Power Containers: An OS Facility for Fine-Grained Power and Energy Management on Multicore Servers

Kai Shen* Arrvindh Shriraman† Sandhya Dwarkadas* Xiao Zhang§ Zhuan Chen*

* University of Rochester † Simon Fraser University § Google

{kshen, sandhya, zchen}@cs.rochester.edu ashriram@cs.sfu.ca xiaozhang@google.com

Abstract

Energy efficiency and power capping are critical concerns in server and cloud computing systems. They face growing challenges due to dynamic power variations from new client-directed web applications, as well as complex behaviors due to multicore resource sharing and hardware heterogeneity. This paper presents a new operating system facility called "power containers" that accounts for and controls the power and energy usage of individual fine-grained requests in multicore servers. This facility relies on three key techniques—1) online model that attributes multicore power (including shared maintenance power) to concurrently running tasks, 2) alignment of actual power measurements and model estimates to enable online model recalibration, and 3) on-the-fly application-transparent request tracking in multi-stage servers to isolate the power and energy contributions and customize per-request control. Our mechanisms enable new multicore server management capabilities including fair power capping that only penalizes power-hungry requests, and energy-aware request distribution between heterogeneous servers. Our evaluation uses three multicore processors (Intel Woodcrest, Westmere, and SandyBridge) and a variety of server and cloud computing (Google App Engine) workloads. Our results demonstrate the high accuracy of our request power accounting (no more than 11% errors) and the effectiveness of container-enabled power virus isolation and throttling. Our request distribution case study shows up to 25% energy saving compared to an alternative approach that recognizes machine heterogeneity but not fine-grained workload affinity.

Categories and Subject Descriptors D.4.1 [*Operating Systems*]: Process Management—Multiprocessing/multiprogramming/multitasking; D.4.8 [*Operating Systems*]: Performance—Measurements, Modeling and prediction

General Terms Design, Experimentation, Measurement, Performance, Reliability

Keywords Multicore, power modeling, power virus, hardware counters, server and cloud computing, operating system

ASPLOS'13, March 16–20, 2013, Houston, Texas, USA.

1. Introduction

Designers of data centers and server systems place high priority on improving energy efficiency, controlling peak power draw, and monitoring resource usage anomalies. Online applications are continuously evolving with new features and some rely on end users to supply content or even content-generating code. For instance, cloud computing platforms run user applications at third-party hosting centers. Online collaboration software (such as the WeBWorK homework management system [37]) allows users (teachers in WeBWorK) to supply scripting programs to run at servers. The workload diversity and dynamic client-directed processing result in large power fluctuations on modern multicore processors with substantial hardware resource sharing and increasing power proportionality [8]. Our measurements found that the idle power is only about 5% of the total CPU package power on a recent Intel SandyBridge processor (the idle power proportion is 32% when counting the full machine power consumption).

Modern processors also exhibit highly varying power consumption during execution, depending on the scope and level of activity in the CPU cores as well as in uncore components such as the memory interconnect. At the same (full) CPU utilization on our SandyBridge processor, a cache/memory-intensive application consumes 49% more power than a CPU spinning program does. On these machines, extreme power-consuming tasks (or "power viruses") [19] may appear accidentally or be devised maliciously. 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 use, under grid power as well as renewable energy sources [35]. Additionally, the economics of incremental upgrades, along with low-power specialization, lead to widespread heterogeneity in server clusters. Exploiting the diverse workload affinity to heterogeneous platforms is beneficial for realizing high energy efficiency.

This paper presents a new operating system facility, called *power containers*, to account for and control the power and energy usage of individual requests in multicore servers. Previous research (particularly, Resource Containers [6], Magpie [7], and our hardware counter modeling [32, 33]) has recognized the need for profiling and isolating per-request resource usage in a server. However, they do not include power or energy as a resource. Fine-grained power and energy management on multicore servers is challenging due to their complex relationship with resource utilization. Concurrent task executions, varying power consumption among resources, and dynamic hardware component sharing in a multicore processor lead to complex per-task power behaviors. Request executions in a concurrent, multi-stage server contain fine-grained activities with frequent context switches, and direct power measurements on such spatial and temporal granularities are not available on today's systems. Hence, request-level power and energy management requires agile, low-cost control to ensure isolation and achieve efficiency.

The characterization of request power and 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 our power containers facility:

- We attribute the multicore power consumption to individual tasks running concurrently on a multicore system. We extend previous event-driven power models [9, 10, 25] to capture 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 [26] 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. Such resource accounting and control is also desirable in cloud computing, especially for those (like the Google App Engine) that do not adopt heavy virtual machine-based isolation.

Our power containers enable the first-class management of multicore server power and energy resources in new ways. In particular, we can pinpoint the sources of power spikes and anomalies, and 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. Furthermore, we improve energy efficiency in a heterogeneous environment through container heterogeneity-aware load distribution. Specifically, request energy profiles on different machines are used to understand each request's cross-machine relative energy affinity and direct its execution accordingly. The rest of this paper will present the design and implementation of our power containers and evaluate its accuracy and effectiveness in several management case studies.

2. Related Work

Power Measurement and Modeling Per-hardware-component power can be measured through elaborate embedded instruments (as in LEAP [27]). However, it is difficult to directly measure per-core power on a multicore chip due to shared use of components such as cache and memory interconnect. Bellosa [9] estimated the processor and memory power using a linear model on hardware event counts. Additional research has supported event-based power phase modeling [10, 25]. On the negative side, McCullough *et al.* [26] identified high power model errors due to multicore complexities and hidden device states. Event-based power modeling has also been included in processor designs like the IBM POWER7 [38] and Intel SandyBridge [30]. Most recently, Huang *et al.* [24] proposed a firmware-level power proxy that estimates per-core power by utilizing specialized activity counters in the IBM POWER7+ chip hardware. Direct measurement or hardware event-based modeling alone is limited by their inability to identify and isolate software-level concurrent resource principals in server and cloud computing environments.

System-Level Energy Accounting By coordinating the external power measurement with interrupt-triggered program sampling, Flinn and Satyanarayanan [16] were able to profile the energy usage of application processes and procedures. Chang *et al.* [12] further enhanced the statistical sampling accuracy using energy-driven sampling intervals. ECOSystem [39] is a unified framework of whole-system energy accounting to support energy management policies. In ECOSystem, the CPU power consumption is assumed to be constant during busy periods. Quanto [18] 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. The Cinder operating system [31] 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.

Data Center Power Management In light of the load burstiness at large-scale service sites, Chase *et al.* [13] and Pinheiro *et al.* [29] 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.* [15] and Govindan *et al.* [20] suggested that the power provisioning should take into account the independence as well as correlation of power fluctuations at individual servers. PowerNap [28] enables fast transitioning between active state and minimum-power nap state, which results in very low server idle power. Research [14, 23] has also recognized the energy effects of load placement in a heterogeneous server cluster. Our power containers are complementary to these techniques. In particular, we enable more fine-grained (request-level) multicore server power tracking and control, a need not addressed by previous techniques.

3. Design and Implementation

We propose a new operating system facility that accounts for and controls the power and 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 (Section 3.1). In addition, we align online power measurements and modeling estimates to recalibrate the power model for better accuracy (Section 3.2). To tackle the second challenge, we build operating system mechanisms (Section 3.3) to track multi-stage request executions on-the-fly, account for request power and energy usage, and apply request-grained power control. We also present container-enabled new multicore server power and energy analysis and management cases (Section 3.4).

3.1 Power Attribution to Concurrent Tasks

While previous work has extensively explored system power modeling [9, 10, 25, 26, 38] and devised on-chip power estimation registers [30], their focus is largely on coarse-grained whole system or full processor power. Since multiple tasks may run concurrently in a multicore system and each task may belong to a distinct user request, there is a clear need for 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 I/O resource accounting will be briefly discussed in Section 3.3.

Our first approach follows Bellosa [9]'s model in 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 cycle ($\mathcal{M}_{\text{float}}$), last-level cache requests per cycle ($\mathcal{M}_{\text{cache}}$), and memory transactions per cycle ($\mathcal{M}_{\text{mem}}$). The constant power term ($\mathcal{C}_{\text{idle}}$) 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:

$$\begin{aligned}\mathcal{P}_{\text{active}} = \mathcal{C}_{\text{core}} \cdot \mathcal{M}_{\text{core}} + \mathcal{C}_{\text{ins}} \cdot \mathcal{M}_{\text{ins}} + \mathcal{C}_{\text{float}} \cdot \mathcal{M}_{\text{float}} \\ + \mathcal{C}_{\text{cache}} \cdot \mathcal{M}_{\text{cache}} + \mathcal{C}_{\text{mem}} \cdot \mathcal{M}_{\text{mem}}\end{aligned} \tag{1}$$

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 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 clocking circuitry, voltage regulators, and other uncore units [21]) consumes some active power as long as one core is running, which does not change proportionally with core-level event rates. Figure 1 illustrates this symptom using a simple CPU spinning microbenchmark. This workload scales perfectly on a multicore so all 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 on the quad-core SandyBridge machine show that the power increment from idle to one utilized core is substantially larger than further power increments, which suggests an active power component that does not scale with core-level physical events. The same experiment on a dual-socket machine (with two dual-core Woodcrest processors) shows that the power increments from idle to two utilized cores are higher than further increments. This indicates that both sockets become active when two cores are utilized, which matches the Linux operating system (used in our experiments)'s multicore scheduling policy that maximizes performance.

We 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 [8]. 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 $\mathcal{M}_{\text{chipshare}}$ ($0.0 \le \mathcal{M}_{\text{chipshare}} \le 1.0$) to denote the proportion of a given task's share of on-chip maintenance power. If a core is busy while all siblings are idle, the full chip power should be attributed to

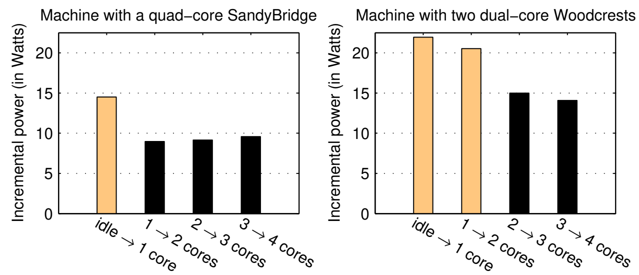

Figure 1. Incremental (per-core) power consumption increases on two machines (one with a quad-core SandyBridge processor and the other with two dual-core Woodcrest processors).

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:

$$\begin{aligned}\mathcal{P}_{\text{active}} &= \mathcal{C}_{\text{core}} \cdot \mathcal{M}_{\text{core}} + \mathcal{C}_{\text{ins}} \cdot \mathcal{M}_{\text{ins}} + \mathcal{C}_{\text{float}} \cdot \mathcal{M}_{\text{float}} \\ &+ \mathcal{C}_{\text{cache}} \cdot \mathcal{M}_{\text{cache}} + \mathcal{C}_{\text{mem}} \cdot \mathcal{M}_{\text{mem}} \\ &+ \mathcal{C}_{\text{chipshare}} \cdot \mathcal{M}_{\text{chipshare}}\end{aligned} \tag{2}$$

Unlike the core-level event metrics that can be simply acquired through hardware counters on the CPU core, the chip power share $\mathcal{M}_{\text{chipshare}}$ does not correspond to any processor hardware counter. Further, precise sharing information in a dynamic system depends on time-synchronized global activities across multiple cores. In our implementation, each core independently makes an approximate estimation in order to avoid 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. We approximate the number of busy sibling cores using the sum of the 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. We do so without any cross-core synchronization for efficiency. Note that each 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.

Our discussion so far applies to the power modeling of a single multicore chip. On a multi-socket machine with more than one multicore chips, each chip's power will be modeled using our approach described above and the full power will include the power from all multicore chips on the machine.

3.2 Measurement Alignment for Recalibration

Despite the wide uses of event-based power models [9, 10, 25, 38], we found that large errors may arise in practice. Recent research [26] 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

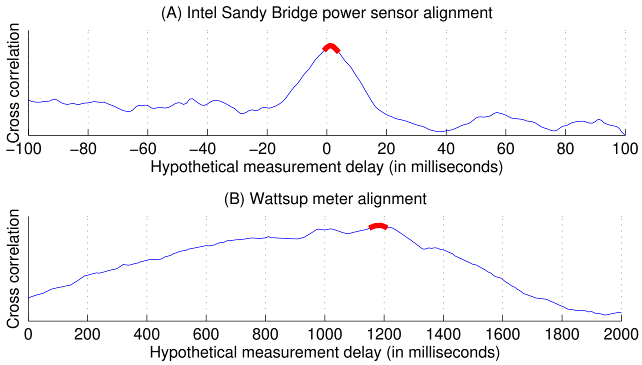

Figure 2. Measurement/model alignment cross-correlation for the Intel SandyBridge on-chip power meter (A) and a Wattsup power meter (B). High correlation peaks are specially marked.

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 can be measured through off-the-shelf meters, measurement results may 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 may not accurately predict the power consumption, it may still identify power transitions and phases quite well. In other words, aligned power measurements should follow the fluctuation patterns of real-time power model estimates. We therefore 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 $\mathcal{P}_{\text{measure}}(1)$, $\mathcal{P}_{\text{measure}}(2)$, $\cdots$ be the sequence of recent power measurement samples ($\mathcal{P}_{\text{measure}}(1)$ is the most recent). Let $\mathcal{P}_{\text{model}}(1)$, $\mathcal{P}_{\text{model}}(2)$, $\cdots$ be the sequence of recent modeling samples at the same sampling frequency. Then the cross-correlation at a hypothetical measurement delay t is:

$$\text{Cross-corr.}(t) = \sum_{i=1}^{\text{number of matching samples}} \mathcal{P}_{\text{measure}}(i) \cdot \mathcal{P}_{\text{model}}(i+t). \quad (4)$$

Figure 2 shows a case example of alignment cross-correlation over hypothetical measurement delays. The highest point of the correlation curve (specially marked in the figure) indicates the likely measurement delay. Our results show about 1 msec delay for the Intel SandyBridge on-chip power meter and about 1.2 second (1200 msecs) delay for a Wattsup power meter. Longer delay for the Wattsup meter is due to its coarse-grained measurements and data propagation latency through an external USB connection. Figure 3 shows the aligned measurement/model power samples for the Intel SandyBridge on-chip power meter. Note that the measured power (whether using the SandyBridge's power meter or using off-chip instruments) is for the whole system, necessitating the per-core/per-request model we have designed for fine-grained online control.

Aligned power measurements can identify modeling errors for the currently running workload and help recalibrate our multicore

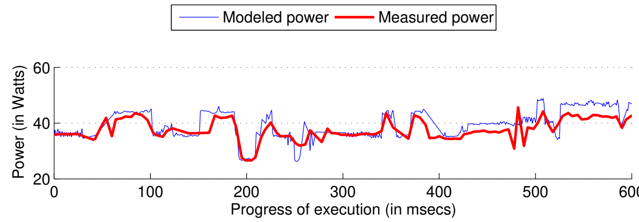

Figure 3. Aligned measurement/model power traces for the Intel SandyBridge on-chip power meter.

power model in Equation 2. The recalibration intuitively means adding new online samples of aligned system metrics and corresponding power measures to the original offline model parameter calibration. 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. The short measurement delay (about 1 msec) for the SandyBridge on-chip power meter makes it suitable for real-time model adjustment and power control.

3.3 Request Power Accounting

We construct operating system mechanisms to support request-level power 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 and 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 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, a request execution may flow through multiple processes in a multi-stage server. For instance, a PHP web processor propagates the request context into a database thread through a socket message. Such a socket connection can be *persistent* in a high-throughput server when each server process can repeatedly serve multiple requests and one socket connection is reused by multiple request propagations over time. The past approach of Resource Containers [6] requires applications to explicitly pass request context bindings across server stages. Google's Dapper tracing infrastructure [34] assumes that applications use a unified RPC framework that can be easily instrumented. Magpie [7] recognizes request context propagations in a server system but it does so out-of-band, which is sufficient for trace analysis but not for applying per-request power control online.

The X-Trace framework [17] is closest to our needs of system-level in-band request context tracking. However, X-Trace was designed for general network systems tracing and we specifically target high-throughput server systems. We highlight two particular differences. First, our request tracking is implemented entirely in the operating system requiring no application change. While the X-Trace propagation primitives provide flexible application control, our transparent mechanism can easily support a large variety of complex (even closed-source) server applications. We achieve application transparency by recognizing key request propagation channels in server systems. Fundamentally, the operating system may recognize request context propagation events as those that indicate causal dependences through data and control flows, specifically, socket communications, IPCs, and process forking. OS-only management, however, cannot track user-level request stage transfers in an event-driven server or a server that employs user-level threads. This is an important limitation, but our mechanism is applicable to a large number of real-world server applications that do not employ an event-driven model (for easier programming with threads and processes) or user-level threads (to avoid performance anomalies due to complex user/kernel thread mapping and lack of full visibility by the OS scheduler and I/O management). Past research [11] suggests that some user-level request stage transfers may be OS-observable by trapping accesses to critical synchronization data structures. We leave its implementation for possible future work. We include a variety of applications supported by our framework in our evaluation in Section 4.2.

We also make efforts to support request tracking over a persistent socket connection in high-throughput servers. We tag each socket message with the sender's request context identifier. We store the identifier in a new TCP option field for protocol compatibility (it will be safely ignored if the message reaches a different machine that does not support our request container mechanism). If the message arrives at the destination socket but the receiver has not made the `read()` call, the message has to be buffered. In a naive implementation, the destination socket simply inherits the request context tag of the buffered message and the receiver will acquire the socket's request context when it calls `read()`. This, however, is unsafe in a persistent connection when the context tag for a new request may arrive before the previously arrived message is read by a receiver (in which case the receiver incorrectly inherits the new request context). Therefore, multiple data segments in a socket buffer must be individually tagged with the corresponding request context. A receiver process will inherit the proper context according to the data it reads.

Our implementation builds on our past work of server system hardware counter modeling [32, 33], with new mechanisms for power and energy accounting. 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 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). Figure 4 illustrates a captured request execution and power and energy attribution in a realistic multi-stage server application containing Apache PHP processing, MySQL database, and various external file manipulation processes.

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, our full system power accounting also considers power-consuming peripheral devices for disk and network I/O. The OS can identify requests responsible 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.

3.4 Container-Enabled Management

By identifying and isolating the power and energy contribution of individual requests in the multicore server, we enable new first-class management of server power and energy resources for efficiency and fairness. We present two particular management cases here.

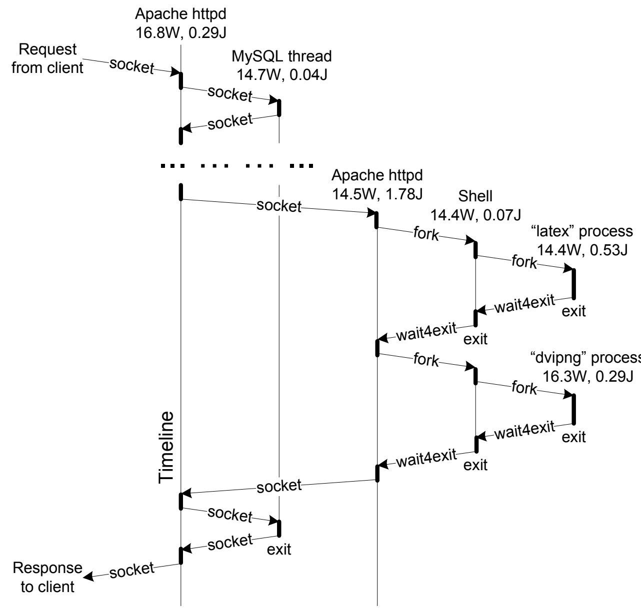

Figure 4. A captured request execution that involves Apache PHP processing, MySQL database, and various external operations on content and image rendering. We mark the attributed power (in Watts) and energy (in Joules) at each request stage. This request is from the WeBWorK online homework system [37]. Identified data and control flows between server components are marked with arrows. Darkened portions of a component timeline indicate active executions (while the rest represent blocking waits).

Fair Request Power Conditioning The infrastructure cost to provision for the system peak power usage is substantial in data centers [15, 22]. While research has looked into cluster-wide load management to control system power [15, 20], a complementary effort would be to condition each server's power consumption to a target level. The system power consumption can be controlled by throttling CPU execution. Specifically, on Intel processors, the OS 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 (approximately linear) relationship between the duty-cycle level and active power consumption, which eases the control policy decision.

CPU duty-cycle modulation can control surging power consumption. However, indiscriminate full-machine throttling 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 can detect sources of power spikes; 2) container-specific power control 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. In our implementation, we apply a particular CPU duty-cycle modulation level to a given request execution according to its power consumption. We also track the request power consumption variations after each periodic counter sampling (typically once per millisecond), and then change the CPU duty-cycle level if desirable. When the CPU core switches to run another request, the core duty-cycle level will be adjusted according to the policy setting and power consumption of the new request.

Heterogeneity-Aware Request Distribution Past work has tackled the problem of energy management in a server cluster, primarily through server consolidation [13, 15, 20, 29] 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 [14, 23] has recognized the importance of energy efficiency optimization in a heterogeneous system. However, heterogeneity-aware request distribution across multicore servers is further made difficult by the need to identify workload cross-machine energy usage tradeoffs during concurrent multicore executions. Our power 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.

A request container can be maintained over multiple machines in a server cluster using socket message-based request context propagation described in Section 3.3. 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 the server machines. By tagging response messages, server machines can pass cumulative power and energy usage information to the dispatcher machine for comprehensive resource accounting.

3.5 Overhead Assessment

We implemented a prototype power 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 SandyBridge processor. Results show that one container maintenance operation takes about 0.95 μsec. 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 generated power and energy profiles. We measure the maintenance-induced event metrics and find that an average container maintenance operation uses 2948 cycles, 1656 instructions, 16 floating point operations, 3 last-level cache references, and no measurable memory transactions. Assuming all four cores are busy (1/4 chip share), the average energy usage for a container maintenance is about 10 micro-Joules according to our active power model. To mitigate the observer effect in the collected statistics, we subtract the maintenance-induced additional event counts from the measured event counts of each sampling period.

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 μsecs per calibration. Its online overhead is negligible if it is performed at a rate of no more than once every 10 msecs.

For container power control, 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 0.2 μsec on a 3.0 GHz machine.

Note that the reported overhead above is incurred on a per-CPU-core (hardware counter sampling, container maintenance, and duty-cycle adjustment) or whole-system basis (model recalibration), rather than for each request in the system. In particular, the hardware counter sampling only occurs for the request that is running on the CPU core at the moment of sampling. Active requests that are not currently running only consume space (but not CPU runtime) costs. This is important for the scalability of our system.

We also report the memory space cost of our system. In our implementation, the state of an active power container is encapsulated in a 784-byte data structure that includes cumulative event counters and statistics, locks, flags, and a reference counter. The data structure is released when all tasks linked to the container exit (when the reference counter is zero) so it does not leak in a long-running server. The space cost is modest and it should not affect the server scalability even if thousands of active power containers exist in the server at the same time.

4. Experimental Evaluation

Our evaluation uses 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 two six-core (12 cores total) Intel Xeon L5640 ("L" for low-power) 2.26 GHz "*Westmere*" processors. The six cores on each processor chip share a 12 MB L3 cache. The third machine contains a quad-core Intel Xeon E31220 3.10 GHz "*SandyBridge*" processor. The four cores share an 8 MB L3 cache. The three processors were publicly released in 2006, 2010, and 2011 respectively. Both Westmere and SandyBridge processors utilize the recent 32 nm technology. We configure hardware event counting registers on each processor to assemble the input metrics for our power model in Equation 2.

We employ two power measurement instruments. First, each machine uses a Wattsup meter that reports the whole machine power once a second. The Wattsup measurements are fed back to the target machine through the USB interface. In addition, we use the on-chip power meter available on the SandyBridge machine, which measures power for the processor socket package including all cores, the uncore components (e.g., Last-Level-Caches), the memory controller, and quickpath interconnects. The SandyBridge meter tracks many (order of 100) microarchitecture events and applies active energy costs to each event [30]. Specifically, it reports accumulated energy once per millisecond and the power is then calculated as the rate of energy changes. Our evaluations sample the SandyBridge power once every 10 milliseconds. Note that the SandyBridge on-chip meter does not report per-core power and it cannot identify and isolate concurrent resource principals in a server environment.

The full power includes a constant idle power consumed when the server exhibits no activity. On the SandyBridge machine, the idle power (26.1 Watts) is about 32% of the full machine power at an observed high load scenario. The idle power ratio goes down to 5% if we only consider the CPU package power (including uncore components, the memory controller, and quickpath interconnects), indicating excellent energy proportionality [8] for the processor sub-system. We note that 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 most evaluation results on the active (full minus idle) power. When using the SandyBridge on-chip power meter, we report the full power reading since its idle component is very small.

The rest of this section will first present the power model calibration results. We will then evaluate the accuracy of our request power and energy accounting using several realistic applications. We will also present two power container-enabled server management case studies—fair request power conditioning and heterogeneity-aware request distribution.

4.1 Power Model Calibration

Our multicore server model in Section 3.1 requires 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 raw CPU spin, 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, high network I/O, and 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 SandyBridge machine. While a model coefficient $\mathcal{C}$ is not intuitively meaningful, $\mathcal{C}\cdot\mathcal{M}^{\max}$ (where $\mathcal{M}^{\max}$ is the maximum observed value for the metric for the whole machine including all cores) represents the maximum active power impact of the metric—

$$\begin{array}{lcr}
\mathcal{C}_{\text{idle}} & = & 26.1\,\text{Watts;}\\
\mathcal{C}_{\text{core}}\cdot\mathcal{M}^{\max}_{\text{core}} & = & 33.1\,\text{Watts;}\\
\mathcal{C}_{\text{ins}}\cdot\mathcal{M}^{\max}_{\text{ins}} & = & 12.4\,\text{Watts;}\\
\mathcal{C}_{\text{cache}}\cdot\mathcal{M}^{\max}_{\text{cache}} & = & 13.9\,\text{Watts;}\\
\mathcal{C}_{\text{mem}}\cdot\mathcal{M}^{\max}_{\text{mem}} & = & 8.2\,\text{Watts;}\\
\mathcal{C}_{\text{chipshare}}\cdot\mathcal{M}^{\max}_{\text{chipshare}} & = & 5.6\,\text{Watts;}\\
\mathcal{C}_{\text{disk}}\cdot\mathcal{M}^{\max}_{\text{disk}} & = & 1.7\,\text{Watts;}\\
\mathcal{C}_{\text{net}}\cdot\mathcal{M}^{\max}_{\text{net}} & = & 5.8\,\text{Watts.}
\end{array}$$

4.2 Request Model Evaluation

Our evaluation uses the following server and cloud computing workloads:

- *RSA-crypto* is a synthetic security processing workload. Each request 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.
- *Solr* [2] is a popular, open-source search platform from the Apache project. It uses the Lucene Java search library as the core for full-text indexing and searching. The search server runs within a Servlet container such as Tomcat. Our deployment uses Solr 3.6.1 and Tomcat 6.0.35 software. We construct a search workload from the Wikipedia data dumps [5]. Our workload has the raw data size of 5.6 GB and indexed data size of 1.7 GB that the Solr search server runs on. The indexed data fits into the memory of our search server. Client queries in our workload are generated by randomly selecting and sequencing article titles in the Wikipedia data dump.
- *WeBWorK* [37] 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 [36]. Our installation runs Apache 2.2.8 web server, a variety of Perl PHP modules, and MySQL 5.5.13 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 [3], is a 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 simultaneously. It generates higher-than-normal power consumption, particularly on our Westmere processor-based machine. We adapted it to a server-style workload with requests each running for about 100 msecs.
- *GAE-Vosao*. Google App Engine (GAE) provides a Platform-as-a-Service cloud computing infrastructure that enables users to build, maintain, and scale web applications. We install the GAE Java software development kit including a Java web server that simulates the GAE runtime environment with a local datastore. On our local GAE setup, we deploy the Vosao content management application [4], which supports collaborative building of dynamic web sites. Our tests use a workload that models the collaborative web content editing using the published revision history of the "Harry Potter" article at Wikipedia. We adopt a 9:1 read/write ratio to mimic the typical higher rate of reads in a mixed read/write scenario.
- *GAE-Hybrid* contains a mixture of GAE-Vosao requests and some artificial GAE-based power viruses. Our power virus design is not optimized as in previous work [19]. Instead, our goal is to show the impact of a very simple power virus that one can easily create. Specifically, our power virus (about 200 lines of Java code) repeatedly writes one of every four bytes over a 16-MByte block. Our measurements find that this workload pattern keeps the cache/memory and instruction pipelining units simultaneously busy. We also use statically allocated data blocks to minimize the impact of Java garbage collection. In our hybrid workload composition, approximately half the load is generated by Vosao requests and power viruses respectively.

Most of our workloads contain only open-source software. One exception is the GAE Java web server, which is released in the form of Java byte code by Google.

Figure 5 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%.

WeBWorK uses multi-stage server architectures including the database. The web server in WeBWorK also pools many request executions on each worker process. Both the Solr search engine and Google App Engine (GAE) workloads run on the Java virtual machine and use Servlets. Our power container facility properly tracks individual request activities and attributes request power consumption during concurrent multicore execution. Figure 6 shows the dis-

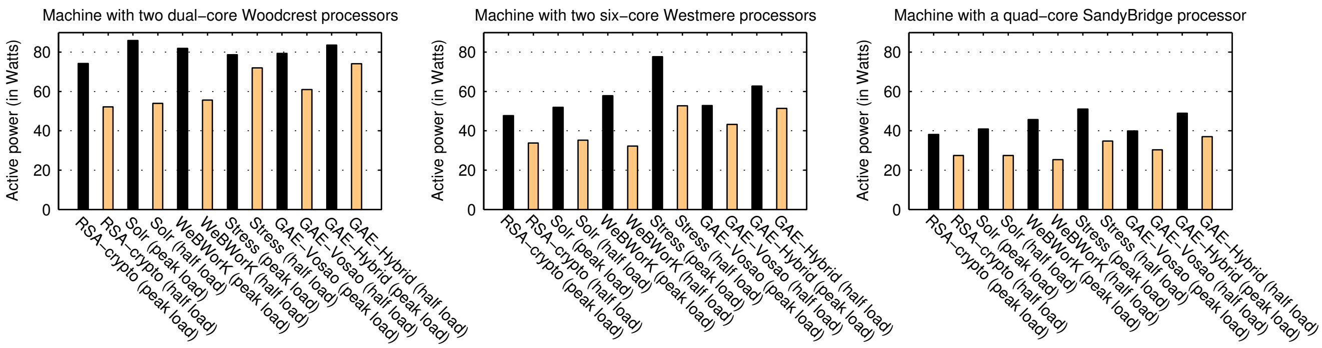

Figure 5. Measured active power of application workloads on three machines and two load levels.

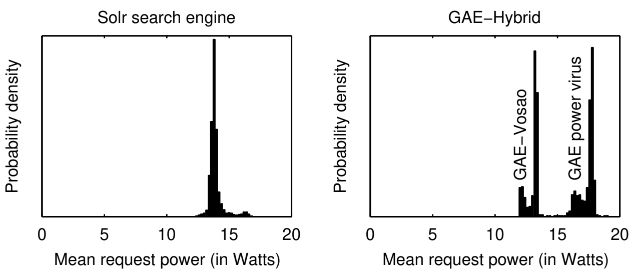

Figure 6. Mean request power distributions (in histograms) for the Solr search engine and GAE-Hybrid workloads on the SandyBridge machine. The mean power for a request is its average power consumption over the course of the request execution. For the GAE-Hybrid workload, we label the major distribution masses for Vosao and power virus requests respectively. Results were collected when each workload runs at half server load. We intentionally do not show the quantitative Y values (histogram heights) in the figures because these values have no inherent meaning and they simply depend on (inversely proportional to) the width of each bin in the distribution histogram.

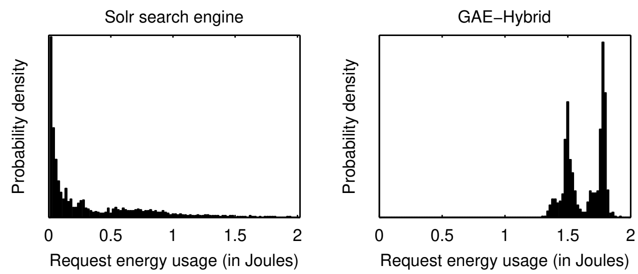

Figure 7. Request energy usage distributions (in histograms) for the Solr search engine and GAE-Hybrid on the SandyBridge machine.

tributions of mean request power for Solr and GAE-Hybrid (mixture of GAE-Vosao and power viruses) workloads on the SandyBridge machine. We observe varying request power consumption for the GAE workload. Specifically, power viruses consume substantially higher power than Vosao requests due to their intense CPU/memory activities. Figure 7 further shows the request energy usage distributions. The varying request energy usage is due both to request power variation (primarily for GAE-Hybrid) and to their execution time difference (primarily for Solr).

Validation By Summing Request Energy Usage A direct validation of our power container modeling is infeasible without a way to 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, we adopt an indirect validation approach. Our power 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 duration, will produce an estimate of the average system power consumption. We validate the accuracy of our request power and energy profiles by comparing this estimate against measured system power.

During this experiment, we identified some substantial background processing by the Google App Engine (GAE) that presents no traceable connections to application request executions. We suspect that these background processes are related to the GAE security management, although we are unable to verify without access to the GAE source code. We account for the resource usage of these background processes in a special power container and include it in our energy usage sum.

To understand the benefits of our proposed techniques, we compare the validation accuracy over three different approaches:

- Approach #1 (described in Section 3.1) employs a linear power model on core-level event metrics. It does not consider the shared chip maintenance power.
- Approach #2 (also presented in Section 3.1) additionally accounts for the multicore chip maintenance power and attributes it to concurrent requests.
- Approach #3 (presented in Section 3.2) further employs measurement-aligned online model recalibration to mitigate the power model errors.

Validation results in Figure 8 show that our techniques are effective in producing accurate request power and energy profiles. Across all workloads, the approach of only modeling core-level events exhibits 29%, 41%, and 20% worst-case validation errors for the three machines respectively. Attributing shared multicore power reduces the worst-case validation errors to 18%, 35%, and 13% for the three machines. The measurement-aligned online model recalibration further reduces the worst-case errors to 8%, 9%, and 6% on the three machines. The measurement-aligned recalibration is particularly effective in improving the request profiling accuracy of high-power workloads like Stress.

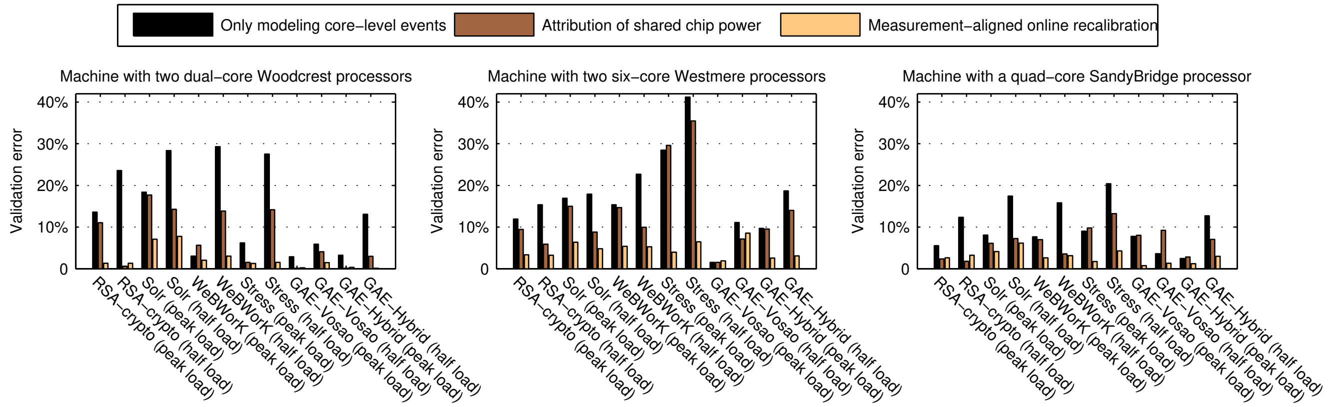

Figure 8. The accuracy of different approaches to estimate system active power from aggregate profiled request energy usage. The error is defined as $\frac{|\text{aggregate profiled request power} - \text{measured system active power}|}{\text{measured system active power}}$.

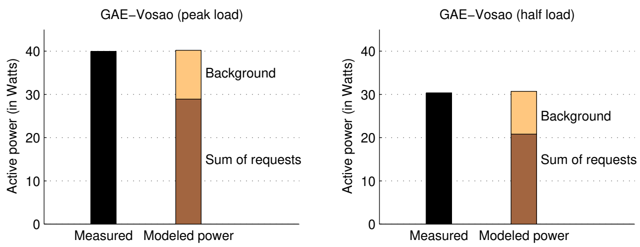

Figure 9. Resource usage of background processes in Google App Engine systems (GAE-Vosao peak load and half load) on the machine with a quad-core SandyBridge processor.

Our evaluation also allows us to understand the significance of GAE background processing on the total system resource usage. Figure 9 illustrates the modeled active power due to sum of requests and due to background processes respectively for GAE-Vosao on our SandyBridge machine. Our power model shows that almost one third of the total system active power can be attributed to background processing in a GAE system.

Prediction At New Request Composition The above validation shows that nearly all measured energy usage is accounted for and attributed. However, it does not validate whether the request power and 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). A successful validation of this prediction would indicate that the profiled per-request energy usage was accurate. Here we assume that the energy usage for each type of request does not change from the profiled system to new workload conditions. Note that this assumption does not hold for workloads (like Stress) that exhibit dynamic behaviors at different resource contention levels on the multicore.

For comparison, we consider two alternative approaches to predicting 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. Note that this approach requires profiling the CPU usage of individual requests through careful request context tracking [6, 7].

We perform evaluation on new RSA-crypto and WeBWorK workload conditions by changing their request type composition. For RSA-crypto, the original workload contains requests with three different encryption keys. The new workload only contains requests with the largest key among the three. For the WeBWorK application, the original problem-solving workload includes thousands of science problem sets used in the real site. We consider a new workload with only the 10 most popular problem sets. Figure 10 shows the predicted power and measured power at several load levels between median and high load. Results show that our request energy profile-enabled prediction achieves higher accuracy than the two alternatives (particularly at high load). Our approach has up to 11% prediction error, compared to up to 19% error for the CPU-utilization-proportional approach and up to 56% error for the request-rate-proportional approach.

4.3 Fair Request Power Conditioning

We evaluate the effectiveness of fair power conditioning using container-specific CPU throttling (presented in Section 3.4). We experiment with our Google App Engine (GAE)-enabled cloud computing workloads. The GAE-Vosao workload fully utilizes all four cores on the SandyBridge machine. In the middle of the experimentation, we inject high-power requests to mimic power viruses. Our GAE 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 msecs. Figure 11(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. If all four cores are being utilized, the per-core active power target is 10 Watts when the system is fully busy. Figure 11(B) shows that our request container-enabled power conditioning can effectively keep power consumption at or below the target level despite the power viruses. It does so by throttling the core running the request that contains the power virus.

While capping the full system power, we further show that the CPU speed adjustment has been applied fairly to each request. Figure 12 plots the applied CPU duty-cycle ratio and original 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-

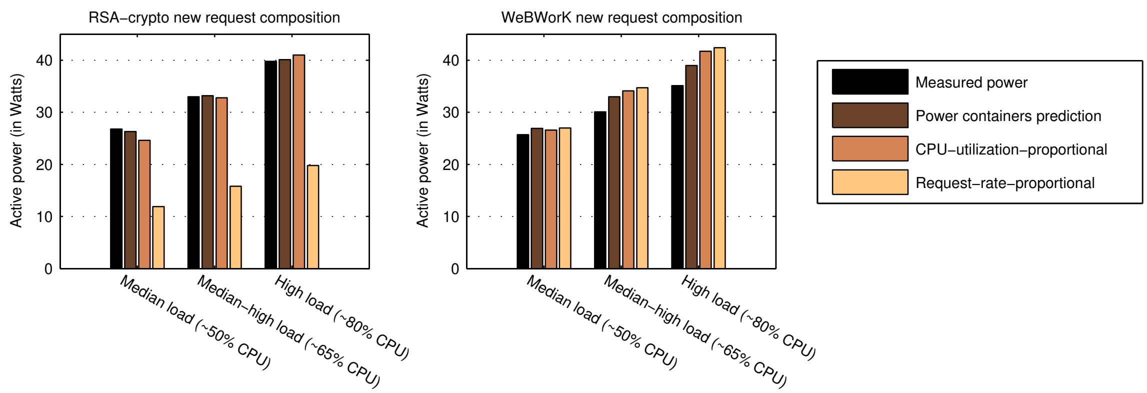

Figure 10. Accuracy of our power containers-produced request energy profiles (and alternative approaches) to predict power at new workload conditions. Experiments were performed on the SandyBridge machine.

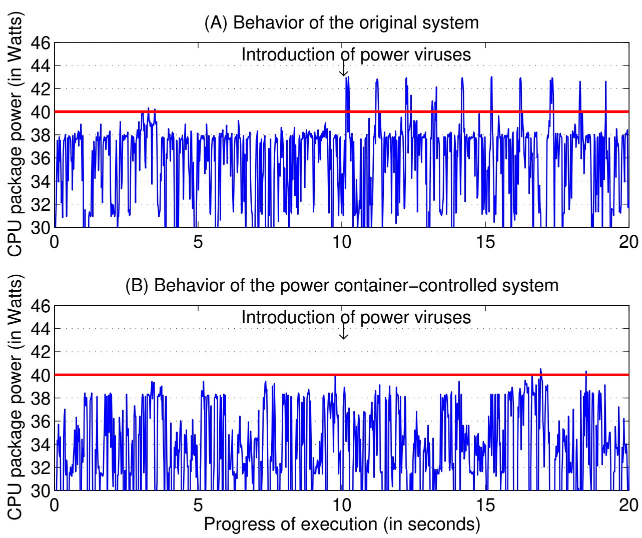

Figure 11. Measured power for original and power-conditioned executions of Google App Engine with power viruses. Experiments were performed on the SandyBridge machine and power is measured using its on-chip power meter. Power viruses are introduced at the 10-second time.

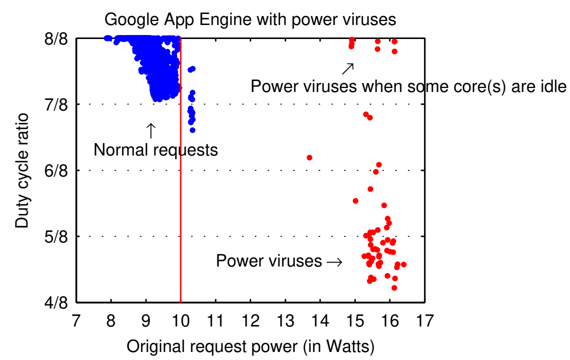

Figure 12. Original power and duty-cycle throttling for Google App Engine with power viruses. Each point represents a sample request. X-coordinate indicates the original (before throttling) request power consumption. Y-coordinate indicates the CPU duty-cycle ratio applied to the request.

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). Results show that low-power GAE-Vosao requests suffer minor CPU speed slowdown (averaged at about 2%). At the same time, the power viruses are subject to substantial (33% on average) slowdown. It may appear surprising that a few power viruses (at Figure 12's top-right corner) are not significantly throttled. This is because they happen to run when some CPU core(s) are idle so each request at the time has a higher-than-10 Watts budget when maintaining the whole system power target of 40 Watts.

Without our container-enabled fair power conditioning, alternatively the peak power can be reduced through full-machine throttling. A full-machine duty-cycle level of 7/8 would be required for such throttling, leading to about 13% slowdown of all requests (low-power normal requests as well as power viruses).

4.4 Heterogeneity-Aware Request Distribution

We evaluate container profiling-enabled energy-efficient request distribution in a heterogeneous server cluster (presented in Section 3.4). We start by assessing the energy efficiency heterogeneity across different machines. While recent processors (*e.g.*, SandyBridge) 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 profiling can quantify such workload-specific relative energy efficiency. Figure 13 shows the cross-machine (SandyBridge over Woodcrest) active energy usage ratio for different workloads. In this evaluation and analysis, we measure the active energy usage that does not include the idle power consumption. We consider the constant idle power as part of the fixed system overhead that is of little interest to our adaptive energy management. Figure 13 shows that the cross-machine energy usage ratio can be as high as 0.91 (for the Stress workload) and as low as 0.22 (for RSA-crypto). When distributing some load from SandyBridge to Woodcrest becomes necessary, placing a Stress request on Woodcrest would be four times more energy-efficient than placing a RSA-crypto request.

We evaluate request distribution over a small heterogeneous cluster consisting of two machines—the newer (more energy-efficient) SandyBridge and the older Woodcrest-based machine

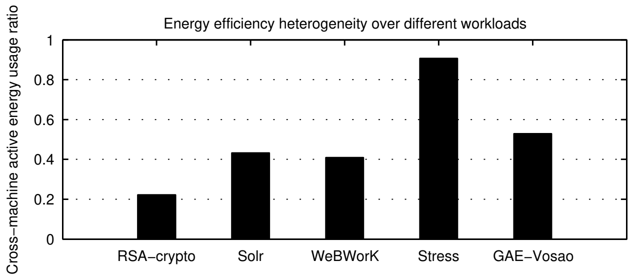

Figure 13. Cross-machine active energy usage ratio (energy usage on SandyBridge over that on Woodcrest) for different workloads. The energy usage on each machine is measured at the peak load level.

from our experimental platform. Our 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* balances the server load by directing an equal amount of load to each of the two machines. It is oblivious to the energy efficiency heterogeneity in the cluster.
- *Machine heterogeneity-aware* approach loads up the more energy-efficient SandyBridge to a healthy high utilization (about 70% CPU utilization to prevent overloading) before loading Woodcrest. It is oblivious to workload energy profiles so it distributes the exact same input request composition to both machines.
- *Workload heterogeneity-aware* approach also first loads up the more energy-efficient SandyBridge. Beyond that, it recognizes the request energy usage profiles using our power containers and it preferentially places requests with higher relative energy efficiency (lower energy usage ratio in Figure 13) on SandyBridge.

Our experiment utilizes a combined GAE-Vosao and RSA-crypto workload (with an approximately 50%–50% load composition). The volume of incoming work in our experiment is the maximum volume that can be supported (without excessive timeout) under the simple load balance. Figure 14 shows the energy usage rate under the three load distribution approaches. Our workload heterogeneity-aware approach saves 30% in combined two-machine energy usage compared to the simple load balance. The saving is 25% compared to the machine heterogeneity-aware approach that cannot recognize diverse workload-to-machine affinity. Our container-enabled request distribution achieves these energy savings by preferentially loading each machine with requests of high relative energy efficiency.

Beyond high energy efficiency, our workload heterogeneity-aware approach also maintains high performance. Table 1 shows the average request response time for GAE-Vosao and RSA-crypto under the three request distribution approaches. Both heterogeneity-aware approaches achieve good performance by keeping the machines under healthy utilization levels. The simple load balance suffers from poor performance because it fails to consider machine heterogeneity and overloads the Woodcrest machine.

5. Conclusion

This paper presents an operating system facility (power containers) to account for and control the power and energy usage of individual requests in multicore servers and cloud computing platforms. Power containers utilize an online per-core power estimation model that includes cross-core environmental effects, measurement-aligned online recalibration, and mechanisms within the operating system 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 with the help of online re-calibration, the acquired request power and energy usage profiles can be aggregated to match measured system power (with no more than 9% error) and predict system power at new, hypothetical workload request compositions (with no more than 11% error).

Power containers enable the OS to better manage online applications with dynamic power profiles as well as new hardware platforms with resource sharing and heterogeneity. When running a Google App Engine-based cloud computing workload, our power containers can cap the system power in a fair fashion—throttling power viruses (using processor duty-cycle modulation) while allowing normal requests to run at almost full speed. Further, the acquired request energy profiles enable energy-efficient request distribution on heterogeneous server clusters, saving up to 25% energy usage compared to an alternative approach that recognizes machine heterogeneity but not fine-grained workload affinity.

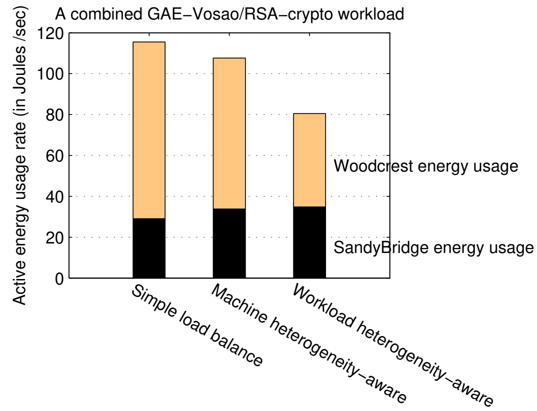

Figure 14. Measured active energy usage rate under three request distribution approaches in a heterogeneous server cluster. Usage for the two machines are marked in different colors.

Load distribution approach	GAE-Vosao	RSA-crypto
Simple load balance	537 msecs	1,728 msecs
Machine heterogeneity-aware	159 msecs	66 msecs
Workload heterogeneity-aware	131 msecs	50 msecs

Table 1. Average request response time for the two applications under three request distribution approaches in a heterogeneous server cluster.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation grants CNS-0834451, CCF-1016902, CNS-1217372, CCF-1217920, CNS-1239423. Kai Shen was also supported by a Google Research Award. Arrvindh Shriraman was supported by the Canadian NSERC Discovery grant, CRD 614093/569226, and Strategic 612142.

References

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

[2] Apache Solr search server. http://lucene.apache.org/solr/.

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

[4] Vosao content management system. http://www.vosao.org.

[5] Wikipedia data dumps. http://dumps.wikimedia.org/enwiki/.

[6] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Third USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 45–58, New Orleans, LA, Feb. 1999.

[7] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modeling. In *6th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 259–272, San Francisco, CA, Dec. 2004.

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

[9] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *ACM SIGOPS European Workshop*, pages 37–42, Kolding, Denmark, Sept. 2000.

[10] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *24th ACM Int'l Conf. on Supercomputing (ICS)*, pages 147–158, Tsukuba, Japan, June 2010.

[11] A. Chanda, A. Cox, and W. Zwaenepoel. Whodunit: Transactional profiling for multi-tier applications. In *Second EuroSys Conf.*, pages 17–30, Lisbon, Portugal, Mar. 2007.

[12] F. Chang, K. I. Farkas, and P. Ranganathan. Energy-driven statistical sampling: Detecting software hotspots. In *Second Workshop on Power-Aware Computer Systems*, pages 110–129, Cambridge, MA, Feb. 2002.

[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 *18th ACM Symp. on Operating Systems Principles (SOSP)*, pages 103–116, Banff, Canada, Oct. 2001.

[14] 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*, Big Sky, MT, Oct. 2009.

[15] X. Fan, W.-D. Weber, and L. Barroso. Power provisioning for a warehouse-sized computer. In *34th Int'l Symp. on Computer Architecture (ISCA)*, pages 13–23, San Diego, CA, June 2007.

[16] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *17th ACM Symp. on Operating Systems Principles (SOSP)*, pages 48–63, Kiawah Island, SC, Dec. 2001.

[17] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-Trace: A pervasive network tracing framework. In *4th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, Cambridge, MA, Apr. 2007.

[18] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *8th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 323–338, San Diego, CA, Dec. 2008.

[19] 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 *19th Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT)*, pages 19–28, Vienna, Austria, Sept. 2010.

[20] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *4th EuroSys Conf.*, pages 317–330, Nuremberg, Germany, Apr. 2009.

[21] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, and G. Srinivasa. The forgotten 'uncore': On the energy-efficiency of heterogeneous cores. In *USENIX Annual Technical Conf.*, Boston, MA, June 2012.

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

[23] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *10th ACM Symp. on Principles and Practice of Parallel Programming (PPoPP)*, pages 186–195, Chicago, IL, June 2005.

[24] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock. Accurate fine-grained processor power proxies. In *45th Int'l Symp. on Microarchitecture (MICRO)*, pages 224–234, Vancouver, Canada, Dec. 2012.

[25] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *12th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 121–132, Austin, TX, Feb. 2006.

[26] 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.*, Portland, OR, June 2011.

[27] 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 *5th Int'l Conf. on Information Processing in Sensor Networks*, pages 449–457, Nashville, TN, Apr. 2006.

[28] D. Meisner, B. Gold, and T. Wenisch. PowerNap: Eliminating server idle power. In *14th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 205–216, Washington, DC, Mar. 2009.

[29] 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*, pages 75–93, 2003.

[30] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power Management Architecture of the 2nd Generation Intel Core microarchitecture, formerly codenamed Sandy Bridge. In *Hot Chips: A Symposium on High Performance Chips*, Aug. 2011.

[31] 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 *6th EuroSys Conf.*, pages 139–152, Salzburg, Austria, Apr. 2011.

[32] K. Shen. Request behavior variations. In *15th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 103–116, Pittsburg, PA, Mar. 2010.

[33] K. Shen, M. Zhong, S. Dwarkadas, C. Li, C. Stewart, and X. Zhang. Hardware counter driven on-the-fly request signatures. In *13th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 189–200, Seattle, WA, Mar. 2008.

[34] 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.

[35] 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*, Big Sky, MT, Oct. 2009.

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

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

[38] 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 *16th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Bangalore, India, Jan. 2010.

[39] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *10th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 123–132, Boston, MA, Oct. 2002.