2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Application-Agnostic Power Monitoring in Virtualized Environments

James Phung* Young Choon Lee** Albert Y. Zomaya§
Email: james.phung@sydney.edu.au *Email: young.lee@mq.edu.au* *Email: albert.zomaya@sydney.edu.au*
*§School of Information Technologies, The University of Sydney, NSW 2006, Australia
**Department of Computing, Faculty of Science and Engineering, Macquarie University, NSW 2109, Australia

***Abstract*—Many servers use technologies such as virtualization or containerization to improve server utilization. These technologies pose challenges for power monitoring since it is not possible to directly measure the power use of an abstraction such as a virtual machine. Much work has been done in modeling the power use of CPUs, virtual machines and entire servers; however, there is a scarcity of work in building lightweight power monitoring middleware that can be deployed across a range of systems. In this paper, we present *cWatts+* as a prototype lightweight software-based virtual power meter. Utilizing a simple but powerful application-agnostic power model, it offers comparable performance to existing "more complex and heavier-weight" power models. It uses a small number of widely available CPU event counters and the Performance Application Programming Interface Library to estimate power usage on a per-thread basis. It has minimal overhead and is portable across a variety of systems. It can be used in containerized or virtualized environments. We evaluate the estimation performance of cWatts+ for a variety of real-world benchmarks that are relevant to large distributed systems. Also, we examine the importance of including CPU core temperature data in the power model. We demonstrate that our power model has an average error of less than 5%. This result compares favorably with existing state-of-the-art power models and is achieved using a relatively simple power model that exhibits minimal power consumption (overhead). Consequently, our power monitoring middleware is viable for use in real-world applications such as power estimation for energy-aware scheduling.**

1. Introduction

In an increasingly environmentally conscious and resource constrained world, performance and security are not the only concerns for large-scale systems such as cloud systems. The number and size of datacenters have exploded and continue to grow, leading to concerns over the rapidly increasing energy footprint of cloud datacenters [1]. These concerns are reflected in various proposals aimed at reducing the energy footprint of large datacenters and other distributed systems, e.g., [2] [3] [4]. Energy-aware scheduling algorithms such as [5] have been developed. Integral to the concept of power management is that power consumption of a particular user must be quantifiable in real-time. The widespread use of virtualization in distributed computing leads to some challenges for effective power monitoring.

The measurement of power of an entire computer is easily accomplished by using a commercially available power meter. However, it is not possible to attach a power meter to a virtualized system such as a Xen virtual machine [6]. Furthermore, a power meter does not enable power monitoring of individual threads, processes or applications. This is a serious limitation, given the rise in the use of virtualization technologies and demand for accurate real-time energy monitoring on a per-user basis, particularly in cloud and other large-scale distributed systems [7].

Power estimation has emerged as a standard approach for measuring energy consumption of servers in distributed networks such as cloud networks [8]. It involves applying a model to a set of performance parameters. These parameters may be software based such as a CPU utilization metric that is provided by the operating system or hardware based such as the event counters that are provided by modern CPUs. A typical approach to building a power model is to "train" it using benchmark data and correlational analysis . A trained power model can then be evaluated using unseen benchmarks. Power estimation tools can be applied at various system levels such as an entire server or a CPU [9]. Some power models involve multiple layers of abstraction such as the one in [7].

Consider a system that is running many applications concurrently. Our goal is to build a prototype lightweight power monitoring middleware called *cWatts+* that is capable of accurately estimating the power use of each of these applications in real-time. In order to minimize the middleware's power use, all parameters used for power estimation must be derived from data that is provided by hardware; no software-defined counters such as interrupt event counters may be used. Also, the number of layers of abstraction must be minimized. In addition, no assumptions may be made in relation to typical energy use patterns of particular classes of programs, i.e. the middleware must be 'application-agnostic'. It needs to be designed with portability in mind.

At its core, cWatts+ uses an application-agnostic reprogrammable power model. This model uses CPU performance event counters and CPU temperature sensors as input. It must be trained first before it can be used to estimate power use of threads and the entire system. To train the power model for our experiments, we use a series

978-1-5090-6611-7/17 $31.00 © 2017 IEEE
DOI 10.1109/CCGRID.2017.100

of benchmarks that we have developed in-house to simulate various types of real-world workloads. Data on actual system power consumption, CPU performance events and CPU temperatures are collected and analyzed off-line. Multi-step regression is used to determine the coefficients that best fitted the collected data. These coefficients are programmed into the power model.

cWatts+ is designed to operate in virtual environments without requiring proprietary or difficult-to-configure hardware or software. It is written in C and relies on Performance Application Programming Interface (PAPI), an open source library that operates on a wide range of platforms. More importantly, cWatts+ is temperature-aware; it takes into account the impact that CPU core temperatures have on fan speed and, therefore, power usage of individual threads and overall power consumption.

cWatts+ has a number of other key strengths. Firstly, it is application-agnostic; no application profiling is employed. The lack of application profiling in cWatts+ is an advantage since this allows it to estimate power consumption of a wide range of different types of applications including new types without the need for reconfiguration. Secondly, it is designed to be lightweight with a limited number of abstraction layers to ensure that its power consumption is minimal. Lastly, it relies on a limited set of widely available performance event counters to maximize portability.

Specific contributions of this paper are as follows:

- We develop a power monitoring middleware including an application-agnostic power model that is capable of accurately predicting power use of each concurrent user application in real-time;
- We evaluate our power model in a range of situations that are representative of real-world applications;
- We demonstrate the importance of including CPU core temperatures in the power model with regard to the reliability of the predicted power values; and
- We quantify the power use of our middleware.

We evaluate our power model using benchmarks from the PARSEC benchmark suite as well as our in-house benchmarks. We use cWatts+ to estimate the power use of the entire system while these benchmarks were running for various CPU frequencies. The differences between our estimates and the actual power use data (measured by the Cabac Power-Mate power meter[1]) are compared.

Our evaluation shows that cWatts+ estimates power use with an average error of 4.67%. It consumes 0.10 W, 0.12 W and 0.72 W when the CPU is running at 0.8 GHz, 2.2 GHz and 3.3 GHz respectively, frequencies that are representative of the full range of non-turbo frequencies that the CPU may operate at. These power usages represent less than 1% of overall system power consumption. Also, we find that for CPU temperatures over 46°C, power use increases linearly with increases in temperature. This illustrates the importance of including CPU temperature data in our power model to maintain its accuracy over a range of CPU temperatures.

1. http://www.cabac.com.au/products/electrical-test-and-measurement/power-meters/PM10AHDS

The remainder of this paper is organized as follows. Section 2 presents cWatts+ including our application-agnostic power model. Section 3 describes the procedures we have employed to build and evaluate our power model. Section 4 summarizes and discusses the results of our experiments. Section 5 discusses related work. This paper concludes with Section 6.

2. Power Monitoring Middleware and Modeling

In this section, we will firstly describe cWatts+, our power monitoring middleware. Secondly, we will describe and justify the parameters that were used for our power model that is used by cWatts+. Thirdly, we will describe our power model formulation. The power model is a quadratic function of various event counters provided directly by the CPU and built-in CPU temperature sensors.

2.1. Middleware – cWatts+

cWatts+ is a power monitoring middleware prototype that is designed to be deployable on a range of platforms. It consists of a reprogrammable application-agnostic power model, an interface that allows the middleware to interact with the user through the console and another interface to allow the middleware to access CPU event counters and temperature sensors via the PAPI library, UNIX *sensors* utility and operating system kernel. It estimates the power use of specified user-level threads and overall system power use.

cWatts+ operates as follows. Firstly, the user invokes it on the command line in a UNIX shell with one or more thread identifiers passed as arguments. These thread identifiers correspond to the user-level threads that the user wishes to monitor. Once launched, the middleware sends the appropriate calls to the PAPI library and *sensors* utility every second for sampling the CPU event counters and CPU temperature sensors respectively. These calls are translated into operating system calls. The kernel then translates them into hardware calls to the CPU, requesting the event counters and temperature sensors to be read. Once the event counters and temperature sensors are read, the reverse of the aforementioned process is used to deliver the results to the middleware. These results are used by the power model to estimate the power use of each selected thread as well as the overall power use. The estimates are displayed on the console in real-time. The relationships between cWatts+ and other system components are depicted in Figure 1.

2.2. Event Counters

For our power model, we rely primarily on performance event counters that are widely supported by modern Intel and AMD x86 based CPUs. A performance event counter counts the number of times that a particular event occurs such as last level cache (LLC) misses and mispredicted

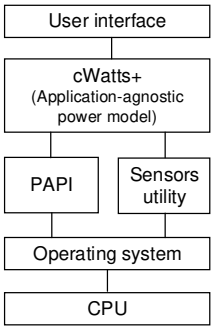


Figure 1: Relationships between cWatts+ and other system components.

branches. Most modern CPUs are capable of counting multiple events in real-time simultaneously. The event counters can be accessed directly through the use of software libraries such as those provided by PAPI or *perfmon*. Different CPU models support different events; nevertheless, a few events such as *microinstructions executed* are almost universally supported by modern CPUs. Since only a finite number of events can be counted simultaneously, prudent selection of events for counting is essential in building a reliable power model.

We select event counters that meet the following criteria:

- Applicability to workloads – There must be a causal relationship between an event counter and power consumption;
- Stability of event counters – Each event counter must be stable enough so that the results can be analyzed; and
- Portability of event counters – The selected event counters must be available on a range of platforms.

Therefore, we select the following event counters for our power model:

- Microinstructions;
- Unhalted cycles;
- Reference cycles;
- Last level cache (LLC) read misses; and
- Translation lookaside buffer (TLB) misses.

A microinstruction is the basic unit of work for a CPU. Each instruction consists of one or more microinstructions [10]. Consequently, the number of executed microinstructions is more likely to be a good proxy for CPU power use than the number of executed instructions.

An unhalted cycle is a clock cycle "when the clock signal on a specific core is running" [11]. The reference cycles counter is incremented at the rate of 100 cycles per second whenever power is applied to the CPU. When multiple running threads compete for CPU time, the effective CPU frequency F may be calculated as follows:

$$F = \frac{U}{R} \tag{1}$$

where U is the sum of unhalted cycles across all applications; and R is the sum of reference cycles across all applications.

We assume that the CPU power consumption is a quadratic function of CPU frequency and varies *linearly* with the number of executed microinstructions. This assumption is consistent with previous work on CPU power modeling such as [7] and [10].

Since memory accesses consume a significant amount of energy, we need to account for this usage in our power model. While there is no event counter that measures memory accesses directly, the number of LLC read misses provides a good proxy for the number of memory accesses. If a LLC read miss occurs, then a read access to main memory must occur (this is not necessarily the case for LLC write misses under a write-back cache policy as explained in [10]).

TLB misses incur memory accesses in similar fashion to cache misses but typically involve larger amounts of data such as a memory page as opposed to a cache line for cache misses [10]. The larger amount of data involved in a TLB miss means more energy is consumed. Additionally, the resolution of a TLB miss may involve retrieving a virtual memory page from the hard disk [10].

Given the CPU intensive nature of distributed computing applications and the fact that the CPU often represents more than half of overall system power consumption [8], we hypothesize that there is a strong correlation between CPU power use and overall system power use. Thus, we use CPU power as a proxy for system power minus the power consumed by the memory sub-system.

Unlike many existing power monitoring middleware, cWatts+ operates by considering only user-level events; kernel-level events are not monitored at all. This has one key advantage: it does not matter whether the existing system configuration will allow monitoring of all kernel-level events or not. This is important since with virtualization technologies such as virtual machines and containers, it is not always practical to access information relating to all kernel-level threads; some of them may be hidden by design for security reasons.

Being a prototype, cWatts+ does not support the automatic real-time detection of user-level processes and threads that belong to a particular user. For deployment in real-world systems, such functionality would be crucial.

2.3. Power Model

To build the power model, it is convenient to decompose power use into three components: idle power use, additional power use due to workloads excluding temperature-dependent effects and temperature-dependent additional power use due to higher fan speeds. When a computer is idle, its power consumption is not trivial; typically, it represents a significant fraction of the power consumption under maximum workload. We hypothesize that the idle system power P_{idle} is a quadratic function of CPU frequency (the constant term captures power use due to other system components). A quadratic term is included to capture nonlinear impacts on CPU power use (and, hence, system power

use) of varying the CPU frequency. We define P_{idle} as follows:

$$P_{idle} = aF + bF^2 + c \tag{2}$$

where F is the effective frequency of the CPU as defined in (1); and a, b, and c are coefficients to be determined.

Each workload requires additional system power. In general, multiple workloads are running at any given time. Suppose that the workloads $0, \ldots, n$ consume P_k respectively where $k = 0, \ldots, n$. In consideration of this, we define P_{work} and P_k as follows:

$$P_{work} = \sum_{k=0}^{n} P_k \tag{3}$$

$$P_k = (dF + eF^{3/2} + fF^2)\mu + g\alpha + h\beta \tag{4}$$

where:

F is the effective frequency of the CPU as defined in (1);
μ is the number of microinstructions issued;
α is the number of LLC read misses;
β is the number of TLB misses; and
d, e, f, g and h are coefficients to be determined.

Again, we have taken into account non-linear relationships between power use and varying CPU frequencies.

We need to factor in additional power consumption due to increased fan speeds when CPU temperatures are very high. We hypothesize that for sufficiently low CPU temperatures, the amount of energy needed for cooling does not vary appreciably as the CPU temperature changes; in contrast, for high CPU temperatures the amount of energy needed for cooling varies signifiantly with small changes in CPU temperature. Accordingly, we define P_{heat} as follows:

$$P_{heat} = \begin{cases} j(T - T_o) + k(T - T_o)^2 & \text{if} \quad T > T_o \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

where T is the average temperature of the CPU cores at a particular instant; and T_o is a value that will be determined experimentally. Now, the total power P_{total} is given by the following relation:

$$P_{total} = P_{idle} + P_{work} + P_{heat} \tag{6}$$

where:

P_{total} is the total power consumption of the system;
P_{idle} is the power consumed by the system when it is idle;
P_{work} is the additional power consumed by the system due to one or more workloads; and
P_{heat} is the extra power consumed by the system due to increased fan speed when CPU core temperatures exceed a certain threshold.

Equations (1) to (6) collectively represent the proposed power model.

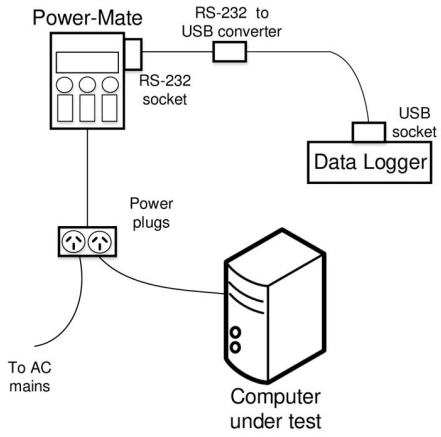


Figure 2: Experimental setup for power estimation experiments.

3. Experiments

In this section, we describe the experimental procedure that we have used to build and evaluate our power model. We have implemented cWatts+ in `C` language. To calculate the coefficients in (2) and (4), the power model was "trained" using a variety of test data as described in Section 3.2. Additionally, the value of T_o in (5) was determined by observation during the training stage. Then, the power model was evaluated as described in Section 3.3.

3.1. Experimental setup

The experiments were run on a desktop computer that contained a 3.3 GHz Intel Core i5 Haswell CPU. A *Cabac Power-Mate* power meter was connected between the mains socket and the computer to enable real-time power measurements of the computer to be made. A USB link between the power meter and a laptop allowed the measurements to be recorded in real-time via proprietary software (supplied with the power meter). The readings were saved to comma separated values (CSV) files for later offline analysis and evaluation. The experimental setup is depicted in Figure 2.

To capture data provided by the CPU's event counters and core temperature sensors, the PAPI library and *sensors* utility were used. The event counters and core temperatures were monitored whilst workloads were running; however, only events that were caused by these workloads were captured. The captured data were saved to CSV files for later offline analysis and evaluation. The CPU frequency was varied using the *cpufreq-set* utility.

3.2. Training Stage

To build the power model, we assumed that the power model can be described by (1) to (6) as defined in Section 2. To obtain the coefficients, the model needs to be "trained". Training involved two steps. Firstly, data for power consumption and event counters were collected for a diverse

Workload	Estimated Power Use (W)			Power-Mate Reading (W)			Relative Error (%)		
	0.8 GHz	2.2 GHz	3.3 GHz	0.8 GHz	2.2 GHz	3.3 GHz	0.8 GHz	2.2 GHz	3.3 GHz
Blackscholes	28.00	40.20	58.02	29.36	43.17	62.77	-4.60	-6.88	-7.57
Bodytrack	28.66	43.71	65.84	29.72	43.27	61.14	-3.56	1.03	7.69
Facesim	29.40	45.77	65.28	30.84	47.27	67.80	-4.67	-3.17	-3.71
Ferret	31.89	52.31	77.74	31.07	46.38	67.90	2.64	12.79	14.50
Fluidanimate	29.63	46.90	69.77	30.15	45.90	67.28	-1.73	2.17	3.70
Freqmine	29.91	48.50	74.41	30.12	46.42	69.61	-0.69	4.48	6.90
Matrix Multiplication	33.00	48.20	59.86	30.24	44.22	62.78	9.15	9.00	-4.64
Sieve of Eratosthenes	29.26	47.11	74.20	30.27	47.59	74.25	-3.36	-1.01	-0.06
Strings	29.25	47.06	74.33	29.87	46.39	72.31	-2.10	1.46	2.80

TABLE 1: Estimated power use readings, actual power use readings and relative errors for various workloads.

range of workloads (that were developed in-house) and CPU frequencies. The number of workloads running was varied between one and four to capture power use due to non-CPU components. The types of workloads were varied to ensure that they were representative of real-world applications but were stable enough to ensure that unintended noise was not introduced into the training process. Varying the CPU frequency allowed for variations in idle power use due to the former to be captured. The following workloads were used:

- Dummy – This workload consisted of a single infinite loop containing no operations;
- Floating – This workload consisted of a series of floating-point operations (division, square roots, trigonometric);
- Integer – This workload consisted of a series of addition, subtraction and division operations involving integers;
- Matrix multiplication – This workload consisted of a series of naive matrix multiplication operations that resulted in poor temporal and spatial locality of memory accesses;
- Sieve of Eratosthenes – This workload consisted of a series of operations aimed at finding all the primes numbers from 2 to n inclusive where n was a randomly chosen integer that was greater than unity; and
- Strings – This workload consisted of a series of string operations.

We have used six CPU frequencies: 0.8 GHz, 1.5 GHz, 2.2 GHz, 2.8 GHz, 3.1 GHz, and 3.3 GHz. The use of multiple CPU frequencies was intended to allow both linear and quadratic relationships to be captured.

Event counters and real-time power meter readings were captured at one second intervals. The collected data was saved for processing. MATLAB scripts were used to estimate the coefficients of (2) and (4). The estimation of coefficients was performed using multi-step regression. Prior to performing the regression, outliers and other irrelevant data were manually removed. To estimate c, real-time power measurements were collected for a range of frequencies when the system was idle. It was assumed that the idle power consumption is a linear function of the frequency. The constant c is the estimated idle power usage for the hypothetical case where the CPU frequency is zero.

An appropriate value of T_o is the lowest temperature at which a strong correlation between power use and temperature existed. To determine this value, we examined the changes in CPU core temperatures and power use when the Sieve of Eratosthenes workload was run at maximum nominal core frequency (3.3 GHz) over an eight-minute interval. We chose this workload as its CPU-intensive nature tended to cause the CPU cores to become increasingly hot when it was run for a few minutes at maximum CPU speed. For this case, we observed temperatures in the 49-76°C range. However, we could not reliably reproduce this effect for relatively low core frequencies. Consequently, an appropriate value of T_o was 50°C.

Next, the coefficients j and k were estimated by regression using data from the experiment as outlined in the previous paragraph. Before regression was performed to find the remaining coefficients, the raw real power readings were adjusted to take into account the additional energy use due to higher fan speeds that occur when $T_o > 50°\mathrm{C}$. This step was necessary since we needed the power consumption data for the case where $T_o < 50°\mathrm{C}$. Without this correction, the coefficients for P_{work} would include noise due to high CPU core temperatures. Lastly, the remaining coefficients were estimated by regression using power readings and performance event counter readings for a range of workloads with c being fixed to the value that was determined earlier.

3.3. Evaluation Stage

To verify the correctness and robustness of our power model, we devised a series of experiments that involved a variety of unseen workload types. These workload types covered a representative range of operations that reflected real-world tasks. The use of unseen workloads was necessary to minimize the effect of trainer bias that tends to affect power models that are trained using benchmarks [9]. The following workloads from the PARSEC benchmark suite [12] were used:

- Blackscholes – Performs option pricing calculations involving Black-Scholes partial differential equations;
- Bodytrack – Performs computations associated with body tracking tasks;
- Facesim – Performs computations associated with animating a human face;

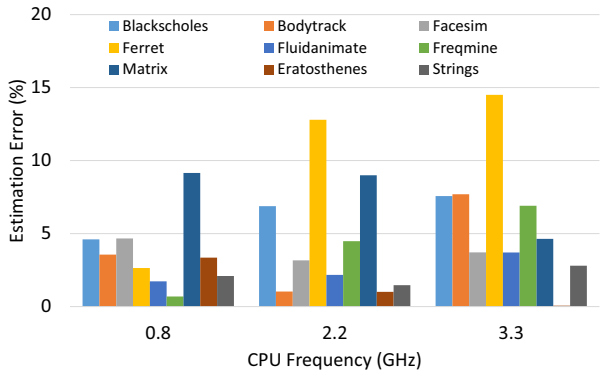


Figure 3: Power estimation performance for various workloads with respect to different CPU frequencies.

- Freqmine – Performs computations associated with data mining for frequent items;
- Ferret – Performs computations associated with identifying similar content; and
- Fluidanimate – Performs computations associated with animating fluid dynamics phenomena using the Smoothed Particle Hydrodynamics method.

Additionally, we used the *Matrix multiplication*, *Sieve of Eratosthenes* and *Strings* workloads in our evaluation.

The experiments were run for the following core frequencies: 0.8 GHz, 2.2 GHz and 3.3 GHz unless stated otherwise. These frequencies were chosen as they were representative of the CPUs frequency range. The coefficients determined during the training stage were used for the power model here. Experiments were run in a similar manner to those in the training stage. The event counter values served as inputs to the power model and the power model returned estimated power consumption values. The estimated power values and actual power readings were collected in real-time and saved for offline analysis and evaluation.

4. Results and Discussion

In this section, we will discuss the accuracy of our power model cWatts+. In particular, we will examine three important issues that we encountered during our experiments: (1) stability of LLC and TLB event counters, (2) temperature dependence and (3) power consumption of cWatts+ itself. Our experiments validated the need to allow for temperature-dependent variations in power use and cWatts+ uses only a very small amount of power. However, there were some issues with the stability of LLC and TLB counters; they will be discussed in more depth later in this section.

Our power model used the following coefficients to satisfy (3) and (5) as defined in Section 2: $a = 0$, $b = 0.5$, $c = 25$, $d = 1.157$, $e = -0.9158$, $f = 0.2739$, $g = 0.00000025$, $h = 0.00000025$, $j = 0$ and $k = 1.1$.

Table 1 and Figure 3 show the average power estimation errors for cWatts+ for various evaluation workloads and CPU frequencies. The maximum measurement uncertainties for estimated power use, actual power use (as measured by the Cabac Power-Mate power meter) and relative error are 0.11 W, 0.24 W and 0.07 % respectively. The overall absolute error for this model is 4.67 %, which is comparable to results in [13] and [14]. Since the individual errors can be positive or negative, the overall error was calculated using the average of the absolute values of the respective individual errors.

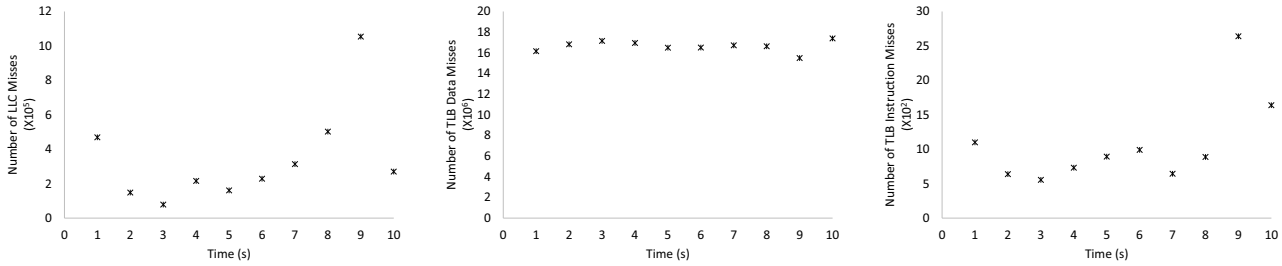
Power estimation performance for a range of workloads was generally stable. Estimation errors mostly remained as little as 0.06% up to 9.15% of the actual value, except two cases with the *Ferret* workload at CPU frequencies of 2.2 GHz and 3.3 GHz, respectively. These exceptions may have arisen from the fact that the CPU utilization of the Ferret workload varies considerably during execution. In other words, the sample event counters might not accurately represent instantaneous CPU activity in these two cases since the events are sampled every second. A possible solution would be to sample the events more frequently; however, more frequent sampling would mean more CPU resources must be allocated to cWatts+, which would lead to increased power consumption. This outcome would be contrary to our goal of building lightweight power monitoring middleware.

4.1. Stability of LLC and TLB event counters

Power use due to LLC read misses and TLB misses was significant for some workloads. For these workloads, there were considerable short-term variations in these values, which led to some instability in the estimated power values. Samples of LLC read misses and TLB misses are given in Table 2 and Figure 4 for the PARSEC *freqmine* workload at a core frequency of 3.3 GHz (this effect is noticeable for all frequencies but is more pronounced at higher frequencies).

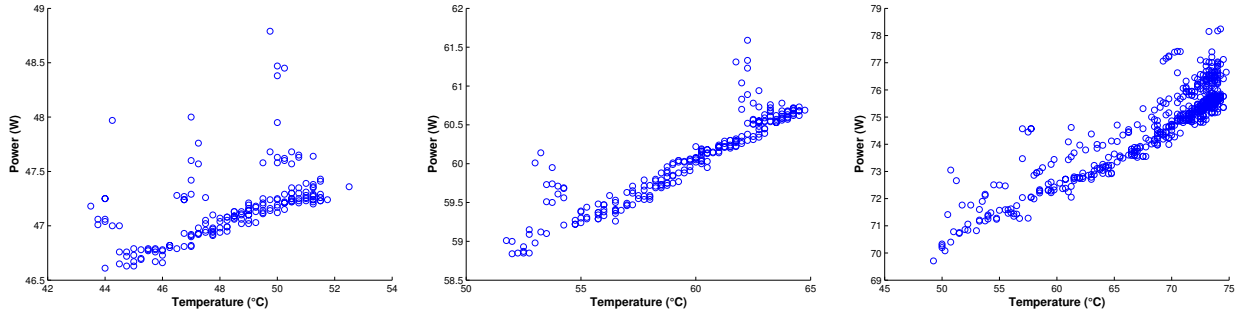
The variations are particularly evident for LLC misses as Figure 4(a) clearly shows, where the highest number of LLC misses is over 13 times the smallest number of LLC misses over the sample ten second interval and the variations exhibit a chaotic pattern. Likewise, a similar chaotic pattern in variations for TLB instruction misses can be observed in Figure 4(c). In contrast, there was less variation in TLB data misses as illustrated in Figure 4(b) where the plots approximate a straight line; in spite of this, the largest recorded number of TLB data misses was still over 12% greater than the smallest recorded number of TLB data misses for the sample ten second interval. Such variation is sufficient to induce significant oscillations in power estimates.

To reduce instability in power estimates due to large short-term variations in LLC read misses and TLB misses, a possible approach would be to compute running averages of these two parameters, subject to the proviso that other parameters such as the number of executed microinstructions remain stable. As soon as this proviso is no longer met, these running averages would be discarded. This approach assumes that as long as the nature of the workload on the CPU is the same, some key parameters such as microinstructions executed should not exhibit drastic changes. However, such an approach would introduce more complexity into the power model, potentially increasing its power consumption.



(a) LLC misses. (b) TLB data misses. (c) TLB instruction misses.

Figure 4: LLC and TLB misses over a 10 second period for the *freqmine* workload at a CPU frequency of 3.3 GHz.



(a) 2.2 GHz. (b) 2.8 GHz. (c) 3.3 GHz.

Figure 5: Relationship between temperature and power consumption for Sieve of Eratosthenes workloads at different CPU frequencies.

Time (s)	LLC misses	TLB data misses	TLB instruction misses
1	468729	16148764	1099
2	148123	16817811	637
3	78056	17147106	555
4	215681	16942203	731
5	160687	16487624	892
6	228419	16505663	990
7	313280	16715658	643
8	502146	16626997	887
9	1053705	15485417	2639
10	269771	17390167	1638

TABLE 2: Ten second sample of LLC misses and TLB misses for the *freqmine* workload at a CPU frequency of 3.3 GHz.

Also, the concept of stability here lacks a firm definition. For these reasons, this potential enhancement was not incorporated into cWatts+.

4.2. Temperature Dependence

Since the system is expected to operate over a wide range of ambient temperatures and the required fan speeds to cool the CPU and other system components increases with ambient temperature, the actual fan speeds for a given CPU frequency will depend on the ambient temperature. As higher fan speeds lead to higher energy use, it follows that actual system power consumption will vary with ambient temperature. An intuitive approach to capture temperature-dependent power variations would be to include the speeds of CPU and system fans in the power model.

CPU frequency (GHz)	Power use increase (W) per 1°C increase in temperature	Correlation coefficient
2.2	0.13	0.5172
2.8	0.15	0.9132
3.3	0.22	0.9125

TABLE 3: The correlations between power use and temperature, and correlation coefficients for Sieve of Eratosthenes workloads.

Because many systems, including our test system, do not have sensors to monitor fan speeds, we used CPU core temperatures as proxies for fan speeds in cWatts+. As the workload on the CPU increases, the energy dissipated increases due to the increased number of computations that need to be performed. The increased energy dissipation means the CPU needs to dissipate more heat. The increased heat dissipation leads to a rise in core temperatures. In the case where the ambient temperature rises, the surrounding air is less able to cool the CPU via the CPU fan and heatsink, leading to a rise in core temperatures. Note that while the CPU fan speed increases in response to higher CPU core temperatures, this only shortens the time taken for the CPU to approach an equilibrium temperature (which depends on workload and core frequencies) by making convective heat transfer more efficient.

To effectively show the relationship between temperature and power use (Figure 5 and Table 3), we have run a CPU-

bound workload (*Sieve of Eratosthenes*) at core frequencies of 2.2 GHz, 2.8 GHz and 3.3 GHz respectively. In particular, this workload consistently maintains a very high CPU utilization, leading to more CPU heat dissipation. Also, we chose these three frequencies since the CPU exhibits more heating at higher frequencies. Maximizing heat dissipation allowed us to observe changes in overall system power use over a range of CPU temperatures. For all three CPU frequencies, we observed a linear relationship between CPU temperature and overall power use.

These results demonstrate the importance of including CPU core temperatures in our power model in order to maintain its accuracy over a wide range of temperatures. For core temperatures over 46°C, power consumption rises with increases in temperature. For core temperatures below 46°C, no clear correlation between temperature and power use is observed.

Given a sufficiently high temperature, the power model's performance is improved by adding a linear term for average core temperature. However, this implication may not hold fully for higher core temperatures due to limited cooling fan speed, non-linear cooling response and automatic core frequency reductions if core temperatures increase beyond predetermined thresholds. Similarly, for relatively low core temperatures, fan speed may not change appreciably as core temperature changes leading to minimal changes in power use.

4.3. Power Consumption of cWatts+

cWatts+ incurs CPU computations as it runs; therefore, like all other processes, it consumes energy (power use overhead). To quantify this energy use, we ran four instances of the Sieve of Eratosthenes workload simultaneously with and without cWatts+ running at core frequencies of 0.8 GHz, 2.2 GHz and 3.3 GHz. Average power usages for these frequencies when cWatts+ was not running were 29.47 W, 47.86 W and 74.66 W respectively. Average power usages when cWatts+ was running were 29.57 W, 47.98 W and 75.38 W respectively. These results imply that for the CPU frequencies 0.8 GHz, 2.2 GHz and 3.3 GHz, cWatts+ consumes 0.10 W, 0.12 W and 0.72 W respectively. In all cases, the power used by cWatts+ represents less than 1% of the power used by the system when the middleware is not running, demonstrating that power use incurred by cWatts+ is minimal.

cWatts+ does not explicitly measure the power consumption of hard disks. This limitation is not a significant shortcoming since power consumption of modern hard disks represents only a small fraction of overall system power use [15].

5. Related Work

There is a large body of research on power modeling and estimation. Previous papers have focused on power monitoring at three different granularities: CPU, virtual machine and server. The techniques involved differ for these granularities. There are wide variations in architecture, choice of performance events and regression techniques.

In [16], Lim et al. adopted a similar approach to our work. Event counters provided by the CPU were used to estimate the power use of an entire system. A power model was trained and then evaluated. The authors used a power meter that required physical reconfiguration of the computer in question in order to connect the power meter; in contrast, our power model uses a power meter that measures power use at the wall socket.

Because CPUs account for a large proportion of the energy consumption of servers, CPU power estimation is a key tool in estimating the power usage of other components such as server clusters [9]. A large number of papers have been published on this topic. Examples of notable research in CPU power modeling include [17], [18], [19] and [20]. Most researchers working in this area have employed linear regression for constructing their power models, albeit with some variations. For example, piecewise linear regression is used in [18] while nonlinear parameter transformation is combined with linear regression in [20] in order to model the power consumption of each core in a multi-core processor; our work follows this pattern, combining piecewise linear regression with nonlinear parameter transformation. Researchers tend to employ CPU performance event counters such as the number of micro-operations or number of instructions retired for their CPU power models; operating system provided event counters are mostly avoided due to latency considerations. For our work, we have adopted the same approach. However, the authors of [19] use an entirely different approach: they consider the power consumption of simple arithmetical instructions in order to construct their power model. We did not use this technique since it requires application profiling, which is incompatible with the requirement that cWatts+ is application-agnostic.

Numerous researchers have explored the concept of estimating the power usage of an entire server or cluster of servers. As a representative example, the authors of [10] observe that there is a strong correlation between certain CPU performance event counter values and the events that take place outside the CPU. For example, the counter corresponding to the highest level cache misses may serve as a reliable proxy for the number of memory accesses; thus, it is possible to estimate the energy consumption of memory by building a power model that correlates power use of memory to the number of highest level cache misses. The server is considered to be a composition of five sub-systems: CPU, chipset, memory, disk, and network. In order to estimate the power consumption of a server, the researchers take the following approach: for each sub-system, a power model is constructed using the CPU performance event counters that are most closely related to the activities that take place within the sub-system in question. For example, the power model for the memory sub-system is a quadratic expression in terms of the number of level 3 cache misses incurred by a Pentium 4 CPU. Then, the power model for the entire server is simply the sum of all sub-system power models. Thus, the

power consumption of the server can be estimated by adding the estimated power consumption for each sub-system.

We use a similar approach as described above for our power model but on a per-thread basis; our power model estimates the power use incurred by each running user thread. Also, we consider the CPU and memory sub-systems only. Our justification for considering these sub-systems only is that together they account for a large proportion of overall power use and that our goal of building a lightweight power model demands that we minimize the amount of monitoring that is required to produce reliable results.

Similar concepts are used in [13], [15], [21] and [22] to estimate the power usage of servers. However, whilst the use of CPU utilization as a model input parameter is near-universal in modeling a server's energy use, the use of other utilization metrics varies considerably. The model developed in [21] makes use of the number of concurrently open sockets in addition to other standard utilization metrics concerning the memory and the input/output sub-systems while the model in [22] uses CPU utilization and frequency only. Also, different modeling techniques are used; whereas linear regression is used in [21], linear programming is employed in [13] and [22] assumes that the relationship between CPU frequency and utilization is quadratic.

Approaches to modeling the power consumption of virtual machines are considerably more varied. In [23], researchers adopt a black-box approach to estimating the power used by each virtual machine. Resource mapping is used to construct power models for virtual machine power consumption by resource type, using linear regression. Then, a weighted sum of these power models is used to estimate the power usage of each virtual machine. The appropriate weightings are determined by the resource utilizations of each virtual machine. Power models for the CPU and the memory sub-system are used in order to estimate the power usage of various system resources.

Colmant et al. presented BitWatts, a working prototype that estimates the power consumption of each process running within a virtual machine [7]. The ability to monitor power use of each process running within a virtual machine contrasts with the black-box approach to power estimation of virtual machines in [23]. BitWatts is a middleware implementation that runs beneath the host operating system and within virtual machines. An instance running directly beneath the host operating system monitors the relevant hardware counters for CPU activity and converts the collected data into estimates of the total CPU energy used. To determine the CPU energy usage for each virtual machine, the utilization of each virtual machine is also tracked. Also, each virtual machine has an instance of BitWatts running that operates similarly to the host instance; however, these instances monitor the virtualized CPUs rather than the physical CPUs. VirtioSerial is used to allow the host instance to notify each virtual machine instance of the amount of CPU energy used. Since the virtual utilization of each process is also tracked, this hierarchical arrangement enables the CPU energy usage of each process to be estimated as it runs. Another key strength of BitWatts (and our power model) is that it does not require changes to the operating system's hardware interfaces [7].

Somewhat different approaches for modeling virtual machine energy consumption are used in other works. In [24], resource mapping is used but kernelized canonical correlational analysis instead of linear regression is used. The derived model input parameters involve the CPU, disk, memory and network utilizations, and the overall server power consumption. In [25], CPU utilization is the sole model input and linear interpolation is used. In contrast, a larger set of performance metrics is used in [26] and Gaussian Mixture Vector Quantization is used to build the virtual machine power model.

For our power model, we need to minimize the number of abstractions in order to reduce overhead. Simultaneously, our power model needs to be portable across a range of platforms and be able to operate in both physical and virtualized environments. For these reasons, we have chosen to use Performance Application Programming Interface (PAPI) for reading event counters. This approach would allow event counters to be accessed in virtual environments without introducing substantial overhead.

6. Conclusion

We have developed and evaluated a prototype power monitoring middleware cWatts+ that is capable of measuring energy use on a per-thread basis. In spite of its simplicity, it is capable of producing power estimates that are of comparable reliability to existing power models that are often far more complex. The use of a limited number of CPU counters means that it is expected to work on a variety of modern x86 based systems. Crucially, cWatts+ consumes very little energy. The lightweight and portable nature of cWatts+ means that it is well suited for deployment in a variety of systems.

We have shown that the power model that cWatts+ uses can easily be extended to account for additional energy use due to increased fan speeds due to high CPU core temperatures. Such an extension sufficiently improves the power model's estimation accuracy to make it worthwhile.

In the future, we will evaluate our powel model against a larger range of workloads. Also, we will evaluate the ability of cWatts+ to estimate power use under continuously varying workloads. These further evaluations will enable futher validation of our power model. Additionally, we will extend cWatts+ to enable it to perform energy accounting on a per-user basis and evaluate it in virtualized environments. In real-world distributed systems, particularly cloud systems, accurately attributing energy use for each user is important since such systems consume large quantities of energy and fairness in charging users is a key business concern.

Acknowledgments

This work was supported by an Australian Research Council (ARC) Discovery Grant (DP1097110) and an ARC Linkage-Industry Grant (LP140100980).

The authors would like to thank Omer Adam for his assistance in formatting this paper.

References

[1] J. Baliga, R. W. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[2] D. Su and Y. Lu, "Greenmap: mapreduce with ultra high efficiency power delivery," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[3] C. Wang, B. Urgaonkar, G. Kesidis, U. V. Shanbhag, and Q. Wang, "A case for virtualizing the electric utility in cloud data centers," in *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, 2014.

[4] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11227-010-0421-3

[5] ——, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, Aug 2011.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, 2003, pp. 164–177.

[7] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, "Process-level power estimation in VM-based systems," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 14.

[8] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 47, 2014.

[9] C. Möbius, W. Dargie, and A. Schill, "Power consumption estimation models for processors, virtual machines, and servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1600–1614, 2014.

[10] W. L. Bircher and L. K. John, "Complete system power estimation: A trickle-down approach based on performance events," in *2007 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, 2007, pp. 158–168.

[11] Intel Corporation, "Intel® 64 and ia-32 architectures software developers manual," 2016.

[12] Princeton University, "PARSEC benchmark suite," 2016. [Online]. Available: http://parsec.cs.princeton.edu/.

[13] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proceedings of the Workshop on Modeling, Benchmarking, and Simulation (MoBS)*.

[14] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, ser. HotPower'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855610.1855613

[15] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[16] M. Y. Lim, A. Porterfield, and R. Fowler, "Softpower: fine-grain power estimations using performance counters," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 308–311.

[17] W. L. Bircher, M. Valluri, J. Law, and L. K. John, "Runtime identification of microprocessor energy saving opportunities," in *ISLPED'05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.* IEEE, 2005, pp. 275–280.

[18] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003, p. 93.

[19] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, "Characterizing the energy consumption of data transfers and arithmetic operations on x86- 64 processors," in *Green Computing Conference, 2010 International*. IEEE, 2010, pp. 123–133.

[20] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[21] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 186–195.

[22] V. Petrucci, E. V. Carrera, O. Loques, J. C. Leite, and D. Mosse, "Optimized management of power and performance for virtualized heterogeneous server clusters," in *Cluster, cloud and grid computing (CCGrid), 2011 11th IEEE/ACM international symposium on*. IEEE, 2011, pp. 23–32.

[23] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan, "Vm power metering: feasibility and challenges," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 56–60, 2011.

[24] Q. Zhu, J. Zhu, and G. Agrawal, "Power-aware consolidation of scientific workflows in virtualized environments," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.

[25] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, vol. 4. ACM, 2010.

[26] G. Dhiman, K. Mihic, and T. Rosing, "A system for online power prediction in virtualized environments using gaussian mixture models," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 807–812.