulator for the intel xscale core," in *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, June 2004.

[21] Czajkowski, K., Foster, I., Kesselman, C., Sander, V., and Tuecke, S., "Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems," in *Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.

[22] Diao, Q. and Song, J., "Prediction of cpu idle-busy activity pattern," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2008.

[23] Dropsho, S., Buyuktosunoglu, A., Balasubramonian, R., Albonesi, D., Dwarkadas, S., Semeraro, G., Magklis, G., and Scott, M., "Integrating adaptive on-chip storage structures for reduced dynamic power," in *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[24] Dropsho, S., Semeraro, G., Albonesi, D., Magklis, G., and Scott, M., "Dynamically trading frequency for complexity in a gals microprocessor," in *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2004.

[25] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R., and Sastry, A., "The cops (common open policy service) protocol." RFC 2748, January 2000.

[26] Elnozahy, E. N., Kistler, M., and Rajamony, R., "Energy-efficient server clusters," in *Proceedings of the Workshop on Power-Aware Computing Systems*, February 2002.

[27] Fan, X., Ellis, C., and Lebeck, A., "The synergy between power-aware memory systems and processor voltage scaling," in *Proceedings of the Workshop on Power-Aware Computer Systems (PACS)*, December 2003.

[28] Fan, X., Weber, W.-D., and Barroso, L., "Power provisioning for a warehouse-sized computer," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2007.

[29] Femal, M. and Freeh, V., "Boosting data center performance through non-uniform power allocation," in *Proceedings of the Second International Conference on Autonomic Computing (ICAC)*, 2005.

[30] Flautner, K. and Mudge, T., "Vertigo: Automatic performance-setting for linux," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[31] Flinn, J. and M., S., "Energy-aware adaptation for mobile applications," in *Proceedings of the Symposium on Operating System Principles (SOSP)*, December 1999.

[32] Flinn, J., Park, S., and Satyanarayanan, M., "Balancing performance, energy, and quality in pervasive computing," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[33] Fryman, J., Huneycutt, C., Lee, H.-H., Mackenzie, K., and Schimmel, D., "Energy efficient network memory for ubiquitous devices," *IEEE MICRO Special Edition*, September/October 2003.

[34] Ghiasi, S., Keller, T., and Rawson, F., "Scheduling for heterogeneous processors in server systems," in *Proceedings of the International Conference on Computing Frontiers*, 2005.

[35] Ghosh, M., Ozer, E., Biles, S., and Lee, H.-H., "Efficient system-on-chip energy management with a segmented bloom filter," in *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS)*, March 2006.

[36] Graupner, S., Konig, R., Machiraju, V., Pruyne, J., Sahai, A., and Moorsel, A. V., "Impact of virtualization on management systems," tech. rep., Hewlett-Packard Labs, 2003.

[37] Greenberg, S., Mills, E., Tschudi, B., Rumsey, P., and Myatt, B., "Best practices for data centers: Results from benchmarking 22 data centers," in *Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings*, 2006.

[38] Heath, T., Centeno, A. P., George, P., Ramos, L., Jaluria, Y., and Bianchini, R., "Mercury and freon: Temperature emulation and management in server

systems," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2006.

[39] Heath, T., Diniz, B., Carrera, E. V., Meira Jr., W., and Bianchini, R., "Energy conservation in heterogeneous server clusters," in *Proceedings of the 10th Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2005.

[40] Hellerstein, J., Diao, Y., Parekh, S., and Tilbury, D., *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[41] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, "Advanced configuration and power interface specification." http://www.acpi.info, September 2004.

[42] Irani, S., Shukla, S., and Gupta, R., "Algorithms for power savings," in *Proceedings of the 14th Symposium on Discrete Algorithms*, 2003.

[43] Isci, C., Contreras, G., and Martonosi, M., "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proceedings of the 39th International Symposium on Microarchitecture (MICRO-39)*, December 2006.

[44] Isci, C. and Martonosi, M., "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the IEEE International Symposium on Microarchitecture (MICRO-36)*, December 2003.

[45] Jejurikar, R. and Gupta, R., "Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, August 2004.

[46] Jejurikar, R., Pereira, C., and Gupta, R., "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the IEEE Design Automation Conference (DAC)*, 2004.

[47] Joseph, R. and Martonosi, M., "Run-time power esimtation in high-performance microprocessors," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 135–140, August 2001.

[48] Joshi, Y., Samadiani, E., and Mistree, F., "Reduced modeling based robust thermal design of energy efficient data centers," in *Proceedings of the Eighteenth International Symposium on Transport Phenomena*, August 2007.

[49] Kannan, K., Jones, C., Lee, N., Leontiades, K., Novak, F., and Sharma, V., "System for distributed power management in portable computers." United States Patent 5423045, June 1995.

[50] Kephart, J. and Chess, D., "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[51] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C., "An analysis of performance interference effects in virtual environments," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2007.

[52] Kotla, R., Ghiasi, S., Keller, T., and Rawson, F., "Scheduling processor voltage and frequency in server and cluster systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[53] Krashinsky, R. and Balakrishnan, H., "Minimizing energy for wireless web access with bounded slowdown," in *Proceedings of ACM MOBICOM*, pp. 119–130, September 2002.

[54] Kravets, R. and Krishnan, P., "Application-driven power management for mobile communication," in *Proceedings of the Fourth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pp. 263–277, October 1998.

[55] Kumar, R., Farkas, K., Jouppi, N., Ranganathan, P., and Tullsen, D., "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings of the 36th International Symposium on Microarchitecture*, December 2003.

[56] Kumar, R., Tullsen, D., Ranganathan, P., Jouppi, N., and Farkas, K., "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2004.

[57] Kumar, S., Talwar, V., Ranganathan, P., Nathuji, R., and Schwan, K., "M-channels and m-brokers: Coordinated management in virtualized systems," in *Proceedings of the Workshop on Managed Many-Core Systems (MMCS)*, June 2008.

[58] Lebeck, A., Fan, X., Zeng, H., and Ellis, C., "Power aware page allocation," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 105–116, November 2000.

[59] Lee, K.-J. and Skadron, K., "Using performance counters for runtime temperature sensing in high-performance processors," in *Proceedings of the Workshop on High-Performance, Power-Aware Computing (HP-PAC)*, April 2005.

[60] Lefurgy, C., Wang, X., and Ware, M., "Server-level power control," in *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2007.

[61] Li, H., Cher, C., Vijaykumar, T., and Roy, K., "Vsv: L2-miss-driven variable supply-voltage scaling for low power," in *Proceedings of the IEEE International Symposium on Microarchitecture (MICRO-36)*, 2003.

[62] Li, T. and John, L., "Run-time modeling and estimation of operating system power consumption," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2003.

[63] Li, X., Li, Z., David, F., Zhou, P., Zhou, Y., Adve, S., and Kumar, S., "Performance directed energy management for main memory and disks," in *Proceedings of the Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2004.

[64] Li, Y., Brooks, D., Hu, Z., and Skadron, K., "Performance, energy, and thermal considerations for smt and cmp architectures," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA-11)*, February 2005.

[65] Lim, M., Freeh, V., and Lowenthal, D., "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *IEEE/ACM Supercomputing*, November 2006.

[66] Lu, Y., Benini, L., and Micheli, G., "Low-power task scheduling for multiple devices," in *Proceedings of the 8th International Workshop on Hardware/Software Codesign*, pp. 39–43, 2000.

[67] Lu, Y., Benini, L., and Micheli, G., "Operating system directed power reduction," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 37–42, 2000.

[68] Lu, Y., Benini, L., and Micheli, G., "Requester-aware power reduction," in *Proceedings of the International Symposium on System Synthesis*, pp. 18–23, 2000.

[69] Lu, Y., Benini, L., and Micheli, G., "Power-aware operating systems for interactive systems," *IEEE Transactions on Very Large Scale Integration Systems*, pp. 119–134, April 2002.

[70] Lu, Y., Chung, E., Simunic, T., Benini, L., and Micheli, G., "Quantitative comparison of power management algorithms," in *Proceedings of the Design, Automation, and Test in Europe*, pp. 20–26, 2000.

[71] Luo, K., Gummaraju, J., and Franklin, M., "Balancing throughput and fairness in smt processors," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, November 2001.

[72] Mahesri, A. and Vardhan, V., "Power consumption breakdown on a modern laptop," in *Proceedings of the Workshop on Power-Aware Computer Systems (PACS)*, 2004.

[73] Mallik, A., Cosgrove, J., Dick, R., Memik, G., and Dinda, P., "Picsel: Measuring user-perceived performance to control dynamic frequency scaling," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.

[74] Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G., "A case study of mobile robot's energy consumption and conservation techniques," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[75] Meng, Y., Sherwood, T., and Kastner, R., "On the limits of leakage power reduction in caches," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA-11)*, February 2005.

[76] Mishra, R., Rastogi, N., and Zhu, D., "Energy aware scheduling for distributed real-time systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2003.

[77] Miyoshi, A., Lefurgy, C., Van Hensbergen, E., Rajamony, R., and Rajkumar, R., "Critical power slope: Understanding the runtime effects of frequency scaling," in *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, June 2002.

[78] Mohaban, S., Parnafes, I., and McCloghrie, K., "Method and apparatus for communicating cops protocol policies to non-cops-enabled network devices." United States Patent 6839766, January 2005.

[79] Moore, J., Chase, J., Ranganathan, P., and Sharma, R., "Making scheduling cool: Temperature-aware workload placement in data centers," in *Proceedings of USENIX '05*, June 2005.

[80] Mudge, T., "Power: A first-class architectural design constraint," *IEEE Computer*, vol. 34, no. 4, 2001.

[81] Nathuji, R., Isci, C., and Gorbatov, E., "Exploiting platform heterogeneity for power efficient data centers," in *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2007.

[82] Nathuji, R., O'Hara, K., Schwan, K., and Balch, T., "Compatpm: Enabling energy efficient multimedia workloads for distributed mobile platforms," in *Proceedings of the ACM Multimedia Computing and Networking Conference (MMCN)*, 2007.

[83] Nathuji, R. and Schwan, K., "Reducing system level power consumption for mobile and embedded platforms," in *Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*, March 2005.

[84] Nathuji, R. and Schwan, K., "Virtualpower: Coordinated power management in virtualized enterprise systems," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, October 2007.

[85] Nathuji, R. and Schwan, K., "Vpm tokens: Virtual machine-aware power budgeting in datacenters," in *Proceedings of the ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC)*, June 2008.

[86] Nathuji, R., Seshasayee, B., and Schwan, K., "Combining compiler and operating system support for energy efficient i/o on embedded platforms," in *Proceedings of the International Workshop on Software and Compilers for Embedded Systems (SCOPES)*, September 2005.

[87] Neiger, G., Santoni, A., Leung, F., Rodgers, D., and Uhlig, R., "Intel virtualization technology: Hardware support for efficient processor virtualization," in *Intel Technology Journal (http://www.intel.com/technology/itj/2006/v10i3/)*, August 2006.

[88] Nutch. http://lucene.apache.org/nutch.

[89] O'Hara, K., Nathuji, R., Raj, H., Schwan, K., and Balch, T., "Autopower: Toward energy-aware software systems for distributed mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[90] Pillai, P. and Shin, K., "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, October 2001.

[91] Poellabauer, C. and Schwan, K., "Power-aware video decoding using real-time event handlers," in *Proceedings of the 5th International Workshop on Wireless Mobile Multimedia (WoWMoM)*, September 2002.

[92] Poellabauer, C. and Schwan, K., "Energy-aware traffic shaping for wireless real-time applications," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2004.

[93] Poellabauer, C., Singleton, L., and Schwan, K., "Feedback-based dynamic frequency scaling for memory-bound real-time applications," in *Proceedings of the 11th Real-Time Embedded Technology and Applications Symposium (RTAS)*, March 2005.

[94] Puttaswamy, K. and Loh, G., "Thermal herding: Microarchitecture techniques for controlling hotspots in high-performance 3d-integrated processors," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2007.

[95] Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., and Zhu, X., "No "power" struggles: Coordinated multi-level power management for the data center," in *Proceedings of the 13th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.

[96] Rajamani, K. and Lefurgy, C., "On evaluating request-distribution schemes for saving energy in server clusters," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2003.

[97] Ranganathan, P., Leech, P., Irwin, D., and Chase, J., "Ensemble-level power management for dense blade servers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006.

[98] Sachs, D., Yuan, W., Hughes, C., Harris, A., Adve, S., Jones, D., Kravets, R., and Nahrstedt, K., "Grace: A hierarchical adaptation framework for saving energy," tech. rep., University of Illinois at Urbana-Champaign, 2004.

[99] Seshasayee, B., Nathuji, R., and Schwan, K., "Energy-aware mobile service overlays: Cooperative dynamic power management in distributed mobile systems," in *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2007.

[100] Shen, J. P. and Lipasti, M. H., *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw Hill, 2005.

[101] Shih, E., Bahl, P., and Sinclair, M., "Wake on wireless: An event driven energy saving strategy for battery operated devices," in *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pp. 160–171, September 2002.

[102] Singleton, L., Poellabauer, C., and Schwan, K., "Monitoring of cache miss rates for accurate dynamic voltage and frequency scaling," in *Proceedings of the Multimedia Computing and Networking Conference (MMCN)*, 2005.

[103] Squillante, M. and Lazowska, E., "Using processor-cache affinity information in shared-memory multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 131–143, 1993.

[104] Stoess, J., Lang, C., and Bellosa, F., "Energy management for hypervisor-based virtual machines," in *Proceedings of the USENIX Annual Technical Conference*, June 2007.

[105] Sugerman, J., Venkitachalam, G., and Lim, B.-H., "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor," in *Proceedings of USENIX*, 2001.

[106] Swaminathan, V., Chakrabarty, K., and Iyengar, S., "Dynamic i/o power management for hard real-time systems," in *Proceedings of International Symposium on Hardware/Software Codesign*, pp. 237–243, 2001.

[107] Swaminathan, V., Schweizer, C., Chakrabarty, K., and Patel, A., "Experiences in implementing an energy-driven task scheduler in rt-linux," in *Proceedings of the Real-time and Embedded Technology and Applications Symposium (RTAS)*, pp. 229–239, 2002.

[108] Waldspurger, C. and Weihl, W., "Lottery scheduling: Flexible proportional-share resource mangement," in *Proceedings of the First Symposium on Operating System Design and Implementation (OSDI)*, 1994.

[109] Walsh, W., Tesauro, G., Kephart, J., and Das, R., "Utility functions in autonomic systems," in *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, 2004.

[110] Weiser, M., Welch, B., Demers, A., and Shenker, S., "Scheduling for reduced cpu energy," in *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pp. 13–23, November 1994.

[111] Weissel, A., Beutel, B., and Bellosa, F., "Cooperative i/o-a novel io semantics for energy-aware applications," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[112] West, R., Schwan, K., and Poellabauer, C., "Scalable scheduling support for loss and delay constrained media streams," in *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*, June 1999.

[113] West, R., Zhang, Y., Schwan, K., and Poellabauer, C., "Dynamic window-constrained scheduling of real-time streams in media servers," *IEEE Transactions on Computers*, vol. 53, no. 6, 2004.

[114] Williams, B., Graf, L., Hollis, M., and Rytina, I., "Policy co-ordination in a communications network." United States Patent 7246165, July 2007.

[115] Witten, I. H. and Frank, E., *Data Mining: Practical machine learning tools with Java implementations*. San Francisco: Morgan Kaufmann, 2000.

[116] Wolf, M., Cai, Z., Huang, W., and Schwan, K., "Smartpointers: personalized scientific data portals in your hand," in *IEEE/ACM Supercomputing*, 2002.

[117] Ye, Z., Moshovos, A., Hauck, S., and Banerjee, P., "Chimaera: A high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2000.

[118] Yuan, W. and Nahrstedt, K., "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[119] Zeng, H., Ellis, C., Lebeck, A., and Vahdat, A., "Currentcy: A unifying abstraction for expressing energy management policies," in *Proceedings of USENIX*, pp. 43–56, June 2003.

[120] Zeng, H., Fan, X., Ellis, C., Lebeck, A., and Vahdat, A., "Ecosystem: Managing energy as a first class operating system resource," in *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, October 2002.

[121] Zhang, W., "Linux virtual server for scalable network services," *Ottawa Linux Symposium*, 2000.

[122] Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., and Wilkes, J., "Hibernator: Helping disk arrays sleep through the winter," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, October 2005.

VITA

Ripal Nathuji was born in California, USA, but soon thereafter moved to rural Texas where he grew up. After graduating from the Texas Academy of Mathematics and Science, he attended the Massachusetts Institute of Technology where he obtained a B.S. in Electrical Engineering and Computer Science. Subsequently, he received his M.S. in Computer Engineering from Texas A&M University before going to the Georgia Institute of Technology for his Ph.D. Ripal is married to Amanda Nathuji, and is anxiously awaiting the birth of their daughter, Elena.