Virtual Machine Power Metering and Provisioning

Aman Kansal, Feng Zhao, Jie Liu
Microsoft Research
Redmond, WA, USA
kansal@microsoft.com

Nupur Kothari
University of Southern California
Los Angeles, CA, USA
nkothari@usc.edu

Arka A. Bhattacharya
Indian Institute of Technology
Kharagpur, West Bengal, India
arka@cse.iitkgp.ernet.in

ABSTRACT

Virtualization is often used in cloud computing platforms for its several advantages in efficiently managing resources. However, virtualization raises certain additional challenges, and one of them is lack of power metering for virtual machines (VMs). Power management requirements in modern data centers have 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, VM power cannot be measured purely in hardware. We present a solution for VM power metering, named *Joulemeter*. We build 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. Our 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, a technique to reduce power provisioning costs in data centers. Experiments are performed on server traces from several thousand production servers, hosting Microsoft's real-world applications such as Windows Live Messenger. The results show that not only does VM power metering allow virtualized data centers to achieve the same savings that non-virtualized data centers achieved through physical server power capping, but also that it enables further savings in provisioning costs with virtualization.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement Techniques; K.6.2 [**Computing Milieux**]: Management of Computing and Information Systems—*Installation Management*

General Terms

Measurement, Performance

Keywords

power capping, virtualization, datacenter power management

1. INTRODUCTION

Most current cloud platforms use virtualized data centers. This paper considers power management for virtualized data centers. The importance of power management for data centers is well known and most computational platforms now provide multiple power management options. A key component of power management in such large scale computational facilities is the visibility into power usage. This visibility 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. Equivalent visibility is, however, lacking in virtualized platforms. The use of virtual machines (VMs) comes with many benefits such as safe isolation of co-located workloads, enabling multiple workloads to be consolidated on fewer servers, resulting in improved resource utilization and reduced idle power costs. However, the lack of VM power metering capability takes away the advantages of server power metering that were available in non-virtualized settings.

In this paper we propose a mechanism for VM power metering, named Joulemeter, and show how it enables virtualized platforms to use some of the same power management mechanisms that have proved beneficial on physical servers. As an example, we show how this mechanism can enable power caps to be enforced on a per-VM basis, resulting in a significant reduction of power provisioning costs. Experiments on real-world workloads from several thousand production servers show similar savings as achieved with physical server capping but with the benefits of VM isolation. Also, additional savings are realized due to the more aggressive over-subscription in the virtualized world enabled by the tighter control and safety mechanisms offered by Joulemeter. The energy metering mechanism also enables several other power management tasks (Section 7.3).

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. Our mechanism does not require any additional instrumentation of application workloads or operating systems within the VMs. Our approach can naturally adapt to changes in application characteristics and even hardware configurations. While prior works have proposed mechanisms to measure VM energy usage [29], they assumed the availability of detailed power models from each hardware component, that are not available in practice, and are in fact difficult to provide.

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. 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.

Second, we experimentally evaluate the accuracy of the VM energy metering mechanism using benchmark applications on commonly used platforms. We also show why generic power models, as used in prior work and effective for physical servers running a single application, are not well-suited to cloud platforms where a server may be shared by many different VMs each running a markedly different application. The Joulemeter mechanism is shown to achieve much higher accuracy, achieving errors within 0.4W - 2.4W.

Third, we show how VM power metering improves power management for cloud platforms. Specifically, 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 [6] to be an important concern, and solutions based on server power capping have not only been proposed [7, 18] but are already available commercially [11, 12]. However, these existing server power capping techniques cannot directly be applied to virtualized environments. Joulemeter helps realize similar savings in virtualized environments as existed in physical servers with hardware power capping, 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 also discuss the use of power metering in taking advantage of renewable energy and variable electricity pricing for data centers. We describe additional 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 [33, 13] that can be extended to VMs.

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

2. RELATED WORK

Energy Metering: Energy measurement for VMs was considered in [29]. That work assumes that the correct power usage of each hardware resource in different usage modes is known to its device driver and exposed to the OS. Accuracy of power estimates was thus not a concern and not evaluated. However, most hardware drivers do not provide such power data. Providing such data is difficult because the power used depends on the details of how an application uses a component. For instance, a processor driver cannot declare a fixed value for its active mode power usage since power usage will depend on which sub-units and accelerators within the processor are activated by the application instructions. 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 experimentally investigate the resultant errors and overheads, using benchmark application in VMs running on representative hardware. Further, we also present applications of the VM power metering mechanism, including a power oversubscription and capping solution for data center power provisioning.

Additional closely related work is found in tracking application energy usage [25, 33, 8] that, in concept, can be compared to VM energy usage. Some of these methods [25, 8] assume that the energy usage at a current time 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 [33]. However, the main focus of the work in [33] was on application energy budgeting and the power metering itself used data-sheet numbers for key hardware components. All required application-software instrumentation that tracked an application's hardware component use was assumed to be available. Our goal is to build the relevant power models using only instrumentation that is available in hypervisors and servers, 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 [33] 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 [30, 2, 14, 26, 4, 24], memory [22, 15], and disks [32, 17]. 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 [5, 6, 23, 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 [16].Our goal is to provide visibility into VM power usage.

Energy models based on extensive hardware and software instrumentation have also been considered [9, 27, 19]. The Quantto [9] 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 cloud platforms because the cloud operator cannot instrument applications executed within the VM. Also, even if each cloud customer was willing to allow such instrumentation on their applications, the kind of fine grained source code annotations used in [9] would incur a very high overhead in cloud applications due to the complexity of software applications, abstraction stacks used, and inclusion of third party modules. Several issues such as monitoring memory activity, or buffering of disk IO by the OS, do not arise in Quantto but are significant in virtualized systems. The LEAP platform [27] presents a system where each hardware resource is individually instrumented in hardware to track energy usage. The method in [19] used laboratory instrumentation for hardware power metering. Our focus is on methods that do not depend on extensive instrumentation and leverage available instrumentation, making them significantly easier to adopt.

3. POWER METERING CHALLENGES

Existing hardware instrumentation, such as motherboard or power supply based power sensors, allow measuring a physical computer system's power consumption [16]. 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 level the resource was operating at 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.

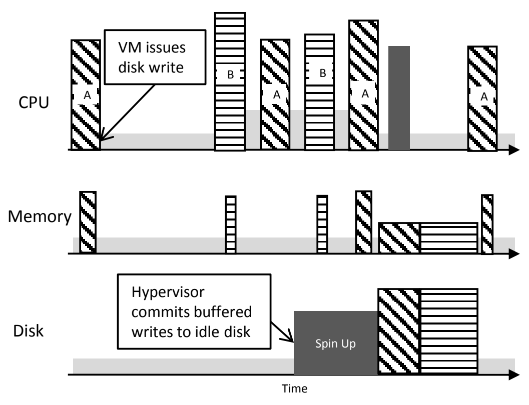


Figure 1: Metering individual VM energy usage.

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 milliseconds), we can compute the *active* energy usage of each VM. The active energy is defined here 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 length of idle time durations between VM active 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 fine time granularity:* The height of the rectangles (instantaneous power) in Figure 1 must be determined at a fine enough granularity for each resource, to distinguish between the energy used by different VMs (depending on time quanta used for context switching).

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 fine time 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 [29] but existing systems do not support this. Providing accurate power models would be hard due to the large number of factors affecting the power level.

2. Use the data sheets of the hardware resources [33]. 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.

4. 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, 12GB RAM, and four 300GB 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.

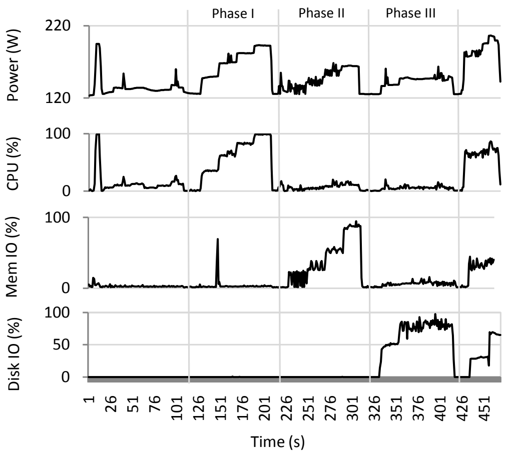


Figure 2: Power impact of the three major system resources. While memory throughput shown above is obtained using hardware specific performance counters, a simpler approach to infer memory energy impact is presented in Section 4.2 to avoid such hardware specific instrumentation.

Prior work has proposed various power models for deriving full system energy usage [2, 5, 6, 3] and some of these models have been compared in [23]. 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. Our approach takes advantage of built-in server power sensors that were not available in older servers used in prior work.

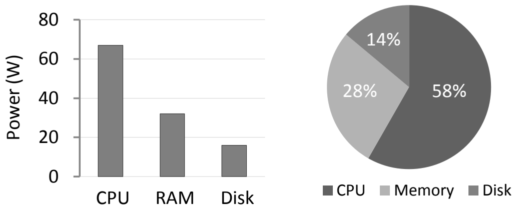


Figure 3: Relative power impact of different resources on dynamic power consumption.

4.1 Practical Power Models

4.1.1 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 Performance states, or P-states, corresponding to different DVFS frequencies and Throttle-states, or T-states, corresponding to DFS frequencies. 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 power metering.

A lighter weight alternative is to track processor active and sleep times, often available easily from the OS as processor utilization. Let u_{cpu} denote processor utilization. Then, in this approach, for a given processor frequency, the CPU energy model becomes:

$$E_{cpu} = \alpha_{cpu} u_{cpu} + \gamma_{cpu} \quad (1)$$

where α_{cpu} and γ_{cpu} are model specific constant. The method to learn parameters the model constants is described in Section 4.2 together with the power models for other resources.

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. The energy used can be tracked using Equation (1) during the durations when the VM is active on the processor. The Windows Hyper-V hypervisor creates virtual processors that can span a whole or fractional logical core[1] and allocates the specified number of these to each VM. Hyper-V allows tracking the usage of the virtual processors from within the root VM through its performance counters (under its `Hyper-V Hypervisor Virtual Processor` counter category, with counter instances denoting each individual guest VM virtual processor, and a dedicated category named `Hyper-V Hypervisor Root Virtual Processor` for the root VM's virtual processors). Using the hypervisor VM settings that relate the virtual cores to a whole or fractional logical core, the virtual processor usage can be mapped to physical processor utilization. Similar data is also available for the Xen hypervisor, for instance, through Xentrace. If the processor utilization of VM A, is represented by $u_{cpu,A}$, then the energy usage of a VM A, denoted $E_{cpu,A}$, becomes:

$$E_{cpu,A} = \alpha_{cpu} u_{cpu,A} \quad (2)$$

[1]A logical core may differ from a physical core when the processor uses hyper-threading, such as on the Intel Nehalem.

Note that the processor idle energy (spent in sleep states) is not included above as we have chosen to report it separately (Section 3).

4.1.2 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:

$$E_{Mem}(T) = \alpha_{mem} N_{LLCM}(T) + \gamma_{mem} \tag{3}$$

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.

Tracking the LLC misses corresponding to each VM is less straightforward than tracking the VM processor usage. Since the meory accesses are managed by the processor hardware, the OS or hypervisor does not have direct visibility into this. Most processors do expose the LLC misses as a hardware performance counter and modern ones such as Intel Nehalem offer it on a per core basis. If we include the additional software tooling required to access the per core LLC misses and track its value with each VM context switch on every core, we can obtain the LLC misses corresponding to a VM. Then, the memory energy used by a VM A becomes:

$$E_{Mem,A}(T) = \alpha_{mem} N_{LLCM,A} \tag{4}$$

where $N_{LLCM,app}$ represents the number of LLC misses for a VM across all cores used by it during time period T, and α_{mem} is the same as in (3).

The above approach is feasible but has limitations for practical use. Tracking for LLC misses requires knowledge regarding processor specific counters that vary not only across processor manufacturers but even among processor families and model series from the same manufacturer, limiting portability and maintainability. Once processor specific instrumentation is included, the hypervisor must also be modified to track these counters at VM context switches. This can be achieved by modifying the hypervisor source code. To avoid some of these overheads that may restrict practical use, we also consider an alternative that avoids hardware performance counters (Section 4.2).

4.1.3 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:

$$E_{Disk}(T) = \alpha_{rb} b_R + \alpha_{wb} b_w + \gamma_{disk} \tag{5}$$

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.

As for other resources, we need to track the disk usage parameters in (5) 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, the Windows Hyper-V hypervisor already does most of this tracking and VM specific disk usage can be obtained from Hyper-V performance counters in the `Hyper-V Virtual Storage Device` category for certain types of VMs and the `Hyper-V Virtual IDE Controller` category for other types of VMs.

This yields the following disk energy model for a particular VM A:

$$E_{Disk,A} = \alpha_{rb} * b_{r,A} + \alpha_{wb} b_{w,A} \tag{6}$$

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}, obtained by taking the sum of the above mentioned VM disk counters, can be used to represent the sum of bytes read and written, simplifying the model to:

$$E_{Disk}(T) = \alpha_{io} b_{io} + \gamma_{Disk} \tag{7}$$

VM disk energy can then be computed using:

$$E_{Disk,A} = \alpha_{io} * b_{io,A} \tag{8}$$

4.1.4 Other Components

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. Note that our approach is focused only on the local energy consumed by a VM and not the additional energy use caused by it in a SAN device or a back-end database. Additional power models for those devices with VM request tracing would be required to include that impact.

4.2 Model Parameter Training

The power models in equations (1), (3), and (7) 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 $E_{sys}(T)$. In Windows Server 2008 R2, E_{sys} is exposed using a counter named `Power Meter.Power`, on supported hardware models, as well as Windows Management Instrumentation (WMI) based API calls.

Suppose we use a quantity $E_{static}(T)$ to represent the idle system power (this also includes power used by the non-modeled resources in the system). Then, assuming a time duration $T = 1$, we can measure:

$$\begin{aligned} E_{sys} &= E_{cpu} + E_{Mem} + E_{Disk} + E_{static} \\ &= \alpha_{cpu} u_{cpu}(p) + \gamma_{cpu} + \alpha_{mem} N_{LLCM} \\ &\quad + \gamma_{mem} + \alpha_{io} b_{io} + \gamma_{disk} + E_{static} \end{aligned} \tag{9}$$

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 γ_{cpu}, 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, the magnitude of u_{cpu} 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:

$$E_{sys} = \alpha_{cpu} u_{cpu} + \alpha_{mem} u_{mem} + \alpha_{io} u_{disk} + \gamma \tag{10}$$

where u_{mem} and u_{disk} represent the normalized value of N_{LLCM} and b_{io} respectively.

Two methods to learn the parameters for the above models are described below.

4.2.1 Base Server Model Learning

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. To generate a sufficient number of observations resulting in linearly independent equations and spanning a large range of resource usage values, we load each resource using a controlled workload (such as shown in Figure 2). This can be carried out once at server installation time before any VMs are instantiated, and the model parameters used at run time for VM power metering using equations (2), (4), and (8).

We refer to the model learned using this method as *base server model*. This method is conceptually similar to prior methods for full system energy estimation [23].

4.2.2 Refined Model Learning

The base model learning approach gives good estimates when linearity holds. Such linearity holds when the servers are used for homogeneous workloads such as used in [6], where power was obtained as a linear function of CPU utilizations.

The problem with the base method however is that linearity does not necessarily hold across multiple workloads. For a cloud platform hosting multiple VMs, the workload is no longer homogeneous for an entire server and different VMs may internally run very different applications. Also, the dynamic power used by a single VM is smaller than a full server and hence to obtain reasonable percentage errors, a much smaller absolute error is required in the estimate.

The simple utilization based models do not capture all aspects of the resource power states. Consider, for instance, the three benchmarks from the SPEC CPU 2006 suite shown in Table 1. The table shows the processor utilization, memory utilization, and the measured power (averaged over one run of each benchmark and excluding the idle power of the platform) for these benchmarks (IO is negligible for SPEC CPU benchmarks). Comparing `gobmk` and `omnetpp`, in the linear model of (10), 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. We could try a negative value for α_{mem} but then, 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. The linear model in 10 cannot capture these effects because certain

	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.

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. The use of the processor's floating point unit may on the other hand increase power usage at the same utilization. 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.

To capture such differences across different workloads, we 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.

In this approach, equation (10) is modified to be written as a sum of individual VM power values. Each unknown model parameter is assigned a separate variable for each VM. Suppose there are n VMs, then α_{cpu} is split across these VMs into n distinct parameters $\alpha_1, \alpha_2, ...\alpha_n$. The root VM, which typically has very low resource usage, is assigned the base model parameters.

Whenever a new VM enters the system, power and resource usage values are collected for some time and linear regression is repeated to determine the unknown parameters for that VM. Model parameters for multiple new VMs can be learned at the same time as demonstrated in our experiments. The model parameters specific to each VM are used for metering its power for as long as that VM resides on the server or when it returns to the same server. Servers with similar configurations could share the model parameters for a VM when if it migrates across homogeneous hardware. This models learned by this method are denoted *refined models*.

Model Adaptation: This approach naturally allows for model updates when application characteristics change, or hardware configurations change. Joulemeter can continuously track the error between the sum of estimated power values for all VMs running on a server and the measured server power. Whenever the error exceeds a threshold (based on previously observed error values), the model can be re-learned from the recent data.

5. IMPLEMENTATION

A block diagram of the JoulemeterVM energy metering system is shown in Figure 4. The block labeled *workload* represents the set of VMs hosted on the server.

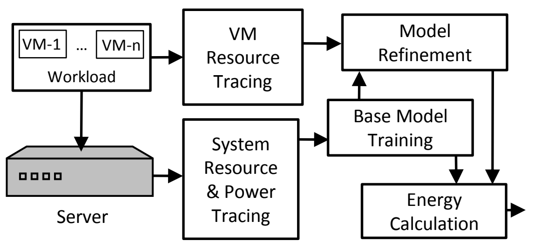


Figure 4: Joulemeter block diagram.

The *system resource and power tracing* module reads the full system CPU, disk, and power usage.

The *VM resource tracing* module uses hypervisor counters to track individual VM resource usage. Tracing of LLC misses was included in our lab prototype for experimentation (Figure 2) but is not intended to be used in Joulemeter.

The *base model training* module implements the learning approach from Section 4.2.1 and the *model refinement* block implements the learning method from Section 4.2.2. The output of base model training and previously learned VM models are used to reduce the number of unknowns in regression when a new VM model is being learned.

The *energy calculation* block uses the resource tracing data and model parameters in equations (2), (4), and (8) to output VM energy usage.

The system is implemented using Windows Server 2008 R2 Hyper-V but the concepts extend to other hypervisors as well.

6. EVALUATIONS

This section presents our experiments to evaluate Joulemeter. We investigate the errors in power metering for the base method as well as the refined method with various mixes of VM workloads.

6.1 Experimental Setup

The metering prototype is tested on a Dell PowerEdge R610 rack-mount server with two Intel Nehalem L5520 quad core hyper-threaded processors, resulting in 16 logical cores. The server includes 12GB RAM and 4 SAS 300GB 10k hard drives. The first drive is used for the host OS (Windows Server 2008 R2 with Hyper-V) and the other three are in a RAID 5 array exposing a single local drive. Storage for all VMs was allocated from the RAID5 array. Power data exposed to the OS is internally measured by the server in its two power supplies, at the AC input, and thus represents total power drawn including any losses in the power supply itself [16].

The built-in power sensors in different servers update their readings at different intervals. Hence we implemented a controlled excitation scheme that pulses the CPU load, generating rapid changes in power, at a known rate. By tracking the power updates, we can learn the update interval of the power sensor. We then average all hypervisor resource usage counters over the same update interval, to remove this non-linearity from the model training data. For the Dell server used, the update interval was 2s.

The benchmarks used to represent applications hosted inside the VMs consist of the SPEC CPU 2006 (`www.spec.org`) suite (processor intensive workloads), and IOmeter (`www.iometer.org`) benchmarks (IO intensive workloads). Multiple VMs were hosted on the server resulting in various mixes of workloads. Each VM was assigned 4 virtual processors allowing it to run at most 4 hardware threads. For single threaded benchmarks, multiple instances of the benchmark hosted in a VM were run to use multiple cores. Given 16 logical cores on the server, up to 4 VMs could be hosted without sharing any cores.

6.2 Base Power Model Training Error

We first evaluate the accuracy of power models, using the base learning method. The server was loaded using controlled loads that exercised the multiple cores and disks, while power was metered. Figure 5 shows an example trace of the power estimate produced using the base model, when the server was subjected to randomly varying loads, generated using the same controlled load generator as was used for generating the training loads. The power measured by an externally connected power meter (WattsUp PRO ES) is also shown for comparison. The base server model can be used to get

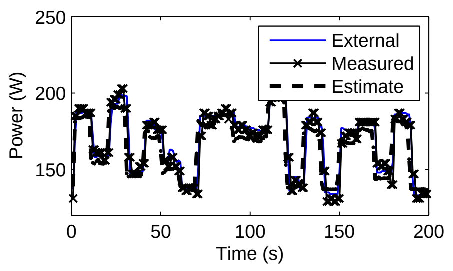


Figure 5: Base server model used for a sample random run.

approximate power usage statistics before VM workload specific refined models have been learned. Also, this power model (such as learned using an external power meter in a lab before production use or at the manufacturer site) can be used to obtain server power for older servers that do not have built-in power sensors built-in power meters or for servers that do not expose their power meter to the OS.

Base Error: To evaluate the base model error, we perform measurements similar to that in Figure 5 for multiple benchmarks. Figure 6 shows the errors for various workloads. The error is calculated as follows. For each run, the instantaneous error between the measured power and estimated power at each time step is measured. The absolute values of the instantaneous errors are averaged over the entire run. The errors reported are averaged over multiple benchmark applications comprising a particular workload. The workload labels SPEC and IO correspond to SPEC CPU 2006 and IOmeter based benchmarks. Label 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. The error is comparable to prior works [23]. A 5% error on this server corresponds to approximately 10W at high utilization levels.

Tracing Overhead: Running Joulemeter only involves reading the performance counters used in the power models and performing scalar multiplications. Occasionally regression matrices are inverted. With n VMs and 2 unknowns per VM, even if regression is performed for all VMs at once, the matrix is only $2n \times 2n$. With the number of VMs on a server usually being small, up to a few 10's of VMs per server, this matrix inversion is trivial. Since Joulemeter runs in the root VM, we tracked the resource usage of this VM itself both 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 root VM was too small to be measurable with respect to noise.

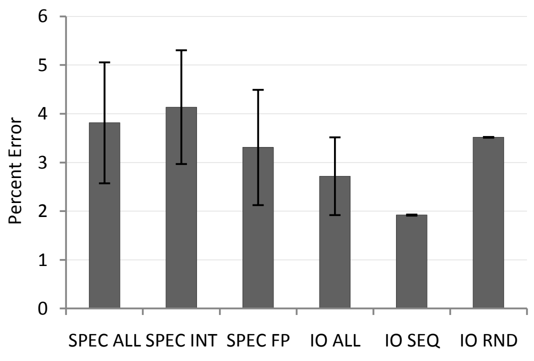


Figure 6: Power model errors for Dell PowerEdge. Error bars show standard deviation.

6.3 VM Energy Measurement

VM energy measurement uses the refined models. As an illustration, consider the CPU utilization and power tracked for multiple CPU intensive workloads from SPEC CPU 2006, shown in Figure 7. As can be seen, at any given CPU utilization, the actual power used can differ significantly across workloads. The spread can be greater when other types of workloads are included. The spread may increase further with greater use of heterogeneous processor cores, specialized hardware accelerators, and dedicated co-processors in newer architectures, that different applications leverage to varying degrees.

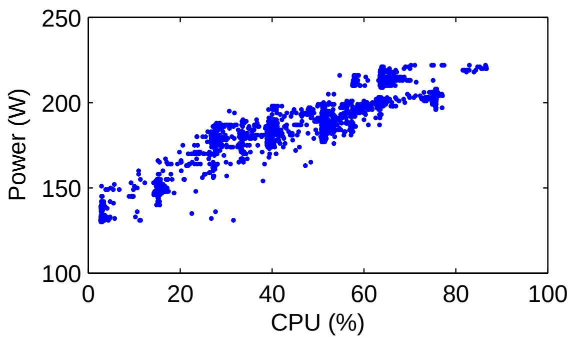
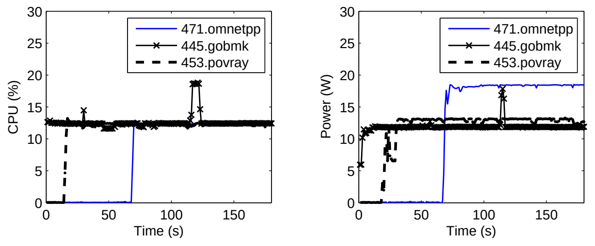


Figure 7: Power usage is not exactly linear with processor utilization. Power differences of exceeding 30W are seen at certain CPU utilizations. (Processor frequency was set to a single level for all measurements.)

For learning refined models, in our experiments, after a new VM comes in, its resources are tracked for 200s and the tracked data is used for regression. Here, 200s suffice because the VM was actively running benchmarks after instantiation. However, practical systems may have to wait longer as the VM may remain idle for long periods after starting up and the model learning can only take place after enough resource usage samples have been collected[2].

After the refined model is learned, the VM resources are tracked for an additional testing period. Figure 8(a) shows the CPU resource usage as an example for 3 VMs on a server, with each VM running a different benchmark from the SPEC CPU 2006 suite. Power data estimated using refined models in test runs is shown in Figure 8(b).

[2]The base model can provide approximate power usage during the waiting time.

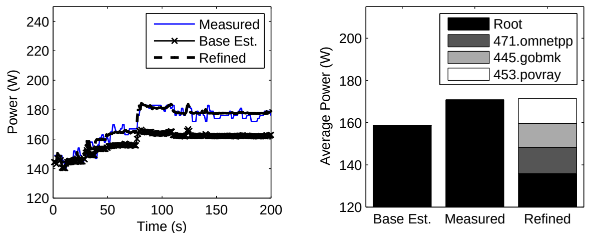


(a) CPU (b) VM Power

Figure 8: CPU and power usage for three VMs. The refined model assigns different power usage even when the CPU usage is similar.

Ground Truth: To evaluate the accuracy of VM energy usage, we wish to compare the Joulemeter VM power estimates to a ground truth. Obviously, no physical power meter can be connected to a VM to obtain a direct ground truth. Only the total physical server power can be measured. However, if the VM powers are estimated correctly and added to the static idle server power usage, the sum should match the total power usage of the server. When the sum is correct for multiple VM combinations at different times, it is very unlikely (though possible) that incorrect VM power estimates add up to the correct total power at all times. We can thus use the error between the estimated sum of power consumptions of multiple VMs and the measured server power as an estimate of Joulemeter accuracy.

The estimated and measured power values from a test run are shown in Figure 9(a). The estimate using the base server model is also shown for comparison. Average power usage across the entire trace assigned to different VMs for this run is shown in Figure 9(b). The static idle server power component is attributed to the root VM in these figures.



(a) Instantaneous Power (b) Average

Figure 9: Comparing the power measurement from the refined model with ground truth.

VM Power Errors: Measurements similar to Figure 9(a) are repeated for various mixes of workloads on multiple simultaneously active VMs on a server. The benchmarks running within VMs are listed in Table 2. A mixture of integer, floating point, and IO intensive workloads is used. The IO workloads include random and sequential access. The error results are shown in Figure 10.

Clearly, the refined models yield significantly improved errors in most cases. The error in total power is within a range of 1.6W-4.8W across all cases tested. We expect the error for each VM would to be even smaller: equally dividing the total error by the number of VMs, the per VM error is expected to be in the 0.4W-2.4W range.

Mix	# VMs	Benchmarks
(a)	2	471.omnetpp, IOmeter
(b)	2	445.gobmk, 444.namd
(c)	2	444.namd, IOmeter
(d)	2	453.povray, IOmeter
(e)	2	445.gobmk, 471.omnetpp
(f)	4	445.gobmk, 453.povray, IOmeter, 471.omnetpp
(g)	4	IOmeter, 471.omnetpp, 444.namd, IOmeter

Table 2: Workload mixes for errors shown in Figure 10.

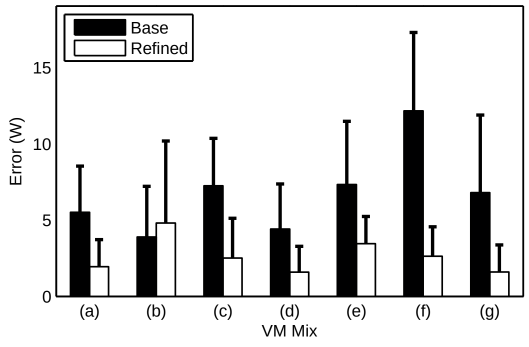


Figure 10: VM error evaluation on multiple VM mixes.

7. APPLICATIONS IN POWER MANAGEMENT

VM power metering has many potential applications in cloud computing and virtualized data centers. We discuss two concrete examples in detail: power provisioning and power tracking, and describe several others briefly.

7.1 Power Provisioning

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.

Power infrastructure is a major capital investment of a data center. It can take up to 40% of the total cost of data center construction [10]. Wastage of this expensive power capacity due to underutilization is a prevalent and well recognized problem [6]. In traditional power provisioning practice, the number of servers allocated to a provisioning unit (e.g. a rack or a power distribution unit) is calculated using the total usable power divided by the peak power consumption of a server. Since a server hardly runs at the peak power, nor will all servers on a circuit reach their peak at the same time, the traditional provisioning mechanism leaves a large amount of available power unused almost all the time. This is usually referred to as *stranded power*. Provisioning costs wastage includes the costs of power distribution circuits, UPS systems, backup generators, and fuel reserves maintained at the increased capacity.

Power *over-subscription* is actively being considered for reducing such waste [7, 18]. Since servers can only use their rated peak power if each server component is used to its peak, the rated peak is rarely achieved by practical applications. Thus more servers can be hosted if provisioned for their actual peak. Further, by taking advantage of statistical variance among power usage of multiple servers, additional servers can be included, say provisioned to the $x-$th percentile of their peak, using resource overbooking techniques such as [31]. However, all over-subscription techniques require a fail-safe, to manage the low probability event where power usage exceeds the capacity. This safety mechanism is called *power capping*, which restricts the server power usage to a specified cap. In physical servers, capping is implemented in hardware, and reduces the power used by throttling processor frequency lower levels. Many server manufacturers now offer this capability [11, 12].

7.1.1 VM Power Provisioning and Capping

Hardware throttling, however, is too rigid for virtualized environments. Reducing CPU frequency inevitably affects the performance of all VMs running on that server. Excesses of one VM can thus cause other VMs to suffer, violating the isolation property of virtualization, causing an undesirable noisy neighbors effect. 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.

Suppose, given a set of VMs and their resource usage traces, it has been consolidated to physical servers using any existing consolidation method. The resource constraints on the VMs automatically ensure that the server rated power is not exceeded. However, using Joulemeter data, we can determine the actual power usage, denoted $P_{act,VM}$ and its x-th percentile, denoted $P_{x,VM}$, for each individual VM. Other VM power consumption statistics, including variance and correlations among VMs, can also be computed from the Joulemeter power data. As an example for two VMs, let P_A and P_B be random variables representing the dynamic power consumption of VMs A and B, realized by their Joulemeter data trace from the past week. Then the consolidated server $A + B$ has statistics:

$$\begin{aligned} E(P_{A+B}) &= E(P_A) + E(P_B) \\ Var(P_{A+B}) &= Var(P_A) + Var(P_B) + 2Cov(P_A, P_B) \end{aligned}$$

The above formulas are easy to generalize to n VMs. The statistics computed using Joulemeter data can be used to set power caps to get two benefits:

Capping with Isolation: Joulemeter allows reproducing the benefits of physical server capping but with VM isolation. The total power consumption for the server can be calculated as $P_S = \sum_{i=1}^{n} P_{act,VM(i)}$. The server power cap may be set to $P_{server} = P_S + \Delta_m$, where Δ_m is a safety margin. Joulemeter tracks the actual power use per VM and allows the cap to be enforced on each VM separately, maintaining isolation of workloads. P_{server} set in hardware acts purely as a fail-safe as capping happens at the VM level. The VMs to be capped may be selected based on which ones are violating their budgets or based on importance of their workloads.

Risk-taking Benefit: With a good power capping technique, the data center operators can take more risk when consolidating VMs, and even set $P_{server} < P_S$. For instance, they could set the cap to $P_{server} = \sum_{i=1}^{n} P_{x,VM(i)} + \Delta_m$, which ensures that each VM is provisioned to its $x-$th percentile.

This will allow even more VMs to be consolidate into fewer servers, much like statistical over-subscription of physical servers to a rack. However, the benefit of virtualization is that the peak of the total power will exceed the total server cap much less than (100-x)% of the time due to statistical variations among VMs. The risk taking benefit will be higher if correlations among VMs are low.

7.1.2 Capping Mechanism

Joulemeter also facilitates the capping mechanisms. With virtualization, in case the peak is exceeded, and the cap has to be enforced, the capping mechanism is not restricted purely to throttling the processor. One may migrate certain VMs to servers that are not exceeding their caps. Again, Joulemeter power metering outputs help determine how the migration of each VM will affect power levels. Without the VM power data one would be restricted to using resource utilizations for making power related migration decisions and that resource usage is only an approximate indicator of actual power use (Figure 7).

Of course, migration may not always be feasible such as when no server within the migration cluster has excess capacity and then power use has to be reduced locally. While frequency scaling is not possible in current hardware for individual VMs, the CPU usage of individual VMs can be controlled resulting in per VM power caps [20]. Joulemeter models can help determine how much the CPU usage of different VMs will have to be reduced to achieve the desired power level.

7.1.3 Production Traces

To evaluate the benefits of VM power capping, we use server resource usage traces from three thousand servers from a production data center serving real world applications hosted in Microsoft data centers. In particular, we use two datasets:

Online Application: We collected resource utilization traces from 1000 servers hosting a single large online application, Windows Live Messenger, 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 2000 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 11.

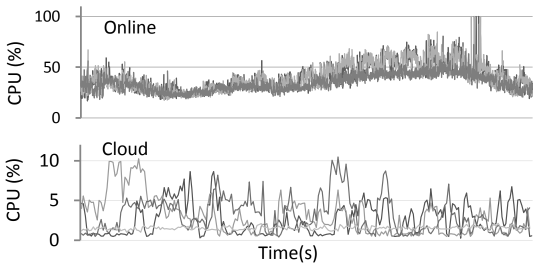


Figure 11: Sample server usage traces.

7.1.4 Evaluation

Isolation Savings: Using the above provisioning and capping method, we can achieve the same kind of over-subscription savings on VMs, as 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.

Even though these savings are similar to what was earlier achieved with physical servers, they 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[3], 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 metering and capping, one could achieve either the consolidation savings (15% to 63% consolidation via virtualization, no capping) or the over-subscription savings (Table 3 savings via hardware capping, no consolidation), but not both. Using Joulemeter allows achieving both at the same time.

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

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

Risk-taking Savings: As mentioned, VMs make it easier to provision for x-th percentile instead of actual peak. Figure 12 shows the additional reduction in provisioned power achieved for the Online and Cloud application datasets when provisioning for $P_{x,VM}$, compared to provisioning for $P_{act,VM}$, for varying x and with $\Delta_m = 10\%$. The power characteristics of the Dell PowerEdge server are used in computing the savings.

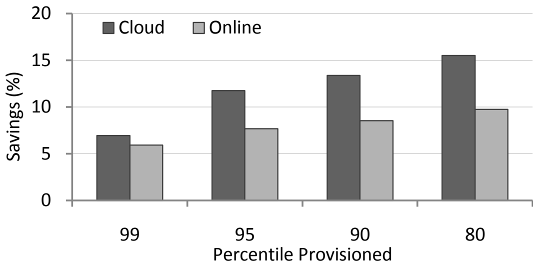


Figure 12: Provisioned capacity savings with Joulemeter.

The savings are shown when every 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 13. As expected, for the Online application, since the VMs are highly correlated in usage, the cap is enforced more often than for the Cloud applications. However, in either case the cap is enforced significantly less often than the tolerable limit $(100 - x)\%$ set during provisioning.

Figures 12 and 13 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 er-

[3]Considerable prior work exists in determining appropriate consolidation strategies for VMs, and for this experiment and we use a commonly used heuristic, first fit decreasing, for consolidation.

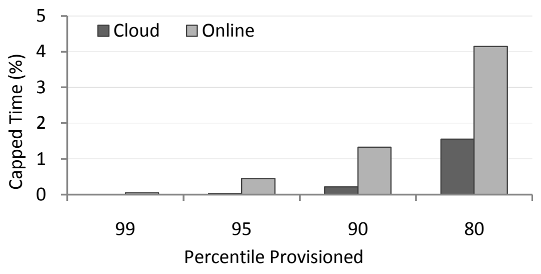


Figure 13: Percent of time capping had to be enforced.

ror 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 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 $(100-y)\%$.

7.2 Power Tracking

Another application of VM level power measurement is to follow a particular power availability level. Many data centers are considering using renewable energy such as solar, wind, or tidal power as part of the power supply [28]. Usually renewable energy production varies with weather conditions. Also, electricity price varies every few minutes on the whole sale market and can be exploited by large data centers to reduce operating costs [21].

To take advantage of these variations, we can control resources such that certain VMs consume only the amount of energy available from the low cost source. Among data center workloads, some are customer facing and must be processed in real time, but there are many others, such as search index crawlers, financial data analysis, database back-ups, and disk scrubbing, that are flexible and can be scheduled to opportunistically take advantage of lower power price or renewable energy availability.

This can be modeled as a control problem that tracks a target energy consumption level based on renewable energy availability. We describe a simplified version in this section, to illustrate the benefit of using VM power metering. Assume a server is hosting two kinds of workload. Let $x_1(k)$ be the amount of high priority workload at discrete time step k, and $x_2(k)$ be the amount of low priority workload at k. Let $d_1 k$ and $d_2(k)$ be a amount of workload arrived between time k to $k+1$. Assume that we use CPU utilization caps u_1 and u_2 as the control knobs for resources allocated to workload 1 and 2. Then we have,

$$x_1(k+1) = x_1(k) + a_{12}x_2(k) + d_1(k) - b_1u_1(k) \quad (11)$$
$$x_2(k+1) = x_2(k) + a_{21}x_1(k) + d_2(k) - b_2u_2(k) \quad (12)$$

where a_{12} and a_{21} represents the cross interference between the two classes of workload, and b_1 and b_2 represents how much workload can be processed by per unit of CPU allocation.

Now the total power consumption of the server is

$$P(k) = \alpha_1 x_1(k) + \alpha_2 x_2(k).$$

where α_1 and α_2 are power model coefficients for the two VMs. Assume that we can only measure the total power consumption P and have to use that to determine u_1 and u_2 such that $x_1(k+1) = 0$. From control theory, we know that if $a_{12} = 0$ and $a_{21} = 0$, i.e., there is no interference between the two VMs, then the system is not observable and one cannot design a controller based on P alone. If there is interference among the workloads, then designing a controller is possible but the controller is sensitive to the accuracy of the interference model. On the other hand, if we can measure VM power values $\alpha_1 x_1$ and $\alpha_2 x_2$ directly, then designing a stable controller is trivial.

7.3 Discussion

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.

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 or in-house servers, by connecting hardware power meters, 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: The cloud operators can use Joulemeter mechanisms to offer energy use based pricing schemes. This will enable them to extend their own benefits from demand-response electricity pricing and renewable energy availability to their customers and align the usage patterns in a jointly optimal manner.

Energy Budgets: Prior work [33] 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.

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 addressed the challenges involved in obtaining a low estimation error for cloud scenarios with several different types of workloads sharing the same hardware. Our solutions also adapts to changes in workload characteristics and hardware configurations. 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 virtualized data centers. Experiments using production server data sets from a commercial data center showed 8% to 12% additional savings aside from reproducing the savings that existed without virtuzlization but with the benefits of application isolation. Several other uses for VM energy metering were discussed. We believe that providing VM level energy visibility can enable both cloud platform operators and cloud application developers to improve the efficiency of their systems.

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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

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

[14] 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.

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

[16] J. Jenne, V. Nijhawan, and R. Hormuth. Dell energy smart architecture (desa) for 11g rack and tower servers. http://www.dell.com.

[17] 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.

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

[19] 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.

[20] 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.

[21] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the Electric Bill for Internet-Scale Systems. In *ACM SIGCOMM*, Barcelona, Spain, August 2009.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] C. Stewart and K. Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *Workshop on Power Aware Computing and Systems (HotPower), at SOSP*, October 2009.

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

[30] 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.

[31] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in a shared internet hosting platform. *ACM Trans. Internet Technol.*, 9(1):1–45, 2009.

[32] 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.

[33] 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.