Joulemeter: Virtual Machine Power Measurement and Management

Aman Kansal, Feng Zhao, Jie Liu
Microsoft Research

Nupur Kothari
University of Southern California

Arka Bhattacharya
IIT Kharagpur

ABSTRACT

The importance of power management has led to most new servers providing power usage measurement in hardware and alternate solutions exist for older servers using circuit and outlet level measurements. However, the power measurement and management capability is severely handicapped when the servers are virtualized because virtual machine (VM) power cannot be measured purely in hardware. We present a solution for VM power metering. We use low-overhead power models to infer power consumption from resource usage at runtime and identify the challenges that arise when applying such models for VM power metering. We show how existing instrumentation in server hardware and hypervisors can be used to build the required power models on real platforms with low error. The entire metering approach is designed to operate with extremely low runtime overhead while providing practically useful accuracy. We illustrate the use of the proposed metering capability for VM power capping leading to significant savings in power provisioning costs that constitute a large fraction of data center power costs. Experiments are performed on server traces from several thousand production servers, hosting real-world applications used by millions of users worldwide. The results show that not only the savings that were earlier achieved using physical server power capping can be reclaimed on virtualized platforms, but further savings in provisioning costs are enabled due to virtualization.

1. INTRODUCTION

This paper considers power management in data centers. With the emergence of online applications and services, data centers are serving an increasingly greater fraction of our computational needs. The importance of power management for data centers is well known and most computational platforms now support multiple power management options. One of the key components of power management in such large scale computational facilities is the visibility into power usage that is used to make automated and manual power management decisions, including power over-subscription and server capping. Modern server hardware has built-in power metering capabilities and several solutions exist to monitor power for older servers using power distribution units (PDUs) that measure power at the outlet. This capability is, however, lacking in virtualized platforms that are increasingly being used in data centers. The use of virtual machines (VMs) allows safe isolation of multiple co-located workloads, enabling multiple workloads to be consolidated on fewer servers, and results in improved resource utilization. However, the lack of visibility into VM power usage takes away the many advantages of server power metering that were available without virtualization.

In this paper we propose a mechanism for VM power metering, named Joulemeter, and show how it enables virtualized platforms to reclaim some of the power management advantages of physical servers. As an example, this mechanism can enable power caps to be enforced on a per-VM basis, resulting in a significant reduction of power provisioning costs. Not only are the savings that existed due to power measurement and capping of physical servers reclaimed, but additional savings are realized due to capping with virtualization. Experiments on real-world workloads from several thousand production servers show reclaimed savings in provisioned server power ranging from 13% to 27%. Also, additional savings of 8% to 12% (Section 6) are realized due to the more aggressive over-subscription in the virtualized world enabled by the tighter control and safety mechanism made feasible by Joulemeter. Further, the energy metering mechanism enables other power management tasks, such as managing VM migrations among different data center power circuits to keep each circuit within allocated capacity and providing power information to applications within VMs so that they may better manage their cost and performance. Such fine-grained visibility is also likely to benefit shared cloud environments such as EC2 [5], by making the hosted applications aware of their energy footprints. The proposed capability allows VMs to use power management strategies previously developed for applications [35, 17]. The metering mechanism may also be leveraged for thermal management, since power usage directly impacts heat generated.

Clearly, a hardware power measurement instrument cannot be connected to a VM. In principle, VM power can be metered by tracking each hardware resource used by a VM and converting the resource usage to power usage based on a power model for the resource. However, in practice this approach involves several challenges since accurate power models are not computationally feasible for run time use and may not work when using only realistically available instrumentation. We address these challenges and design a power metering mechanism for existing hardware platforms and hypervisors without requiring additional instrumentation of application workloads.

1.1 Contributions

Specifically, we make the following contributions:

First, we present a solution, named Joulemeter, to provide the same power metering functionality for VMs as currently exists in hardware for physical servers. The solution uses power models in software to track VM energy usage on each significant hardware resource, using hypervisor-observable hardware power states. We select practically feasible power models and show how they can be learned using only existing instrumentation.

Second, we experimentally evaluate the accuracy of the energy metering mechanism using benchmark applications on commonly used platforms including rack mounted servers and a mobile laptop. We discuss the impact of various choices of using hypervisor-observable events and additional hardware-specific events, along with an approach to improve accuracy. While prior work has used performance counters for metering full system power, we address the additional challenges that are faced when applying such approaches to VM energy metering. The design keeps the runtime overhead of the proposed energy metering system very low.

Third, we use Joulemeter to solve the power capping problem for VMs. This enables significant reductions in power provisioning costs. Power provisioning costs are well known [7] to be an important concern, and solutions based on server power capping have not only been proposed [8, 23] but are already available commercially [13, 15]. However, these existing server power capping techniques cannot directly be applied to virtualized environments. Joulemeter reclaims the savings from server capping that would have been achieved in non-virtualized environments, and also offers additional savings that are achievable only with virtualization, due to VM migration capability and temporal multiplexing of VM peak loads. Actual savings are evaluated through experiments using traces from commercial production servers. The impact of metering error is considered in the provisioned power reduction mechanism.

Finally, we discuss power management techniques that can be realized if the proposed VM energy metering capability is available, including its applicability to previously proposed application energy management strategies [35, 17] that can be extended to VMs.

Most techniques discussed are also applicable to metering the energy usage of individual applications running on a computer system, and may be applicable to battery power management for laptops.

1.2 Related Work

Energy Metering: Energy measurement for VMs was considered in [32]. That work assumes that the correct power usage of each hardware resource in different usage modes is provided by its device driver. Accuracy of power values was thus not a concern. We develop methods that work with realistic platforms and build all required power models based on existing instrumentation in the hypervisor and the hardware. We consider the design choices involved and experimentally study the resultant errors and overheads. Further, we also present a power over-subscription and capping solution for data center power provisioning.

Additional closely related work is found in tracking application energy usage [29, 35, 9] that, in concept, can be compared to VM energy usage. Some of these methods [29, 9] assume that the energy usage at a current instance can be attributed to the thread that has the active context on the processor and ignore issues of asynchronous activities such as buffered IO and parallel tasks on multi-core processors. This results in high estimation error as already shown in [35]. However, the main focus of the work in [35] was on application energy budgeting and the power metering itself used data-sheet numbers for key hardware components. All required software instrumentation was assumed to be available. Our goal is to build the relevant power models using available instrumentation, and extend the metering capability to VMs. We also evaluate errors on standard benchmarks. We discuss several VM energy cost reduction opportunities, in addition to the application energy management scenario considered in [35] that can potentially be applied to VMs instead of applications.

Power Modeling: Power models of varying complexity and accuracy have been proposed before for several hardware resources including processors [33, 2, 19, 30, 4, 28], memory [26, 20], and disks [34, 21]. We focus on applying such models to VM energy metering. This involves adding the required monitoring capabilities for VM energy tracking, selecting the appropriate power models with regards to their runtime overhead and accuracy, and using these models on realistic hardware with practically feasible instrumentation.

Power usage of full systems has been explored before through software-based modeling [6, 7, 27, 3]. While those methods are very useful for the platforms they were developed for, on most newer platforms, the server hardware already measures full system power usage and makes it accessible through manufacturer specific tools [14, 15] or even open interfaces in emerging standards [18]. Our goal is to provide visibility into VM power usage.

Energy models based on extensive hardware and software instrumentation have also been considered [10, 31, 24]. The Quantto [10] approach is well-suited to the platforms targeted in that work: low power embedded systems. The source code was instrumented to track all application and system activities. Such instrumentation is impractical in larger systems due to the complexity of software stacks used consisting of many modules, potentially from multiple vendors. Several issues such as monitoring memory activity, or buffering of disk IO by the OS, do not arise in Quantto but are significant in larger systems. The LEAP platform [31] presents a system where each hardware resource is individually instrumented in hardware to track energy usage. The method in [24] used laboratory instrumentation for hardware power metering. Our focus is on methods that do not depend on extensive instrumentation and largely leverage available instrumentation, making them significantly easier to adopt.

2. POWER METERING CHALLENGES

Existing hardware instrumentation, such as motherboard or power supply based power sensors, allow measuring a physical computer system's power consumption. While currently accessible only through vendor specific tools [14, 16], open access is likely to become feasible through adoption of standards such as DCMI [18]. On battery powered laptops, open access is already possible using the smart battery interface accessible via the ACPI standard. Our goal is to convert such full system power measurement capability into virtual machine (VM) energy usage.

While no hardware power measuring device can be connected to an individual VM, we can think of the following two approaches to measure its power usage:

Dedicated run with hardware measurement: We could host the VM alone on a server, measure the server power usage and subtract the server's measured idle power usage to obtain the VM's power usage. This could be repeated for all servers the VM will be used on. If the VM power usage over time stays the same for every run, this would work. But in practice, the VM behavior will change significantly from one run to another depending on user workload and which other VMs are running alongside.

Infer energy from resource usage: We can, in principle, track the resources used by each VM in software and then convert the resource usage to energy by leveraging the power models of individual resources.

We consider the second approach as more feasible and pursue it in more detail. For this approach, we need to track the usage for all the hardware resources, such as the CPU cores, disk arrays, memory banks, network cards, and graphics cards, to determine what power state the resource was in and what VM was it working for. Figure 1 represents this approach, depicting some of the hardware resources and their usage by different VMs over time.

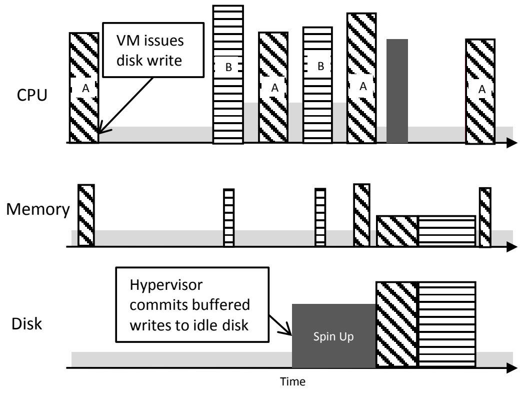

Figure 1: Metering individual VM energy usage.

The rectangles labeled A and filled with the corresponding pattern represent the times when VM A used the CPU, disk, and memory respectively. The height of a rectangle corresponds to power draw during that duration, depending on the power state of the resource. For instance, for the CPU, the power state could depend on the DVFS level used and the processor subcomponents exercised by the workload. If we can correctly label each resource according to which VM was using it and also measure the power draw during that duration (typically a few microseconds), we can compute the *active* energy usage of each VM. The active energy may be defined as the energy consumed by a resource when working on behalf of some VM or the system. For instance, the active energy used by VM A is the summation of energies under the diagonally hashed rectangles.

In addition to the active energy, the system also uses some *idle* energy, shown as light gray shaded regions, and some *shared* system energy, shown as dark gray shaded regions. For the CPU, the idle energy could include the energy spent in a sleep state for a small duration between active states. The shared energy includes energy used by a resource when the work performed benefits multiple VMs, such as the energy spent on disk spin-up before IO operations from multiple VMs. Some of the idle energy is spent regardless of whether any VM is running or not while part of it is affected by the presence of VMs. For instance the time durations between VM idle periods may affect whether a deeper sleep state is used by the processor or not during the idle time, such as depicted using varying heights of the light gray rectangles. The idle and shared energies can be reported separately or divided among VMs either equally or proportional to their active energy usage, depending on how the metering information is to be used. For the power capping application illustrated later, reporting separately works best.

Considering the previous figure, tracking VM energy boils down to two challenges:

1. *Power measurement at microsecond granularity:* The height of the rectangles (instantaneous power) in Figure 1 must be determined every few microseconds for each resource, since the VM using the resource may change every few microseconds (depending on time quanta used for context switching) causing the component usage and power state to change.

2. *Label resource use per VM:* We must determine which VM was responsible for using each resource. This step should not require the application source code running inside a VM to be instrumented, since the platform developers may not have access to it and requiring application developers to instrument their source code is unlikely to scale in practice.

To address the first challenge, we leverage *power models* that relate the software-observable state of a resource to its power usage. If the state changes can be observed at microsecond granularity, we can infer the power usage at that fine time scale. Observing the power states is non-trivial and several approximations must be made. For instance, the resource states, such as the clock gating of sub components within the processor or exact nature of mechanical motions within a disk array, may not be completely known, and the hypervisor may not have visibility into power states of certain hardware resources within the platform, such as hardware-controlled device power states or graphics processor activity.

After observing the resource state, the next step is to determine the power usage in that state using its power model. However, power models are not readily available. The following methods could be employed to obtain power models:

1. Let the device drivers provide the power usage in different resource states [32] but existing systems do not support this.

2. Use the data sheets of the hardware resources [35]. However, detailed power data for different power states is rarely available in data sheets. If data is available for only some of the resources, the power used by the whole system cannot be determined.

3. Build the power models in situ. We pursue this approach and design the metering method to observe the resource states and learn power models using available platform instrumentation.

To address the second challenge, we leverage the knowledge that the hypervisor has regarding scheduling of resources for VMs. Again, complete visibility into all resources is lacking and trade-offs in instrumentation overhead and accuracy must be made.

3. JOULEMETER SYSTEM DESIGN

The largest dynamic energy consuming resources in a computer server (without displays) are the processor, memory, and disk. The server of course has a non-trivial idle energy consumption, often exceeding 50% of its peak power, but the idle energy is relatively static and can be measured in hardware. The energy impact of the VM can thus be considered in terms of the dynamic energy used.

To visualize the energy impact of the processor, memory, and disk subsystems in a realistic data center server, consider the workloads and power data shown in Figure 2. The figure shows the power consumption measured in hardware for a Dell PowerEdge server with two quad core processors, 16GB RAM, and two 1TB disks. The figure also shows the processor, memory and disk utilizations over time. The workload shown was specifically generated to first exercise the processor (denoted Phase I in the figure), then the memory (Phase II), followed by the disk (Phase III), and finally a combination of all three resources. The resource utilizations and power data are aligned in time. It is easy to see the increase in power with varying processor utilization in phase I. Similarly, phases II and III show the energy impacts of memory and disk respectively. The dynamic energies of these three resources under this workload are shown in Figure 3, found using the power models discussed next.

Prior work has proposed various power models for deriving full system energy usage [2, 6, 7, 3] and some of these models have

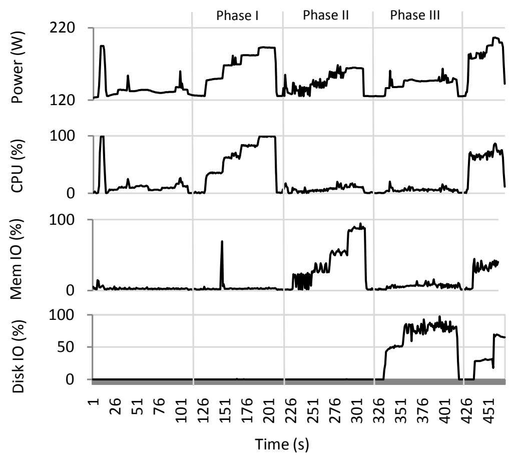

Figure 2: Power impact of the three major system resources.

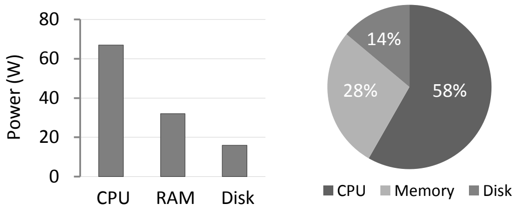

Figure 3: Relative power impact of different resources.

been compared in [27]. The models used are linear. As we see in experiments and as is consistent with prior work, the linearity assumptions made in these models do lead to errors. The magnitude of errors was small compared to full system energy but is much larger compared to the energy used by an individual VM. Also, the errors reported were averaged across multiple workloads but can be higher on specific workloads. We discuss methods to overcome those errors without requiring extensive additional instrumentation. Also, we make the model DVFS-aware by using a piecewise linear model where different coefficients are used for every frequency. The design of the power models is discussed next.

3.1 Practical Power Models

3.1.1 CPU

The power usage of the CPU depends on several factors, such as the subunits within the processor that are active, specific instructions executed, on-chip cache usage, frequency used, denoted by *P-state*,[1] if the processor has dynamic voltage frequency scaling (DVFS), and sleep state usage over time. An accurate power estimation considering these factors can be achieved using a cycle accurate simulator. However, that requires a complete architectural model of the processor, and has high processing overheads, making it unsuitable for runtime VM metering.

A lighter weight alternative is to track processor states that capture the major power usage changes. Suppose there are m P-states, denoted $P0, P1, ..., Pm$ representing progressively lower frequencies. When the processor is not actively executing any thread, it enters a low power sleep state, referred to as a *C-state*. By convention, $C0$ is used to denote the active state and $C1, C2, ..., Cn$ represent progressively deeper sleep states, where n is the number of sleep states (typically $n \in \{3, ..., 6\}$). Suppose the instantaneous power used by the processor at time t is $P_{cpu}(t)$. Assuming that the power depends on only the P- and C-states, it can be denoted by a constant in each different combination of frequency $p(t)$ and sleep state $c(t)$ at time t:

$$P_{cpu}(t) = k_{p(t),c(t)} \tag{1}$$

where $k_{p,c}$ represents the power used in P- and C-state combination p, c. If one could measure $P_{cpu}(t)$ with every P- and C-states change, learning the values of $k_{p,c}$ would be easy. Measurement at microsecond granularity is feasible on specially instrumented motherboards but not on realistic platforms. Instead, power can only be measured at coarser time granularity, of the order of a second. Hence, we integrate the above power model over a longer time duration, T, say:

$$\int_0^T P_{cpu}(t)dt = \sum_{p,c} k_{p,c} * \Delta T_{p,c} \tag{2}$$

where $\Delta T_{p,c}$ represents the time spent in state $\{p, c\}$ during the duration T and the summation is taken over all (p, c) combinations. We now express the above equation in terms of observable quantities that are available on real systems.

For a given P-state, the energy spent in all sleep states $C1$ and higher may be approximated as $k_{p,sleep}$ since the differences are small relative to other approximations made. The active energy is $k_{p,C0}$. Let $E_{cpu}(T)$ denote the integral of power over T. Also, P-state changes are slow and hypervisor counters often allow updates at interval T such that the P-state is constant over T. Then, assuming a single P-state p for the duration T, we get:

$$\begin{aligned} E_{cpu}(T) &= k_{p,C0}u_{cpu}(p)T + (1 - u_{cpu}(p))k_{p,sleep}T & (3)\\ &= (k_{p,C0} - k_{p,sleep})u_{cpu}(p)T + k_{p,sleep}T & (4) \end{aligned}$$

where $u_{cpu}(p)$ represents the percentage of time the processor was active during T, in fixed P-state p. Let $\alpha(p) = (k_{p,C0} - k_{p,sleep})$. Also let the sleep state energy in lowest frequency state, $k_{Pm,sleep}$, be denoted as $\gamma_{cpu,idle}$ and the increase in sleep state energy for higher frequencies be denoted $\gamma(p) = k_{p,sleep} - k_{Pm,sleep}$. Then, taking $T = 1$ second for convenience:

$$E_{cpu} = \alpha(p)u_{cpu}(p) + \gamma(p) + \gamma_{cpu,idle} \tag{5}$$

where $u_{cpu}(p)$ is readily available as an hypervisor performance counter and may thus be used on a real platform to monitor the processor state. If the model parameters are known, the processor energy can be estimated. The method to learn parameters $\alpha(p), \gamma(p)$ and $\gamma_{cpu,idle}$ is described in Section 3.2 together with the power models for other resources.

VM Labeling: Assigning the CPU usage to relevant VMs requires accounting for the exact chip resources used by the VM, including the shared caches and processing components. We choose a light weight approach that simply tracks when a VM is active on a processor core, as is possible by tracking the context switches for every core. The energy used can be tracked using Equation (1) during the durations when the VM is active on the processor. Rewriting in terms of a more readily available performance counter, the processor utilization of VM A, denoted $u_{cpu,A}(p)$, and assuming that the P-state stays constant during interval T, the energy usage of a VM A, denoted $E_{cpu,A}$, becomes:

$$E_{cpu,A} = \alpha(p)u_{cpu,A}(p) \tag{6}$$

[1]To be precise, only DVFS frequency states are referred to as P-states and DFS states are referred to as Throttle, or T-states. We abuse P-state to represent all frequency levels as the distinction is not relevant in our context.

again taking $T = 1$s. Note that the processor idle energy (spent in sleep states) is not included above as we have chosen to report it separately (Section 2).

3.1.2 Memory

A true estimate of memory energy usage may use a cycle accurate simulation of its hardware design. However, prior memory power models have found that the key factor that affects memory energy usage is the read and write throughput. While external instrumentation has been attempted to capture memory throughput accurately [1], a low overhead estimate of memory throughput is the last level cache (LLC) miss counter available in most processors. Using this metric, memory power consumption may be written as:

$$E_{Mem}(T) = \alpha_{mem} N_{LLCM}(T) + \gamma_{mem} \quad (7)$$

where $E_{Mem}(T)$ represents the energy used by memory over duration T, $N_{LLCM}(T)$ is the number of LLC misses during T, and α_{mem} and γ_{mem} are the linear model parameters.

VM Labeling: Since we use LLC misses as an approximation for throughput, we need to track the misses corresponding to each VM. On a single core system, we can assign the LLC misses during the the time a VM has the active context on the processor as the LLC misses for that VM. Thus, the memory energy used by a VM A becomes:

$$E_{Mem,A}(T) = \alpha_{mem} N_{LLCM,A} \quad (8)$$

where α_{mem} is the same as in (7) and $N_{LLCM,app}$ represents the number of LLC misses while the VM was active on the processor.

The above approach has several limitations for practical use. Many multi-core processors share the LLC among multiple cores but do not report the LLC miss count for each core separately, making it hard to attribute it to multiple VMs running on the different cores. Core-level misses are reported by some newer processors such as the AMD Shanghai and the emerging Intel Nehalem, but this functionality was not available in the servers used in our prototype. For the purposes of controlled evaluation, we performed measurements by restricting the active VM to a single core among all cores that shared the LLC.

Another key limitation arises in portability and ease of implementation. The processor performance counters are dependent on the micro-architecture and hence, if these are used, Joulemeter cannot be easily ported across platforms using different processors.

Also by default, most hypervisors hide the hardware performance counters and their use is thus not feasible on production systems. Modifying the hypervisor source code does allow accessing hardware counters but portability still remains a problem.

Therefore, we also consider an alternative that avoids hardware performance counters (Section 3.2).

3.1.3 Disk

While several power models have been developed for disks, this subsystem remains the hardest to model well. The difficulty arises due to lack of visibility into the power states of a hard disk and the impact of disks' hardware caches. Further, in data center servers, disks are mostly used in RAID arrays and even when RAID-0 is used, only the RAID controller hardware controls the physical disks while the hypervisor only sees the logical drives. We restrict our design to using hypervisor-observable parameters.

The hypervisor can observe the number of bytes read and written as well as the service times for those reads/writes. However, for individual VMs, current hypervisors only track bytes read or written and we use those in our disk energy model:

$$E_{Disk}(T) = \alpha_{rb} b_R + \alpha_{wb} b_w + \gamma_{disk} \quad (9)$$

where $E_{Disk}(T)$ represents the energy consumed by the disk over time duration T, and b_r and b_w are the number of bytes read and written respectively during interval T. The α parameters and γ_{disk} are model parameters to be learned.

This model involves approximations since disk spin up/down actions, not visible outside of the RAID controller, are not captured. Variable spin speeds are not captured but as multi-speed disks are not commonly used in data centers, this is not a serious concern.

VM Labeling: As for other resources, we need to track the disk usage parameters in (9) for individual VMs. The time at which the disk activity occurs is usually not the same as when the VM is active on the processor, since the hypervisor may batch IO interrupts and buffer IO operations. Thus instead of looking at storage system activity during the active context of a VM, the IO operations need to be explicitly tracked in the hypervisor. Fortunately, Hyper-V already does most of this tracking and Hyper-V performance counters can be used to report the IO activity of each VM separately. This yields the following disk energy model for a particular VM A:

$$E_{Disk,A} = \alpha_{rb} * b_{r,A} + \alpha_{wb} b_{w,A} \quad (10)$$

where $b_{r,A}$ and $b_{w,A}$ represent the number of bytes read and written, respectively, by VM A. Further, in our experiments, we found the difference in energies for disk read and write to be negligible and hence a common parameter, say b_{io}, can be used to represent the sum of bytes read and written, changing the model to:

$$E_{Disk}(T) = \alpha_{io} b_{io} + \gamma_{Disk} \quad (11)$$

VM disk energy can then be computed using:

$$E_{Disk,A} = \alpha_{io} * b_{io,A} \quad (12)$$

3.1.4 Other Resources

The dynamic range of power usage due to other resources on our testbed servers was small and we have not modeled those. The static energy use of those resources is included in the system idle energy in our model. However, some of these resources may be important to model on other platforms. The 1 Gbps Ethernet cards on our servers did not show a wide variation in energy use with network activity but higher speed cards such as 40Gbps and fiber channel cards do use more energy in driving the physical medium, and this energy is likely to vary with network activity. With servers that use multiple such cards, modeling the network energy will thus be important. The testbed servers did not vary their fan speeds but if variable speed fans are used, their contribution to dynamic energy should be modeled. Another effect to consider is the change in power supply efficiency as the power drawn changes, which will introduce errors into our linear power models.

3.2 Model Parameter Training

The power models in equations (5), (7), and (11) use certain coefficients, denoted by α's, and γ's, that need to be learned in situ on the servers.

Realistic platforms do not allow measuring $E_{cpu}(T)$, $E_{Mem}(T)$ and $E_{Disk}(T)$ separately but only the full system power, denoted $E_{sys}(T)$. Suppose we use a quantity $E_{static}(T)$ to represent the energy used by the non-modeled resources in the system. Then, assuming $T = 1$ as before, we can measure:

$$\begin{aligned} E_{sys} &= E_{cpu} + E_{Mem} + E_{Disk} + E_{static} \\ &= \alpha(p) u_{cpu}(p) + \gamma(p) + \gamma_{cpu,idle} + \alpha_{mem} N_{LLCM} \\ &\quad + \gamma_{mem} + \alpha_{io} b_{io} + \gamma_{disk} + E_{static} \end{aligned} \quad (13)$$

The following points are worth noting regarding the above equation and lead to slight modifications for actual implementation. Firstly, with the above summation, since the constants $\gamma_{cpu,idle}$, γ_{mem}, γ_{disk}, and E_{static} do not have any observable parameters that vary across observations, we cannot learn their individual values from measurements of E_{sys}. Their sum can hence be denoted as a single constant, γ. Secondly, note that the magnitude of $u_{cpu}(p)$ is a fraction between 0 and 1 while N_{LLCM} and b_{io} take values of the order of a hundred million. For numerical stability, it is preferable to normalize N_{LLCM} and b_{io} with respect to their maximum values observed on a system, such that the α parameters are scaled to similar magnitudes. The final equation used in learning thus becomes:

$$E_{sys} = \quad \alpha(p)u_{cpu}(p) + \gamma(p) + \alpha_{mem}u_{mem} + \alpha_{io}u_{disk} + \gamma \tag{14}$$

where u_{mem} and u_{disk} represent the normalized value of N_{LLCM} and b_{io} respectively. In summary, there are $m * 2 + 3$ unknown model parameters: $\{\alpha(p), \gamma(p)\}_{p=1}^{m}$, α_{mem}, α_{io}, and γ, where m is the number of P-states, and four observables that are available on real hardware without extra instrumentation: E_{sys}, u_{cpu}, u_{mem}, and u_{disk}.

Taking multiple observations of the observable quantities allows estimating the model parameters using learning techniques such as linear regression. We use linear regression with ordinary least squares estimation. The learned values are used in equations (6), (8), and (12) and the computed resource energies are summed up to determine the energy usage of each VM. In fact the energy used by the VM on each resource can reported separately, if needed.

3.2.1 Design Choices in Model Learning

We compare three approaches to learn the model parameters, with different overheads and errors. The first approach generates linearly independent combinations of power states in (14) through controlled workloads that cause various resource power states to be used. For instance, the three phases shown in Figure 2 could be repeated in each P-state and E_{sys}, $u_{cpu}(p)$, u_{mem}, and u_{disk} recorded, for several one second intervals. The resultant data traces could be used in linear regression to derive the α and γ parameters. This approach has the lowest overhead since one workload is used for training.

However, the controlled workloads may not use the processor and other resources in a manner representative of real workloads. Hence, a second approach is to run several realistic workloads, such as from benchmark suites, and learn the model parameters from those energy and resource usage traces. This approach has the higher overhead of running multiple workloads and possibly running a new mix every time the target applications change.

The third approach is designed to overcome the accuracy limitations of the first two. We tested the above two approaches and found the errors in full system power estimation to match those reported for the best model in [27] for the SPEC CPU 2006 benchmarks (Section 5). However, an important observation made from the evaluations was the following. The absolute error is quite high and can be higher for specific workloads than the average error. This happens because the power models do not capture all aspects of the resource power states. Consider, for instance, the `gobmk` and `omnetpp` benchmarks from the SPEC CPU 2006 suite. Table 1 shows the processor utilization, memory utilization, and the measured power (averaged over the run and excluding the idle power of the platform) for these two benchmarks. Using the linear model of (14), if α_{mem} is positive, would predict a higher power consumption for `omnetpp` since it has the same processor usage but higher memory usage. The real observed power is in fact lower. On the other hand, comparing `omnetpp` with `lbm` one can see that higher memory usage of `lbm` does result in higher energy than `omnetpp`, and so a negative value of α_{mem} will yield high error.

	CPU (%)	Memory (%)	Power (W)
445.gobmk	100	1.8	28
471.omnetpp	100	23	25
470.lbm	100	31	26

Table 1: Model non-linearity exposed using measured power and observed resource states.

The linear model in 14 cannot capture these effects because certain unobserved power states of the processor are not being accounted for. For instance, the large memory usage may correspond to processor pipelines being stalled on cache misses, reducing processor power without reducing processor active time. Some of the unobserved states can be captured through additional processor performance counters but the number of such counters that can be tracked simultaneously is very limited (2 to 4 counters typically) and not portable across processors. This motivates our third approach: learn the model parameters for each VM separately. Since for a given workload, the unobserved power states are highly correlated to the observed ones, the model based on a small number of observed states will capture the power usage more accurately. In fact with this approach, even if the processor counter for LLC misses is not captured due to portability limitations across processors, that simply becomes another unobserved state that for a given workload should be correlated with observed states.

This approach does have the higher overhead of requiring each new VM to first be hosted alone for the model parameters to be learned. The overhead is acceptable in certain scenarios. For large scale data center hosted applications, though a large number of VMs is used to host multiple parallel instances, the number of distinct applications is small. Hosting one of the instances separately for a limited time for training is not a significant overhead. For a cloud environment, with several small customers, with each application only hosting a small number of instances, the applications are still long lived. Hosting a new VM separately for a limited time for training thus becomes acceptable.

4. IMPLEMENTATION

A block diagram of the VM energy metering system, Joulemeter, is shown in Figure 4. The grey module represents a physical server

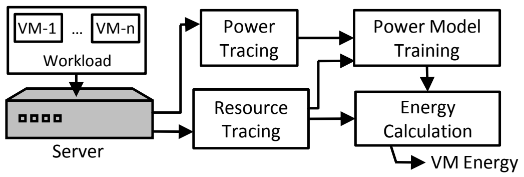

Figure 4: Joulemeter block diagram.

and the other blocks represent software modules executed on it. The block labeled *workload* represents the set of VMs hosted on the server.

The *power tracing* module represents a software driver to read the full system power consumption from an available power sensor on the motherboard. While the power sensors are expected be available through standardized interfaces [18] soon, the server platforms used in our testbeds allowed access to their power sensors only from the manufacturer's proprietary server management software. Hence, we used an external wall power meter, WattsUp Pro ES, that can provide power readings once per second. The serial-API exposed by the WattsUp meter was used to access power data.

A laptop platform used in experiments did allow access to its smart battery interface for power measurement. The test data collected using that interface showed that the smart battery in the particular laptop only updated its power reading every 15 seconds and the reading was averaged over past several 15s intervals. The key impact was that the model learning process was slowed down but Joulemeter could be used without additional hardware.

The *resource tracing* module represents the hypervisor performance counter tracing functionality. To keep Joulemeter usable in existing systems, we use only hypervisor performance counters already available in Hyper-V. Tracing LLC misses did require some additional tools and was only possible on specific processors. Joulemeter operates without processor performance counters when required.

The *power model training* module implements the learning approaches discussed in Section 3.2.1.

The *energy calculation* block uses the resource tracing data and model parameters in equations (6), (8), and (12) to output VM energy usage.

Joulemeter operates in two modes: *learning* and *metering*. In the learning mode, the workload used is as required for the learning approach. The generation of the controlled workload required for the first learning approach is also implemented in Joulemeter. As an example, the processor power model coefficients found for one of the platforms, a Lenovo laptop, are shown in Table 2, using the controlled workload. The other coefficients were $\alpha_{mem} = 8.17$, $\alpha_{disk} = 1.8$, and $\gamma = 25.08$. The laptop LCD screen was maintained at constant brightness throughout the training.

P-State (MHz)	2200	1600	1200	800
$\gamma(p)$	8.98	3.51	1.20	0
$\alpha(p)$	22.70	12.24	4.37	1.10

Table 2: Processor power model coefficients.

In the metering mode, the VMs to be hosted are executed on the server as usual. The resource tracing data and previously learned model parameters are used to generate the metered power output. The model parameters can be updated as needed such as when new VM loads are introduced or when total energy estimation error exceeds a certain threshold.

5. EVALUATIONS

The metering prototype is tested on the following platforms:

1. Dell PowerEdge rack-mount server: 8 cores (2X quad core Xeon 2.5GHz processors), 16GB RAM, and 2x 1TB SATA disks

2. Rackable Systems (SGI) rack-mount server: 4 cores (2X dual core Xeon 2.33GHz processors), 8GB RAM, and 4x 233GB SATA disks

3. Lenovo Thinkpad T61p: 2 cores (Intel Pentium Core-2 Duo processor at 2.2GHz), 4GB RAM, and a 160GB SATA disk

The benchmarks used to represent the applications hosted inside the VMs consist of the SPEC CPU 2006 (`www.spec.org`) suite, representing processor intensive workloads, and IOmeter (`www.iometer.org`) benchmarks, representing storage intensive workloads.

5.1 Power Model Training Error

We first evaluate the accuracy of power models. In particular, we show the errors achieved using the three learning approaches described in section 3.2.1:
Approach 1: Controlled workload
Approach 2: Mix of realistic benchmark workloads
Approach 3: Learn on *similar* workload as the model is to be used on. (We train on one run of a benchmark application and test on multiple other runs.)

The error is evaluated as follows. One run of the training workload is executed and the model parameters are trained on it. Then, multiple test benchmarks are ran. For each test benchmark, multiple test runs are executed. For each run, the instantaneous error between the measured energy and estimated energy (updated at one second interval) is measured. The absolute values of the instantaneous errors are averaged over the entire run. The errors from multiple runs are averaged for each benchmark. The errors reported are averaged over multiple benchmarks.

Figure 5 shows the errors achieved on the Dell PowerEdge server with the three learning approaches. The standard deviations across the multiple runs are also shown. The labels SPEC and IO correspond to SPEC CPU 2006 and IOmeter based benchmarks. ALL refers to use of all applications within the benchmark suite. FP and INT refer to SPEC CPU floating point and integer respectively. RND and SEQ refer to IOmeter load configurations with and without random disk access, respectively.

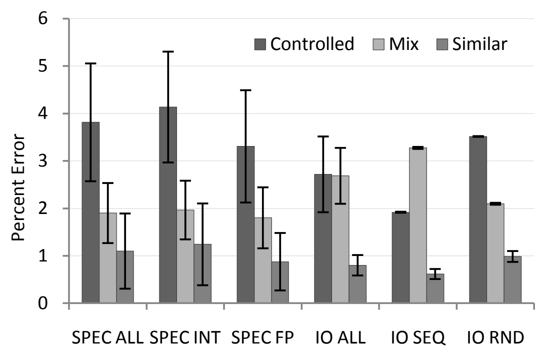

Figure 5: Power model errors (and standard deviations) for Dell PowerEdge.

As expected, the approach of learning the parameters on similar workloads as used for testing yields lowest errors. Also, these errors are low enough to be useful for VM energy metering. The errors shown are without tracking the LLC miss counter; tracking LLC misses did not improve the error for the third learning approach and the improvement was small for other approaches.

In absolute terms a 1% error corresponds to 2.8W of error on the Dell server. It is also worth considering the histogram of the error. While the average is low, one needs to consider the probability that the actual error may sometimes be high. The histogram for one of the runs is shown in Figure 6. We shall use the error histogram to bound the error probability in designing the VM power capping strategy in the next section.

Similarly, the errors measured on the Lenovo laptop for the SPEC CPU 2006 benchmarks are shown in Figure 7. Here, 1% error corresponds to 0.5W. IOmeter based workloads are omitted for the laptop because the disk energy variation with increased disk workload on the laptop was less than 1W and hence the error for the IO intensive workload was very small. The errors measured on the Rackable

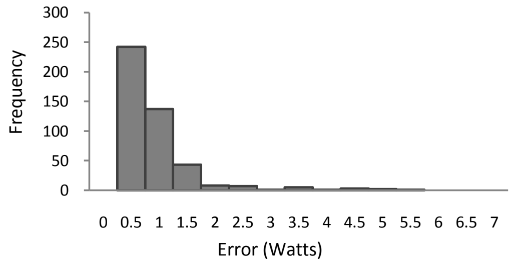

Figure 6: Error histogram for a sample run.

server were similar to those on the Dell PowerEdge.

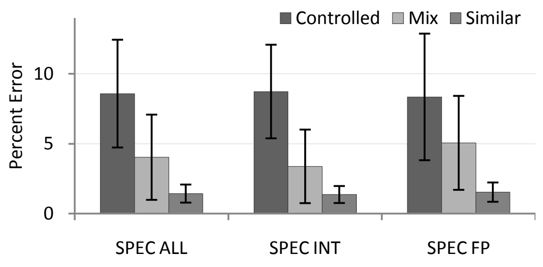

Figure 7: Power estimation errors on a laptop.

Tracing Overhead: The overhead of using the proposed metering mechanism essentially includes reading the performance counters used in the power models and performing scalar multiplications. Since Joulemeter runs in the domain 0 VM, we tracked the resource usage of this VM itself when running Joulemeter and when not, on both an idle server and when other VMs were running some of the benchmarks. The difference in the resource usage of the domain 0 VM was too small to be measurable with respect to the noise.

5.2 VM Energy Measurement

We also wish to evaluate the accuracy of metering for the individual VMs. Ideally, we should compare the Joulemeter estimates to the ground truth. However, as it is not possible to connect a power meter to a VM to measure the ground truth, we compare the Joulemeter estimate of VM power to an alternative estimate. If the difference between the estimates from two methods is small, that will help increase the confidence in the Joulemeter estimate, though without providing an exact measure of error.

The alternative estimation method is as follows. Measure the energy used by the system when it hosts an idle VM and then measure the energy with the VM running a test benchmark workload. Consider the difference between these two measured energies as a hardware estimate of the energy used by the VM.

We perform two types of experiments - one set with only one guest VM (aside from the domain 0 VM) and another with multiple guest VMs where one of the VMs is changed between idle and working. Joulemeter reports the energy used by the domain 0 VM as well as all guest VMs. We noticed that the domain 0 VM energy changed slightly when the guest VM changed from idle to working, and we account for this increase when computing the difference between the two runs.

Experiment 1: Suppose the measured power when the guest VM is idle is denoted $P_m(idle)$ and the power when the guest VM is running benchmark workload A is denoted $P_m(A)$. Then, the hardware estimate of the power attributed to VM A, $P_{hw}(A)$, is:

$$P_{hw}(A) = P_m(A) - P_m(idle)$$

The Joulemeter estimate of the power is obtained from the second run where VM A is running a benchmark, and is denoted $P_J(A)$. Suppose the power estimate of the domain 0 VM is denoted $P_J^{run}(d0)$ in a given run. Then, the change in energy used by the domain 0 VM as estimated by Joulemeter in the two runs is $\Delta_0 = P_J^{run2}(d0) - P_J^{run1}(d0)$. The error, e, in power at a given instant becomes:

$$e = |P_{hw}(A) - (P_J(A) + \Delta_0)|$$

Since the two runs do not overlap in time, we align the start times of the runs before performing the subtraction. The error is averaged over the entire run. In addition to error in average power, we also plot the error in energy over the entire run that may be useful for certain scenarios where the total energy of a VM is required. Figure 8 plots this error for a few of the SPEC CPU 2006 benchmarks.

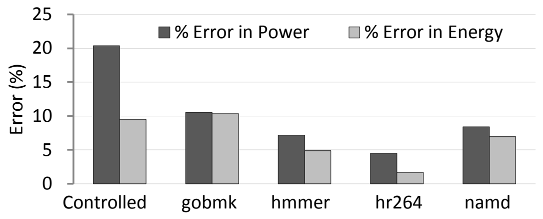

Figure 8: VM power and energy estimation errors.

The error in VM A is reported as a percentage of the average energy used by VM A (and not as a fraction of the full system energy). Here, a 1% error corresponds to 0.2W to 0.3W of absolute error, depending on the VM. The Rackable Systems server was used.

Experiment 2: Here, we use two guest VMs, where VM A is changed between idle and working while VM B runs the same benchmark workload in both runs. Suppose the Joulemeter energy estimate of a VM is denoted $P_J^{run}(B)$ during a given run. Let $P_m(A, B)$ denote the measured power with two VMs. Then, the change in energy between the two runs as measured in hardware is $P_{hw}(A) = P_m(A, B) - P_m(idle, B)$. The change in energy between the two runs as predicted by Joulemeter is: $P_J(A) = P_J^{run2}(A) + P_J^{run2}(B) - P_J^{run1}(B) + \Delta_0$. The error is computed as:

$$e = |P_{hw}(A) - P_J(A)|$$

This error for some choices of A and B from among the SPEC CPU 2006 benchmarks is plotted in Figure 9.

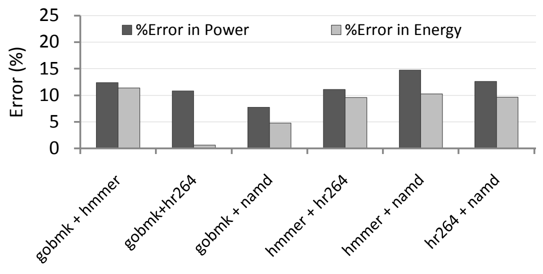

Figure 9: Joulemeter errors in multi-VM experiment.

The results show that a reasonably low error, ranging from 1W to 3W, can be achieved. The errors are similar to that achieved using temperature compensated hardware measurements in high-volume, low-cost servers, reported at 2% (or 5W-6W) in [23]. The applications of the VM metering functionality must be designed to work with this accuracy.

6. POWER CAPPING

We now show how Joulemeter can be used to solve the power capping problem for VMs, leading to significant savings in power provisioning costs in data centers. The wastage in energy provisioning cost due to underutilization of the power infrastructure is well recognized [7] and power over-subscription is actively being considered for reducing such waste [8, 23]. A key enabler for over-subscription is server *power capping* that ensures safe operation of servers on over-subscribed circuits. Server power capping can be implemented in hardware and server manufacturers now offer this capability [13, 15].

However, with virtualization, server hardware based power capping is no longer appropriate as each server is now effectively divided into multiple virtual servers and capping is required on a VM basis. We show how the proposed VM power metering capability allows extending power capping to VMs so that benefits of over-subscription continue to be realized in spite of virtualization. In fact additional benefits can be realized when over-subscribing with virtualization.

6.1 Over-subscription and Power-Capping

To understand the problem exactly, we give a brief overview of how power over-subscription helps reduce data center power provisioning costs. Provisioned power refers to the capacity of the power infrastructure, including the power distribution circuits, UPS systems, and backup generators. For an efficiently designed data center, the provisioning cost is close to 25% of the total cost where the total cost includes the capital cost and recurring operational cost. The absolute numbers are also quite high and a significant cost concern for businesses using the data centers. Amortized over the life of a single mid-sized data center, the provisoning cost accounts for over \$1 million per month and over a hundred million when considered at the time of construction [12]. In addition, provisioning the generator capacity involves recurring fuel expenses since the fuel stored is replenished every month even if not used due to fear of moisture entering the fuel that may destroy the generator equipment. Over-subscription helps reduce the provisioned capacity and all associated costs. In some regions the power infrastructure is saturated, and over-subscription is the only way for businesses to grow their server capacity.

Over-subscription works as follows. Suppose a server is *rated* for peak power consumption P_{rat}, where P_{rat} is the possible peak power the server hardware can consume. The possible peak power is reached only when every hardware component of the server, including every hardware accelerator, processor subcomponent, memory subsystem, storage, and so on, is used to its maximum power level. A real workload, even at peak user load will not reach P_{rat}. Suppose the *actual* power level reached is denoted P_{act}. Rather than provisioning for P_{rat}, one could provision for $P_{act} + \Delta_m$, where Δ_m represents a safety margin. This reduces the required provisioned capacity by $P_{rat} - P_{act} - \Delta_m$. For instance, for the Dell PowerEdge server mentioned before, $P_{rat} = 275$W. For the online application workload described in Section 6.2.1, we measured $P_{act} = 209.5$W, implying a potential saving of 38W assuming $\Delta_m = 10\% * P_{rat}$.

Power Capping: For a given data center, the P_{act} for the various workloads it hosts can be measured and the power distribution circuits provisioned for this lower power per server. Power limits are enforced via circuit breakers on every rack to locally curb the effect of any intermittent power budget violations. A common power limit is used for all racks, so servers can be placed in any rack. When provisioned for P_{act} per server, the circuit breaker is said to be oversubscribed because in principle, it is possible that some change in workload could cause P_{act} to be exceeded. To ensure safety, an additional mechanism, known as *server power capping*, is required to ensure that P_{act} is not exceeded at each server. Server power capping is implemented in most new servers [13, 15] and uses hardware power measurement to determine when capping is to be enforced (via DVFS, for instance).

Further savings can be achieved for many workloads that use P_{act} only during peak user demand, limited to a small duration of time, and operate at well below P_{act} most of the time. Suppose that the application is willing to take a performance hit (due to capping) for 1% (in general, say $(100 - x)\%$) of the time. Then the server could be provisioned for a power level that is not exceeded 99% (in general, $x\%$) of the time. Denoting this power level by $P_{x=99}$, this results in further reduction in provisioning requirement by $P_{act} - P_{x=99}$, though at the cost of a performance hit. For instance, the online application workload (Section 6.2.1) had $P_{x=99} = 198$W, implying a potential for additional 6% savings, compared to P_{act}.

6.2 VM Power Capping

If server hardware capping is used with virtualization, the isolation property of VMs is no longer respected because capping the server will affect all VMs hosted on it. Joulemeter enables the safety mechanisms and control required to cap individual VMs within a server, allowing the advantages of over-subscription and power capping to be extended to the virtualized world while maintaining the VM isolation property. This yields two benefits:

Reclaimed Benefit: Joulemeter allows allocating power budgets to VMs by measuring their actual power usage, $P_{act,VM}$. The cap can then be enforced on each VM separately, maintaining isolation of workloads.

Additional Benefit: As mentioned before, excessive power penalty due to occasional peaks can be avoided by provisioning for $P_{x,VM}$ instead of $P_{act,VM}$, and forcing the cap during the small fraction $(100 - x)\%$ of time that peaks do occur. The same technique can be extended to VMs but with two additional advantages:

1. *Statistical Multiplexing:* While on a physical server the x-th percentile would be exceeded $(100 - x)\%$ of the time by definition, in the virtualized setting the x-th percentile would be exceeded much less often. This happens because different VMs peak at different times and server power capacity may not be exceeded when some VMs peak because other VMs may be operating well below their peaks. Since only total server power is constrained, the cap will be enforced less than $(100 - x)\%$ of the time.

2. *Migration:* While for a physical server the cap is enforced by reducing its performance, the cap can be met in virtualized servers by migrating a VM to an underutilized server or rack. Migration penalty is typically preferred over taking a performance hit. If migration is not feasible, certain low priority VMs could be shut down rather than having to throttle the server exceeding the cap even if it is a high revenue server.

Both these advantages can only be realized if VM power is metered individually and not using server hardware based capping.

These benefits are of course in addition to the other advantages of virtualization such as reduction in number of servers required due to consolidation, which itself reduces power provisioning and usage costs.

6.2.1 *Production Traces*

To evaluate the benefits of VM power capping, we use server resource usage traces from two thousand servers from a production data center serving real world applications used by millions of users worldwide. In particular, we use two datasets:

Online Application: We collected resource utilization traces from 1000 servers hosting a single large online application[2] that serves millions of users. We use the utilization traces as if they came from 1000 VMs hosting the same application. The peaks and resource usage in these traces are highly correlated since the servers are hosting the same application and serving the common user workload.

Cloud Applications: The same data center also contains other servers hosting a variety of small applications that perform very different functions, ranging from serving media files and internal enterprise data processing. We use the resource usage traces from 1000 of these servers to represent heterogeneous customer VMs hosted in a cloud computing environment. These traces are not significantly correlated in their usage patterns.

Sample CPU utilizations from both sets are shown in Figure 10.

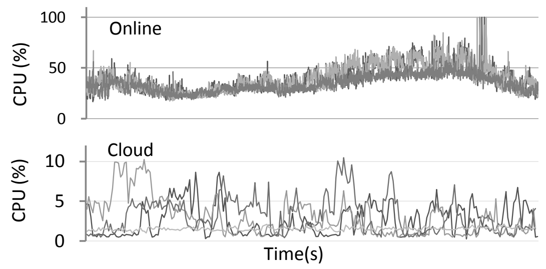

Figure 10: Sample server usage traces.

6.2.2 *Capping Method*

Given a set of VMs and their resource usage traces, we first determine their peak resource requirements (CPU, memory space, and disk bandwidth) and consolidate them. Considerable prior work exists in determining appropriate consolidation strategies for VMs and we use a commonly used heuristic, first fit decreasing, for consolidation. The resource constraints on the VMs automatically ensure that the P_{rat} of a server is not exceeded. However, using Joulemeter, we can determine the peak power usage, $P_{act,VM}$ and its x-th percentile, $P_{x,VM}$, for each individual VM.

Suppose for a given server, n VMs are placed on it. Then, we can set the hardware power cap for the server as:

$$P_{server} = \sum_{i=1}^{n} P_{act,VM(i)} + \Delta_m$$

P_{server} is only a failsafe, should the VM cap enforcement have errors. The actual capping is performed on the VMs individually.

The VM capping mechanism is as follows. When the server capacity is exceeded, the appropriate VM is migrated to a different server on another circuit that has excess capacity. If no circuit has excess capacity, the VM's execution is postponed to after the peak subsides. The appropriate VM need not be the one exceeding its capacity but a different low priority VM.

[2]Names of the application and collaborating commercial provider are suppressed for blind review.

If no migration is possible, and no low priority VM can be postponed, reduction of resources for the VM exceeding its capacity is required, akin to physical server capping. While in physical servers resource reduction can be implemented using DVFS [23], most hypervisors do not allow DVFS control for individual VMs. Here, the processor cycles allocated to a VM can be reduced to achieve the desired power reduction [25].

6.2.3 *Evaluation*

Reclaimed Savings: Using the above provisioning method, we can reclaim the savings that were earlier achieved with physical server power capping. For the above datasets the savings in provisioned capacity, with respect to provisioning for possible peak, are shown in Table 3 (at $\Delta_m = 10\%$), where Cloud and Cloud' represent two different sets of 1000 servers from the cloud dataset.

Workload	Online	Cloud	Cloud'
Savings	13.81%	24.18%	27.80%

Table 3: Savings due to VM power metering and capping.

Even though these savings are only reclaiming what was earlier achieved with physical servers, these over-subscription savings are important because they are in addition to the consolidation savings from virtualization and would have been lost without VM power metering. The consolidation savings are themselves significant. For instance, for the Online application workload virtualization allowed consolidating the servers to 15% fewer servers, leading to a reduction in the idle power required for those 15% servers. The Cloud workload had much lower CPU utilization and was consolidated to 63% fewer servers. Without the capability of VM power capping, one would have to choose only one of consolidation savings (using virtualization without capping) or over-subscription savings (using hardware capping without virtualization), but Joulemeter allows achieving both, due to safe over-subscription on virtualized platforms.

Additional Savings: As mentioned, VMs make it easier to provision for x-th percentile instead of actual peak. Figure 11 shows the additional reduction in provisioned power achieved for the Online and Cloud application datasets when provisioning for $P_{x,VM}$ (with $\Delta_m = 10\%$), compared to provisioning for $P_{act,VM}$, for varying x. The power characteristics of the Dell PowerEdge server are used in computing the savings. The savings are shown when every

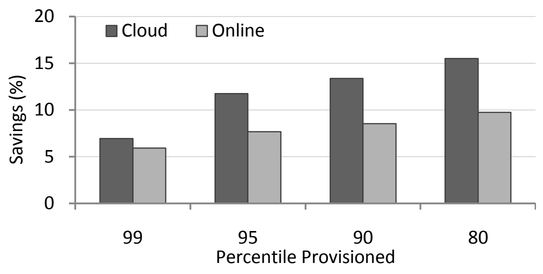

Figure 11: Provisioned capacity savings with Joulemeter.

server is provisioned for a *common* capacity, based on the capacity required for the server drawing the highest power. This makes it easier for the consolidation algorithms to treat all servers homogeneously. Joulemeter is itself used to profile the power usage of VMs and find the x-th percentiles of their peak powers.

The actual percent of the time when the VMs have to be migrated (i.e., the cap has to be enforced) are plotted in Figure 12. For the

Online application, since the VMs are highly correlated in usage, the cap is enforced more often than for the Cloud applications, but in general significantly less often than the tolerable limit $(100-x)\%$ set during provisioning.

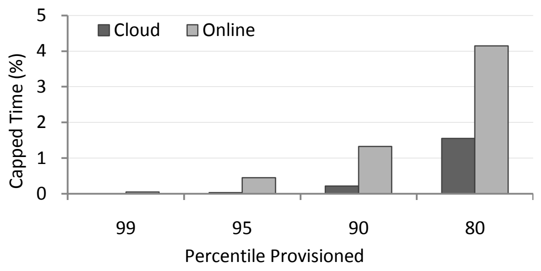

Figure 12: Percent of time capping had to be enforced.

Figures 11 and 12 show that additional savings of 8% to 12% of provisioned power can be achieved, with migrations required for less than 0.5% of the time ($x = 95$).

The reductions shown in provisioning cost can be used to add more servers thereby postponing the construction of the next data center saving significant capital expense. For a mid sized data center, typically provisioned at 15MW, a 10% reduction in provisoned capacity amounts to an additional capacity of 1.5MW that is sufficient for approximately 5400 servers.

Metering Error: Note that the metering mechanism has some errors as shown in the previous section. An over-estimation error does not lead to any problems, other than reduction in savings. However, sometimes the error can cause the peak usage of a VM to be under-estimated, causing a server's allocated cap to be exceeded even when no VM is inferred by Joulemeter to exceed its allocated VM cap. To address this concern, we can use the histogram of the VM energy metering error (Figure 6) and determine the y-th percentile of the error, say Δ_y. Then, if we provision with a margin Δ_y, the provisioned peak will not be exceeded due to metering errors, more than $(100 - y)\%$ of the time. When exceeded, the hardware cap may kick in. Of course, in reality not all VMs peak simultaneously, and the hardware cap will kick in much less frequently than $(100 - y)\%$.

7. DISCUSSION

Applications: In addition to the VM power capping and provisioning cost savings enabled using Joulemeter, the same metering capability can also be leveraged for several other uses in virtualized environments, as follows.

QoS Control: When multiple VMs are hosted in a virtualized environment, certain VMs may be lower priority that others. For instance, VMs hosting interactive online services may need high performance but batch jobs performing internal enterprise processing may have a high tolerance for latency. Joulemeter allows selectively determining how much resource reduction is required for low priority VMs to achieve desired aggregate power limits.

Migration Control: The provisioned power in a data center is often allocated among different racks, each with its circuit breaker to ensure that a power surge is locally controlled rather than affecting the entire data center. Using Joulemeter to meter each VM individually allows determining which VMs may be migrated from highly utilized racks to under-utilized ones to match the power allocation.

Energy Budgets: Prior work [35] proposed using fixed energy budgets for applications. Similar budgets can be applied to VMs to control energy costs. Joulemeter can be combined with the hypervisor's resource scheduling mechanisms to enforce the energy budgets on each VM.

Developer Visibility: Providing visibility into energy use is often a valuable motivator for developers to redesign their applications for saving energy. While developers can measure the energy use on their developer workstations, they do not have that visibility on the actual cloud servers, where the VM behavior could be different due to differences in processor micro-architecture, cache hierarchies, or IO speeds. Joulemeter can provide energy visibility for individual workloads, even on a shared cloud platform.

Pay-as-you-go billing: An online application often needs to have multiple instances hosted on multiple machines due to reasons of reliability, such as ensuring availability during system updates on some of the machines, or hardware repairs on some. If the user load is not sufficient, many application instances are underutilized. In cloud systems, the application owner pays for the number of cores reserved even if they are not utilized fully. Using Joulemeter, the cloud operator can consolidate more VMs in fewer cores and pass on the savings to customers by introducing usage based billing. Savings for the data center and more competitive prices for customers are likely to benefit the growth of the industry, allowing more and more legacy applications using inefficient custom data center deployments to move to highly optimized cloud environments.

Thermal Optimizations: Cooling consumes a significant fraction of energy in data centers. Cooling efficiency depends on the distribution of heat in the data center. For single cooling unit based data centers, having a uniform heat distribution is often most efficient as it avoids hot-spots and the required over-cooling in other area to deal with hotspots. On the other hand, when multiple smaller cooling units are used, each cooling a few racks, the efficiency of the cooling mechanism is increased with higher temperature differentials. In these cases, it is best to operate a few racks, that share a cooling unit, at full power and shutting others down. Integrating Joulemeter with consolidation algorithms can allow generating the required power distribution profiles across the data center.

Future Work: The power capping scheme presented was optimal assuming the VM consolidation and capping steps are disjoint. The server was provisioned for a summation of $P_{act,VM}$ (or $P_{x,VM}$) and the consolidation scheme could have placed these VMs without optimizing the capping. Instead, if the consolidation method could ensure that VMs that are placed together are such that they peak at different times [11], we could provision for actual power required by the selected VM combination, and provision for even lower capacity.

The time delay of VM migration in enforcing the cap is not considered in the current work. The migration could be initiated at a power level slightly lower than the set cap, so that the migration takes place before the server hardware cap kicks in.

We saw that the learning approach with least errors had the additional overhead of requiring separate training for each different application. We could reduce the overhead by determining similarities among applications based on their processor usage characteristics. These characteristics can be determined using the processor performance counter traces for each application and matched to previously known applications. Such an approach was previously tried for predicting VM performance [22]. Model parameters could then be re-used for workloads with similar characteristics.

8. CONCLUSIONS

We presented a VM power metering approach that can be implemented on current virtualized platforms without adding any additional hardware or software instrumentation. The guest VMs are not modified and existing hardware capabilities are used. We discussed the challenges involved in obtaining a low estimation error and avoiding extensive instrumentation. The end result is a mechanism that extends the current physical server power monitoring solutions to VM power monitoring.

The proposed VM power metering capability was used to realize significant power provisioning cost savings in data centers. Experiments using production server data sets from a commercial data center operator showed 8% to 12% additional savings aside from reclaiming savings that existed without virtuzlization. Several other uses for VM energy metering were discussed that add to the many advantages of virtualization.

The design of Joulemeter also revealed certain characteristics that are desirable for the design of hypervsiors and processors used in virtualized platforms. For instance, processor performance counters if exposed for individual cores help attributing resource usage to individual VMs that are using the cores in parallel. Also, currently hypervisors hide the hardware performance counters while tracking and exposing the counter values on a per-VM basis would be beneficial.

9. REFERENCES

[1] Y. Bao, M. Chen, Y. Ruan, L. Liu, J. Fan, Q. Yuan, B. Song, and J. Xu. HMTT: A platform independent full-system memory trace monitoring system. In *ACM Sigmetrics*, June 2008.

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

[3] W. L. Bircher and L. K. John. Complete system power estimation: A trickle-down approach based on performance events. In *International Symposium on Performance Analysis Systems and Software (ISPASS)*, 2007.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, 2000.

[5] Amazon elastic compute cloud (EC2). http://aws.amazon.com/ec2/.

[6] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Workshop on Modeling, Benchmarking and Simulation (MoBS)*, June 2006.

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

[8] M. Femal and V. Freeh. Safe overprovisioning: Using power limits to increase aggregate throughput. In *Workshop on Power-Aware Computer Systems (PACS)*, Portland, OR, December 2004.

[9] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, 1999.

[10] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Symposium on Operating System Design and Implementation (OSDI)*, December 2008.

[11] L. Ganesh, J. Liu, S. Nath, G. Reeves, and F. Zhao. Unleash stranded power in data centers with rackpacker. In *Workshop on Energy-Efficient Design (WEED), at ISCA-36*, June 2009.

[12] J. Hamilton. Cost of power in large-scale data centers. Blog entry dated 11/28/2008 at `http://perspectives.mvdirona.com`. Also in Keynote, at ACM SIGMETRICS 2009.

[13] HP. Dynamic power capping TCO and best practices white paper. http://h71028.www7.hp.com/ERC/downloads/4AA2-3107ENW.pdf.

[14] HP. Hp integrated lights-out (iLO) standard. http://h18013.www1.hp.com/products/servers/management/ilo/.

[15] IBM. IBM active energy manager. http://www-03.ibm.com/systems/management/director/about /director52/extensions/actengmrg.html.

[16] IBM. IBM PowerExecutive installation and user's guide version 2.10.

[17] C. Im and S. Ha. Energy optimization for latency- and quality-constrained video applications. *IEEE Des. Test*, 21(5):358–366, 2004.

[18] Intel. Data center manageability interface. http://www.intel.com/technology/product/DCMI/.

[19] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *36th annual International Symposium on Microarchitecture (MICRO)*, 2003.

[20] J. Janzen. Calculating memory system power for ddr sdram. *Micro Designline*, 10(2), 2001.

[21] Y. Kim, S. Gurumurthi, and A. Sivasubramaniam. Understanding the performancetemperature interactions in disk i/o of server workloads. In *The Symposium on High-Performance Computer Architecture*, pages 176– 186, February 2006.

[22] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pages 200–209, 2007.

[23] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Fourth International Conference on Autonomic Computing (ICAC)*, page 4, 2007.

[24] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop. In *Power-Aware Computer Systems, 4th International Workshop (PACS)*, Portland, OR, USA, December 2004.

[25] R. Nathuji, P. England, P. Sharma, and A. Singh. Feedback driven qos-aware power budgeting for virtualized servers. In *Fourth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, April 2009.

[26] F. Rawson. Mempower: A simple memory power analysis tool set. Technical report, IBM Austin Research Laboratory, 2004.

[27] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *HotPower'08: Workshop on Power Aware Computing and Systems*, December 2008.

[28] A. Sinha and A. P. Chandrakasan. Jouletrack: a web based tool for software energy profiling. In *38th Conference on Design Automation (DAC)*, pages 220–225, 2001.

[29] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th EuroSys Conference*, Nuremberg, Germany, April 2009.

[30] P. Stanley-Marbell and M. Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the International Symposium on Low power Electronics and Design*, pages 141–146, 2001.

[31] T. Stathopoulos, D. McIntire, and W. J. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *7th international conference on Information processing in sensor networks (IPSN)*, pages 383–394, 2008.

[32] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In *USENIX Annual Technical Conference*, pages 1–14, 2007.

[33] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. Instruction level power analysis and optimization of software. In *9th International Conference on VLSI Design*, page 326, 1996.

[34] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In *2nd USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[35] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: managing energy as a first class operating system resource. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 123–132, 2002.