VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems

Ripal Nathuji
CERCS Research Center
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30032
rnathuji@ece.gatech.edu

Karsten Schwan
CERCS Research Center
College of Computing
Georgia Institute of Technology
Atlanta, GA 30032
schwan@cc.gatech.edu

ABSTRACT

Power management has become increasingly necessary in large-scale datacenters to address costs and limitations in cooling or power delivery. This paper explores how to integrate power management mechanisms and policies with the virtualization technologies being actively deployed in these environments. The goals of the proposed *VirtualPower* approach to online power management are (i) to support the isolated and independent operation assumed by guest virtual machines (VMs) running on virtualized platforms and (ii) to make it possible to control and globally coordinate the effects of the diverse power management policies applied by these VMs to virtualized resources. To attain these goals, VirtualPower extends to guest VMs 'soft' versions of the hardware power states for which their policies are designed. The resulting technical challenge is to appropriately map VM-level updates made to soft power states to actual changes in the states or in the allocation of underlying virtualized hardware. An implementation of VirtualPower Management (VPM) for the Xen hypervisor addresses this challenge by provision of multiple system-level abstractions including VPM states, channels, mechanisms, and rules. Experimental evaluations on modern multicore platforms highlight resulting improvements in online power management capabilities, including minimization of power consumption with little or no performance penalties and the ability to throttle power consumption while still meeting application requirements. Finally, coordination of online methods for server consolidation with VPM management techniques in heterogeneous server systems is shown to provide up to 34% improvements in power consumption.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General—*System architectures*; D.4.7 [**Operating Systems**]: Organization and Design; K.6.4 [**Management of Computing and Information Systems**]: System Management

SOSP'07, October 14–17, 2007, Stevenson, Washington, USA.

General Terms

Design, Experimentation, Management

Keywords

Power management, Virtualization

1. INTRODUCTION

The necessity of power management in computing systems has become increasingly evident in both mobile and enterprise environments. Limitations in battery capacities and demands for longer device lifetimes have motivated research for mobile and embedded systems [12, 34, 37, 38]. For server systems, online management is important for two significant reasons. First, there are power delivery and cooling limitations in datacenter environments that result from meeting the ever increasing performance and scalability demands of enterprise applications with a combination of technological advances like increased densities, small form factor blade servers, and software methods for server consolidation through virtualization. Here, the nearly 60 Amps per rack currently provisioned in datacenters could become a bottleneck for high density configurations, even if the related cooling issues can be solved [33]. Online methods for minimizing power consumption and, when necessary, enforcing limits on rack power draws can help deal with such bottlenecks. A second motivation for online power management is the cost of power and cooling capabilities in datacenters. A datacenter consisting of 30,000 square feet and consuming 10MW, for instance, requires an accompanying cooling system that costs from $2-$5 million [25]. The yearly cost of running this cooling infrastructure can reach up to $4-$8 million [33].

To manage the power properties of server systems, researchers have considered issues that include managing thermal events [25] and utilizing hardware management support like processor frequency scaling for power budgeting [10, 33]. An aspect of future datacenter environments not addressed by previous work, however, is the proliferation of virtualization techniques into the enterprise domain. Specifically, the growing capabilities of hardware support for virtualization [28] and of software solutions like Xen [4] or VMware [36] have made it easier to flexibly use datacenter resources for applications [3]. 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 [30, 31] or meet application requirements with minimum power consumption [11] 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 [26], 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 paper 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, this paper's 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 [15], and can address the multiple machines, racks, and enclosures found in modern datacenters.

The remainder of this paper is organized as follows. Section 2 reviews related work. This is followed by a discussion in Section 3 on how virtualization impacts power management and the resulting technical challenges. Section 4 outlines the components of the VirtualPower architecture and describes the manner in which they address these challenges. After describing experimental methodology in Section 5, we evaluate the viability of soft scaling as a power management mechanism in Section 6, and the performance of coordinated policy-based management with VirtualPower in Section 7. Conclusions and future work are reviewed in Section 8.

2. 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 [5]. 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 [23] and OS-level techniques that set processor modes based on predicted application behavior [18]. Power budgeting of SMP systems with a performance loss minimization objective has also been implemented via CPU throttling [20]. VirtualPower leverages the invest-

ments 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 [24]. For server systems, Chase *et al.* discuss how to reduce power consumption in datacenters by turning servers on and off based on demand [6]. Identifying the need for disk power management in datacenters, support for energy-efficient disk arrays has been extended [39]. Incorporating temperature awareness into workload placement in datacenters was proposed by Moore *et al.* [25], along with emulation environments for investigating the thermal implications of power management [15]. Researchers have also investigated using such cluster-level reconfiguration in conjunction with DVFS [8], or by using spare servers [32] 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 [10], at the granularity of blade enclosures [33], or even amongst virtual machines using energy accounting capabilities [35]. The experimental evaluations performed in this paper 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 [21] and in cluster environments, the latter proposing a scheduling approach for power control of processors with varying fixed frequencies and voltages [13]. In enterprise systems, intelligent methods for request distribution have leveraged hardware heterogeneity to curtail power usage [16]. The ability to exploit heterogeneity in platform hardware and underlying power management capabilities for datacenters has also been investigated [26]. Experiments presented in this paper 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.

3. 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.

3.1 Scalable Enterprise Environments

Desired benefits from virtualization include improved fault isolation and independence for guest VMs [4], performance isolation [19], and the ease of seamless migration of VMs across different physical machines [7]. 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 [2, 14] 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 1.

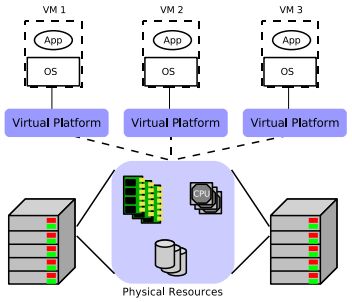


Figure 1: Scalable Enterprise Infrastructure.

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 4.

3.2 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 [30, 31]. 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 [17] 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.*

3.3 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.

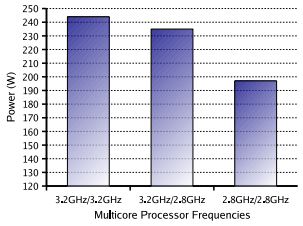


Figure 2: DVFS Limitations on a Dual Core Chip.

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 2, 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.*

4. VIRTUALPOWER ARCHITECTURE

In keeping with common practice, VirtualPower does not require modifications to guest operating systems. Further, while implemented with the Xen [4] 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.*

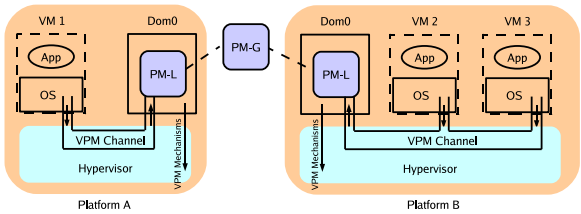


Figure 3: VirtualPower Management Architecture.

Figure 3 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, com-

posed of sets of VPM rules running in Dom0, which finally, use VPM mechanisms to actually carry out management decisions. As described further in Section 7, 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.

4.1 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 [17]) 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.

4.2 VPM Channels

VPM channels are the abstraction via which guest VMs provide input on power management to VPM rules, as illustrated in Figure 3. 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.

4.3 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 [27] 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`.

5. EVALUATION METHODOLOGY

5.1 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.

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 4 summarizes our power measurement configuration.

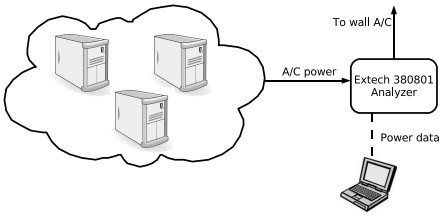


Figure 4: Power Measurement Setup.

5.2 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 backend subsystem responsible for tracking operational revenue earned from the services offered by the company [1]. 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 [29].

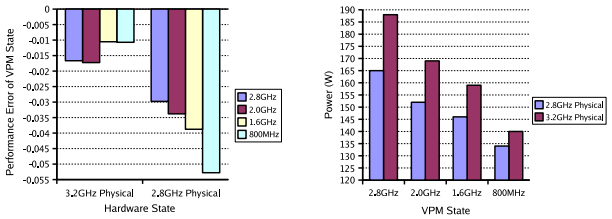
Tiered Web Service Workloads (RUBiS).

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.

6. VIABILITY OF SOFT SCALING

As described in Section 4.1, 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 in this paper 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.



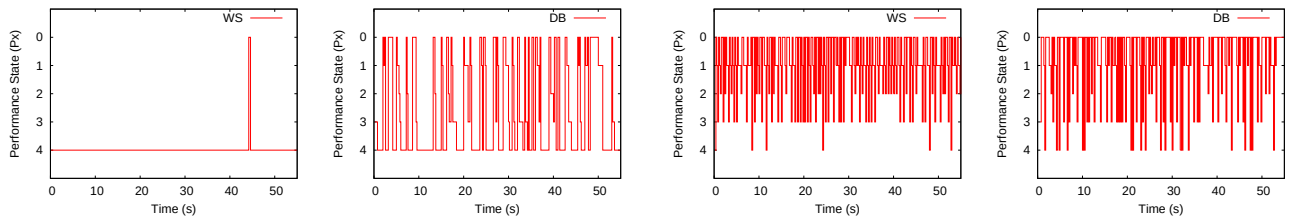
(a) Performance Error (b) Scaling Power Effects

Figure 5: Soft Scaling Characteristics.

In Figures 5(a) and 5(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 5(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 5(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 1: 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



(a) No Active Power Management (b) Active Power Management

Figure 6: VPM Channel Feedback for State-Based Guidance with VirtualPower.

Table 1 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 1. 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 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.

7. POLICY BASED COORDINATION

The VirtualPower infrastructure can be used to implement complex management policies. We demonstrate this fact with hierarchical power management policies [10]. 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.

7.1 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 6. 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 6(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 6(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 [5, 6]. 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 4, 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 7, 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.

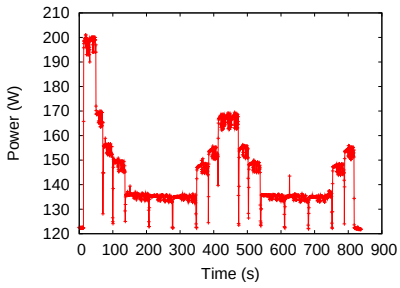
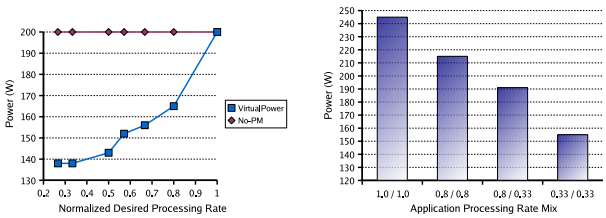


Figure 7: Power Trace of Transactional Application.

The power trace in Figure 7 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.

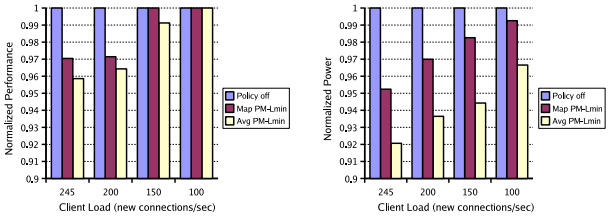


(a) Single Transaction VM (b) Two Transaction VMs

Figure 8: 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 8. 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 8(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 8(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 9: 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 9 (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 6. 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 9 shows that this policy provides similar performance characteristics to the mapping approach, but reduces power consumption further, by up to 4%. 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 [9]. Ongoing industry research is developing system support for power consumption monitoring that can be used as feedback for such policies [22, 33]. 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.

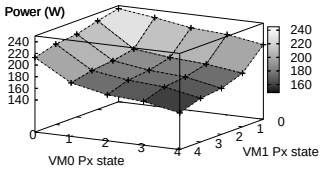
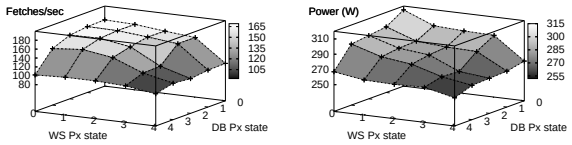


Figure 10: Transactional Workload Management.

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 10 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.*

Next, we again consider the RUBiS setup with two machines in Figure 11. If the power of both machines is limited simultaneously such as in a rack or enclosure setting, from



(a) Performance (b) Power Consumption

Figure 11: RUBiS Management.

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.

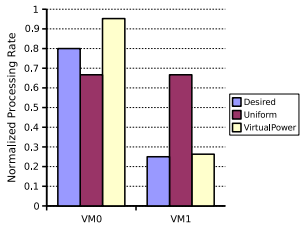


Figure 12: Non-uniform VPM State Allocation.

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 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 12 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.

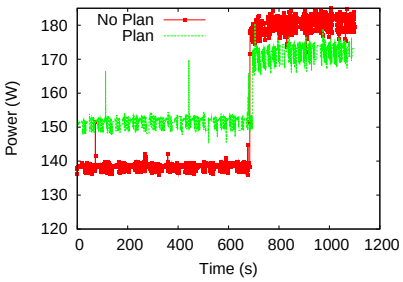


Figure 13: Power Tradeoffs of a Planning Policy.

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 13 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 13, 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.

7.2 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.

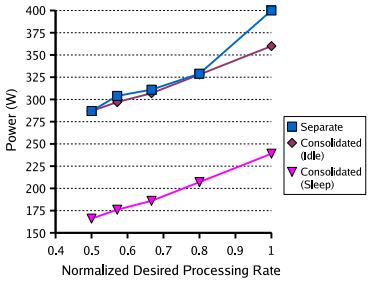


Figure 14: Consolidation with Sleep States.

Soft Scale-Enabled Consolidation.

To understand the benefits of consolidation, particularly in multicore machines, consider Figure 14. 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 2 provides the power consumption of these machines under various configurations.

Table 2: 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 2 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 15 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.

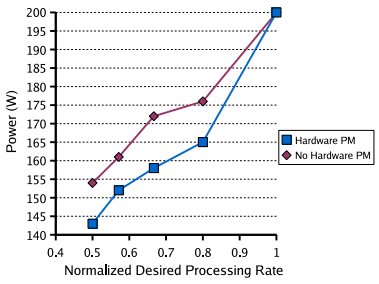


Figure 15: Power Management Heterogeneity.

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 results shown in this paper 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.

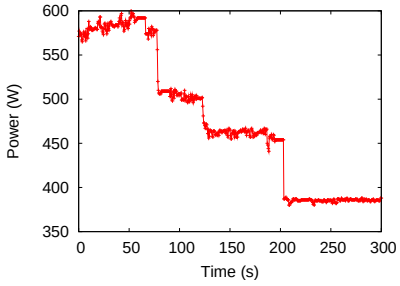


Figure 16: Consolidation with Heterogeneity.

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 16, 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 Virtu-

alPower 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.

8. CONCLUSIONS AND FUTURE WORK

Power management has become an important problem in enterprise systems, where costs as well as limitations in cooling and power delivery have made it necessary to extend active power management support into these environments. At the same time, end users are deploying new methods for system virtualization, to benefit from their isolation properties and/or from the increased levels of flexibility derived from server consolidation. Our research has extended virtualization solutions to support rich and effective policies for active power management. The VirtualPower infrastructure presented in this work advocates two basic ideas: (1) to present guest virtual machines with what appears to be a rich set of 'soft' power states accessible to their application-specific policies, termed VPM states, and then (2) to use the state changes requested by VMs as inputs to virtualization-level management policies. These include local policies architected to efficiently use specific platforms and their power management capabilities, coupled with global policies that consider goals derived from entire applications running across multiple machines and/or derived from global constraints, such as blade- or rack-level limitations on maximum power consumption. Extensive experimentation with multiple processors, including the new power-efficient Intel Core microarchitecture, demonstrate the benefits of active power management with VirtualPower as well as the suitability of VirtualPower's abstractions of VPM states, channels, mechanisms, and rules. They also demonstrate that with the state-based guidance approach to active power management realized in VirtualPower, it is possible to capture and respond to the application-specific power management desires and policies implemented in guest virtual machines without any need for application specificity at the virtualization level. Experimental evaluations also show substantial benefits derived from VirtualPower's use (i.e., with improvements in power consumption of up to 34% without appreciable losses in performance).

Our future work will extend the VirtualPower infrastructure across multiple dimensions. One of these is to include abstractions and mechanism support for VM-specific power throttling and power allocation. The idea is to ensure 'fair' power allocations across different VMs. An outcome of such fairness would be the ability to prevent power viruses from damaging machine performance. Another set of extensions will consider the potential performance and thus, power dependencies in applications like RUBiS, where often, it is only certain application components in certain VMs that critically affect application performance. Runtime methods for identifying these via VirtualPower's state-based guidance approach would permit downscaling of all but the most important VMs for these applications.

9. ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Mendel Rosenblum, for their feedback on improving this paper. This work has been supported by an NSF ITR award and by Intel Corporation via an Intel Foundation Student Fellowship. Extensive exposure to modern multicore platforms and virtualization technologies at Intel Corporation have been invaluable to our research, including discussions with Intel personnel, most notably Eugene Gorbatov and Rob Knauerhase. We also acknowledge additional inputs and support from Dilma Da Silva and Freeman Rawson at IBM. Finally, we would like to thank Intel Corporation for their generous donation of resources and platforms used to perform this work. Ripal Nathuji was awarded an SOSP student travel scholarship, supported by Infosys, to present this paper at the conference.

10. REFERENCES

[1] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. 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.

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

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

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

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

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

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

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

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

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

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

[12] J. Flinn and M. Satyanarayanan. Energy-aware

adaptation for mobile applications. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, December 1999.

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

[14] S. Graupner, R. Konig, V. Machiraju, J. Pruyne, A. Sahai, and A. V. Moorsel. Impact of virtualization on management systems. Technical report, Hewlett-Packard Labs, 2003.

[15] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. 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.

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

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

[18] C. Isci, G. Contreras, and M. Martonosi. 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.

[19] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. 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.

[20] R. Kotla, S. Ghiasi, T. Keller, and F. Rawson. Scheduling processor voltage and frequency in server and cluster systems. In *Proceedings of the Workshop on High-Performance, Power-Aware Computing (HP-HPAC)*, 2005.

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

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

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

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

[25] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling cool: Temperature-aware workload placement in data centers. In *Proceedings of the USENIX Annual Technical Conference*, June 2005.

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

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

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

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

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

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

[32] K. Rajamani and C. Lefurgy. 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.

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

[34] B. Seshasayee, R. Nathuji, and K. Schwan. 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.

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

[36] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, 2001.

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

[38] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Currentcy: A unifying abstraction for expressing energy management policies. In *Proceedings of the USENIX Annual Technical Conference*, June 2003.

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