Power Budgeting for Virtualized Data Centers

Harold Lim
Duke University

Aman Kansal
Microsoft Research

Jie Liu
Microsoft Research

Abstract

Power costs are very significant for data centers. To maximally utilize the provisioned power capacity, data centers often employ over-subscription, that is, the sum of peak consumptions of individual servers may be greater than the provisioned capacity. Power budgeting methods are employed to ensure that actual consumption never exceeds capacity. However, current power budgeting methods enforce capacity limits in hardware and are not well suited for virtualized servers because the hardware is shared among multiple applications. We present a power budgeting system for virtualized infrastructures that enforces power limits on individual distributed applications. Our system enables multiple applications to share the same servers but operate with their individual quality of service guarantees. It responds to workload and power availability changes, by dynamically allocating appropriate amount of power to different applications and tiers within applications. The design is mindful of practical constraints such the data center's limited visibility into hosted application performance. We evaluate the system using workloads derived from real world data center traces.

1 Introduction

Data centers require large amounts of power and the costs of their power supply infrastructure, backup generators and batteries, and power consumption are a significant concern [13]. Aside from costs, the availability of power may be a limiting factor, especially for smaller data centers deployed in enterprise buildings, educational institutions, and for emerging container-based "edge" data centers located close to end users. As a result, data center design must minimize the power capacity requested from utilities. The need to optimize provisioned power capacity has lead to the adoption of a practice known as *over-subscription*. In over-subscribed data centers, the sum of the possible peak power consumptions of all the servers combined is greater than the provisioned capacity. Servers typically operate below their peak power and even when servers from one application are near peak usage, other servers may be well below their peaks, keeping the total power within capacity. To ensure that actual total power use stays below capacity, servers are equipped with power budgeting mechanisms that can throttle the power usage of a server, such as by reducing the processor frequency. Power budgeting has been used for several safe and efficient over-subscription methods [9, 3, 23, 19].

However, the current methods are not well suited to virtualized infrastructures where the servers are shared by virtual machines (VMs) belonging to different applications, due to several reasons. First, in virtualized infrastructures, there is a disconnect between the *physical* server layout and the *logical* organization of resources among applications. Hardware power budgeting used in current power budgeting methods does not respect the isolation among virtual machines with different performance requirements. Second, existing techniques do not explicitly address workload and power dynamics. As input workload volumes change, the power available for different applications changes, as does the optimal distribution of power among an application's constituent tiers. Third, existing designs typically use a single power control knob and do not exploit multiple feasible combinations of power settings for optimizing performance.

In this paper we present a power budgeting solution named virtualized power shifting (VPS) that efficiently coordinates the power distribution among a large number of VMs within given peak power capacity. VPS dynamically shifts power among various distributed components to efficiently utilize the total available power budget, as workloads and power availability vary. Power is distributed among application components in the correct proportions to achieve the best performance. The system respects application boundaries and differentiates

performance based on priorities. In contrast to existing techniques that use only one power control knob, typically frequency scaling, VPS uses multiple power control knobs and selects the optimal combinations of power settings to optimize performance within the available power budget. We describe how the system operates with practical constraints such as limited insight into application performance.

The system tracks dynamic power availability and workload dynamics with low error, as its design is based on well-studied control theoretic algorithms with desirable stability and accuracy properties. We evaluate the system through experiments on a multi-server testbed running a mix of interactive and batch processing benchmarks. Real world data center traces from Microsoft's online services are used.

2 Virtualized Power Budgeting Challenges

The over-arching problem addressed by VPS is to dynamically adjust power allocations in a multi-application scenario. The key challenges presented by this problem are discussed below.

2.1 Server Sharing

The large number of servers in a data center are shared among multiple applications, typically using virtualization. VMs from multiple applications may be co-located on the same physical servers depending on the application characteristics such as complimentary resource usage and data placement needs. At the same time, VMs from one application are spread across many servers based on required minimum spread for hardware redundancy, and minimum number of servers needed to allow for seamless software upgrades. Since different applications have different users and workloads, an increase in the workload and power usage of one application should not negatively impact another application sharing the same hardware. In some scenarios, different applications may have different priorities. For example, customer-facing online services may have higher priority than batch processing or internal enterprise applications.

The power budgeting mechanism must therefore enforce power limits at *application granularity* rather than at the hardware level. Power budgets enforced in hardware, such as using dynamic voltage and frequency scaling (DVFS), impact an entire server or all processor cores supplied from the same voltage rail[1]. This will cause a performance drop for all application VMs sharing those processors. Additional power capping mechanisms that operate at the individual VM level must be used. Coordination of power allocations must also follow the application VM layout across the server infrastructure.

[1] In most servers, an entire processor chip or socket is supplied from a single voltage rail and hence the supply voltage and DVFS can only be controlled for the entire socket consisting of multiple cores.

2.2 Multi-dimensional Power Control

To respect application boundaries, a combination of hardware-based (e.g. DVFS) and software-based (e.g. VM CPU time allocation) power control knobs is used. Multiple knobs imply that more than one combinations of power settings may achieve the same power level. However, application performance may be different for each feasible combination. We illustrate this phenomenon through experimental measurements in a later section (Figure 5): at a given power level, performance varies up to 25% depending on power settings. Power budgeting design has the opportunity to *maximize performance*, if it intelligently selects the best combination of power settings to satisfy the power budget. Optimization of performance brings with it challenges of measuring and modelling performance. These measurements may not be available in certain scenarios, especially when the applications are not owned by the same entity that manages the data center, as is often the case for large organizations and cloud based infrastructures. The VPS system includes different modes of operation to work with or without such information.

2.3 Dynamic Power Proportions

The input workload volume for each application changes over time, implying that the power used, and as a result the power available for other applications, changes. The power budgets must be dynamically adapted, requiring run time coordination across all applications.

Within an application, allocation of power to its VMs is also non-trivial since the best allocation may vary with workload volume. This happens because different VMs may be hosting different tiers of the application. As a toy example, consider a two tier application with the front-end tier executing a processor intensive stage and the back-end tier providing data storage. Power usage of the front-end tier depends on processor utilization, and as an example suppose it changes between 50W and 100W. The back-end comprises disk storage, and has a high idle power for keeping the disks spinning, say 80W, with an additional power usage of up to 20W that varies with the volume of I/O activity. At peak load, the allocation of power is 100W to each tier, while at idle, the power allocation is 50W and 80W to the two tiers respectively. The power distribution proportion among tiers is not constant.

Changes in workload not only change the application power consumptions but also influence the power

used by the shared components such as cooling equipment. Another dynamic factor is the power capacity available. For example, if environmentally-harvested energy is used, the available supply varies over time. In view of the workload and power capacity dynamics, to maximally utilize the available capacity, the budgeting mechanism must *dynamically adjust the power allocation proportions* across applications and application tiers.

3 System Design

The VPS system for power budgeting is designed to address the challenges described in the previous section. To support multiple distributed applications in a scalable manner, we use a hierarchical approach. The hierarchy is designed to follow the application layout, with the total power budget being divided dynamically among applications, and within the application among the different tiers, following down to the individual VMs comprising those tiers. This hierarchy is independent of the server and rack layout. Power is tracked at the VM level allowing each application to be budgeted independently of others by leveraging VM specific power control knobs (such as VM CPU time allocation). The dynamics of the system, including workload variations and power capacity changes, are handled through feedback controllers that monitor and control the power usage in real time. Application performance is optimized through a combination of control algorithms based on proportional-integral-derivative (PID) and model predictive control (MPC). In this section, we describe the design of the VPS architecture and the control algorithms used.

3.1 Power Budgeting Architecture

Figure 1 shows the overall structure of the VPS system. The white boxes correspond to VPS components while the gray boxes show the underlying physical hierarchy. The VPS control mechanism consists of a multi-level hierarchy where the topmost level is a *data center level controller*[2]. This controller receives the *total power capacity* as its input. The controller allocates it among the *application level controllers* that comprise the next level in the hierarchy. The top level controller is scalable since it monitors and controls only a small number of applications rather than the thousands of individual servers or VMs. Each application may in turn consist of hundreds or even thousands of VMs. The application is thus further divided into tiers and the application level controller monitors and controls only the *tier-level controllers*. The tiers of the application are typically arranged in a pipeline that, as we describe in the detailed controller design, facilitates our method for optimal power allocation among the application components. This approach utilizes the available power more efficiently than following a physical server or rack layout based hierarchy. The tier level controller in turn controls each VM belonging to the corresponding application tier. Within a tier, the constituent VMs are often load balanced and similar in behavior. If a particular tier has a reasonably small number of VMs, the tier level controller can perform a nearly uniform allocation within the tier, and directly command the VM power settings affecting the power consumption. If the number of VMs within a tier is very large, or the VM roles are distinct, further levels may be added to the hierarchy, based on network proximity or VM roles. Without loss of generality, we focus on a three-level hierarchy: data center level, application level, and tier level. VPS operates independent of the physical hierarchy comprising of servers, racks, and power distribution units (PDUs).

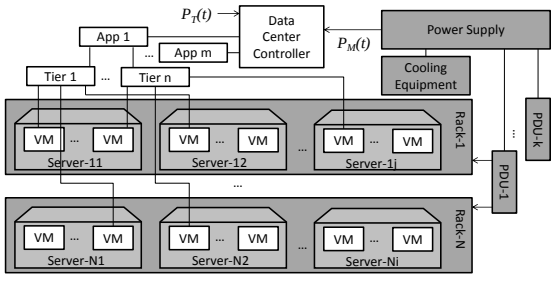


Figure 1: VPS power control hierarchy. The white boxes constitute VPS while the gray boxes represent the physical hierarchy. PDU stands for power Distribution unit.

The above architecture assumes that power can be measured and limits enforced at the application granularity. Measurement of an application's power consumption may not be possible at a physical wire in virtualized servers since the physical server components are shared across multiple applications. The VPS system relies on VM power measurement methods such as [8, 18] that report the individual power usage of each VM on a shared server, as well as, the base power that the hardware platform consumes. The individual VM power measurements are propagated up the VPS hierarchy to obtain the power consumption of each application or application tier.

Actual power reduction can only be realized at the lowest layer that controls the power consuming resource. In our implementation, the resource whose power is varied is the processor since in current platforms, the proces-

[2] A data center may be divided into sections referred to as "colos" where the power infrastructure and backup is separate for each colo: in this case one data center level controller would operate in each colo. The top level controller may be applied at any physical boundary representing an independent unit in terms of application deployment and power constraints.

sor is the resource with the most advanced power management options. By controlling the CPU active time allocated, i.e., the CPU utilization, the power consumption of the processor can be varied from near zero to its peak power. Similarly, DVFS also allows varying the processor power over a large range. We use both these knobs to control power. For memory, power varies directly with the number of IOs performed [8], which can effectively be throttled by the number of CPU cycles allocated. Additional power control knobs can be included in our framework as they become available since it is already designed to use multiple knobs. Applications whose power use does not change with any available power control knob can of course not be throttled. Their power use is measured, similar to idle power and cooling equipment power, and changes are compensated for by VPS controllers.

The application-based hierarchy is more natural from a performance perspective because resource allocation decisions are typically made for an application as a whole and each application has a different business functionality with its own priority, revenue, and QoS expectation. The implementation of such a hierarchy is however more sophisticated since it incorporates knowledge about the application's VM layout and the hypervisor's VM CPU time allocation. As a result, the VPS system operates in the data center management plane rather than in the server motherboard or blade-enclosure based firmware.

The power budget at the highest layer, denoted $P_T(t)$, is an input to VPS. This is the hard constraint that must be satisfied at all times. It could be based on the static capacity built for the facility, or could be dynamic, based on time-of-day based power prices, or amount of environmentally harvested energy [16, 2] available.

3.2 Top Level Controller

The data center level controller, placed the the top level in the design, determines the amount of power allocated to each application. If applications have different priorities, the controller takes those into account.

Naïvely partitioning the power budget $P_T(t)$ among applications, say based on statically assigned shares, does not work well in practice because of the following factors. First, the application workloads are dynamic. An application may not be using its fixed allocation at time t if incoming workload is low. The power allocation mechanism must adapt dynamically, to assign the unused power to another application if needed. Second, the measurement of application power consumption may have some errors. Additionally, a measurable power allocation increase to an application may lead to associated hidden power level increases in shared infrastructure, such as due to non-linear changes in transformation losses across power supplies and changes in cooling load, that are hard to assign to any single application. Such errors directly affect the total power used and must be compensated to satisfy the hard limit of $P_T(t)$.

VPS design uses feedback control to address the above factors. The top level controller receives measurements of each application's power consumption from the respective application level controllers. It also receives the total data center power consumption from hardware instrumentation at the power circuits supplying the servers and cooling equipment. This hardware measurement includes power consumption that is not directly attributed to any specific application VM. The output is the power allocation to each of the applications, at each time instance, that is then enforced by the application level controllers. Figure 2 shows the feedback loop involved.

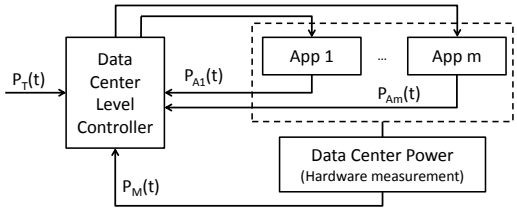


Figure 2: Block diagram of data center level feedback.

The only output action available at this controller is the power allocation and hence the performance objective is only to allocate the maximum possible power, up to the application demands, and minimize the workload throttling. Any controller that can closely track the available power limit and is robust to errors in measurements can be used, and a PID controller is thus appropriate at this layer. Other control algorithms that make optimal decisions by choosing among multiple control knobs are employed in VPS at lower layers. Ad hoc algorithms such as those based on rules that actuate power increases and decreases based on observed consumption levels may become unstable or oscillate as shown in [21]. VPS uses a control theoretic framework that enables stable operation by design, over the range of practical operating constraints (the specific methods used to tune the controllers in our prototype are outlined in Section 3.5).

3.2.1 Application Budgets and Priorities

Note that while the control algorithm adapts total power consumption to operate close to P_T, the PID controller output here is the sum of all the applications' power consumptions. There is no knob available to control this sum; only the power consumptions of individual applications can be affected through their respective application level controllers. The control output is split across the

application level controllers using application priorities as well as the measured application and total hardware power consumptions, as follows.

At power provisioning time, each application is assigned a maximum budget, based on expected usage at, say, 99-th percentile peak load. While in non-virtualized settings, application budgets are typically assigned simply based on number of servers allocated and measured power for those servers when running the relevant application, with virtualization the actual power impact must be profiled on the appropriate infrastructure. When a new application is accepted, its power can be profiled for each type of VM instance used, and extrapolated to number of VM instances at 99-th percentile of peak load. Such provisioning is typically required by a data center before it can accept an application to be hosted. Suppose the assigned power is denoted P^0_{Ai}.

Suppose $\Delta(t+1)$ denotes the desired change in total power at the next time step (PID controller output). To determine the per application power split, the amount of uncontrollable power, denoted $P_U(t)$, is first estimated by subtracting the sum of application power consumptions from the measured total power consumption, $P_M(t)$:

$$P_U(t) = P_M(t) - \sum_{i=1}^{m} P_{Ai}(t) \quad (1)$$

where $P_{Ai}(t)$ represents the power consumption of application i, and m is the number of applications. This uncontrolled portion of power includes all shared infrastructure power as well as errors in application power measurement. The estimate of the total power consumption at the next time step becomes $P_M(t) + \Delta(t+1)$. The power budget available to be allocated to the applications, denoted P_{app}, at the next time step, is estimated as:

$$P_{app}(t+1) = P_M(t) + \Delta(t+1) - P_U(t) \quad (2)$$

This is only an estimate since it does not include the unknown change in $P_U(t)$ at the next time step, and that change will act as an error for the feedback controller, to be compensated as the controller converges. $P_{app}(t+1)$ is distributed among the applications according to the desired priortization policy.

In our implementation, the priortization policy is as follows. The controller allocates power to each application based on its current demand, subject to a maximum of P^0_{Ai}, starting with the highest priority applications. If at any priority level, there is not enough power budget to satisfy all application power demands, then we use *weighted fair sharing* to distribute the remaining power, with weights set proportional to the initial provisioned application budgets. With this policy, lower priority applications are affected first. Similarly, any excess power left over is also assigned using weighted fair sharing to applications with unsatisfied demand (excess power may be available when some applications are below their initial budget). The assigned shares are sent to each application level controller to be enforced. Effectively, priority levels determine the split of P_{app} across applications while the feedback controller tunes the value of P_{app} to meet the target total power.

3.3 Application Level Controller

The VMs comprising an application are typically divided into a number of tiers. Each tier has a different role, and consequently a different power requirement. The application-level controller distributes the application power budget received from the top level controller among each of its application tiers. This controller only communicates with a small number of tier level controllers and is thus scalable in number of VMs.

The controller must determine the correct proportion in which power is allocated to the different tiers. One design option for this controller is to learn a model of power usage across tiers, and use that to determine the appropriate ratio in which power should be split among the tiers. This approach can be used when a detailed model of application performance and resource utilization at each tier can be learned. This is feasible for a specific power control knob at a given workload volume [7]. However, as illustrated in Section 2.3, the best power sharing proportion changes with workload volume. The model may also depend on the power control knob used at the lower layer, such as DVFS or CPU time allocation. Further, the application behavior may change over time with software upgrades. In a virtualized infrastructure supporting multiple applications, with little control over application internals, learning this model is difficult.

VPS design dynamically tunes the power allocations without relying on previously learned models. The key challenge of course is to determine the correct sharing ratio. Our design is based on the observation that the multiple application tiers are arranged in a pipeline, and throttling one tier will directly affect the workload flowing into other tiers. The relationship among power changes at different tiers need not be known a-priori, as long as the pipeline assumption holds. The VPS application-level controller measures the total application power usage but controls only one of the tiers. As the power allocation to the controlled tier is changed, the power consumed by other tiers changes in the right proportion required to serve the throttled workload volume passed on by the controlled tier. This automatically maintains the optimal power sharing proportion.

An experimental illustration of the pipeline assumption is shown in Figure 3, using a two tier application described in Section 3.5. As the power usage of the controlled tier is reduced, the power usage of the uncontrolled tier changes as well. The figure also shows that the ratio of power consumptions is not constant at different power levels and further depends on the lower layer power control knob used (the figure shows two different DVFS levels), implying that learning a model for this relationship would be non-trivial.

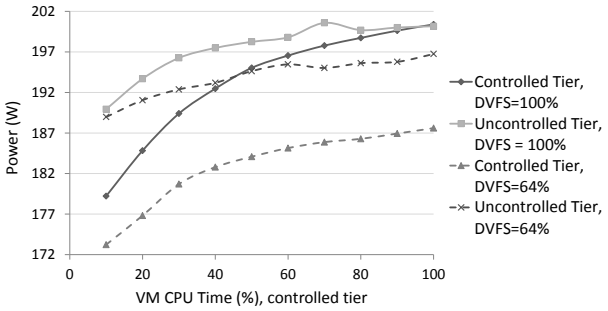


Figure 3: Server power variation with throttling of the controlled tier VM CPU time.

While the pipeline assumption holds in many practical cases and is used in our description, theoretically, the only assumption required is controllability, which is a less stringent requirement. The analysis of controllability conditions is beyond the scope of this paper.

The specific feedback controller used here is based on proportional-integral-derivative (PID) control (Figure 4), governed by:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \qquad (3)$$

where $u(t)$ represents the change in power allocated to the controlled tier, $e(t)$ represents the difference between the desired application power budget and the currently measured application power consumption, and the parameters K_p, K_i and K_d are PID controller parameters for the proportional, integral, and derivative terms respectively. The tuning of the controller parameters follows known control theoretic methods and is discussed in Section 3.5.

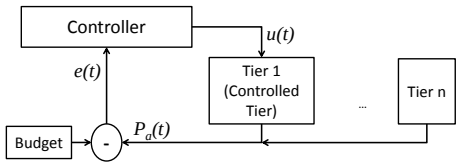


Figure 4: Application level feedback controller.

Conceptually, any of the tiers in the pipeline can be used as the controlled tier. From a practical standpoint, selecting the tier with the largest power variation is desirable as that will provide finer control over the power consumption, leading to lower error in tracking the power budget. The controlled tier can be selected by recording the power variation changes with workload from the power readings provided by each tier.

3.4 Tier Level Controller

The tier level controller, at the controlled tier, controls each of its VMs to track the tier power budget. It communicates with the servers hosting the tier's VMs, to actuate the power control knobs.

The control algorithm at this tier may have multiple power control knobs at its disposal. In our prototype, we use two knobs: VM CPU time allocation and DVFS.

VM CPU time allocation controls the maximum CPU time allocated to a VM, and the processor can enter low power sleep states (also known as C-states) for the unallocated time, reducing the CPU utilization and power consumption [11]. This knob can control the power consumption of an individual VM without affecting other VMs sharing the same processor.

DVFS controls the processor frequency (P-state) to scale CPU power. This knob affects all CPU cores supplied from a single power rail, and thus impacts all VMs sharing those cores.

While we use only CPU-based power knobs, these indirectly influence other components such as the storage subsystem by limiting the workload volume processed and in our experiments we found that power consumption of the storage intensive database tier does vary with CPU power scaling. However, in the future, if the storage subsystem provides direct power control knobs those can be directly used in the VPS framework.

Performance Optimization: Use of multiple power control knobs opens up the opportunity to affect performance. Figure 5 shows the performance of one of the application VMs (StockTrader application, Section 4.1), with different settings of the two knobs. Different types of marks correspond to different DVFS levels while multiple marks of the same type correspond to different VM CPU time allocations at one DVFS setting. The key observation is that a given power level may be achieved at multiple combinations of the two control knobs, yielding different performance levels[3].

[3]The absolute power variation here is small compared to typical server power because the graph only shows the power variation of one VM, and the range of power is restricted to the changes in power of one core. Power is measured in hardware with only one VM allocated. Only change in power is shown.

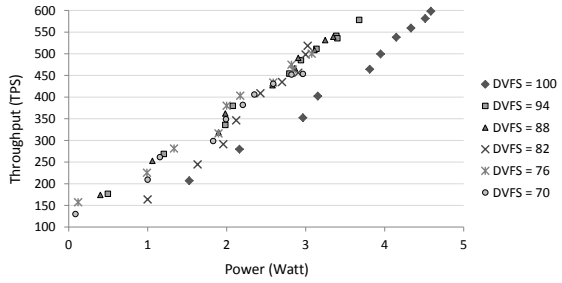


Figure 5: Performance vs power at different DVFS and VM CPU time allocation combinations.

One of the challenges here is to select the power settings for optimal performance. Another challenge is to coordinate the use of the hardware level DVFS knob with power budgeting at application and VM boundaries. A practical constraint for VPS design is that an application performance model and a real time measurement of performance may not be available. This is often the case when data center power and the hosted applications are managed by different entities, and is especially true for cloud platforms.

We study three design options for the tier level controller, making different trade-offs in terms of performance achieved and implementation constraints. All of these designs assume that VMs within a tier are largely homogeneous and load balanced, though they may have small instantaneous variation in their activity.

3.4.1 Open Loop Control

The open loop design assumes that a power model relating the power consumption to the power control knob setting is available for the server hardware. Such models could be learned in-situ using known methods [8]. For instance, if the power knob is VM CPU time, this model may be represented as:

$$P_{VM} = c_{freq} * u_{cpu} \quad (4)$$

where P_{VM} denotes VM power, u_{cpu} is the CPU utilization of the VM, and c_{freq} is a processor frequency dependent power model parameter. Any single control knob can be handled similarly. Multiple simultaneous knobs are considered in Section 3.4.3.

No visibility into application performance is assumed. Only the VM CPU time allocation knob, that acts at VM granularity, is used. VM power allocation is obtained by uniformly dividing the tier power among the tier VMs. For each VM, the assigned power is converted to VM CPU time allocation using (4). The controller is easy to implement and acts instantaneously, but it does not compensate for errors in the model equation (4).

3.4.2 PID Control

Accuracy of the open loop controller can be improved using feedback. Since a feedback controller uses real time power measurements to tune the power setting, it can in fact work even without a power model. VPS uses a PID controller (Figure 6), with one variation that the control output is sent to multiple homogeneous VMs. The control output, $u(t)$, is the VM CPU time allocated to each VM and is assumed to be the same across all VMs within the tier. Small instantaneous differences in VM activity are acceptable since the controller uses only the sum of the powers of all VMs as feedback (small VM variations are averaged out), but the overall VM CPU time to power relationship must be similar for all VMs, implying that a common hardware configuration is used and the software running is the same (since different software functionality can lead to different power consumption even at the same CPU utilization [8]).

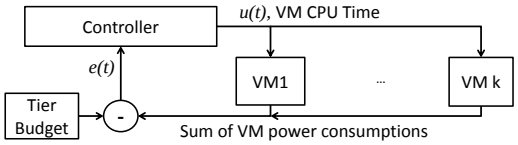


Figure 6: Tier level controller PID feedback loop.

The PID based design provides the advantage of accurate power control. However, since it relies on feedback measurements to reach the desired setting (i.e., tier power budget), it is slower than an open loop control leading to longer control intervals at higher levels of the hierarchy. Also, since it manipulates a single variable, it does not incorporate any notion of optimality for application performance.

3.4.3 Model Predictive Control

The third design option used is based on model predictive control (MPC). MPC allows computing an optimal setting among multiple power control knobs. The optimum is defined in terms of application performance, and hence this design option requires a mechanism to measure application performance.

The cost function optimized by MPC typically consists of two terms: an error term that quantifies the difference from the desired state, and a performance term. The MPC objective function includes not just the current time step but the system state at future time steps, requiring a system model that relates the control knobs to the system state, in this case the target power level and performance. At each time step, the controller solves for the optimal outputs for the next N time steps, applies the solution for only the next time step, and repeats the process to ensure smooth convergence to the desired state.

Cost Function: Suppose $dvfs(i)$ denotes the DVFS, and $v(i)$ denotes the VM CPU time limit, at time step i for a VM. Suppose f_{power} denotes the power model, e.g., equation (4), and f_{perf} the performance model. The cost function, J, is:

$$J = \sum_{i=1}^{N} ||f_{power}(dvfs(i), v(i)) - P_{VM}|| + w \sum_{i=1}^{N} ||f_{perf}(dvfs(i), v(i)) - \alpha_{max}|| \quad (5)$$

where P_{VM} is the VM power to be tracked, α_{max} is the maximum possible performance, and w is a weight that determines the relative importance of the two terms. The first term optimizes the error between the target and predicted power levels, and the second term optimizes performance along the predicted N step control trajectory.

The cost function is minimized to find $dvfs(i)$ and $v(i)$ for best performance. The optimization is solved individually for each VM to keep the size of the optimization search space independent of the number of VMs, ensuring scalability to applications with large number of VMs within a tier.

Hardware Coordination: A block diagram of the control system is shown in Figure 7. The DVFS knob acts at the hardware level and may not respect VM boundaries. Thus, the settings computed above are not applied directly but through a *coordination service*, hosted at each physical server. The service receives DVFS requests from the MPC output for each VM on the server (potentially belonging to different applications) and sets the server DVFS to the highest frequency among the DVFS levels requested. This ensures that no VM is unduly throttled down. The applied DVFS level is reported back. The MPC controllers that receive back a different DVFS setting than the one requested, solve their optimization problem again, with the reported DVFS setting added as a constraint, tuning the VM CPU time knob for the current DVFS setting. The process is repeated at each control iteration yielding the combination of DVFS and VM CPU time allocations that maximizes performance within hardware sharing constraints.

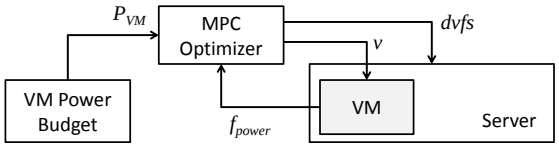


Figure 7: Tier level controller MPC block diagram.

While the MPC based design can yield higher performance than the previous two options, it requires the application performance to be exposed to VPS. Certain cloud platforms such as Microsoft Azure do provide APIs for applications to expose custom performance counters and can be used when available. Power and performance models are also needed. A third consideration is that while the PID controller will always provide a best effort solution within the range of the power settings available, the optimization step in MPC can fail if the optimization is infeasible, and a backup control method may have to be employed. Table 1 summarizes the pros and cons of the above design options.

	Pros	Cons
Open Loop	Fast	Needs power models Higher error
PID	Low error	No performance optimization Slower
MPC	Optimizes performance	Needs system models Needs performance measurement

Table 1: Summary of controller design options.

3.5 Implementation

VPS controllers are implemented as network services on the same physical servers as running the workload. The tier level controller also runs a service in the privileged VM (root VM in Windows Hyper-V) on each physical server to actuate the VM CPU time allocations and DVFS. The network services implementing the controllers also log power and performance data for the experiments. The various parameters and system models needed in the implementation as acquired as follows.

Controller Parameters: The feedback controllers used in the implementation are tuned using known methods from control system design literature. For the PID controllers employed at various layers, the parameters K_p, K_i, and K_d are tuned using the Ziegler-Nichols method [25], on test runs with one of our applications. This method is known to yield robust parameters, keeping the controller stable as workloads change. However, this method does not necessarily yield the fastest convergence or minimum overshoot. Other tuning heuristics available for control system design may be employed as desired. The MPC controller is tuned to operate with a prediction horizon of $N = 1$. Longer time horizons are helpful for ensuring smoother convergence. In VPS, the MPC control is applied only at the individual VM level, where the models are relatively accurate, and hence a short time horizon suffices. The optimization effectively uses the error term as a constraint and maximizes the performance, implying a weight factor w that emphasizes

accurate power tracking over application performance. The detailed optimization and tuning of controller parameters is beyond the scope of this work.

Power and Performance Models: For MPC, the performance model $f_{perf}(dvfs(i), v(i))$ is learned using a test run where each $dvfs(i)$ and $v(i)$ setting is exercised. For each application, this is simply represented as a table with the performance metric listed at each DVFS and VM CPU time setting of the controlled tier. Only a few discrete DVFS levels are available in hardware, and for VM CPU time, nine discrete levels ranging from 10% to 100% are measured. The power model f_{power} is learned using the methods from [8] and is represented using an equation of the form (4). These models need to be learned for each application only if the MPC design option is used. The models depend on hardware used. Large data centers typically have a large numbers of servers for each configuration, and servers are updated in bulk with a single configuration. This means that the models have to be learned on a small number of servers and updated only incrementally.

Additionally, the server infrastructure provides for measuring the total power (from the circuits supplying the servers, cooling, and network equipment). The root VM in each server implements the VM power measurement technique from [8]. The maximum power allowed for each application, denoted P^0_{Ai}, is assumed known and may be determined using the technique described in Section 3.2.1. If multiple applications have different priorities, these are assumed known. In practice, customer facing interactive applications may be assigned one priority level, and batch processing tasks such as MapReduce jobs, data mining, test and development, and internal enterprise applications, could be assigned a second, lower, priority level.

Coordination Across Levels: The controllers at multiple levels are coordinated by setting the control interval of the higher layer controllers to be larger than the convergence time of the lower layer ones. This ensures that the lower layer controller has converged before the higher layer controller receives feedback and actuates, thus avoiding instability. In our prototype, we found the lowest layer controller, when using PID, has a convergence time of 6 seconds and hence, the application level controller uses 6 seconds as its control interval. The application level controller also uses 6 control steps to converge, leading to a control interval of 36 seconds at the data center layer controller. The control algorithm at each layer updates its output at the assigned control interval.

4 Evaluation

4.1 Workloads and Experiment Setup

We use two types of applications for our experiments – an interactive multi-tier application that represents online services subjected to variable user workload, and a set of computationally-intensive batch processing tasks:

StockTrader: StockTrader [17] is an open source multi-tier clustered web application benchmark that mimics a stock trading website, provided for Windows platforms. It is functionally and behaviorally equivalent to IBM WebSphere Trade 6.1 benchmark that runs on other platforms. The application has two significant tiers: a middle tier that implements business logic and a database tier that provides the storage backend. The front-end is a lightweight presentation layer. The incoming requests can be load balanced among multiple VMs hosting the application.

We modified the workload generator provided with the Stocktrader source code to generate workload volume based on a trace file. The application reports its performance in a graphical user interface that we modified to expose the performance as a performance counter sent to the relevant network services implementing the control algorithms in our experiment.

SPEC: We use multiple applications from the SPEC CPU 2006 benchmark suite [15] to represent background jobs that would typically run with lower priority in a data center.

To simulate realistic workloads that vary with time, we use real world data center traces from Windows Live Messenger, an online service with millions of users worldwide. Sample traces from two of its servers are shown in Figure 8. Each instance of the StockTrader application was loaded using a separate data center trace. While the StockTrader application is different from Live Messenger, generating load proportional to production traces helps simulate realistic variations in workload volume.

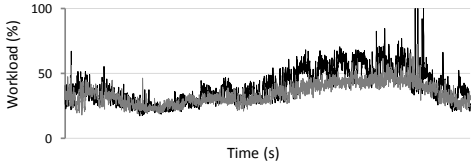


Figure 8: Windows Live Messenger workload trace.

Our testbed consists of seventeen servers, eleven of which host the applications and are subjected to VPS, while the others generate user workload. These are quad core HP ProLiant servers, virtualized using Windows Hyper-V.

Application Deployment: The testbed hosts 4 clustered applications: 3 instances of StockTrader labeled A, B, and C, and a SPEC CPU task set. StockTrader A and StockTrader B are each composed of 13 VMs. StockTrader C is composed of 6 VMs, and the SPEC CPU task set is given 10 VMs. Each VM is assigned one core on one of the quad core servers. StockTrader VMs are allocated to multiple tiers such that each tier reaches high resource utilization at peak load.

Stocktrader B and C are treated as high priority applications while StockTrader A and SPEC are given low priority. The VMs are mixed up across servers such that some servers host both high and low priority VMs while others host VMs only from a single priority level. Each server hosts VMs from more than one application.

Measurements: Power measurements for the hosted VMs are obtained using [8]. This technique obtains a mapping between resource usage, which can be monitored for each VM by the hypervisor, to actual VM power use, since VM power cannot be measured in hardware. Power measurements for the entire testbed are obtained in hardware, using a set of WattsUp PRO [24] meters, connected to each of the servers. This hardware measurement includes the base power consumption of the servers (power consumed when powered on but idle) that is not attributed to any specific VM, and is treated as $P_M(t)$ for equations (1) and (2). Cooling equipment is not part of this testbed. When using MPC at the tier level, the SPEC application's tier level controller is still PID, because the SPEC CPU applications does not expose performance metrics in real time.

Comparison: In addition to the VPS controllers with multiple options from Table 1, we also implemented a power budgeting system that simply follows the hierarchy of the physical layout of the testbed, for comparison. This controller uses only DVFS as its power control knob and operates at the server level, similar to prior works [22]. Servers that are exceeding their allocated budget, i.e., the ones with highest resource usage, are throttled first.

Illustrative Run: We conduct multiple runs with different workload traces and take an average of the measured metrics (5 runs in each experiment). As an illustration, Figure 9 shows the power consumption with and without VPS controllers, for part of a run. Tracking is enforced during time intervals where uncontrolled consumption (dashed line) is above the tracked power level (solid black line). Only two of the controllers are shown for clarity. The controllers do exceed the tracked power level at times, leading to tracking errors. Also, even when the uncontrolled curve exceeds the tracked power level, implying that the workload is high, the controllers sometimes leave power unused below the tracked level, taking an unnecessary performance hit.

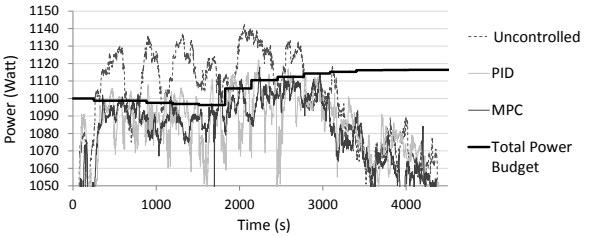


Figure 9: A workload run against various control systems with dynamic power budget.

The figure also illustrates the use of a time varying power capacity availability, as may be useful in scenarios where the utility power is supplemented with environmentally harvested power [16, 2].

4.2 Results

The performance metrics of interest are: total (data center level) and application level *power budgeting errors*, application performance *differentiation* (ability to operate interactive applications on shared infrastructure with low priority tasks), and *performance* achieved within the power budget.

4.2.1 Power Budgeting Errors

Error is defined as the excess power consumed above the assigned power budget, normalized by the power budget:

$$TrackingError = MAX\left\{\frac{P_M(t) - P_T(t)}{P_T(t)}, 0\right\}$$

where $P_M(t)$ represents the measured data center power consumption. Consumption below the target level may result from workload being low or the controller being overly conservative. Being overly conservative is not an error from a budgeting perspective, but penalizes the controller in terms of achieved application performance.

Figure 10 shows the average and standard deviation of the mean error across all experiment runs, for each design choice. The PID-based system has higher error because the PID controller has higher overshoots during its convergence time, compared to MPC and Open Loop systems, and is as expected. Higher oscillations for PID compared to MPC were also seen in [21]. The physical hierarchy based controller has higher error primarily because the control knob it uses, DVFS, is not as fine grained as VM CPU time allocation. Processors have only a few discrete DVFS levels as opposed to CPU time allocation that can varied in fine grained steps. Overall however, each of the design choices yields fairly low error and the choice will thus depend on the other implementation constraints or performance considerations.

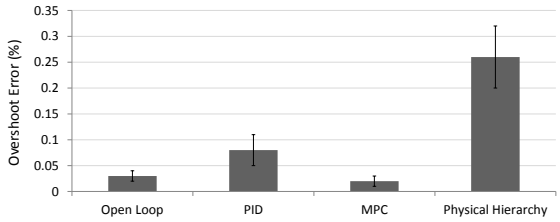


Figure 10: Total power errors of each control system.

It is worth noting that the overall error is low even when using open loop control, because some of its error is compensated by higher layer controllers. The error in open loop control is more apparent at the lower layers. Figure 11 shows the mean error for each hosted application (ST-x refers to StockTrader-x). Here, the PID and MPC based systems have similar application power errors, and both fare better than the open loop VPS system. ST-A and SPEC being the lower priority applica-

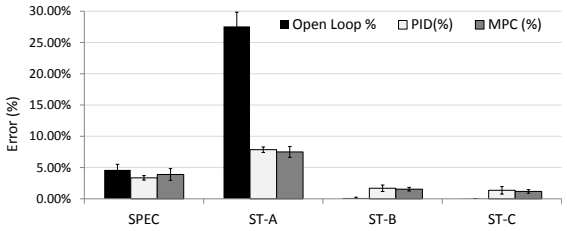


Figure 11: Application power enforcement errors.

tion are subject to greater power reduction. However, the open loop controller has a poorer power model for StockTrader applications than SPEC (the power model accuracy can vary across applications since different applications may use resources differently [8]). As a result, it underestimates the power consumption of ST-A, and does not throttle it sufficiently, leading to high error. Due to this error, the higher layer controller reduces the total power available to all applications, resulting in the higher priority applications, StockTrader B and C, seeing lower budgets under open loop design than their actual limits in other designs. These are throttled more than necessary and stay well below the target level, resulting in tracking error being virtually eliminated for B and C. The physical hierarchy based controller does not apply to individual applications and is omitted in this figure.

4.2.2 Power Differentiation

VPS is designed to respect application priorities and QoS constraints in a shared infrastructure. Figure 12 shows the differentiation between different applications enabled by VPS. Power reduction compared to uncontrolled operation is shown, normalized by the uncontrolled consumption.

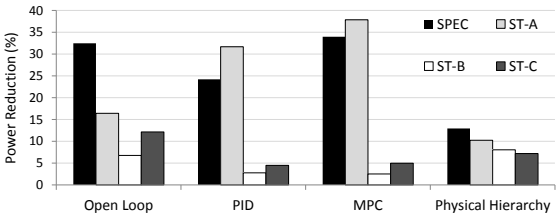


Figure 12: Average application power reduction under each control system. SPEC and ST-A are the lower priority applications and ideally only these two should have their power reduced.

The physical hierarchy based controller, which operates at the hardware level without consideration of application VM boundaries, is unable to differentiate between applications: higher and lower priority applications have their power reduced by similar amounts. In contrast, the PID and MPC based VPS systems show marked application power differentiation.

The open loop system does differentiate, and SPEC is throttled by similar amount as with MPC and PID. However, the power model used does not work as well for the StockTrader applications and we see that StockTrader A is throttled much less, causing the higher priority applications to be throttled more, as explained with Figure 11.

4.2.3 Application Performance

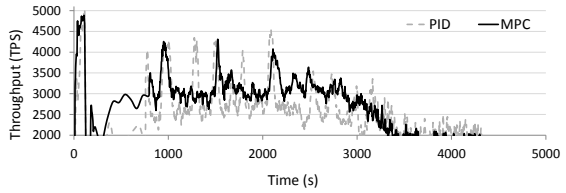
We saw above that both the PID and MPC based VPS systems can perform appropriate application differentiation and achieve low errors. The distinguishing feature of MPC however, is its ability to improve application performance by intelligently selecting the appropriate combination of power settings that yields higher performance for a given power level.

An illustration of this effect is shown in Figure 13, which shows the throughput and response time achieved by StockTrader A, under both MPC and PID based approaches, for the same power budget. MPC yields higher throughput and lower response times, showing a noticeable performance advantage.

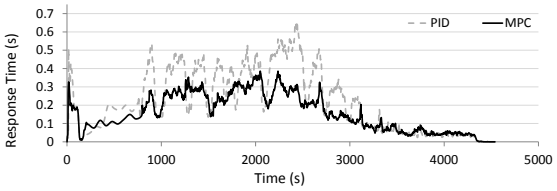
Quantitatively, the performance difference is measured as follows. The degradation, δ, in performance is defined as the fractional reduction in performance compared to when run with unlimited power:

$$\delta = \left| \frac{Perf_{unlimited} - Perf_{VPS}}{Perf_{unlimited}} \right|$$

for both response time and throughput. For each experiment run, we calculate the mean degradation for each application. The degradations in throughput and response time are compared in Figure 14. In each case, the MPC based system shows lower performance degradation, implying higher performance. StockTrader B and C being



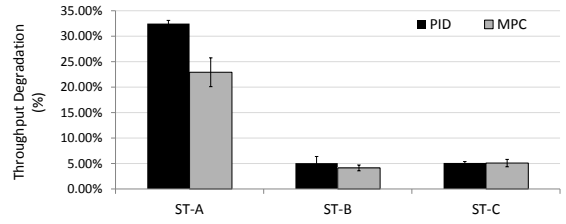
(a) Application Throughput.



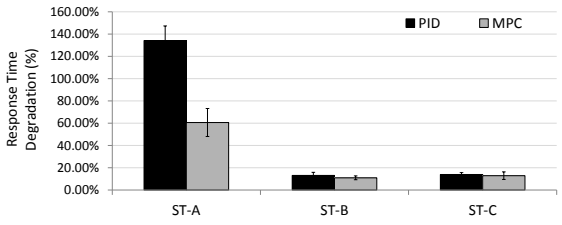
(b) Application Response Time.

Figure 13: Application performance of a low priority (throttled) application, for MPC and PID, with the same power budget.

higher priority applications, are not affected much in either PID or MPC, but StockTrader A shows a marked performance advantage for using MPC.



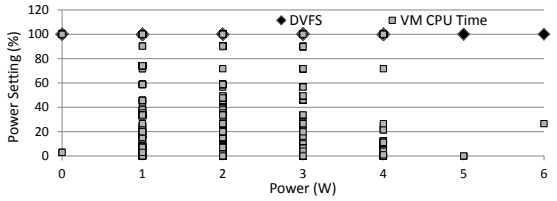
(a) Throughput Degradation



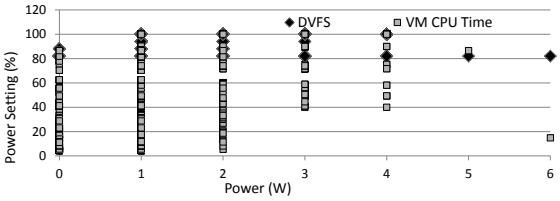
(b) Response Time Degradation

Figure 14: Performance degradation: MPC vs PID. The high priority applications (ST-B and ST-C) are not throttled much but ST-A suffers lower degradation when throttled using MPC than with PID for the same power budget.

We also noted in Section 3.4.3 that MPC is restricted in its use of the DVFS control knob because the hardware is shared among multiple VMs. While the VM CPU time limit can be applied to any VM, the DVFS knob can only be used when allowed by the coordination service. To study the impact of coordination, we track the DVFS and VM CPU time limit settings used at two of StockTrader A's VMs. One of these VMs, labeled VM1, is co-located with VMs from higher priority applications while the other, VM2, is placed on a server where all other VMs belong to lower priority applications. Figures 15(a) and 15(b) show the power control knob settings used by these two VMs during the MPC experiment, at different times during an entire run. We see that VM1 is unable to use the DVFS knob (DVFS is always 100%) because the other VMs on that server require the highest DVFS setting. VM2 on the other hand does use multiple DVFS levels, and its spread of VM CPU time limits is thus different from VM1.



(a) VM1 power settings



(b) VM2 power settings

Figure 15: Power control knob settings used for VM1 (co-located with high priority application VMs) and VM2 (co-located with low priority application VMs).

The performance advantage shown by MPC in Figures 14(a) and 14(b) is obtained using the DVFS knob only in cases where the VM placement allowed its use. The performance advantage may be higher in scenarios where all low priority VMs do not share servers with higher priority VMs.

5 Discussion and Future Work

VPS enables performance aware power budgeting in multi-application virtualized scenarios. The system can be extended to incorporate additional features and application scenarios, as follows.

Server Shutdown: In this paper, we only used two power control knobs: DVFS and VM CPU time allocation. Both these knobs can be applied in real time with low latency. However, these knobs only influence the portion of server power consumption that varies with processor power settings. A significant portion of server power, as high as 50-60%, is spent to simply power up a server, and is referred to as idle power. Therefore, an effective means to reduce power is to shut down some

servers, instead of throttling down servers that are powered up. Consider as a toy example, a set of 10 servers, each of which consumes 50W at idle and 100W at peak load (i.e., server power increases from 50W to 100W as CPU utilization increases from 0 to 100%). The total power consumption is 1000W at peak load. Suppose a reduction of 250W is desired. One option is to reduce the CPU utilization of each server from 100% to 50%, reducing the power level of each server to 75W. The number of CPU cycles available is reduced by 50% in this case. Another option is to shut down three of the servers, reducing the power by 300W but reducing the CPU cycles by only 30% (instead of 50%). The second approach achieves part of its power reduction by eliminating idle power of three servers and can offer higher performance.

Clearly, exploiting server shutdown as a power control knob has a performance advantage. However, server shutdown has high latency. Also, since it is a hardware level knob, coordination among all VMs located on a server to be shut down is required. Commercial products that can automatically migrate or shutdown VMs and servers as resource utilization changes are already available [20]. Incorporating such techniques into application power budgeting presents interesting research challenges and will likely yield significant performance benefits when power throttling is required for longer time durations.

Additional Applications: We explained the choice of control intervals in Section 3.5. The latencies achieved are acceptable for over-subscription with a static power capacity, since the only dynamics come from user workloads and these vary only gradually over the course of a day [1]. Such latencies are also acceptable for additional power budgeting scenarios. VPS may be used to control power usage in a multi-application server cluster powered wholly or in part from environmentally-harvested energy, such as solar power, since it varies relatively slowly. VPS may also be employed when the data center wishes to change its power budgets with demand response based or time of day based power prices. The power prices are adjusted for periods of at least an hour or 30 minutes in most electricity markets, allowing ample time for controller convergence.

6 Related Work

Several prior works have considered the problem of power and performance control of data center servers. Power budgeting for a single server has been considered in [9]. Multi-server power budgeting, sometimes referred to as power shifting, has also been discussed [23, 21, 3, 22]. The power controllers proposed in [21, 23] distribute power proportional to CPU utilization in a cluster of servers. In [3], workload differences among multiple nodes are used to allocate different power limits. In [22], a hierarchical approach is used where the controller hierarchy follows the physical server and rack layout. In all these works, only DVFS is used as the power control knob and application differentiation is not considered. The use of multiple power control knobs is also not considered. The controller centrally measures and actuates each server, limiting scalability. Optimal allocation of available power to maximize performance was also considered in [4], where a choice was made between the number of active servers and their processor frequencies. A single application running on homogeneous servers was considered and power allocations were made centrally. We extend the above works to allow multiple applications sharing a common server infrastructure. We also design a method to select an optimal combination of power settings when multiple options exist for achieving the same power level, rather than always using DVFS. We further adapt power allocations dynamically across applications and application tiers to improve performance.

The performance of multi-tier applications has also been considered in [7, 10]. The method in [7] tunes the power settings at each tier to meet an overall performance objective by determining coordinated frequency levels for each tier. In [10], one controller is used to set the processor frequency of one of the tiers to meet performance requirements and another controller tunes the frequency of the other tier to minimize overall energy. These methods require detailed performance models across multiple tiers. We optimize performance across multiple tiers using low overhead mechanisms that do not require learning multi-tier performance models. Our methods work with dynamic workloads and can also use multiple power control knobs.

Partitioning of power due to limitations of power distribution may also lead to inefficient operation because unused power capacity in one part of the data center cannot be delivered to other parts. Solutions to this problem have been discussed before [13]. We assume that such solutions have been deployed, and the distribution infrastructure is not a limiting factor.

In addition to the above works, several others have addressed various related aspects of power control. Coordination of multiple controllers for joint objectives of power capacity, energy consumption, and thermal management was presented in [14]. Our solution addresses multi-application scenarios with dynamic workloads and application performance optimization. Design time analysis of coordinated controllers for detecting unwanted positive feedbacks and instability was presented in [6]. We use multiple coordinated controllers in a hierarchy such that they do not lead to positive feedbacks and ensure stability through known methods. Design time

methods for efficient power provisioning, based on statistical profiling, have also been studied [5] but are complementary to our work. VPS power tracking methods are designed to be employed at run time, after the design time provisioning limits have been determined. Modeling and control of application performance and resource usage in a virtualized infrastructure has also been considered before [12]. Our focus is specifically on power tracking, with appropriate mechanisms for performance differentiation and optimization.

We also consider several practical aspects not previously considered. For instance, the platform providing and controlling the power limits has very limited visibility into the application performance metrics. This is especially true for cloud environments where the applications may not be owned by the same entity that manages the data center and its power usage. Further, the workload for one application may change, causing the power availability for other applications to change and hence we dynamically adapt to such changes. We also do not assume that detailed models for power distribution across multiple application tiers can always be learned.

7 Conclusion

We presented a power budgeting system, VPS, for virtualized data centers hosting multiple applications. VPS can significantly improve the power capacity utilization by providing effective power budgeting in multiple scenarios including over-subscription, energy harvesting data centers, and variable power pricing. VPS allocates available power efficiently among multiple applications sharing the same servers and adapts to dynamic workload variations. The pipelined organization of large scale applications into tiers is used to automatically distribute power among the application tiers in appropriate proportions. Multiple power control knobs are exploited for optimizing performance. The algorithms used are based on control theoretic techniques to help ensure stable and robust operation. VPS offers multiple implementation options to adapt to practical design constraints such as lack of detailed system models and limited visibility into application performance.

8 Acknowledgments

The authors are grateful to Prof Xenofon Koutsoukos (Vanderbilt University) for his insightful comments.

References

[1] CHEN, G., HE, W., LIU, J., NATH, S., RIGAS, L., XIAO, L., AND ZHAO, F. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI* (2008), pp. 337–350.

[2] CLIDARAS, J., STIVER, D. W., AND HAMBURGEN, W. Water-based data center. US Patent Application, February 2007.

[3] E. FEMAL, M., AND W. FREEH, V. Boosting data center performance through non-uniform power allocation. In *ICAC* (2005).

[4] GANDHI, A., HARCHOL-BALTER, M., DAS, R., AND LEFURGY, C. Optimal power allocation in server farms. In *SIGMETRICS* (2009).

[5] GOVINDAN, S., CHOI, J., URGAONKAR, B., SIVASUBRAMANIAM, A., AND BALDINI, A. Statistical profiling-based techniques for effective power provisioning in data centers. In *EuroSys* (2009).

[6] HEO, J., HENRIKSSON, D., LIU, X., AND ABDELZAHER, T. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *RTSS* (2007).

[7] HORVATH, T., ABDELZAHER, T., SKADRON, K., AND LIU, X. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput. 56*, 4 (2007), 444–458.

[8] KANSAL, A., ZHAO, F., LIU, J., KOTHARI, N., AND BHATTACHARYA, A. A. Virtual machine power metering and provisioning. In *SoCC* (2010).

[9] LEFURGY, C., WANG, X., AND WARE, M. Server-level power control. In *ICAC* (2007).

[10] LEITE, J. C., KUSIC, D. M., AND MOSSÉ, D. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *ICAC* (2010).

[11] NATHUJI, R., ENGLAND, P., SHARMA, P., AND SINGH, A. Feedback driven qos-aware power budgeting for virtualized servers. In *FeBID* (2009).

[12] PADALA, P., HOU, K.-Y., SHIN, K. G., ZHU, X., UYSAL, M., WANG, Z., SINGHAL, S., AND MERCHANT, A. Automated control of multiple virtualized resources. In *EuroSys* (2009).

[13] PELLEY, S., MEISNER, D., ZANDEVAKILI, P., WENISCH, T. F., AND UNDERWOOD, J. Power routing: dynamic power provisioning in the data center. In *ASPLOS* (2010).

[14] RAGHAVENDRA, R., RANGANATHAN, P., TALWAR, V., WANG, Z., AND ZHU, X. No "power" struggles: coordinated multi-level power management for the data center. In *ASPLOS* (2008).

[15] SPEC CPU2006. `http://www.spec.org/cpu2006`.

[16] STEWART, C., AND SHEN, K. Some joules are more precious than others: Managing renewable energy in the datacenter. In *HotPower* (2009).

[17] .NET StockTrader Sample Application. http://msdn.microsoft.com/en-us/netframework/bb499684.aspx.

[18] STOESS, J., LANG, C., AND BELLOSA, F. Energy management for hypervisor-based virtual machines. In *USENIX Annual Technical Conference* (2007).

[19] URGAONKAR, B., SHENOY, P., AND ROSCOE, T. Resource overbooking and application profiling in a shared internet hosting platform. *ACM Trans. Internet Technol. 9*, 1 (2009), 1–45.

[20] VMWARE. VMware distributed power management concepts and use. http://www.vmware.com/files/pdf/DPM.pdf.

[21] WANG, X., AND CHEN, M. Cluster-level feedback power control for performance optimization. In *HPCA* (2008).

[22] WANG, X., CHEN, M., LEFURGY, C., AND KELLER, T. W. Ship: Scalable hierarchical power control for large-scale data centers. In *PACT* (2009).

[23] WANG, X., AND WANG, Y. Coordinating power control and performance management for virtualized server clusters. *IEEE Transactions on Parallel and Distributed Systems 99* (2010).

[24] WattsUp PRO ES. https://www.wattsupmeters.com/secure/index.php.

[25] ZIEGLER, J., AND NICHOLS, N. B. Optimum settings for automatic controllers. *Transactions of the ASME 64* (1942), 759–768.