Power and Energy Containers for Multicore Servers*

Kai Shen   Arrvindh Shriraman†   Sandhya Dwarkadas   Xiao Zhang‡

Technical Report #970
Department of Computer Science, University of Rochester

November 29, 2011

Abstract

Energy efficiency and power capping remain growing concerns in server systems. Online applications continue to evolve with new features and dynamic client-directed processing, resulting in varying power profiles. New computing platforms with multicore resource sharing and heterogeneity further obfuscate the system behaviors, presenting challenges for request/client-based energy accounting, identification and control of power viruses, as well as energy-efficient load management.

This paper presents a new operating system facility we call "power and energy containers" that accounts for and controls the power/energy usage of individual fine-grained requests in multicore servers. Our power and energy containers are enabled by three techniques— 1) online attribution of multicore power (including shared maintenance power) to individual tasks running concurrently on the multicore, 2) alignment of actual power measurements and model estimates to enable online model recalibration, and 3) on-the-fly request tracking in multi-stage servers to isolate the power/energy contributions and customize control of individual requests.

We evaluate our request-level power/energy containers on three different multicore processors (Intel Woodcrest, Nehalem, and Westmere) using a number of open-source application workloads including a high stress benchmark that can mimic a power virus. Our power containers can be used to cap server power consumption in a fair manner by penalizing only power-hungry requests. The container energy profiles can also identify request affinity and guide request distribution between heterogeneous servers. Our case study shows up to 18% energy saving compared to an alternative approach that recognizes machine heterogeneity but not per-request affinity.

*This work was supported in part by the National Science Foundation (NSF) grants CCF-0448413, CNS-0834451, CCF-0937571, and CCF-1016902. Shen was also supported by an IBM Faculty Award and a Google Research Award.

†Shriraman is currently affiliated with Simon Fraser University.

‡Zhang is currently affiliated with Google.

Keywords: multi-core, server system, power throttling, energy optimization.

1 Introduction

Designers of data centers and server systems put high priorities on improving energy efficiency, controlling peak power draw, and monitoring resource usage anomalies. Online applications are continuously evolving with new features and some (like social networking, Wiki sites, and online collaboration) rely on end users to supply content or even content-generating code. The workload diversity and dynamic client-directed processing can result in large power fluctuations on modern processors with increasing specialization and power proportionality [6]. In particular, extreme power-consuming tasks (or "power viruses") [25] may appear accidentally or be maliciously devised. Isolating per-client power attribution to identify such tasks so as to cap the system power draw in a fair fashion is highly desirable. Further, recognizing the energy usage of individual requests helps inform the full costs of web uses, as illustrated vividly in the public exchanges between a scientist [50] and Google [29]. Additionally, the economics of incremental data center upgrades, along with low-power designs and specialization, lead to widespread heterogeneity in server clusters. Exploiting the affinity of the diverse workload to heterogeneous platforms is beneficial for realizing high energy efficiency.

Previous research (particularly resource containers [4], Magpie [5], and our hardware counter signatures [43, 44]) has recognized the need for profiling and isolating per-request resource usage in a server. Applying this concept to request power/energy accounting and management on multicore servers is challenging. Concurrent task executions, varying power consumption, and dynamic hardware component sharing in a multicore processor leads to complex per-task power behaviors. Direct power measurements on such spatial and temporal granularities are not available on today's systems. Further, request executions in a concurrent, multi-

stage server contain fine-grained activities with frequent context switches. Finally, request-level power and energy management requires agile, low-cost control to ensure isolation and achieve efficiency.

This paper presents a new operating system facility, called *power and energy containers*, to account for and control the power/energy usage of individual requests in multicore servers. The characterization of request power/energy behaviors provides a detailed understanding of the server system power profile, and facilitates fine-grained attribution of energy usage to clients and their individual requests. We develop three key techniques to support power and energy containers:

- We attribute the multicore power consumption to individual tasks running concurrently on a multicore system. Beyond previous event-driven power models [7, 8, 16, 18, 30], we capture the shared multicore maintenance power and dynamically attribute it to actively running tasks at runtime. For low overhead, the power accounting is performed independently at each CPU core without global coordination.

- Power modeling inaccuracy [36] may result from different characteristics between calibration workloads and production workloads, particularly for unusually high-power applications. Online power measurements can help recalibrate power modeling but measurement results often arrive with some delays. We align power measurements and modeling estimates using signal processing cross-correlation.

- Utilizing an application-transparent, online request context tracking mechanism, we isolate the power consumption contribution of each individual request, and enable client/request-oriented accounting of power and energy usage. Online request context tracking also allows the selective adoption of power and energy control mechanisms for certain requests.

Our power and energy containers enable the first-class management of multicore server power/energy resources in unprecedented ways. This paper will demonstrate that:

- It can pinpoint the sources of power spikes and anomalies. It can further condition the request power consumption in a fair fashion—throttling the execution of power viruses (using processor duty-cycle modulation) while allowing normal requests to run at full speed.

- It can improve energy efficiency in a heterogeneous environment through *container heterogeneity-aware* load distribution. Request energy profiles on different machines are used to understand each request's cross-machine relative energy efficiency and direct its execution accordingly.

We will present the design and implementation of our power/energy containers and examine the above case studies in this paper.

2 Related Work

Power measurement and modeling: Per-hardware-component power can be measured through elaborate embedded instruments (as in LEAP [37]). The Intel Sandy Bridge processors can read voltage and current via on-chip digital voltage regulators. It provides power measurement at the level of multi-core package (but not per-core measurement). Fundamentally, directly measuring per-core power on a multicore chip is difficult due to shared use of components such as cache and memory interconnect.

Bellosa [7] suggested that the processor and memory power may be estimated using a linear model on hardware event counts. Economou *et al.* [18] combined hardware event counters with software metrics to create full-system power models. Isci and Martonosi [30] and Bertran *et al.* [8] further utilized the hardware counter statistics not only to estimate per-component power usage but also to predict power phases. Alternatively, accurate per-component power estimation can be acquired through detailed processor and memory models [9, 33]. On the negative side, McCullough *et al.* [36] identified several factors for high power model errors (including multicore complexities and hidden device states) and advocated direct power measurement. Power modeling has also been included in processor designs like the IBM POWER7 [48] to monitor runtime activities and adaptively configure the processor frequency.

Direct measurement or hardware event-based modeling alone is limited by their inability to identify and isolate concurrent resource principals in a server environment. This paper improves multicore server power modeling by considering cross-core environmental factors and aligning measurements and modeling estimates for online recalibration.

System-level energy accounting: By coordinating the external power measurement with interrupt-triggered program sampling, Flinn and Satyanarayanan [22] were able to profile the energy usage of application processes and procedures. ECOSystem [51] was proposed as a unified framework of whole-system energy accounting to support energy management policies. The ECOSystem work used relatively simple component power models (for instance, the CPU power consumption is assumed to be constant during busy periods). The Quanto system [23] combined the information of component power states, high-resolution energy metering, and causal tracking of system activities to profile energy usage in embedded network devices. Kansal *et al.* [31] employed a re-

source usage-based power model to track virtual machine power consumption in cloud computing platforms. Most recently, the Cinder operating system [42] employed new control abstractions (isolation, delegation, and subdivision) to account for and manage energy in mobile devices. In comparison to these techniques, our work tackles the challenges of power attribution in two new dimensions—over shared-resource multicore processors and among concurrently running fine-grained requests.

Adaptive power management: Weiser *et al.* first proposed adjusting the CPU speed according to its utilization [49]. The basic principle is that when the CPU is not fully utilized, the processing capability can be lowered to improve the power efficiency. Later efforts included automatic setting of performance goals at the operating system [21], request batching to extend the CPU low-power state [19], energy management directed by performance guarantees [34], coordination of multiple interacting power management techniques [14, 41], and fast transitioning between active state and minimum-power nap state [38]. In comparison to these studies, our contribution is the online power/energy profiling of individual request executions in a server system. Our model provides fine-grained accounting of power consumption and energy usage to user requests. It can also enable new system support for request-level quality-of-service and power/energy policies.

Cluster/warehouse-level energy/power management: In light of the load burstiness at large-scale service sites, Chase *et al.* [13] and Pinheiro *et al.* [40] proposed to consolidate services to a subset of servers at load troughs while the remaining servers can be shut down to conserve energy usage. In terms of data center power provisioning, Fan *et al.* [20] and Govindan *et al.* [26] suggested that the power provisioning should take into account the independence as well as correlation of power fluctuations at individual servers. On the system architecture, Lim *et al.* [35], Caulfield *et al.* (Gordon) [10], and Andersen *et al.* (FAWN) [3] showed that low-power processors and flash memory can significantly enhance the data center energy efficiency. In addition, Heath *et al.* [28], Nathuji *et al.* [39], and Chun *et al.* [15] have recognized the energy effects of load placement in a heterogeneous server cluster. Our request profiling and management on multicore servers are complementary to these systems. Specifically, the capture and control of power virus requests is an important part of the system power management goals. Further, our container-enabled request energy usage profiles can identify the request energy tradeoff across heterogeneous multicore servers and thereby address a key challenge for heterogeneity-aware request distribution.

3 Power and Energy Containers

We propose a new operating system facility that accounts for and controls the power/energy usage of individual requests in a multicore server. We tackle the challenges of power attribution and control in two new dimensions—1) over concurrent executions on a shared-resource multicore and 2) among fine-grained requests in a multi-stage server application. To tackle the first challenge, we model per-task power consumption from core-level activities and shared multicore chip power. In addition, we align online power measurements and modeling estimates to recalibrate the power model for better accuracy. To tackle the second challenge, we build operating system mechanisms to track multi-stage request executions on-the-fly, account for request power/energy usage, and apply request-grained power control.

3.1 Power Attribution to Concurrent Tasks

While the system power modeling has been extensively addressed in previous research [7, 8, 16, 18, 30, 36, 48], they largely focus on the whole system or full processor power consumption. However, multiple tasks may run concurrently in a multicore system and each task may belong to a distinct user request, which desires separate accounting. We present new techniques to attribute the power consumption to individual tasks running concurrently on a multicore system. This section focuses on the processor and memory power attribution while the accounting of I/O power consumption will be discussed in Section 3.3.

Our first approach follows Bellosa [7] and others [8, 16, 18, 30]'s model that the processor/memory power consumption is linearly correlated with the frequency of relevant hardware events. Example metrics of interests include the core utilization or the ratio of non-halt core cycles over elapsed cycles ($\mathcal{M}_{\text{core}}$), retired instructions per CPU cycle ($\mathcal{M}_{\text{ins}}$), floating point operations per CPU cycle ($\mathcal{M}_{\text{float}}$), last-level on-chip cache requests per CPU cycle ($\mathcal{M}_{\text{cache}}$), and memory transactions per CPU cycle ($\mathcal{M}_{\text{mem}}$). The constant power term in the linear relationship represents the idle power consumed when zero values for all metrics are observed. The remaining *active* (full minus idle) power can be modeled as:

$$\mathcal{C}_{\text{core}}\cdot\mathcal{M}_{\text{core}}+\mathcal{C}_{\text{ins}}\cdot\mathcal{M}_{\text{ins}}+\mathcal{C}_{\text{float}}\cdot\mathcal{M}_{\text{float}}+\mathcal{C}_{\text{cache}}\cdot\mathcal{M}_{\text{cache}}+\mathcal{C}_{\text{mem}}\cdot\mathcal{M}_{\text{mem}} \quad (1)$$

where $\mathcal{C}$'s are coefficient parameters for the linear model that can be calibrated offline (once for each target machine configuration). Equation 1 models the full system active power if $\mathcal{M}$'s capture the summed event metrics over all cores. We can also use it to account for the active power of an individual task if $\mathcal{M}$'s capture the metrics on the CPU core where the target task is currently running.

We implement such event-based power accounting in the operating system. Each core performs accounting for its local task independently without cross-core synchronization or coordination. We acquire per-core system metrics ($\mathcal{M}$'s) online by reading processor hardware counters and computing relevant event frequencies. The continuous maintenance of the power model and hardware counter statistics requires periodic counter sampling. We configure the core-local Programmable Interrupt Controller for threshold-based event counter overflow interrupts. Specifically, we set the interrupt intervals to a desired number of non-halt core cycles. *Non-Halt* cycle-based triggers have the benefit of suppressing the interrupts when the CPU core has no work to do (so it can continuously stay in the low-power idle state).

The above approach assumes that the power accounting for a task only depends on core-level physical events. On a multicore processor chip with intricately shared hardware resources, however, chip-wide environmental factors also affect per-core power accounting. In particular, the maintenance of shared multicore resources (including at least clocking circuitry and voltage regulators) consumes some active power as long as one core is running. But such consumption does not change proportionally with core-level event rates. Figure 1 illustrates this symptom using a very simple CPU spinning microbenchmark. This workload scales perfectly on the multicore so all observed event metrics scale proportionally with the CPU core utilization. The Equation 1 model would suggest a linear relationship between the core utilization and power. However, measurements show that the power increment from idle to one utilized core is at a much higher rate than further power increments, which suggests an active power component that does not scale with core-level physical events.

It is intuitive to evenly attribute the chip maintenance power at each time instant to the currently running tasks. The system utilization level fluctuates over time in production server environments [6]. Proper accounting and attribution of shared chip maintenance power is challenging because one task's share may change depending on activities (or the lack thereof) on other cores. We use a new metric $\mathcal{M}_{\text{chipshare}}$ to denote the proportion of a given task's share ($0.0 \le \mathcal{M}_{\text{chipshare}} \le 1.0$). If a core is busy while all other siblings are idle, the full chip power should be attributed to the task on the busy core ($\mathcal{M}_{\text{chipshare}}=1.0$). If multiple ($k$) cores are busy, then each running task on one of the busy cores has $\mathcal{M}_{\text{chipshare}}=\frac{1.0}{k}$. Our new active power model adds the shared chip maintenance power to the original model:

$$[\text{Equation } 1] + \mathcal{C}_{\text{chipshare}} \cdot \mathcal{M}_{\text{chipshare}} \tag{2}$$

Unlike the core-level event metrics that can be simply acquired through hardware counters on the CPU core,

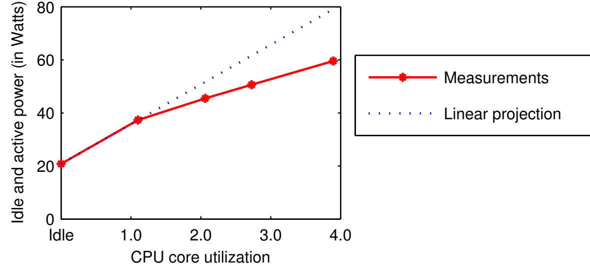

Figure 1: Relationship between processor/memory power and core-level activities on a quad-core Nehalem processor. We highlight its mismatch with a hypothetical linear projection.

the chip power share $\mathcal{M}_{\text{chipshare}}$ does not correspond to any processor hardware counter. Further, a precise sharing in a dynamic system depends on time-synchronized global activities across multiple cores. In our implementation, each core independently makes an approximate estimation while avoiding expensive global coordination or cross-core interrupts. We discretize the computation of $\mathcal{M}_{\text{chipshare}}$ over time intervals that match the hardware counter sampling periods for collecting core-level events. When the discretized time intervals (for example, 1 millisecond intervals) are much shorter than a typical CPU scheduling quantum, the probability for a scheduling decision made during an interval is low. Therefore each core is likely to be fully busy or fully idle within each interval. We approximate the number of busy sibling cores using the sum of latest core utilization ratios at all siblings on the multicore chip. Formally on an n-core processor, the task currently running on CPU core c has:

$$\mathcal{M}_{\text{chipshare}}(c) = \mathcal{M}_{\text{core}}(c) \cdot \frac{1}{1 + \sum_{1 \le i \le n, i \ne c} \mathcal{M}_{\text{core}}(i)}, \tag{3}$$

where $\mathcal{M}_{\text{core}}(x)$ indicates the core utilization ratio on CPU core x.

We check a sibling's core utilization by reading its most recent hardware counter sample in memory. Note that each CPU core performs independent sampling at non-halt cycle-triggered interrupts, which stop when the core is idle. Therefore an idle sibling may have stale sample statistics. To address this problem, we check whether the OS is currently scheduling the idle task on a sibling core and consider its current activity rate as zero if so.

3.2 Measurement Alignment for Model Recalibration

Despite the wide uses of event-based power models [7, 8, 30, 48], we found that large errors may arise in

practice. Recent research [36] has also raised questions on the accuracy of event-based power models. Beyond superficial problems like insufficient coverage of modeled events, significant modeling inaccuracy also results from differing characteristics between calibration workloads and production workloads. This is particularly the case for unusually high power-consuming production workloads that demand careful attention in server system power management. To address the modeling inaccuracy, we utilize online power measurements to adjust and recalibrate the offline-constructed power models.

While whole-system power measurements can be acquired through off-the-shelf meters, measurement results often arrive with some lag time due to the meter reporting delay and data I/O latency (*e.g.*, through a USB interface). On the other hand, processor event counter-based power models can be maintained at the much shorter latency of reading CPU registers and computing simple statistics. In order to use the power measurements to identify modeling errors and recalibrate the model, the measurement results and modeling estimates need to be properly aligned.

While a poorly calibrated power model does not accurately predict the power consumption, it may still identify power transitions and phases quite well [8, 30]. In other words, aligned power measurements should follow the fluctuation patterns of real-time power model estimates. This motivates us to employ a signal processing approach to align measurement samples and model estimates. Specifically, we compute the cross-correlation metric between measurement and model power samples at different hypothetical measurement delays. A higher cross-correlation would indicate better matching of the measurement/model fluctuation patterns.

Formally, let $\mathcal{P}_{\text{measure}}(0)$, $\mathcal{P}_{\text{measure}}(1)$, $\cdots$ be the sequence of recent power measurement samples ($\mathcal{P}_{\text{measure}}(0)$ is the most recent). Let $\mathcal{P}_{\text{model}}(0)$, $\mathcal{P}_{\text{model}}(1)$, $\cdots$ be the sequence of recent modeling samples at the same sampling frequency. Then the cross-correlation at a hypothetical measurement delay t is:

$$\sum_{i=0}^{\text{number of matching samples}} \mathcal{P}_{\text{measure}}(i) \cdot \mathcal{P}_{\text{model}}(i+t). \quad (4)$$

Figure 2 shows a case example of alignment cross-correlation over hypothetical measurement delays from 0.0 to 5.0 seconds. The highest point of the high correlation peak (specially marked in the figure) is about 1.3 seconds, which indicates the likely measurement delay. Figures 3(A) and (B) show the original and aligned (with 1.3 second delay) measurement/model power samples respectively. These results are for an measurement using a current clamp and an external Agilent multimeter with a USB data connection. Alignment for a different setup using a Wattsup meter shows a measurement delay of about 2.1 seconds. Details on the power measurement setups will be provided later in Section 4. Note that the power measurements are for the whole system so the alignment uses the modeled whole system power (by adding per-core modeled power over all CPU cores).

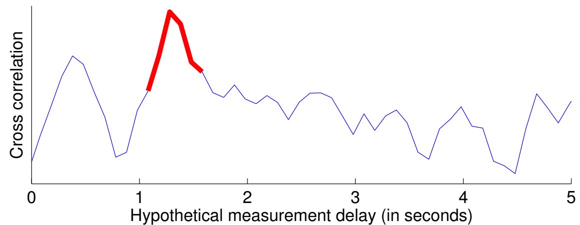

Figure 2: Measurement/model alignment cross-correlation for an Agilent power meter on a Nehalem machine. High correlation peak is specially marked in the figure.

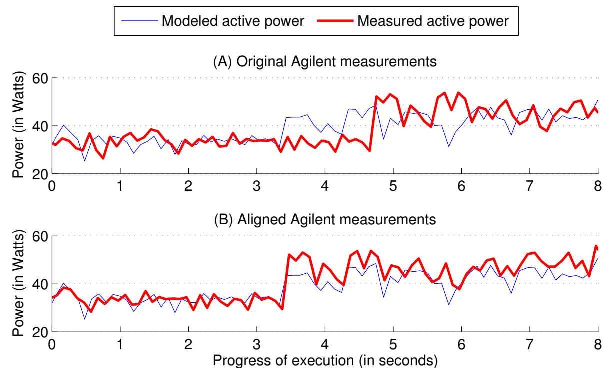

Figure 3: Original (A) and aligned (B) measurement/model power traces.

Aligned power measurements can identify modeling errors for the currently running workload and help recalibrate our multicore power model in Equation 2. The recalibration intuitively means adding new online sample of aligned system metrics and corresponding power measures to the original offline model parameter calibration. Specifically, we perform online least-square-fit linear regression to recalibrate the model parameters when new online samples are available. Our parameter recalibration includes both offline workload samples and current online measurements, weighed equally in the square error minimization target.

3.3 Request Power Accounting and Control

We construct operating system mechanisms to support request-level power/energy containers in a server system. Our first goal is to account for request power consumption and cumulative energy usage. Our second goal is to

enable request-specific power/energy control (*e.g.*, speed throttling) according to request-level policies on resource usage and quality-of-service. The online control requires each request's power/energy container to be maintained *on-the-fly*—while the request executes.

Our OS support tracks request execution contexts so that power-relevant metrics can be properly attributed and control mechanisms can be properly applied. A request context coincides with a process (or thread) context in some cases. However, request context tracking is challenging in multi-stage servers where a request execution may flow through multiple processes. For instance, a PHP web processor propagates the request context into a database thread through a socket message. In another example, a web server handler forks an external process to render an image for desired client presentation. The past approach of Resource Container [4] requires applications to explicitly pass request context bindings across server stages. To relieve the burden on applications, Magpie [5] logs system events transparently and analyzes their request context relations afterwards. Magpie's asynchronous analysis is highly flexible, but it is unable to control a request's power/energy behaviors while it executes.

We may identify a request context on-the-fly with recognized patterns on how it propagates over multiple server stages. Specifically, we recognize request context propagation events as those that indicate causal dependences through data and control flows between processes. The basic idea for online flow tracking through multiple system components isn't new, as in the X-Trace framework [24], our past work of workload signature identification [44], and Google's Dapper tracing infrastructure [45]. Our contribution is to develop the first system that supports request-level power accounting and control.

In our implementation, a process (or thread) binds to the context of the request it currently executes. Request contexts propagate between processes through socket communications, IPCs, and process forking. A request context is created when a socket connection is established on the designated server listening port. At the initialization time, the request context is bound to the process receiving the socket connection. A request context ends when no process binds to it. A process ends its binding to a request context when it exits or when a new request context binding arrives (*e.g.*, through a socket message from another request). The latter is important to support request pooling when a single process executes multiple requests over its lifetime. Our approach also supports persistent connections between server stages where one socket connection is reused by multiple request propagations over time. In such cases, the socket's old request binding is replaced when a message with a new binding (inherited from the sender) flows through the connection. Figure 4 illustrates a captured request execution and identified request context propagations across multiple components in a realistic server application. Note that the Apache processes support request pooling so their request bindings do not end until new bindings arrive.

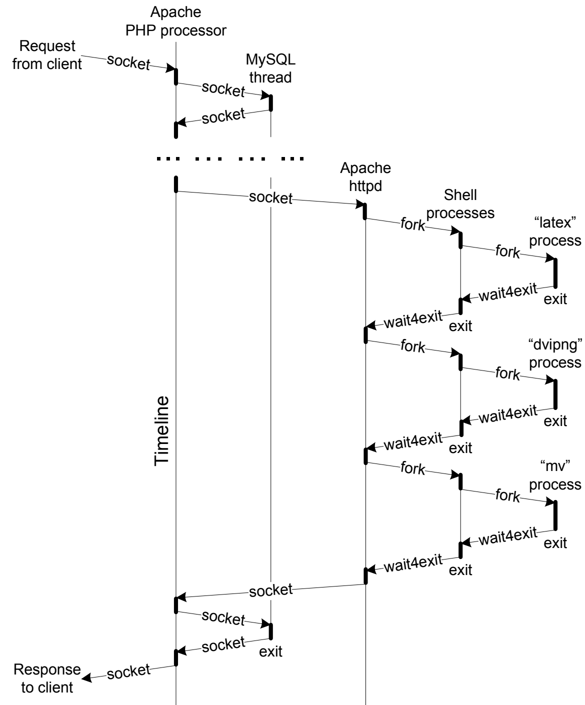

Figure 4: A captured request execution that involves Apache PHP processing, MySQL database, and various external operations on content and image rendering. This request is from the WeBWorK online homework system [47]. Identified data and control flows between server components are marked in arrows. Darkened portions of a component timeline indicate active executions (while the rest represent blocking waits).

Our current system is implemented entirely in the operating system requiring no application change. OS-only management, however, cannot track user-level request stage transfers in an event-driven server. Encouragingly, past research [11] suggests that some user-level request stage transfers may be observed by trapping accesses to critical synchronization data structures. We leave its implementation for possible future work.

During the course of a request execution, our system samples cumulative processor hardware event counters including elapsed non-halt CPU cycles, retired instructions, floating point operations, last-level cache reference counts, and memory transactions. It samples at multiple moments and calculates the counter metric for

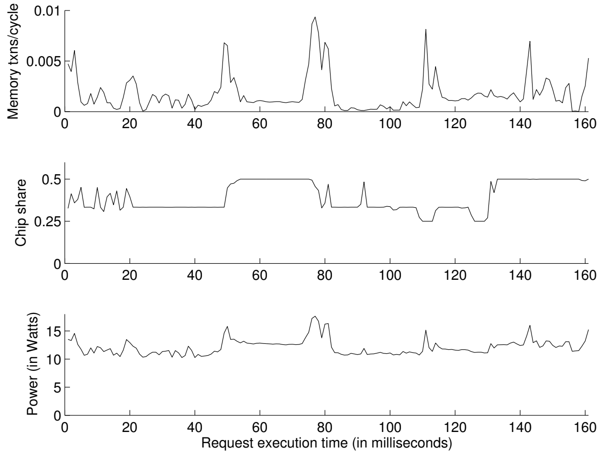

Figure 5: Power-related metrics and modeled active power over the course of a WeBWorK request on a Nehalem machine. The chip share metric indicates the task's share of the chip maintenance power ($\mathcal{M}_{\text{chipshare}}$ in Equation 3). This illustration serializes all request execution samples according to sample wall clock timestamps. Note that we do not suggest a canonical serialization of the request execution stages. Indeed some stages of a request may even execute in parallel.

each period between consecutive sampling. To maintain per-request event metrics, we sample the counter values at the request context switch time to properly attribute the before-switch and after-switch event counts to the respective requests. A request context switch on a CPU occurs in two scenarios—1) when two processes bound to different request contexts switch on the CPU; 2) when the running process receives a new context binding (*e.g.*, by an arriving socket message). In addition to context switch sampling, we sample at periodic interrupts (triggered by a desired number of non-halt core cycles) to capture fine-grained behavior variations that affect power.

With per-request event sampling, our power model in Equation 2 specifies the contribution from collected metrics to the request power. Beyond the processor/memory power, the full system power accounting should also consider power-consuming peripheral devices. Specifically we include the active power consumption of disk and network I/O operations in our power model. The OS can identify responsible requests for I/O operations by tracking the requests that consume the data received at I/O interrupts. With the estimation of request power at each sampling period, the cumulative energy usage can be simply calculated as the integral of power consumption over time. Figure 5 illustrates the modeled active power along with some power-related metrics over the course of a request execution.

In addition to power accounting, the on-the-fly request tracking allows us to selectively apply power and energy control mechanisms on certain requests. For example, we can apply a particular CPU duty-cycle modulation level to a given request execution. Effectively, when the request starts executing on a CPU core, the core duty-cycle level will be set appropriately. When the CPU core switches to run another request, the core duty-cycle level will be adjusted according to the policy setting of the new request. Our request-specific CPU duty-cycle modulation can limit the power consumption of selected requests without hurting the performance of others. We provide a case study of such request-level control in Section 5.1. We can similarly support request-specific dynamic voltage/frequency scaling if the multicore processor allows such scaling on a per-core basis.

A request container can be maintained over multiple machines in a server cluster. Our single-machine request container mechanism already tracks socket messages for request context propagation over multiple server stages. When a socket message crosses the machine boundary, we tag it with local request statistics including the cumulative runtime, cumulative energy usage, and most recent power usage. Additional information about request execution control may also be included. By tagging request messages, a dispatcher machine can pass container identifier and control policy settings to execution host machines. By tagging response messages, the execution host machines can pass cumulative power/energy usage information to the dispatcher machine for comprehensive resource accounting.

3.4 Overhead Assessment

We implemented a prototype power/energy container facility including per-request hardware counter sampling, power modeling, and statistics maintenance in Linux 2.6.30. Online container maintenance introduces overhead in the system. A container maintenance operation typically includes reading the hardware counter values, computing modeled power values, and updating request statistics. We measure its overhead on a machine with a quad-core Nehalem processor. Results show that one container maintenance operation takes about 1.3 microseconds. If the maintenance (hardware counter sampling) occurs at the frequency of once every millisecond (sufficiently fine-grained for many accounting and control purposes), the overhead is only around 0.1%.

Besides the overhead, the hardware counter sampling and statistics maintenance also produces additional activities (and energy usage) that do not belong to the inherent application behaviors. This behavior, called the *observer effect*, introduces perturbation in the power/energy profiles generated. We measure such maintenance-induced

event metrics and find that an average container maintenance operation produces 2948 cycles, 1656 instructions, 16 floating point operations, 3 last-level cache references, and no measurable memory transaction. Assuming all four cores are busy (1/4 chip share), the average energy usage for a container maintenance is about 11 micro-Joules according to our active power model in Equation 2. To mitigate the observer effect in our statistics collection, we subtract the measured event counts of each sampling period by the maintenance-induced additional event counts.

The measurement alignment and model recalibration (Section 3.2) also introduce online overhead. The measurement alignment does not need to occur frequently because the measurement lag time on a given system is unlikely to change dynamically. Our least-square fit model recalibration requires linear algebra computation that consumes about 16 microseconds per calibration. Its online overhead is negligible if it is performed at the frequency of once per second.

4 Request Power Evaluation

We evaluate our request power model on three different machines. The first is a multichip/multicore machine with two dual-core (four cores total) Intel Xeon 5160 3.0 GHz "*Woodcrest*" processors. Two cores on each processor chip share a single 4 MB L2 cache. The second machine contains a quad-core Intel Xeon E5520 2.26 GHz "*Nehalem*" processor. The four cores share an 8 MB L3 cache. The third machine contains two six-core (12 cores total) Intel Xeon L5640 2.26 GHz "*Westmere*" processors. The six cores on each processor chip share a 12 MB L3 cache. The three processors were publicly released in 2006, 2009, and 2010, using 65 nm, 45 nm, and 32 nm technologies respectively. And our Westmere processors are Intel-designated low-power version. We configure hardware event counting registers on each processor to assemble the input metrics for our power model in Equation 2.

We employ two different power measurement instruments. First, each machine uses a Wattsup meter that reports the whole machine power consumption once a second. In addition, we use a Fluke i410 current clamp applied on the 12V wires that supply power to the processor sockets. The clamp detects the magnetic field created by the flowing current and converts it into voltage levels (1mV per 1A current). The voltage levels are then monitored by an Agilent 34410a multimeter at the granularity of 100 samples per second. This measurement captures the power to the processor package, including cores, caches, Northbridge memory controller, and the quickpath interconnects. Both Wattsup and Agilent measurements are fed back to the target machine through the USB interface.

The full power includes a constant idle power consumed when the server exhibits no activity. The idle power is of little interest to modeling or resource provisioning since it is a constant. Including it in the power metric would make modeling errors look artificially small. Therefore we present evaluation results on the active (full minus idle) power. For interested readers, we briefly describe the significance of idle power on the Nehalem machine. For the processor/memory sub-system that our work mostly targets, its idle power is 20.8 Watts, or about 22% of total processor/memory power at an observed high load scenario. The idle power proportion increases to 58% when the full machine is considered. While the idle power is still significant on today's production machines, we are encouraged by continuous efforts and advances toward better energy proportionality [6].

4.1 Model Calibration

Our multicore server model in Section 3.1 requires an offline calibration to acquire the coefficient parameters. This calibration is performed once for a target machine configuration but is subject to measurement-based online recalibration as described in Section 3.2. We design a set of microbenchmarks that stress different parts of the system (including CPU spin with no cache access, CPU spin with high instruction rate, CPU spin with high floating point operations, high last-level cache access, high memory access, high disk I/O access, high network I/O, and finally a benchmark with a mixture of different workload patterns). For each microbenchmark, we use several different load levels (100%, 75%, 50%, and 25% of the peak load) to produce calibration samples. We use the least-square-fit linear regression to calibrate the coefficients for Equation 2.

As an example, we list the coefficient parameters of the calibrated offline model for the Nehalem machine. While a model coefficient $\mathcal{C}$ may not be intuitively meaningful, $\mathcal{C} \cdot \mathcal{M}^{\max}$ (where $\mathcal{M}^{\max}$ is the maximum observed value for the respective metric on the whole machine) would approximate the maximum active power impact of the metric—

$$\begin{aligned}
\mathcal{C}_{\text{core}} \cdot \mathcal{M}_{\text{core}}^{\max} &= 23.2\,\text{Watts;} \\
\mathcal{C}_{\text{ins}} \cdot \mathcal{M}_{\text{ins}}^{\max} &= 16.3\,\text{Watts;} \\
\mathcal{C}_{\text{float}} \cdot \mathcal{M}_{\text{float}}^{\max} &= 3.5\,\text{Watts;} \\
\mathcal{C}_{\text{cache}} \cdot \mathcal{M}_{\text{cache}}^{\max} &= 8.8\,\text{Watts;} \\
\mathcal{C}_{\text{mem}} \cdot \mathcal{M}_{\text{mem}}^{\max} &= 22.8\,\text{Watts;} \\
\mathcal{C}_{\text{chipshare}} \cdot \mathcal{M}_{\text{chipshare}}^{\max} &= 7.5\,\text{Watts;} \\
\mathcal{C}_{\text{disk}} \cdot \mathcal{M}_{\text{disk}}^{\max} &= 2.7\,\text{Watts;} \\
\mathcal{C}_{\text{net}} \cdot \mathcal{M}_{\text{net}}^{\max} &= 3.6\,\text{Watts.}
\end{aligned}$$

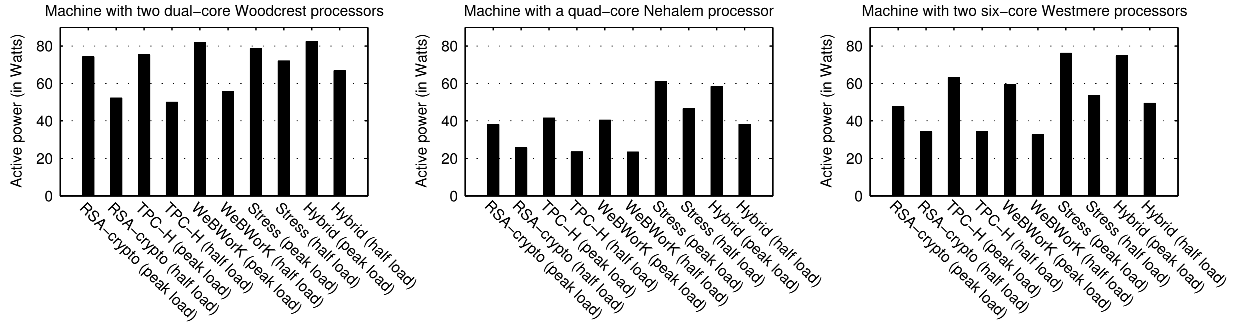

Figure 6: Measured active power of application workloads on three machines and two load levels.

4.2 Request Model Evaluation

Our evaluation employs several server-style workloads. All applications contain only open-source software.

- *RSA-crypto* is a synthetic security processing workload. Each request in this workload runs multiple RSA encryption/decryption procedures from OpenSSL 0.9.8g. It contains three types of requests each uses one of the three encryption keys provided as examples in OpenSSL. Each request typically runs for tens of milliseconds on our experimental machines.

- *TPC-H* is a database-driven decision support benchmark processing 22 complex SQL queries. Some queries require too much CPU time to be appropriate for interactive server workloads. We slightly modify them so each query takes no more than 2 seconds of CPU time. Our workload contains an equal proportion of requests of each query type. We use MySQL 5.5.13 database. We disable the index key cache in MySQL, which we found scales poorly on multiprocessors due to high contention on the key cache mutex lock.

- *WeBWorK* [47] is a web-based teaching application hosted at the Mathematical Association of America and used by over 240 colleges and universities. WeBWorK lets teachers post science problems for students to solve online. It is unique in its *user (teacher)-created content*—considered by some a distinctive "Web 2.0" feature [46]. Our installation runs Apache 2.2.8 web server, a variety of Perl PHP modules, and MySQL database. Tests are driven by around 3,000 teacher-created problem sets (ranging from pre-calculus to differential equations) and user requests logged at the real site.

- *Stress*, or Stressful Application Test [2], is a Google-released benchmark that runs the Adler-32 checksum algorithm over a large segment of memory with added floating point operations. It stresses the CPU core units, floating point unit, and cache/memory accesses simulatanously. This workload generates higher-than-normal power consumption, particularly on recent machine models (Nehalem and Westmere). We adapted it to a server-style workload with requests each running for about 100 milliseconds.

- *Hybrid* contains a mixture of WeBWorK and Stress requests. Approximately half the load is generated by each application.

Figure 6 shows the measured active power consumption for all the application workloads on the three machines. Our experiments employ a test client that can send concurrent requests to the server at a desired load level. We show results at two load levels—peak load when the target server is fully utilized, and half load when the server utilization is about 50%.

Both TPC-H and WeBWorK use multi-stage server architectures. The web server in WeBWorK also pools many request executions on each worker process. Our power container facility is able to properly track individual request activities and attribute request power consumption during concurrent multicore execution. Figure 7 shows the distributions of mean request power for TPC-H and Hybrid (mixture of WeBWorK and Stress) workloads on the Nehalem machine. We observe varying request power consumption in the results. In the Hybrid workload, Stress requests consumes substantially higher power than WeBWorK requests due to their intense CPU/memory activities. Figure 8 further shows the request energy usage distributions. The varying request energy usage is due both to request power variation and to their execution time difference.

We validate the accuracy of acquired request power/energy profiles. A direct validation is infeasible for several reasons. First, it is challenging to directly

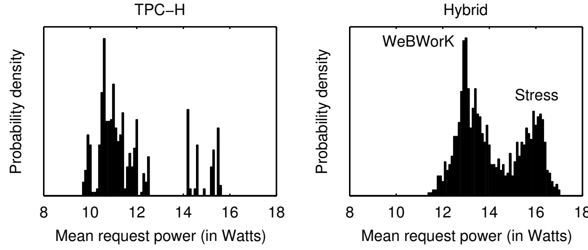

Figure 7: Mean request power distributions (in histograms) for TPC-H and Hybrid workloads on the Nehalem machine. The mean power for a request is the average power consumption over the course of the request execution. For the Hybrid workload, we label the major distribution masses for WeBWorK and Stress requests respectively. Results were collected when each workload runs at half server load.

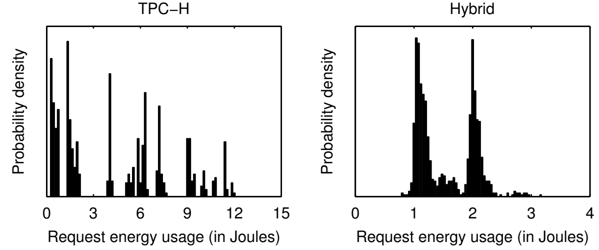

Figure 8: Request energy usage distributions (in histograms) for TPC-H and Hybrid workloads on the Nehalem machine.

measure power attribution to concurrently running tasks on a hardware resource-sharing multicore. Even if such a measurement mechanism exists, it must coordinate with fine-grained request activities (particularly frequent context switches) to directly measure the power profile for individual requests. Given such difficulties, our validation uses the acquired request energy profiles to estimate the full system active power, which can then be compared to the measured system active power.

Specifically for a running system, our energy containers can profile the energy usage of all request executions that fall into a given time duration. The sum of all request energy usage, divided by the time length, will produce an estimation of the average system power consumption. We validate such an estimation with measured system power. To understand the benefits of our proposed techniques, we compare the validation accuracy over three different approaches:

- The first approach (presented in Section 3.1) employs a linear power model on a set of core-level event metrics. It does not consider the shared chip maintenance power.
- The second approach (also presented in Section 3.1) additionally accounts for the multicore chip maintenance power and attributes it to concurrently running requests.
- The third approach (presented in Section 3.2) further employs measurement-aligned online model recalibration to mitigate the impact of differing characteristics between offline calibration and production workloads.

Validation results in Figure 9 show that our techniques are effective in producing accurate request power/energy profiles. Averaging over all workloads, the approach of only modeling core-level events exhibits 17%, 18%, and 24% validation errors for the three machines respectively. Attributing shared multicore power reduces the validation errors to 8%, 12%, and 19% for the three machines. The measurement-aligned online model recalibration further reduces the errors to 4% or so on all machines. The measurement-aligned recalibration is particularly effective to improve the request profiling accuracy of the high-power Stress workload.

4.3 Prediction At New Request Composition

Validation in the previous subsection shows that nearly all measured energy usage is accounted for and attributed. However, it does not validate whether the request power/energy attribution is properly done. We address this issue by validating power prediction at new request compositions. Specifically, we can learn the energy profiles of different types of requests from a running system. By assembling such per-request energy profiles, we can predict the system power in new, hypothetical workload conditions (different composition/ratios of request types, as well as different request rates). Here we assume that the energy usage for each type of requests does not change from the profiled system to new workload conditions. A successful validation of this prediction would indicate that the profiled per-request energy usage was accurate. Beyond the purpose of validation, the power prediction at new request compositions may also be utilized to assess possible request distribution strategies and enhance online load management.

For comparison, we consider two alternative approaches to predict system active power at hypothetical request compositions/rates. The *request-rate-proportional* approach simply assumes that all requests have a uniform effect on the total system energy usage so the active power consumption is exactly proportional to the request rate. The other approach, *CPU-utilization-proportional*, assumes that the active power consumption is proportional to the CPU utilization level. It requires

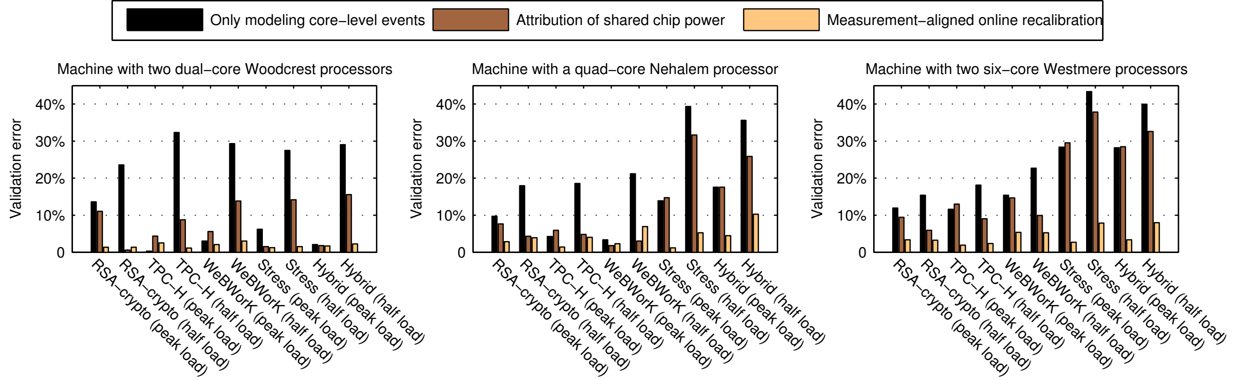

Figure 9: The accuracy of different approaches to estimate system active power from aggregate profiled request energy usage. The error is defined as $\frac{|\text{aggregate profiled request power} - \text{measured system active power}|}{\text{measured system active power}}$.

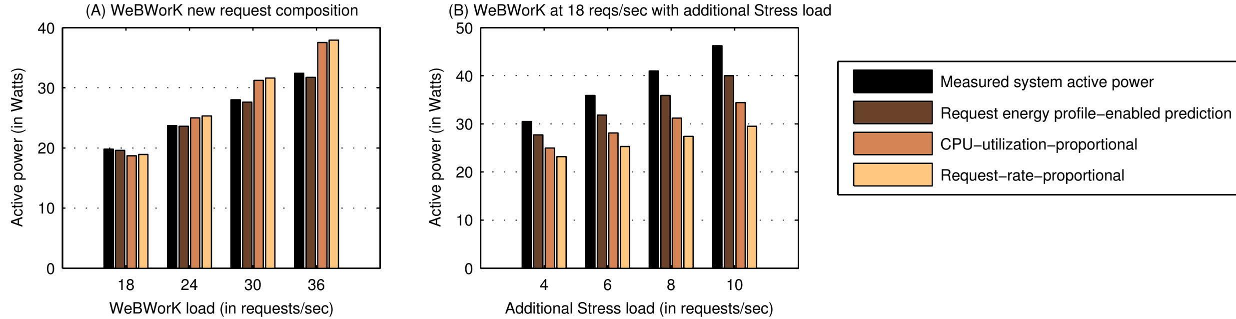

Figure 10: Accuracy of our request energy profile-enabled approach (and alternative approaches) to predict power at new workload conditions. Experiments were performed on the Nehalem machine.

profiling the CPU usage of individual requests through careful request context tracking [4, 5].

We perform evaluations on two scenarios of new workload conditions. First, we change the request type composition within a single application. Specifically, the original WeBWorK problem-solving workload includes thousands of science problem sets used in the real site. We consider a new WeBWorK workload with only the 10 most popular problem sets. In the second scenario, we add additional high-power Stress requests into an existing WeBWorK workload. Figure 10 shows the predicted power and measured power for the two workload scenarios on the Nehalem machine. In each workload scenario we experiment with several request rates that lie between half and full load of the machine.

Results show that our request energy profile-enabled prediction achieves higher accuracy than the two alternatives (particularly at high load). For the new WeBWorK request composition shown in Figure 10(A), at 36 requests/sec load, our approach has a 2% prediction error, compared to 16% or so errors for the two alternatives. For WeBWorK with additional Stress load shown in Figure 10(B), at the highest load, our approach has a 13% error, compared to 26% error for CPU-utilization-proportional and 36% error for request-rate-proportional.

It is worth discussing the higher prediction errors on the second workload scenario (WeBWorK with additional Stress load). An important assumption in our request energy profile-based prediction is that the energy usage for each type of requests does not change from the profiled system to new workload conditions. However, the request energy usage can be affected by the runtime condition. In particular, resource contention between the simultaneously running requests on the multicore server may lead to additional power-consuming activities. The Stress requests generate substantial load on the multicore-shared cache and memory interconnects, which leads to additional contention-induced power consumption. Such prediction errors are not due to the inaccuracy of our request energy profiles. Rather, it calls for an accurate prediction of dynamic multicore resource contention, which is the target of intense ongoing research [12, 17, 52]. Note that while the resource contention model is relevant to power prediction

at new, hypothetical workloads, it is not needed for online power/energy accounting and management in our request containers.

5 Container-Enabled Management

By identifying and isolating the power/energy contribution of individual requests in the multicore server, we enable first-class management of server power and energy resources for efficiency and fairness. This paper presents two case studies—fair request power conditioning using container-specific CPU execution throttling, and container profiling-enabled energy-efficient request distribution in a heterogeneous server cluster.

5.1 Fair Request Power Conditioning

The infrastructure cost to provision for the system peak power usage is substantial in today's data centers [20, 27]. While research has looked into cluster-wide load management to control system power [20,26], a complementary effort would be to condition each server's power consumption at a target level.

The system power consumption can be controlled by throttling CPU execution. Specifically, our case study employs the mechanism of CPU duty-cycle modulation. On Intel processors, the operating system can specify a portion (a multiplier of 1/8 or 1/16) of regular CPU cycles as duty-cycles. During each non-duty-cycle period (on the order of microseconds [1]), the processor is effectively halted and no memory operations are issued. This would lead to fewer activities (including memory transactions) and consequently lower power consumption, at the cost of slower application execution. The duty-cycle modulation can be independently adjusted on a per-core basis. It also exhibits a simple relationship between the duty-cycle level and active power consumption, which eases the control policy decision. Figure 11 shows the power effects of duty-cycle modulation for two (one CPU-bound and another memory-bound) workloads. An approximate linear power relationship exists for the memory-bound workload because memory transactions are not issued during non-duty-cycle periods.

CPU duty-cycle modulation can control surging power consumption. However, indiscriminate full-machine throttling [32] would lead to slowdowns of all running requests regardless of their power use. In particular, the occurrence of a power virus could force speed reductions on all concurrently running normal requests. Our power container provides two mechanisms to enable power conditioning in a fair fashion—1) request power accounting allows us to detect sources of power spikes; 2) the use of a request container can precisely throttle execution of power-hungry requests. In practice, we maintain a power consumption target for each request. Those that exceed the specified target will be subject to request-specific CPU duty-cycle modulation while other requests will run at full speed.

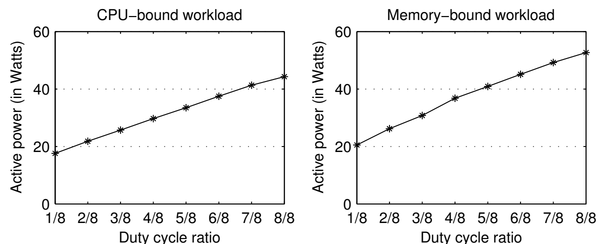

Figure 11: Power effects of duty-cycle modulation for two (one CPU-bound and another memory-bound) workloads on the Nehalem machine.

In our implementation, we check for possible change of CPU duty-cycle level due to request power consumption variations after each periodic counter sampling (typically once per millisecond). To enforce request-specific speed control, a request switch on a CPU core requires adjusting the core duty-cycle level according to the new request's power consumption. On our machines, configuring the CPU duty-cycle level requires reading and then writing a control register. The read/write operations take about 265 and 350 cycles respectively, or less than 0.3 microsecond on our 3.0 GHz and 2.26 GHz machines. The overhead is negligible (less than 0.03%) when the change in duty-cycle level is controlled to occur at a rate of no more than once every millisecond.

We evaluate the effectiveness of fair power conditioning. We experiment with WeBWorK on the Nehalem machine. The WeBWorK workload fully utilizes all four cores on the machine. In the middle of the experimentation, we inject high-power Stress requests to mimic power viruses. The power viruses arrive in a sporadic fashion at an average rate of one per second. Each power virus occupies a CPU core for about 100 milliseconds. Figure 12(A) shows that the introduction of power viruses lead to substantial power spikes. We apply our container-based fair power conditioning with a system active power target of 40 Watts. Since four requests may be running simultaneously on the quad-core system, the per-request active power target is 10 Watts when the system is fully busy. Figure 12(B) shows that our request container-enabled power conditioning can effectively keep power consumption at or below the target level despite the power viruses.

While the above results demonstrate the effectiveness of power conditioning, we next show that the CPU speed adjustment has been applied fairly to each request. Figure 13 plots the applied CPU duty-cycle ratio and orig-

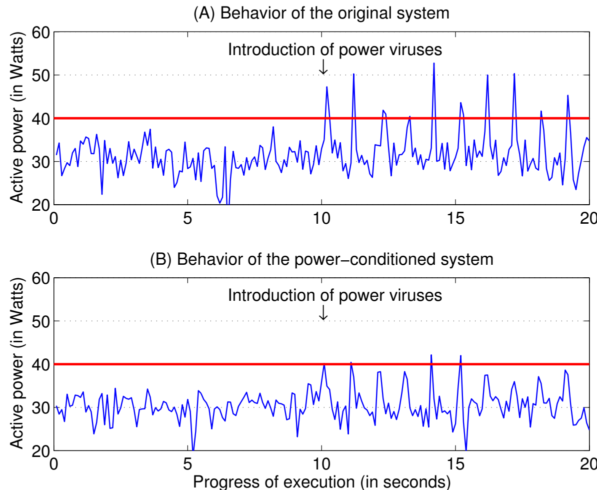

Figure 12: Measured active power for original and power-conditioned executions of WeBWorK with power viruses. Experiments were performed on the Nehalem machine with Agilent power meter.

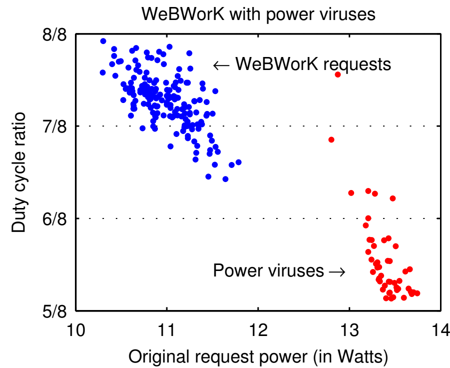

Figure 13: Original power and duty-cycle throttling for WeBWorK requests and power viruses. Each point represents a sample request. X-coordinate indicates the original (before throttling) request power consumption. Y-coordinate indicates the CPU duty-cycle ratio applied to the request.

inal request power (before throttling) for each request. Since a request power consumption may fluctuate over its execution, different duty-cycle levels may be applied over time. We show the time-averaged duty-cycle ratio for each request. We also estimate its power consumption in the original system (assuming a linear relationship between active power and CPU duty-cycle level as shown in Figure 11). Results show that low-power WeBWorK requests suffer small CPU speed slowdown (averaged at about 9%). At the same time, the power viruses are subject to more substantial (31% on average) slowdown.

Without our container-enabled fair power conditioning, the peak power can be reduced through full-machine throttling. A full-machine duty-cycle level of 6/8 would be required for such throttling, leading to about 25% slowdown of all requests (low-power WeBWorK requests as well as power viruses).

5.2 Heterogeneity-Aware Request Distribution

Past work has tackled the problem of energy management in a server cluster, primarily through server consolidation [13, 20, 26, 40] to shut down unneeded machines at load troughs. A production server cluster may contain different models of machines because it is not economical to upgrade all servers at once in a data center. Another possible reason is that each of the machine models has unique characteristics desired in certain workload scenario. In a heterogenous server cluster, the load placement and distribution on available machines (probably after consolidation) may affect the system energy efficiency. Previous research [15, 28, 39] has recognized the importance of energy efficiency optimization in a heterogeneous system. However, heterogeneity-ware request distribution across multicore servers is challenging in the identification of each request's cross-machine energy usage tradeoff during concurrent multicore executions. Our energy containers directly address this challenge by capturing fine-grained request energy usage profiles, which can later enable the preferential placement of each request on a machine where its relative energy efficiency is high.

We assess the energy efficiency heterogeneity across our machines. While recent processors (*e.g.*, Westmere) are generally more energy efficient than older models (*e.g.*, Woodcrest), some applications or application requests may see more substantial cross-machine energy efficiency difference than others do. Our energy container-enabled profiling allows us to quantify such workload-specific relative energy efficiency. Figure 14 shows the cross-machine (Westmere over Woodcrest) energy usage ratio for different workloads. Over different applications, the cross-machine energy usage ratio can be as high as 0.40 (for TPC-H) and as low as 0.11 (for WeBWorK). Over different requests within one application (TPC-H), the cross-machine energy usage ratio ranges from 0.27 (q1 and q22) to 0.61 (q9). When distributing some load from Westmere to Woodcrest becomes necessary, placing q9 on Woodcrest would be 2.3 times more energy-efficient than placing q1 or q22.

We perform a small-scale case study of request distribution over heterogeneous machines. We consider a two-machine cluster containing the newer (generally more energy efficient) Westmere and older Woodcrest

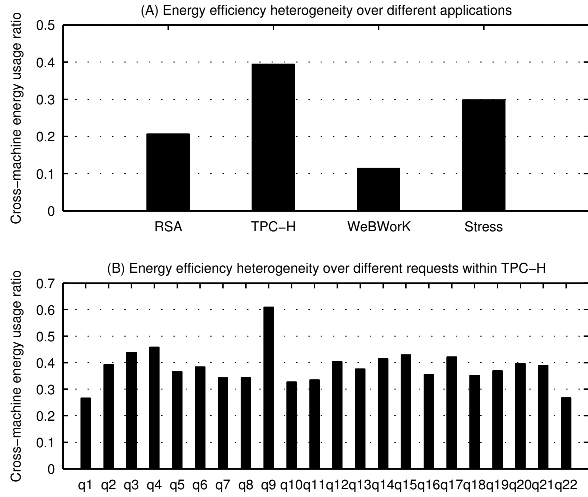

Figure 14: Cross-machine energy usage ratio (energy usage on Westmere over that on Woodcrest) for different applications (A) and for different requests within an application, specifically TPC-H queries (B).

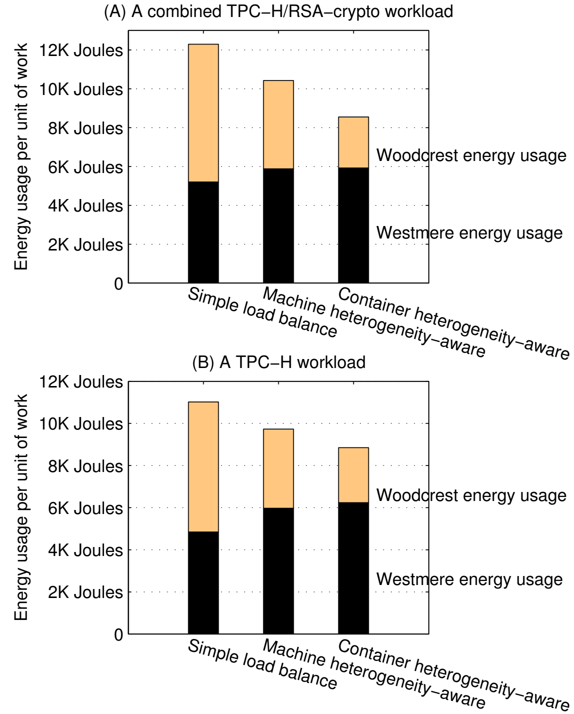

Figure 15: Measured energy usage per unit of work under three request distribution approaches in a heterogeneous server cluster. Usage for the two machines are marked in different colors. (A) shows results of a multi-application workload while (B) shows results of a single-application workload.

machines. Our request distribution goal is to achieve low overall system energy usage without overloading the more energy-efficient machine. We compare three load distribution approaches:

- *Simple load balance*— This approach balances the server load by trying to maintain similar CPU utilization ratios on the two machines. It is oblivious to the energy efficiency heterogeneity in the cluster.

- *Machine heterogeneity-aware*— This approach recognizes the machine heterogeneity so it loads up the more energy-efficient Westmere to a healthy high utilization (about 90% CPU utilization to prevent overloading) before loading Woodcrest. However, it is oblivious to request-level energy profiles so it distributes the exact input request composition to both machines.

- *Container heterogeneity-aware*— This approach also loads up the more energy-efficient Westmere to its full capacity before loading Woodcrest. Beyond that, it recognizes the per-request energy usage profiles using our energy containers and it preferentially places requests with higher relative energy efficiency (lower energy usage ratio in Figure 14) on Westmere.

We evaluate the effectiveness of heterogeneity-aware request distribution on high-utilization workloads (high utilization is typical for systems under energy-conserving server consolidation). Specifically, our cluster receives workloads at about 75% of full system capacity in our experiments. Our first experiment utilizes a combined TPC-H and RSA-crypto workload (with approximately half-half load composition). We define one unit of work as the body of requests arriving in 100 seconds, which includes 1,100 TCP-H requests and 25,000 RSA requests in this experiment. Figure 15(A) shows the energy usage per unit of work under the three load distribution approaches. Our container heterogeneity-aware approach saves 31% in combined two-machine energy usage compared to the simple load balance. The saving is 18% compared to the machine heterogeneity-aware approach that cannot recognize diverse request-to-machine affinity. Our container-enabled request distribution achieves these energy savings by preferentially loading each machine with requests of high relative energy efficiency. Specifically, TPC-H q3, q4, q9, q14, q15, q17 queries are dispatched to the Woodcrest machine while other TPC-H queries and RSA-crypto requests are dispatched to the Westmere machine.

Figure 15(B) illustrates the results of our second experiment with a workload that contains only TPC-H requests. Here one unit of work (arriving load in 100 seconds) includes 2,500 TPC-H queries. Among

the three request distribution approaches, our container heterogeneity-aware approach saves 20% and 9% in combined two-machine energy usage compared to the two alternatives respectively. The savings are less than those in the first experiment due to less request behavior variation in a single-application workload. Nevertheless, our evaluation demonstrates strong benefits of container-enabled heterogeneity-aware request distribution for both multi-application and single-application workloads.

The performance under all three approaches are comparable in both experiments. This is because our heterogeneity-aware approaches keep the machines under a healthy utilization threshold (about 90%) to prevent overloading.

6 Conclusion

This paper presents an operating system facility to account for and control the power/energy usage of individual requests in multicore servers. It utilizes an online per-core power estimation model that includes cross-core environmental effects, measurement-aligned online recalibration, and an operating system mechanism to isolate request-level power for accounting and control. Our system incurs low overhead (on the order of 0.1% for a typical setup). Validation shows that the acquired request power/energy usage profiles can be aggregated to match measured system power (with only 4% error) and predict system power at new, hypothetical workload request compositions (with no more than 13% error).

Power/energy containers enable the operating system to better manage online applications with dynamic power profiles and new computing platforms with hardware resource sharing and heterogeneity. We demonstrate that the request containers can help condition the overall system power in a fair fashion—throttling power viruses (using processor duty-cycle modulation) while allowing normal requests to run at full speed. Further, the acquired request energy profiles can enable energy-efficient request distribution on heterogeneous server clusters (by saving up to 18% energy usage compared to an alternative approach that recognizes machine heterogeneity but not per-request affinity).

While this paper focuses on per-request power accounting and control, the concept of power/energy containers can also be applied to other resource principals in a multicore system, such as virtual machines in a cloud hosting platform. Beyond the two management case studies presented in this paper, the abilities of request power accounting and control can add a new dimension to many classic server management schemes, including classifying resource usage patterns, detecting anomalies, and exploiting tradeoffs between quality-of-service and power. They call for additional research in future work.

References

[1] Intel Core2 Duo and Dual-Core thermal and mechanical design guidelines. http://www.intel.com/design/core2duo/documentation.htm.

[2] Stressful application test. http://code.google.com/p/stressapptest.

[3] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. In *SOSP*, 2009.

[4] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *OSDI*, 1999.

[5] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modeling. In *OSDI*, 2004.

[6] L. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[7] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *SIGOPS European Workshop*, 2000.

[8] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *ICS*, 2010.

[9] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*, 2000.

[10] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *ASPLOS*, 2009.

[11] A. Chanda, A. Cox, and W. Zwaenepoel. Whodunit: Transactional profiling for multi-tier applications. In *EuroSys*, 2007.

[12] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *HPCA*, 2005.

[13] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP*, 2001.

[14] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *SIGMETRICS*, 2005.

[15] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. In *Workshop on Power Aware Computing and Systems*, 2009.

[16] G. Contreras and M. Martonosi. Power prediction for Intel XScaler processors using performance monitoring unit events. In *ISLPED*, 2005.

[17] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. Patt. Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems. In *ASPLOS*, 2010.

[18] 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*, 2006.

[19] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *USITS*, 2003.

[20] X. Fan, W.-D. Weber, and L. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, 2007.

[21] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for Linux. In *OSDI*, 2002.

[22] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP*, 2001.

[23] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *OSDI*, 2008.

[24] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-Trace: A pervasive network tracing framework. In *NSDI*, 2007.

[25] K. Ganesan, J. Jo, W. L. Bircher, D. Kaseridis, Z. Yu, and L. K. John. System-level max power (SYMPO): a systematic approach for escalating system-level power consumption using synthetic benchmarks. In *PACT*, 2010.

[26] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *EuroSys*, 2009.

[27] J. Hamilton. Where does the power go in high-scale data centers? Keynote speech at SIGMETRICS, 2009.

[28] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP*, 2005.

[29] U. Hölzle. Powering a Google search. http://googleblog.blogspot.com/2009/01/powering-google-search.html, 2009.

[30] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Int'l Symp. on Microarchitecture*, 2003.

[31] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. Bhattacharya. Virtual machine power metering and provisioning. In *ACM Symp. on Cloud Computing*, 2010.

[32] C. Lefurgy, X. Wang, and M. Ware. Power capping: A prelude to power shifting. *Cluster Computing*, 11(2):183–195, June 2008.

[33] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Int'l Symp. on Microarchitecture*, 2009.

[34] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar. Performance directed energy management for main memory and disks. In *ASPLOS*, 2004.

[35] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *ISCA*, 2008.

[36] J. C. McCullough, Y. Agarwal, J. Chandrasheka, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conf.*, 2011.

[37] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. Kaiser. The low power energy aware processing (LEAP) embedded networked sensor system. In *Int'l Conf. on Information Processing in Sensor Networks*, 2006.

[38] D. Meisner, B. Gold, and T. Wenisch. PowerNap: Eliminating server idle power. In *ASPLOS*, 2009.

[39] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *4th Int'l Conf. on Autonomic Computing*, 2007.

[40] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*, 2003.

[41] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: Coordinated multi-level power management for the data center. In *ASPLOS*, 2008.

[42] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the Cinder operating system. In *EuroSys*, 2011.

[43] K. Shen. Request behavior variations. In *ASPLOS*, 2010.

[44] K. Shen, M. Zhong, S. Dwarkadas, C. Li, C. Stewart, and X. Zhang. Hardware counter driven on-the-fly request signatures. In *ASPLOS*, 2008.

[45] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Apr. 2010.

[46] C. Stewart, M. Leventi, and K. Shen. Empirical examination of a collaborative web application. In *IEEE Int'l Symp. on Workload Characterization*, 2008.

[47] The Mathematical Association of America. WeBWorK: Online homework for math and science. http://webwork.maa.org/.

[48] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter. Architecting for power management: The IBM POWER7 approach. In *HPCA*, 2010.

[49] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, 1994.

[50] A. Wissner-Gross. How you can help reduce the footprint of the web. *Times Online, UK*, 2009. http://www.timesonline.co.uk/tol/news/environment/article5488934.ece.

[51] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS*, 2002.

[52] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Managing contention for shared resources on multicore processors. In *ASPLOS*, 2010.