Title
Improving the Energy Efficiency of Modern Computing Platforms using High-Resolution Real-Time Energy Measurements

Author
Singh, Digvijay

Publication Date
2014

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Improving the Energy Efficiency of Modern Computing Platforms using High-Resolution Real-Time Energy Measurements

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

Digvijay Singh

2014

ABSTRACT OF THE DISSERTATION

Improving the Energy Efficiency of Modern Computing
Platforms using High-Resolution Real-Time Energy Measurements

by

Digvijay Singh
Doctor of Philosophy in Electrical Engineering
University of California, Los Angeles, 2014
Professor William J. Kaiser, Chair

High-performance computing platforms have become critical in meeting the demands of modern computing applications. Rising performance requirements in a broad range of platforms from mobile devices to server systems combined with the proliferation of these high-performance computing platforms has increased the energy costs incurred and lead to an exigent need for improvement in platform

energy efficiency. This requires infrastructure for monitoring of energy consumption and methods to reduce the platform energy costs. In this dissertation, we present a new measurement infrastructure to provide real-time event-synchronized platform energy measurements, demonstration of these energy measurement capabilities through application to network data transport and an operating system task scheduler that utilizes these energy measurements to greatly improve energy efficiency for multi-core computing platforms.

The energy measurement infrastructure is integrated at the platform level and provides event-synchronized energy measurements for the complete platform along with important components such as the CPU, memory modules, secondary storage, peripherals and others. Furthermore, since modern secondary storage devices have buffering mechanisms that defer data write operations, the energy consumption of these operations is modeled and the model is integrated into the platform to characterize the impact of deferred operations.

The energy measurement capabilities are demonstrated through application to network data transport where a data file is transported over a network link. The

data compression scheme is dynamically selected using real-time energy measurements during transport of the data file to enable adaptation to the dynamic system and network conditions. The energy cost of transporting the data file is significantly reduced through the use of this energy aware compression algorithm.

A novel task scheduler is presented and is designed to improve energy efficiency of multiprocessing platforms. It utilizes real-time energy measurements along with CPU performance monitoring units to identify inefficient tasks that suffer from co-run degradation due to resource contention. These inefficient tasks have their scheduling priority modified to avoid contention. Evaluation of the scheduler demonstrates large energy and execution time benefits on a quad-core platform.

The dissertation of Digvijay Singh is approved.

Gregory J. Pottie

Mani B. Srivastava

Lixia Zhang

William J. Kaiser, Committee Chair

University of California, Los Angeles

2014

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Foremost, I would like to express my deepest gratitude towards Dr. William J. Kaiser for his guidance throughout my graduate education. He has been more than a graduate adviser by helping my developments beyond a researcher with his constant support of all my endeavors from powerlifting to teaching and mentoring other students.

Also, I want to thank my committee members Dr. Gregory J. Pottie, Dr. Mani B. Srivastava and Dr. Lixia Zhang for their excellent graduate courses, which I was fortunate enough to attend, and their feedback throughout the important phases of my graduate research that has finally culminated in this dissertation. Furthermore, our close collaborators, namely Dr. Peter Peterson and Dr. Peter Reiher, have been very encouraging of this research and their inputs have been invaluable.

Finally, words cannot even begin to express the immense amount of support I have received from my family and my lovely wife Ki Young through all the tribulations of my doctoral research. They have made this possible.

VITA

2003 - 2007	Undergraduate study at IIT, Kharagpur
	Best Computer Science Undergraduate Thesis
2007	B. Tech. in Computer Science and Engineering
	Graduated with Honors
2007 - 2008	Graduate study at IIT, Kharagpur
	Teaching Assistant, Operating Systems and Digital Circuits
2008	M. Tech. in Information Technology
2008 - 2014	Graduate study at UCLA
	Graduate Student Researcher
2010	M.S. in Electrical Engineering
	Qualcomm Fellowship Recipient

PUBLICATIONS

D. Singh and W. Kaiser, "Energy efficient task scheduling on a multi-core

platform using real-time energy measurements." Accepted: International Symposium on Low Power Electronics and Design (ISLPED), 2014.

B. Spiegel, M. Kaneshiro, M. Russell, A. Lin, A. Patel, V. Tashjian, V. Zegarski, D. Singh, S.Cohen, M. Reid, C. Whitman, J. Talley, B. Martinez, and W. Kaiser, "Validation of an acoustic gastrointestinal surveillance biosensor for postoperative ileus." Journal of Gastrointestinal Surgery, 2014.

M. Kaneshiro, W. Kaiser, M. Russell, A. Patel, V. Tashjian, V. Zegarski, D. Singh, S. Cohen, M. Reid, C. Whitman, J. Talley, B. Martinez, and B. Spiegel, "Characterizing gastrointestinal (GI) motility with a computer-aided, non-invasive acoustic sensor: proof-of-concept testing in normal controls vs. postoperative patients." Digestive Disease Week (DDW), 2014.

J. Yan, C. Lonappan, A. Vajid, D. Singh, and W. Kaiser, "Accurate and low-overhead process-level energy estimation for modern hard disk drives." IEEE International Conference on Green Computing and Communications (GreenCom), 2013.

D. Singh and W. Kaiser, "Energy efficient network data transport through adaptive compression using the DEEP platforms." IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012.

P. Peterson, D. Singh, W. Kaiser, and P. Reiher, "Investigating energy and security trade-offs in the classroom with the Atom LEAP testbed." USENIX Cyber Security Experimentation and Test (CSET), 2011.

X. Xu, D. Singh, M. Batalin, and W. Kaiser, "StepFit: a novel fitness evaluation system." International Conference on Body Area Networks (BodyNets), 2011.

D. Singh and W. Kaiser, "The Atom LEAP platform for energy-efficient embedded computing." Report, University of California, Los Angeles, 2010.

D. Singh, S. Soundararaj, S. Kundu, and A. Pal, "Low-power microcontroller for wireless sensor networks." IEEE Region 10 Conference (TENCON), 2009.

CHAPTER 1

INTRODUCTION

The energy consumption of computing and communication equipment is increasing at a staggering rate and the resulting costs have become critical to both end-users and corporations alike [Sca06]. For end-users, the introduction and proliferation of a number of personal computing devices, like smartphones, in the last decade has further accelerated the rising energy consumption due to computing equipment. A standard smartphone is now expected to consume the same amount of energy as a refrigerator and can contribute to a significant portion of household's electricity consumption [Phil13].

Large companies and corporations also face urgent problems with the constantly increasing energy costs of data center servers as illustrated in Figure 1.1 [Sca06]. Figure 1.2 shows that the annual worldwide energy cost of only cloud computing has exceeded that of countries like Germany and India in 2013 [Phil13].

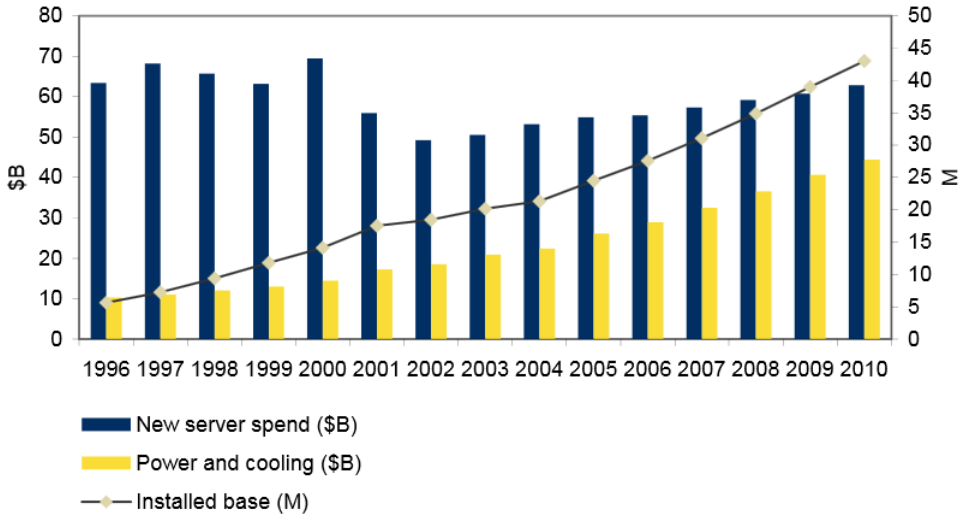

Figure 1.1. Rapidly increasing energy costs of data center server platforms.

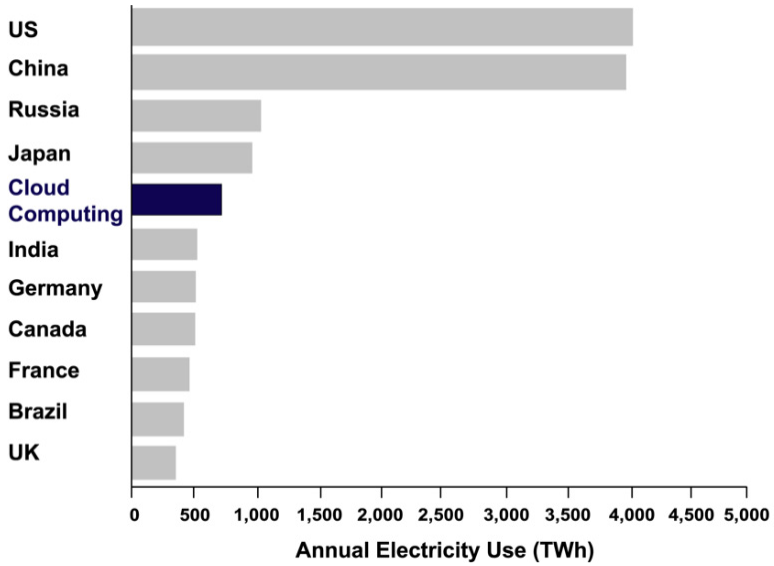

Figure 1.2. Cloud computing energy usage compared to that of various countries.

As illustrated in Figure 1.3, the energy costs associated with computing equipment encompass multiple modes of energy usage and a typical data center server dissipates energy due to the following [Ras09]:

1) Heating, ventilation and air conditioning (HVAC).
2) Computing, network and storage energy consumption.
3) Lighting and auxiliary equipment.
4) Power supply losses.

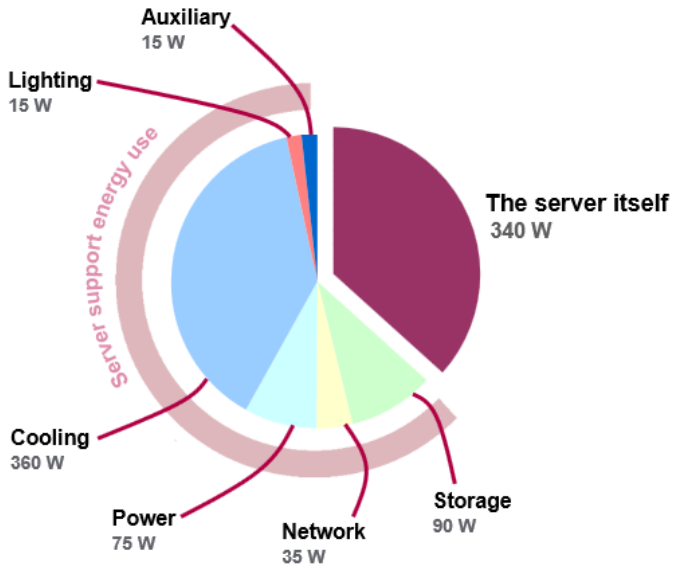

Figure 1.3. The modes of energy consumption due to a typical data center computing platform.

For embedded or mobile platforms, like smartphones, the HVAC or cooling system itself is usually not as important. Instead, batteries and other energy storage techniques become vital for these platforms [PFW11].

This dissertation primarily focuses on the computing, networking and storage energy consumption for modern computing platforms. The design and implementation of a new platform architecture called DEEP (Decision-support for Energy Efficient Processing) to measure and enable optimization of platform energy consumption is detailed. Furthermore, the capabilities of the DEEP platforms are utilized in the important applications of network data transport and operating system task scheduling to improve energy efficiency.

1.1. DEEP Platforms

To provide effective decision support for improved energy efficiency in modern computing and networking, the DEEP platforms are presented. DEEP provides decision support through high-resolution time-synchronized direct energy measurement capabilities for standard computing platforms. Energy measurements for important system components, such as CPU, memory, hard

disk drive and others, are also provided by DEEP. DEEP is an open design that can be rapidly deployed using standard hardware and widely-supported software components.

DEEP includes an innovative utility called *energy calipers* that utilizes time-synchronized energy measurement and kprobes [MPK06] to estimate the energy consumption associated with execution of sections of software application code. Energy calipers have a low-overhead, scalable and non-intrusive design. They do not require any invasive modifications to the algorithm or software source code.

An implementation of the DEEP architecture that uses commodity hardware and software is also presented. Evaluation reveals low processing and energy overheads (less than 5% in most cases) on the computing platform. To enable utilization of the DEEP platforms by other research and student groups, a testbed consisting of multiple DEEP implementations has been created. The testbed has been used in graduate research, collaborations with other research groups and student education through both graduate and undergraduate courses,. The DEEP platforms and the DEEP testbed are detailed in Chapter 2.

1.2. Hard Disk Drive Energy Estimation

Modern hard disk drives and operating systems frequently employ buffering mechanisms to defer data write operations to the secondary storage device to enable re-scheduling and optimization of these operations. This leads to an important issue because the energy consumption of these devices is not synchronized with the actual operating system data write requests generated by software applications or processes. Thus, the energy consumption for these deferred operations must be modeled and appropriately attributed to the causative processes to improve the accuracy of application energy consumption measurement. This is especially important for applications that perform a large number of data write operations to secondary storage. Such an energy estimation system for a modern hard disk drive is presented in Chapter 3. The system implemented using the DEEP platforms and enables process-level accounting of disk drive energy consumption.

1.3. Energy Efficient Adaptive Data Compression

The capabilities of the DEEP platforms are demonstrated through their application to the problem of network data transport. A data file is transported over a network

link and the energy efficiency of the file transfer requires improvement. Data compression is applied as a solution to this problem to reduce the payload being transferred and thus reduce energy costs. Energy measurement using DEEP reveals the impact of changing network and system conditions on the energy efficiency of some widely-utilized compression schemes.

An adaptive data compression algorithm is developed. This algorithm uses the high-resolution time-synchronized energy measurements from DEEP to dynamically select the most energy efficient compression schemes. The compression scheme selection is adapted to changing network and system conditions during the transfer of the data file. Evaluation of the adaptive compression algorithm reveals large energy savings (about 38%) for network data transport. The adaptive data compression algorithm and its implementation are detailed in Chapter 4.

1.4. Energy Aware Task Scheduling

Operating system task scheduling is a critical area of research that has a well-known and significant effect on the energy efficiency and performance of the

computing platform. In addition, other important attributes such as response time, task deadlines, fairness in allocation of resources and interactivity are also impacted by the task scheduler.

An energy aware task scheduler is presented in Chapter 5. DEEP's energy measurements and data from the CPU performance measurement unit are used to identify tasks that create inefficiencies due to resource contention with other tasks. Such tasks are prevented from being scheduled together and tasks that are more efficiently co-scheduled together are selected for execution. This drastically reduces co-run degradation among tasks and significantly improves energy efficiency.

The energy aware task scheduler is implemented on a standard Linux kernel and is compared to the Linux task scheduler using a set of common benchmark applications. Comparison on a quad-core multiprocessing system demonstrates improvements in both performance by about 24% and energy efficiency by about 30%.

1.5. Summary of Contributions

The novel contributions of this dissertation are summarized as follows:-

1) Design of the DEEP platforms, which provide infrastructure for time-synchronized component-level energy measurement in commodity computing platforms.
2) A set of utilities called energy calipers to enable measurement of energy consumption during execution of sections of software or application code without modifications to the source.
3) Implementation of multiple platforms based on the DEEP design to form a networked testbed that has been utilized for both research and student education.
4) Design and implementation of a hard disk drive energy estimation system that complements DEEP to enable accurate process-level estimation of energy usage due to deferred secondary storage operations.
5) An adaptive data compression algorithm that uses energy measurements provided by DEEP to dynamically select compression schemes to adapt to changing network conditions, such as available data transfer bandwidth, and

leads to significant improvement in the energy efficiency of network data transport.

6) An energy aware task scheduler that leverages time-synchronized energy measurements along with the data from the CPU performance measurement unit to advance the energy efficiency of multi-core computing platforms.

CHAPTER 2

DEEP PLATFORMS

This chapters details the DEEP (Decision Support for Energy Efficient Processing) platforms. These platforms enable improvements in energy efficiency for computing platforms by providing system and subsystem energy measurements. This chapter begins with Section 2.1, which overviews previous work related to energy measurement for computing platforms. Section 2.2 summarizes the objectives that have guided the design of the DEEP platforms. Section 2.3 presents the hardware architecture of the DEEP platforms. Section 2.4 details the DEEP platform software architecture. Section 2.5 describes an implementation of the DEEP architecture, called DEEP Atom, that is based on an Intel Atom platform and overviews the DEEP testbed along with its applications in education and research. Section 2.6. presents results of characterization and evaluation of DEEP. Section 2.7 concludes this chapter. A portion of the material presented in this chapter has been published previously [SK12].

2.1. Background and Related Work

Since platform energy monitoring is required to enable effective optimization of energy consumption, a significant amount of the previous research exists on estimating the energy consumption of computing platforms. Each of these works can be categorized by their use of energy models, direct measurement or a combination of both of these methods in the design of their energy monitoring framework.

2.1.1. Energy Models

Model-based energy measurement refers to methods that estimate the energy consumption of computing systems through construction of energy models. Linear state-based models are widely utilized [RSR07] and assume the existence of a relatively small number of power consumption states for a computing platform. Such models have degraded accuracy compared to direct energy measurements, but are utilized in prior research for applications like thermal management [MB05] that don't require extreme accuracy of estimation. Performance counter based energy models are an example where the accuracy of estimation is improved through the use of models based on hardware registers

called performance counters [IM03], [RSM09] that are part of a modern CPU's performance monitoring unit. Intricate non-linear models and machine learning techniques, like Mantis [ERK06] that learns the energy-consumption states of computing systems by utilizing different workloads, have also been utilized in prior research. They are computationally expensive and require extensive training [KOI10], but still demonstrate limited accuracy improvements over simpler models [MAC11].

Battery-based energy measurement systems have also been successfully deployed in prior research. They are based on queries to the ACPI interface of a device's "smart battery," which can periodically report rudimentary statistics such as the current capacity and drain rate. These measurement systems are feasible sources of information for many mobile devices such as smartphones [RSI08]. Applications, such as PowerTOP [Gra08], have utilized the ACPI battery interface and other system statistics to estimate the energy consumption of devices. However, due to inherent limitations in the ACPI battery interface, these techniques suffer from degraded accuracy and extremely limited data sampling rate.

Prior systems, like Sesame [DZ11] and WattProbe [Pra03], utilize hybrid methods where energy measurements (such as from battery monitors) are combined with energy models to improve measurement accuracy. ECOSystem [ZEL02] augments the Linux operating system to consider energy as a first-order resource. PowerScope [FS99] is a hybrid system that utilizes statistical profile-based models for software in combination with limited sampling rate instrumentation.

Simulation based energy estimation frameworks, like SimplePower [YVK00], Wattch [BTM00] and DRAMsim [RCJ05], are also widely utilized in prior research because they enable rapid and effortless deployment. Their critical drawback is the inability to completely capture the run-time dynamism and variability present in the power consumption of modern computing systems [MAC11], [WAB10].

2.1.2. Direct Measurement

Direct energy measurement platforms, which have provisions for the direct measurement of energy consumed by a computing platforms and its subsystems, offers an important advantage over model-based energy estimation methods

because these platforms provide superior measurement accuracy. They also capture the run-time dynamism and variability present in the energy consumption characteristics of modern computing systems. They are also required in constructing and evaluating energy models since they provide the "ground-truth" about platform energy consumption. However, the need for custom-built hardware and the assembly expertise has inhibited their availability [MAC11].

A limited number of direct measurement studies are presented in prior work and they explore the energy consumption characteristics of computing systems from servers [RSM09] to smartphones [BBV09], [CH10]. The LEAP (Low-Power Energy Aware Processing) platforms are a set of direct energy measurement systems that have assisted energy-efficiency focused computing research [MHY06], [SMK08]. The LEAP systems are designed to be extremely integrated custom-designed platforms targeted specifically at wireless sensor network research. This diminishes their portability and prevents their use in decision support for general-purpose computing platforms and applications. Finally, promising new technologies integrating power measurement capabilities with the computing system architecture have emerged in recent years [NRA11].

2.2. Design Objectives

This section describes the important objectives that guide the design of the DEEP platform architecture. Foremost, the DEEP architecture includes provisions for *component-resolved measurement* of energy consumption. Thus, the platform measures the energy consumption for important hardware subsystems, including memory modules, secondary storage devices, and the CPU. The energy consumption data for individual subsystems of a computing system assists with component-resolved energy inspection, fine-grained power management through better guidance in control of each subsystem and opportunities for improved energy efficiency through utilization of component-resolved decision support [MAC11].

Measuring energy consumption associated with the execution of software events on the computing system enables the inspection and improvement of energy efficiency of software applications. The DEEP architecture provides *event-resolved measurement* that is readily applicable to the inspection of the energy consumption during execution of software application code or specific events in the operating system.

Platform architectures designed on custom-built hardware, such as the LEAP platforms [MHY06], are not widely-supported and are unable to explore the energy efficiency of a large space of applications that are developed for commodity hardware. Hence, the ability to inspect the energy consumption of a range of commodity computing platforms is important. Furthermore, modern IT equipment serves in applications beyond simple computing and networking. For example, smartphones are utilized in medical sensing applications through extension of the core computing device by addition of sensing hardware. Platforms designed with custom-built hardware and proprietary software result in diminished extensibility due to the need for a prohibitive amount of effort in making even simple extensions. To ensure accomplishment of this objective, DEEP is not designed using custom-built computing hardware, but instead is designed and implemented using standard commodity computing platforms.

Large-scale adoption of direct energy measurement platforms has not been possible because most prior platforms either don't have an open-source design, are tedious to assemble, or employ custom-built components [MAC11]. In contrast, DEEP presents an open-source design based on commodity hardware,

readily-available measurement instrumentation, widely-supported open-source software, and straight-forward assembly to create a platform that can be *rapidly deployed* by members of the research community.

2.3. Hardware Architecture

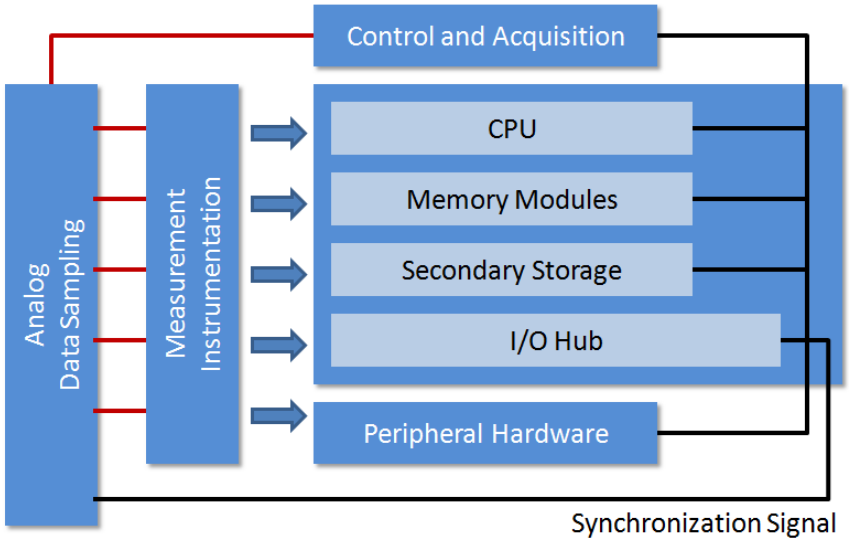

Figure 2.1. The hardware architecture of the DEEP platforms.

The hardware architecture for the DEEP platforms is illustrated in Figure 2.1. The computing platform executes all software applications and networking subroutines, and contains important computing subsystems such as the processor, memory modules and secondary storage units. Rather than targeting a specific

platform or system architecture, an abstraction of a standard computing system is used in the DEEP platform architecture to support an open design, and to preserve portability of the platform design. The DEEP platform architecture is based on standard widely-supported computing hardware rather than custom computing or networking boards with non-standard interfaces. Thus, the platform architecture supports rapid extension of the core design, through addition of peripheral hardware, like sensors, actuators and wireless radios, to meet requirements for a wide range of applications.

2.3.1. Energy Measurement and Data Acquisition Hardware

Low-tolerance current sensing resistors are inserted in each of the input power supply lines of the subsystems of the platform. This current measurement instrumentation is used for component-resolved energy measurement. The instantaneous electric current flowing into a subsystem or component is computed using the voltage difference across the terminals of the respective sense resistor along with its resistance value. Instantaneous power consumption is computed using the obtained instantaneous electric current and supply voltage value for each subsystem.

The data acquisition unit (DAQ) acquires power measurements by periodically sampling the required voltage values from the terminals of the resistor-based energy measurement instrumentation added to the platform hardware. This unit then transfers the acquired data to the computing system for calculation of energy consumption. The DAQ is a separate unit and is not integrated with the rest of the computing hardware in the DEEP platforms.

2.3.2. Timing Mechanism and Synchronization Signal

To enable event-resolved energy measurement, the platform needs to synchronize the timing of software events with that of energy measurements acquired by the DAQ. To accomplish this, a high-resolution timing mechanism is required to time-stamp the occurrence of both software events and energy measurements. Most modern CPUs provide access to hardware timing registers that are utilized as the timing mechanism required for synchronization. The most commonly available high-resolution timing mechanism in commodity hardware is the time-stamp counter (TSC) for modern Intel x86 and AMD K-series CPUs, and the clock-cycle counter for ARM platforms.

To complete the synchronization of energy measurements with software events, a synchronization signal is used in conjunction with the timing mechanism to assign time-stamp values to energy measurements. The synchronization signal is a hardware signal generated by the computing system and sampled by the DAQ along with the component-resolved power measurements. This enables the synchronization of the DAQ data samples with the time-stamps of software events occurring on the system. This is detailed further in the next section.

2.4. Software Architecture

The software architecture of the DEEP platforms provides, in addition to acquisition of the component-resolved energy measurements, an important advance by supporting event-resolved energy measurement. The DEEP platforms are based on the Linux operating system in support of an open design and the software architecture is implemented as a set of loadable Linux kernel modules to enable portability of the modules to other Linux-based computing platforms. The details of the software architecture are presented in the following subsections and are based on Linux kernel version 2.6.32.

2.4.1. Data Acquisition and Synchronization

The DAQ hardware is controlled by software modules that execute on the computing platform. A number of parameters, such as data sampling frequency, are controlled by these modules. The data acquisition and control modules also facilitate the acquisition of data samples containing the energy measurements for each of the hardware subsystems.

The synchronization software module, by using the synchronization signal and timing mechanism, performs the time-synchronization of energy measurements obtained from the DAQ unit. Synchronization is accomplished when the synchronization module changes the value asserted on the synchronization signal while simultaneously recording the time-stamp provided by the timing mechanism. The DAQ acquires the energy consumption measurements along with the value asserted by the synchronization signal. During processing of the obtained data samples when a change in the synchronization signal's value is detected, the corresponding recorded time-stamp value is assigned to the data sample at which the change is detected. This completes the synchronization of the obtained data samples with the corresponding TSC values.

2.4.2. Energy Calipers

Energy calipers are innovative software utilities that present an advance in investigating the energy efficiency of software applications. They use the time-synchronized energy measurements to estimate the energy consumption associated with execution of sections of program code. Energy calipers are based on Kprobes [MPK06] and dprobes [Bha03], which are standard run-time debugging mechanisms for kernel and user-space code in the Linux operating system. Thus, energy calipers can be inserted at any instruction in both kernel and user-space application code during run-time without any need for recompilation, modification or even availability of the application's source code.

A pair of *start* and *end energy calipers* consists of a pair of Kprobes or dprobes that are inserted at the beginning and concluding instructions of a section of kernel or user-space code that is to be monitored for energy measurement. Each of these Kprobes or dprobes records the process ID (PID) and time-stamp value, which is provided by the timing mechanism, corresponding to the instant they are executed on the computing system. This creates a record of the time-stamp values that identify the beginning and completion of execution of the target section of

code. Energy measurements that have time-stamps outside the recorded interval of execution are excluded. The remaining measurements are utilized to compute the energy consumed during execution of the target section of code. Energy calipers can be multi-instantiated and can overlap i.e. multiple pairs of calipers may be used to simultaneously measure the energy consumption during execution of different, and possibly overlapping, sections of code.

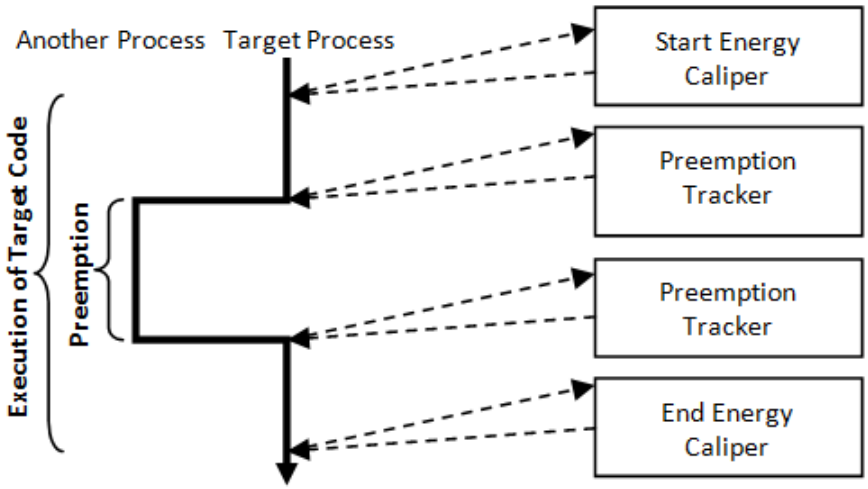

Figure 2.2. Energy caliper usage example through insertion at target code section.

Energy calipers can also be utilized in cases where the target section of code may be preempted before it has completed execution as illustrated in Figure 2.2. The

operating system scheduler's context-switch function is instrumented, without any intrusive kernel modification, with an energy caliper called the *preemption tracker*. When an executing section of code is pre-empted or resumed by a call to the scheduler's context-switch function, it results in the recording of time-stamp and PID values indicating a process' preemption and resumption. Along with the target process' PID reported by the start and end energy calipers, the time-stamps are utilized to compute the energy consumption that occurs during execution of the target code.

2.5. Implementation

The DEEP architecture can be implemented on many standard computing platforms and the ease of assembly enables researchers to rapidly deploy their own DEEP platforms without special assistance or expertise. This section presents an example implementation of the DEEP architecture using an Intel Atom computing platform along with assembly details. A detailed evaluation that characterizes the time synchronization and overhead of the platform implementation is also included in this section. An alternate offline version of the implementation is also detailed for applications that require reduced overhead

2.5.1. DEEP Atom

The DEEP Atom implementation is based on the Intel Atom N330 CPU. The Atom CPU is designed for deployment in portable devices such as netbooks, mobile devices and tablet computer systems. This implementation of the DEEP platform architecture uses the standard Intel D945GCLF2 Atom board. The DEEP Atom was also previously referred to as Atom LEAP [SPR10], but was renamed in 2012 as DEEP Atom to avoid confusion with older LEAP projects [MHY06].

Figure 2.3. SATA hard drive with instrumented power supply cable containing a 0.1Ω current sensing resistor.

Energy measurement instrumentation for the computing subsystems is constructed as a wiring harness that can be rapidly deployed using any off-the-shelf current sensing resistors of resistance values between 0.05Ω and 0.1Ω. The hard-drive is instrumented through insertion of a current sensing resistor into the exposed SATA power cable as shown in Figure 2.3. The memory module utilizes a riser-card with a current sensing resistor and this riser-card is inserted between the Atom board's memory slot and the memory module as illustrated in Figure 2.4.

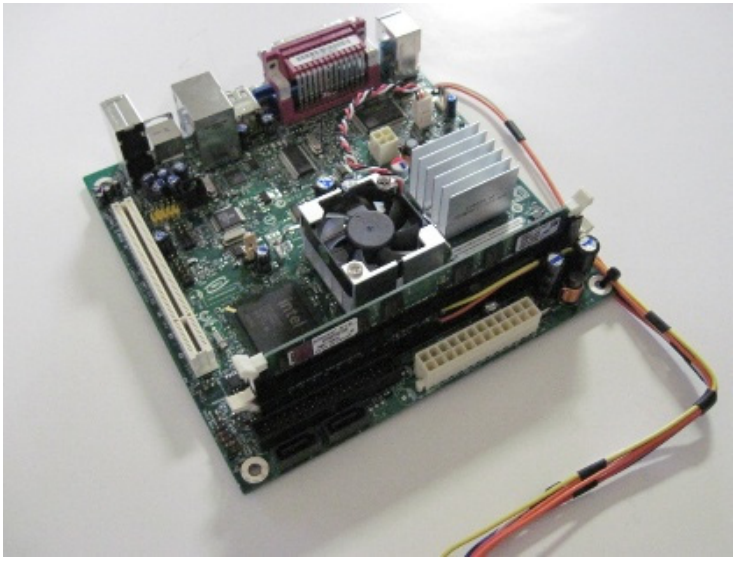

Figure 2.4. The Atom board with the memory module in the foreground. A riser card with a current sensing resistor attaches the memory module to the board.

Figure 2.5. Atom motherboard with attention to the regulator circuit's inductor.

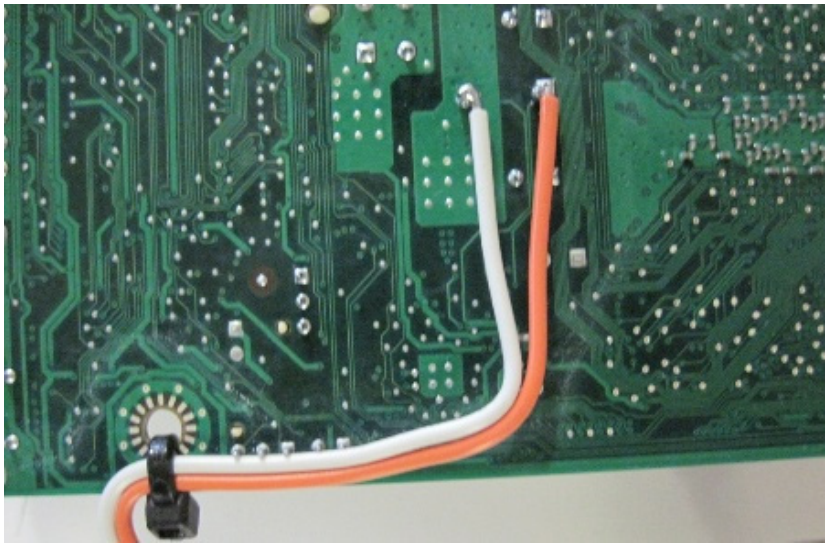

Figure 2.6. Leads enabling use of the inductor's resistance for current sensing.

Modern CPUs enable the CPU core to control its input supply voltage. The CPU accomplishes this through a voltage regulator circuit that provides the CPU with the requested supply voltage. Since there is no exposed power cable for the Atom N330 on the D945GCLF2 motherboard, the voltage regulator circuit on the board is utilized for energy measurements. A current sensing resistor can be placed in series with the supply from the regulator circuit to the CPU. This can be accomplished by making modifications to the terminals of the exposed inductor that is part of such voltage regulator circuits on commodity motherboards. This method requires some minor external modification to the motherboard's circuitry and makes the instrumentation process slightly tedious. Thus, alternatively the inductor's resistance is used directly as a current sensing element as shown in Figure 2.5 and Figure 2.6.

Once the measurement instrumentation is assembled, the leads from the terminals of the current sensing elements for each instrumented subsystem are connected to the input channels of the Data Acquisition unit (DAQ). A National Instruments USB-6215 DAQ is utilized in this implementation and it interfaces to the computing system through one of the available USB ports.

The Atom N330 processor provides a 64-bit hardware register called the time-stamp counter (TSC) that is a high-resolution clock with a resolution of a single processor cycle. The motherboard also provides a serial and parallel port. The TSC is utilized as the timing mechanism and the parallel or serial port is used to generate the synchronization signal needed by the DEEP architecture for synchronization of energy measurements with operating system events. The synchronization signal is connected to an input channel of the DAQ and this signal's value is sampled along with the values from the energy measurement instrumentation.

The DEEP Atom implementation is extensible through addition of peripheral hardware. Standard I/O ports, like USB, are used to add peripherals like wireless radios and sensors that enable the DEEP implementation to meet the requirements of different applications. An inexpensive interface cable, such as a USB cable, is instrumented with a current sensing resistor and used for peripherals powered by the I/O ports. For an externally powered peripheral the external power supply cable is directly instrumented with current sense resistors using a technique similar to the one used for the SATA hard drive previously shown in Figure 2.3.

The hardware for this implementation is rapidly deployed from the constituent components and provides decision-support for portable/netbook-class computing systems or networking equipment like high-end routers. The open design, use of commodity hardware and relatively straightforward assembly instructions [SPR10] encourage large-scale adoption of the DEEP platforms, and make the DEEP Atom implementation deployable by research groups and students.

The DEEP architecture is also implemented as an *offline version* that is ideal for applications that require a reduced overhead. In contrast to the *online version* previously presented in this section, the target computing system contains only the measurement instrumentation, energy caliper data recording and synchronization signal generation while the data sampling and energy caliper computations are performed off-board by another system which controls the DAQ. This reduces both the processing and energy overhead, but requires an external device for data collection from the DAQ, synchronization and energy computation using the energy caliper reports. The delay in obtaining measurements is increased and this version of the implementation is not designed for online power management, but for post-experimental analysis of the energy measurements.

2.5.2. Implementation Details

The DAQ is programmed with a number of different parameters. Among these parameters, the data sampling frequency is most important. High sampling frequencies allow accurate measurement of the power consumption for a subsystem, but generate an increased amount of data. Thus, the sampling frequency is selected such that it can capture the power supply signal for a subsystem without degradation in accuracy while ensuring that the amount of data generated is within manageable limits.

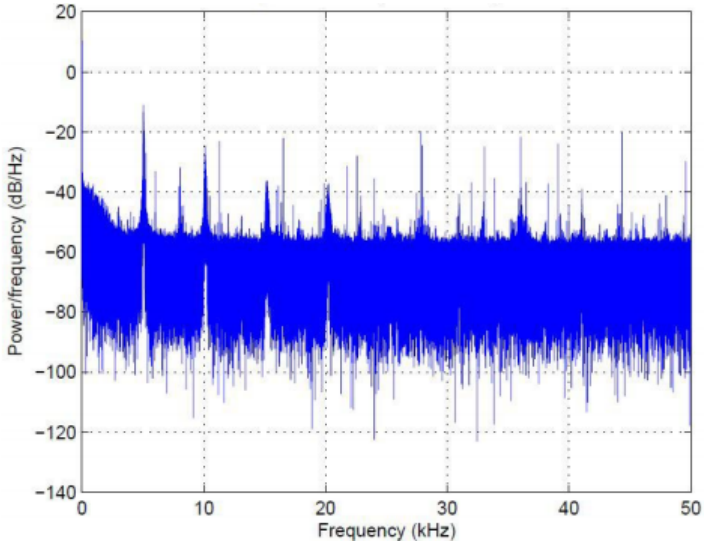

Figure 2.7. Power spectral density of the power supply signal for the Atom board.

The power spectral density analysis of the power supply signals of the subsystems for the Atom N330 computing system reveal a very small power contribution (≤5%) from very high frequency (≥5kHz) components as shown in Figure 2.7. This leads to selection of the optimal sampling frequency as 10kHz so that the signal is sampled at the Nyquist rate for capturing most of the signal's power (>95%). Thus, a 100μs sampling resolution or a 10kHz sampling frequency is sufficient for the DAQ of the DEEP Atom.

To verify the accuracy of the time-stamps provided by the synchronization scheme for each data sample, an event workload is created by the operating system such that it has an immediate impact on the power consumption of a subsystem. The time-stamp value for this event is recorded by the operating system. This value is then matched to time-stamps predicted by the synchronization scheme. The index of the data sample with the closest match to the event's time-stamp is selected as the *predicted index*. This value identifies the data sample at which the synchronization scheme predicts that the operating system event began. The *measured index* is derived as the sample index at which a change in the power consumption of the subsystem caused by the event is

observed. This value identifies the sample at which the operating system event began. Predicted and measured indices are compared to derive the *index error*.

```
Data  # Time sequential array of acquired rows or samples of data
TSC[] # Queue of recorded time-stamps for each 'Sync' toggle
C[]   # Queue of time-stamp off-sets
K[]   # Queue of synchronization scale factors
k = 0 # Current scale factor
t = 0 # Previous time-stamp off-set value
T = 0 # Current time-stamp off-set value
N = 0 # Sample index number of previous 'Sync' toggle
S = 0 # 'Sync' signal toggles between positive & non-positive

for each sample in Data             # SI is sample index here
      Sample = Data[SI]             # Get next data sample
      if ( S*Sample['Sync'] <= 0 )  # Detect toggle in 'Sync'
            T = TSC.pop()           # Time-stamp for the toggle
            C.push( T )
if ( t > 0 )                        # Toggle detected before?
      K.push( ( T-t )/( SI-N ) )
      N = SI
      t = T
      S = Sample['Sync']

S = 0; T = 0; N = 0;
for each sample in Data             # SI is sample index here
      Sample = Data[SI]             # Get next data sample
      if ( S*Sample['Sync'] <= 0 )  # Detect toggle in 'Sync'
            if ( K.isEmpty() )
                  break
            T = C.pop()
            k = K.pop()
            N = SI

      S = Sample['Sync']
      Sample['TSC'] = T + k*( SI-N )# Predicted time-stamp
      Data[SI] = Sample             # Write back time-stamp
```

Figure 2.8. Pseudocode for implementing time-synchronization of energy data.

The implementation of the synchronization algorithm is illustrated using pseudocode in Figure 2.8. The synchronization between the data samples and the time keeping mechanism can drift over time, and this can result in degraded accuracy. To limit this drift, the synchronization module needs to periodically change the value of the synchronization signal to cause re-synchronization. The value of the time period after which this process is repeated is called the *re-synchronization interval* and is determined by the observed drift in the synchronization. As illustrated in Table 2.1, the synchronization is accurate to within one sample's resolution (100μs) and does not drift noticeably for up to a few seconds. Thus, an effective value for the re-synchronization interval is one second and this value is used in the DEEP Atom implementation presented in this chapter.

Table 2.1. Characterization of DEEP Atom synchronization accuracy and drift.

Event Time (s)	**Event Time Stamp**	**Measured Index**	**Predicted Index**	**Index Error**
2.2	27181544764	21769	21769	0
7.4	35473163152	73721	73721	0
16.6	50137785508	165604	165603	1

2.5.3. DEEP Testbed

To enable rapid access to the DEEP platforms, a networked testbed consisting of multiple DEEP Atom implementations and a DEEP x86-64 server-class implementation has been created. This testbed is accessible to students and research groups that are interested in using the DEEP platforms, but lack the resources to assemble their own platforms.

Multiple courses in computer science and electrical engineering (EE180D, CS188 and EE202C) have employed the testbed for education and research. Some of the results pertaining to the trade-offs between energy usage and security from course projects have been published previously [PSK11], [FPR12]. The DEEP testbed has even been used by our collaborator Dr. Peter Peterson during research for his doctoral dissertation concerning the use of adaptive data compression [Pet13].

2.6. Results

This section presents results of evaluation and characterization of the DEEP platform implementation. In particular, the DEEP Atom implementation is analyzed.

2.6.1. Evaluation of Overhead

To evaluate the overhead of the DEEP implementations, two metrics are used: 1) execution time and 2) energy consumption. The execution times for a set of common benchmark applications are measured both with and without the DEEP current sensing instrumentation, synchronization and energy caliper system. The execution times for both cases are compared to determine the overhead introduced by the DEEP platform architecture.

For energy overhead measurement the energy consumption by the offline implementation without synchronization and energy calipers is compared to the energy for complete online and offline implementations. The overhead for using multiple pairs of energy calipers to simultaneously monitor multiple sections of application code is evaluated to characterize the scalability of the system.

We analyze both the online and offline versions of the DEEP Atom implementation and the results are illustrated in Figure 2.9. The DEEP Atom demonstrates low overhead and excellent scalability when simultaneously monitoring multiple code sections with energy calipers.

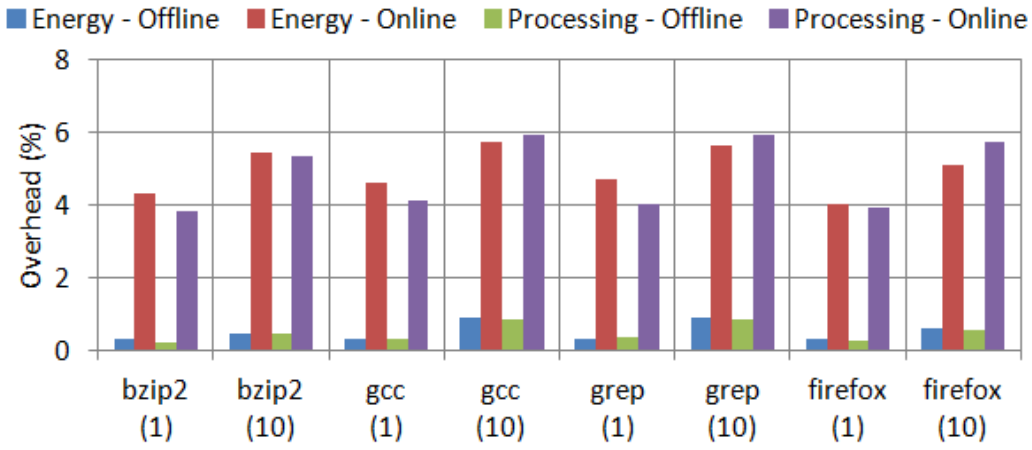

Figure 2.9. Scalability and overhead for the offline and online DEEP Atom. Numbers in parentheses are the number of monitored code sections.

2.6.2. Power Measurement Characterization

To characterize DEEP Atom's power measurement instrumentation's accuracy and precision, an oscilloscope with a current probe and two voltage probes is used. One voltage and current probe is attached to a terminal of the current sensing resistor for the memory module. Thus, both input current and voltage for the memory module are measured by the oscilloscope. The DEEP platform also measures the voltage and current using its own measurement instrumentation. This enables both instruments to calculate the power consumption of the memory

module. DEEP's measurements and the oscilloscope's measurements are synchronized using the synchronization signal generated by DEEP. The DAQ and the second voltage probe of the oscilloscope both measure the value of this signal to enable synchronization of measurements between the two instruments.

The power consumption data for the oscilloscope is used as the baseline or ground truth and the data obtained using DEEP's measurement instrumentation is characterized using this baseline. A million data samples for power consumption for the DAQ are collected and compared to values obtained using the oscilloscope. The results of this comparison are presented in Table 2.2.

Table 2.2. Characterization of DEEP Atom power measurement instrumentation.

Metric	Value
Average Error	0.18 %
Worst-Case Error	0.21 %
Uncertainty Interval	0.01 W

2.7. Conclusions

The DEEP platforms presented in this chapter provide component-resolved direct energy measurements that can be utilized for decision support in important areas

of computing and networking energy efficiency. The rapidly deployed implementation of the platforms using commodity hardware and standard open-source software enables their large-scale adoption by members of the community for their research. DEEP can also measure the energy consumption associated with the execution of software code through utilization of an innovative software utility called energy calipers. This makes the platform ideal for investigating and improving the energy efficiency of software applications.

An implementation of the DEEP platform architecture called DEEP Atom is described. Evaluation of the implementation reveals low processing and energy overhead while demonstrating excellent scalability when using energy calipers to monitor multiple sections of software code for energy consumption. Furthermore, the DEEP Atom's energy measurement instrumentation displays excellent accuracy and precision of measurement.

A networked testbed consisting of multiple DEEP platforms has been created. This enables rapid access to the DEEP platforms for research groups or students

that do not have the resources to implement their own platforms, but still need to measure energy consumption for their applications.

CHAPTER 3

HARD DISK DRIVE ENERGY ESTIMATION

This chapter details the design, implementation and evaluation of the process-level hard disk drive (HDD) energy estimation system for the DEEP platforms presented in the previous chapter. HDD energy estimation is important for applications that perform large amounts of I/O to secondary storage. The chapter begins with an overview of previous work related to energy modeling for modern secondary storage devices in Section 3.1. Section 3.2 presents the architectural design of the HDD energy estimation system. Section 3.2 describes the energy measurement infrastructure used for obtaining HDD energy consumption data. Section 3.4 details the HDD I/O monitoring in the operating system kernel. Section 3.5 presents the HDD energy model created using HDD I/O monitoring. Section 3.6 summarizes the implementation and integration of the HDD estimation system. Section 3.7 demonstrates the effectiveness of HDD energy

estimation system with evaluation results. Section 3.8 concludes this chapter. Some of these results have been published previously [YLV13].

3.1. Background and Related Work

Data storage devices like hard drives create a large portion of the energy consumption in these computing systems. The amount of storage required by modern IT services continues to increase and I/O along with disk storage could account for 30% of the energy consumption in a modern computing platform [RLG08]. By underestimating the storage energy demands due to software, an application developer could considerably negate the efforts of optimized power management algorithms and energy efficient file systems. Thus, accurately determining energy consumed because of an application's processes due to disk I/O operations can lead to improved system energy efficiency.

Measuring and characterizing the energy consumed by modern secondary storage devices and attributing it to the causative processes in the operating system is not a trivial task. This is due to the buffering, rescheduling and optimization of I/O write operations that exist in most modern computing systems. Therefore, actual

write operation of data into the physical sectors of the hard drive and the consequent energy consumption does not occur immediately after the issuing of corresponding write requests in software. Unlike CPU or memory operations that usually occur in tight temporal synchronization with execution of software instructions, the disk drive I/O operations can occur after a non-deterministic delay.

The work presented in this dissertation is based on the GNU/Linux operating system that maintains a page cache in the main memory for fast data access. After issuing a data write request to the HDD, the data resides in the page cache before being divided into multiple I/O requests. The I/O requests are then processed by an I/O scheduler, which reorders and merges the requests to optimize disk operations before dispatching the requests via DMA to the disk drive. In addition, the disk controller also has a disk cache where transferred data resides before finally being written into the physical sectors of the disk drive. Figure 3.1 illustrates the issuing of a write request to the physical sectors of the hard disk and the rise in power consumption indicates when the hard disk drive is active. In this case, the write operations to the disk didn’t start until the application's execution

was close to completion and continued for a significant time interval after the causative program had finished issuing I/O requests and even completed its own execution.

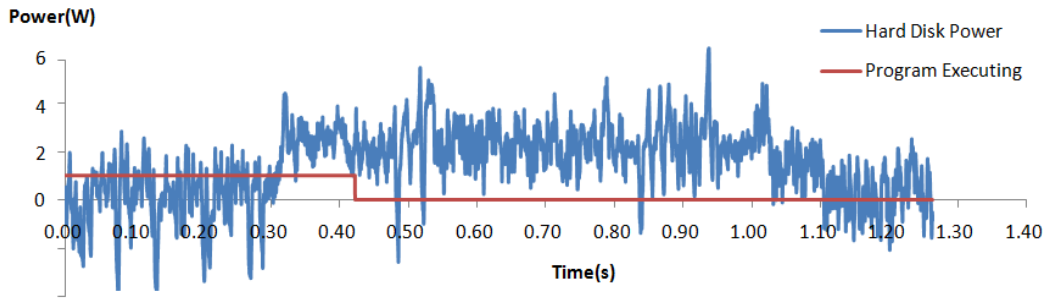

Figure 3.1. The HDD power measurements during execution of an application that issued a data write request.

Previous research on HDD energy modeling has divided the energy consumption of hard disks into various activities [HSR08]. Although the custom built measurement system presented could enable separate measuring of energy for the electrical and mechanical parts of a hard disk drive, it does not attribute the energy measurements to the causative application's processes. Other approaches have focused on modeling the hard drive power consumption [AAF09], [ZSG03] through methods that translate the disk workload to the primitive activities of the

hard disk drive [AAF09] along with simulation-based methods [ZSG03] that use disk energy simulators such as Dempsey to simulate disk I/O operations and power simulators such as DiskSim to read I/O traces to estimate the power consumed by each operation. While these methods were able to achieve superior granularity in their energy models, they still did not attempt to relate the modeled hardware power consumption to causative processes in the operating system.

Power management is also an area of focus in some methodologies [NDR08], [ZDD04] with techniques such as write off-loading, which redirects write requests to active hard disks in a datacenter, and energy aware cache management algorithms that prioritize energy efficiency. A significant amount of work on energy efficient file systems also exists [NF04], [KS92]. Distributed file systems have been employed with the primary goal of achieving superior energy efficiency [NF04]. The Coda file system discusses the feasibility of disconnected operations for portable computers whereby a client can access critical data even during temporary unavailability of shared data repositories via caching to improve performance and possibly energy efficiency [KS92]. These approaches focus on improving energy efficiency in the storage hardware through intelligent power

management, caching and file system design, but do not provide capabilities to determine energy consumption in the disk drive due to a an application's processes.

3.2. Architecture

The HDD energy estimation system's design is based on two critical objectives:-

1) *Construction of a HDD Energy Model*: this has been explored in previous work, but it needs to adapted to the disk drive being used in this DEEP implementation. Furthermore, the model must provide high prediction accuracy while having low overhead.
2) *Mapping of Process ID (PID) to I/O Requests*: the HDD energy model itself can only estimate HDD energy consumption values for given I/O requests, but this energy consumption needs to be attributed to the causative processes in the operating system. The operating system does not maintain the concept of processes at the block device layer of the kernel as shown in Figure 3.2. This leads to obfuscation of the PID and so the PID must be explicitly mapped from the higher Virtual File System (VFS) layer to the lower block device layer's I/O requests.

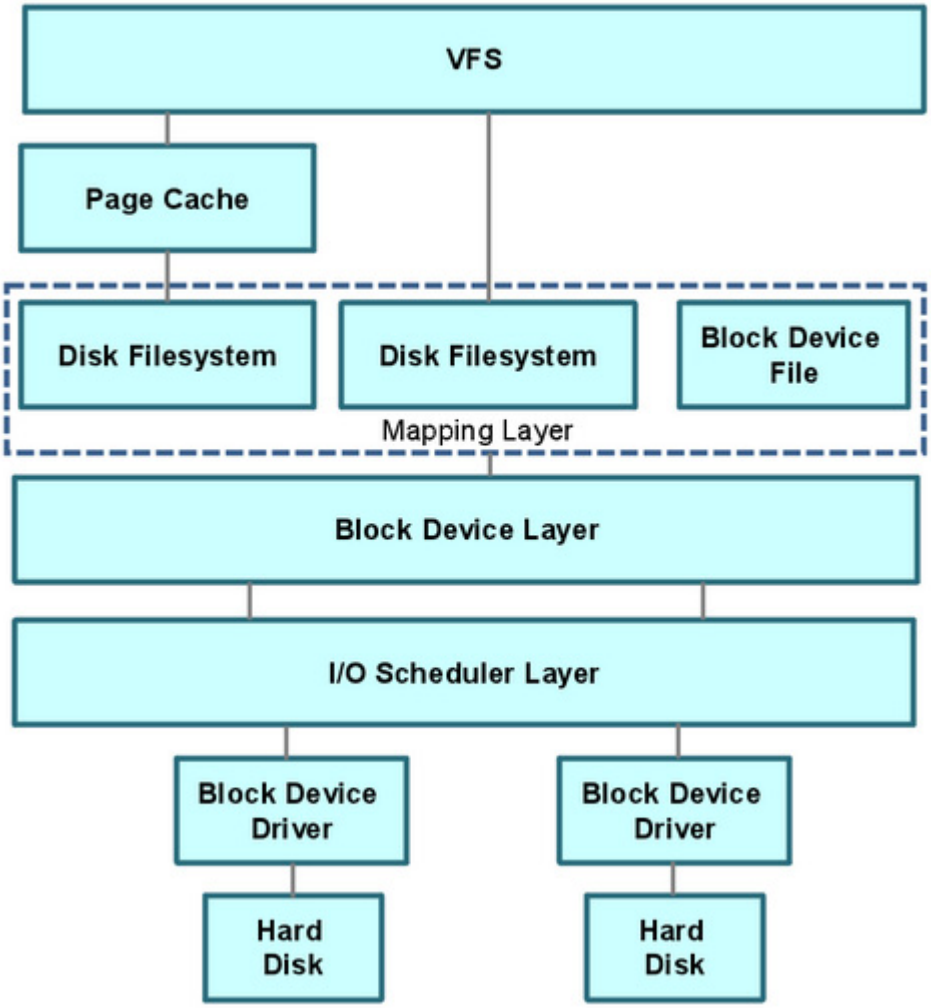

Figure 3.2. The different file system layers that each data I/O request must traverse in Linux (kernel 2.6.31-14) before being issued to the SATA HDD for writing to the physical sectors of the disk.

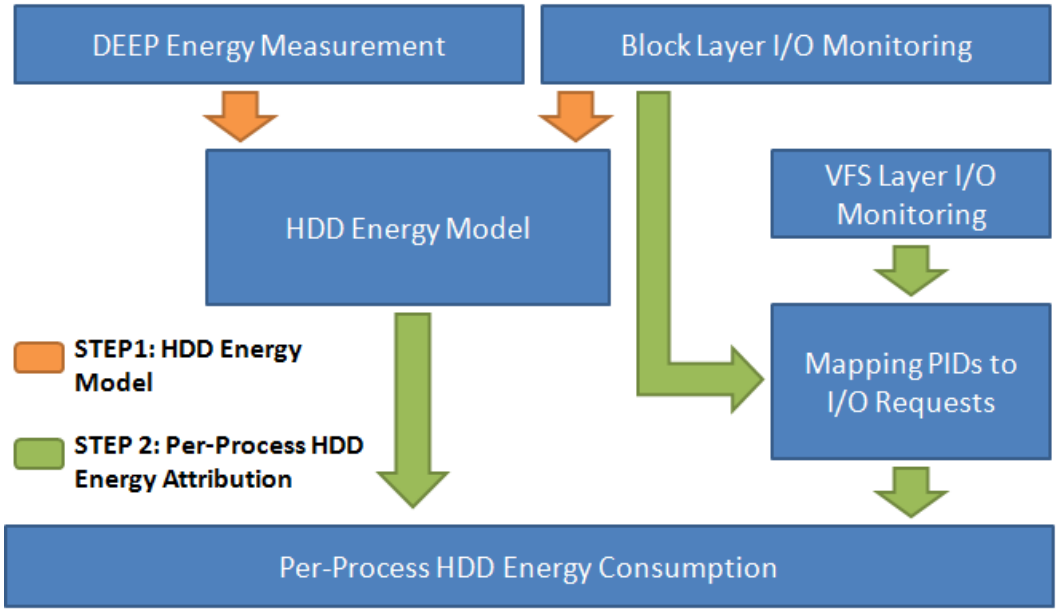

Figure 3.3. The architecture of the HDD Energy Estimation System. The HDD energy model is used with I/O monitoring to enable per-process HDD energy attribution.

The accomplish the previously mentioned design objectives, the HDD energy estimation system's architecture consists of HDD energy measurement, I/O monitoring and the HDD energy model. The HDD energy measurement infrastructure is detailed in the succeeding section and provides energy consumption data for construction of the HDD energy model. The I/O monitoring

is used to monitor the important routines in the kernel to enable both HDD model construction and mapping of the I/O requests to the causative processes. The HDD energy model estimates the energy consumption for an application's HDD I/O requests. The complete system architecture is illustrated in Figure 3.3.

3.3. Energy Measurement

The DEEP Atom, presented in Chapter 2, is based on a computing system with an N330 Intel Atom CPU, SDRAM memory, secondary storage, and network interfaces, which are monitored for power consumption by a high-speed data sampling system. By employing various event synchronization methods, the hardware power dissipation data received from the sampling system is synchronized with the operating system clock and kernel software events. Usage of kprobes, jprobes or dprobes [Bha03], which enable user-defined runtime handler execution at break-points in software code, enable fast instrumentation of the software systems.

The DEEP platform used herein comprises of two separate disk drives: one HDD for the GNU/Linux operating system along with the DEEP software to collect

energy consumption data and the second disk drive is the test HDD for performing the write tests while measuring energy consumption. The test HDD used is a Western Digital Scorpio Blue WD1600BEVS 160GB SATA disk drive that operates at 5,400RPM with an 8MB cache. Isolation of the test HDD from the operating system and energy measurement infrastructure's I/O requests ensure minimal impact on the test HDD when modeling its energy consumption.

DEEP provides a power log at a data sampling rate of 10kHz. Figure 3.4 presents a snippet of a sample power log file. The power measurements for the HDD are collected by the DEEP platform running Linux kernel 2.6.31-14.

CPU	HDD	RAM	Sync	TSC
2.001010	1.615800	1.398510	1.856000	977053835280017.500000
2.120580	1.622380	1.381940	1.857000	977053835439758.250000
2.238990	1.668410	1.357080	1.856000	977053835599499.000000
2.367610	1.655260	1.357080	1.853000	977053835759239.750000
2.484550	1.645400	1.354720	1.853000	977053835918980.500000

Figure 3.4. A snippet of the log of energy measurements provided by DEEP with component-resolved energy data, the synchronization signal values and the TSC values. Only the HDD energy measurements are of concern in this chapter.

3.4. I/O Monitoring

Monitoring of I/O requests is performed through insertion of jprobes or kprobes [MPK06] in the important routines inside the Linux kernel with `printk()` calls in the kprobe handler. Thus, the kernel log is used to report data about I/O requests to the test HDD. This creates a low-overhead method for obtaining a detailed log of data about each of the I/O requests without any invasive modifications to the kernel source code.

3.4.1. Important Kernel Routines

As illustrated in Figure 3.3 previously, a number of important operations take place between a disk I/O request by a process and the actual transfer of the data to physical sectors of the HDD. Figure 3.5 shows a simplified sequence of kernel routines when a user-space process intends to write data to a SATA HDD. Each important step of the sequence is overviewed as follows:-

1) *Before Page Cache*: system call `write()` is issued by a process to create an I/O request, the function `__generic_file_aio_write_nolock()` is invoked, which enables certain flags, performs an I/O request size check and then transfers control to the function `generic_perform_write()` to

update the page cache. Then, the operating system checks if the dirtied page exceeds the allotted limit after copying the data. If not, `write()` returns and the data is not immediately written to the disk. Otherwise, the kernel routine continues by calling `write_page_caches()` to write the pages back to the disk drive. It is important to note that the previous step can greatly impact the time required to write data to the disk drive as the page cache can create a buffering delay before the data is actually passed on the lower file system layers for writing to the HDD.

2) *After Page Cache*: the `generic_writepages()` function checks the list of dirty pages and tries to write them to the block device file representing the disk drive. To communicate with the block device, the operating system kernel uses the `bio` structure to describe each unit of data that is to be written. The `submit_bh()` kernel function initializes these structures, and the `generic_make_request()` function inserts the structures into the request structure in the `request_queue`. Finally, the function `blk_start_request()`, which is usually invoked by interrupts, processes the I/O requests in the request queue by interacting with the block device.

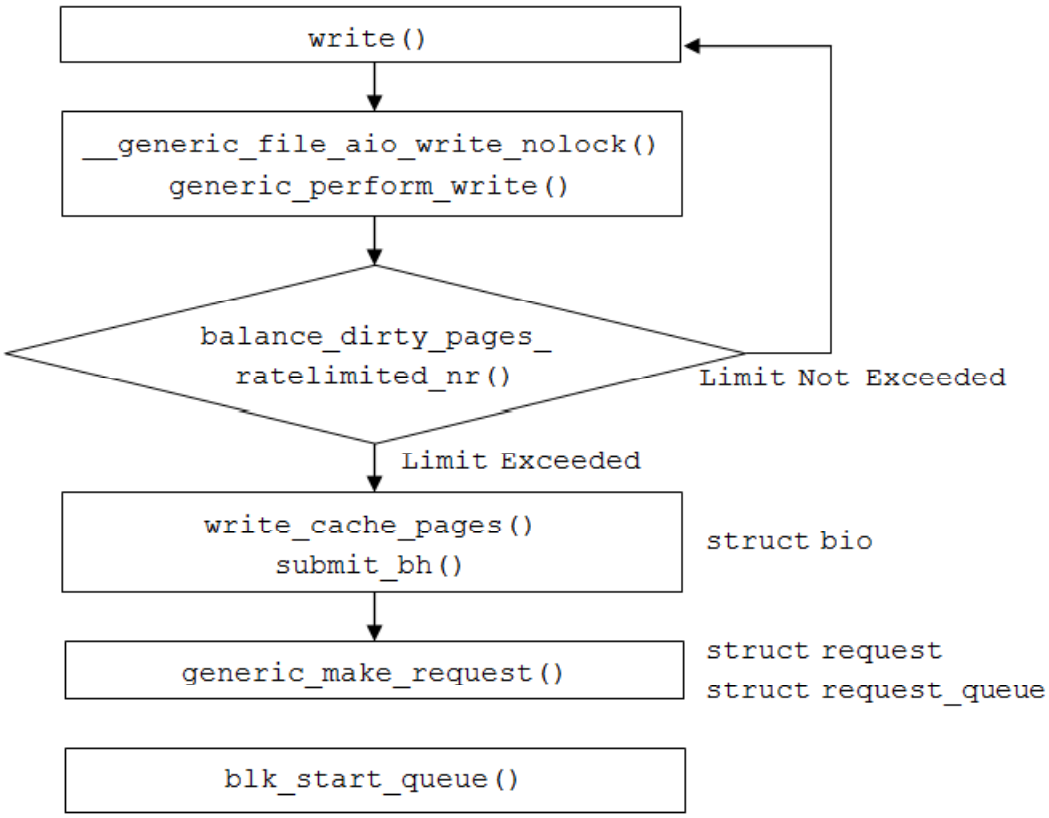

Figure 3.5. The sequence of kernel routines and data structures required to handle a data write request to the HDD.

3.4.2. I/O Monitoring Details

I/O mentoring performs the following two important functions in the HDD energy estimation system:-

1) *Timing of I/O Requests*: determining when the I/O requests are issued is achieved by utilizing kprobes with `printk()` function calls in both the `__generic_file_aio_write_nolock()` kernel routine and the `blk_start_request()` routine to record the timestamps for each request.
2) *Mapping I/O Requests to Causative Processes*: the PID is obtained from the `__generic_file_aio_write_nolock()` function. Mapping of a process to its requests is done by comparing the inode number logged using the `printk()` in both the `__generic_file_aio_write_nolock()` and `blk_start_request()` functions. Figure 3.6 illustrates the snippet an example log for the two functions and Figure 3.7 summarizes the mapping of a PID at the VFS layer to the corresponding I/O requests at the block device layer using the inode number. For the first routine, the inode number is obtained from the `inode` structure that is passed as a parameter to `__generic_file_aio_write_nolock()`. Inside the `inode` structure is the variable `i_ino` that contains the inode number. The size of the data to be written can be accessed from the structure `kiocb`, which is used to track the status of an I/O operation. It has a member variable called `ki_nbytes`,

which contains the data size in bytes. Current process information can be obtained using the `current` macro. For the second routine at the block device layer, the size of requests is obtained by summing the data sizes from each of the `bio` structures by accessing the segment size from the field `bv_len` in the member structure of `bio` called `bio_vec`, which is a pointer to the start of the data segment array of this `bio` structure. To access the inode number of the data abstracted by `bio`, the `page` field in the structure `bio_vec`, which is a pointer to the page descriptor of the segment's page frame, is accessed. Then, the `mapping` field has the corresponding page cache interpretation and the hosting `inode` could be located using the `host` field to obtain the inode number. This process is further detailed in Figure 3.8.

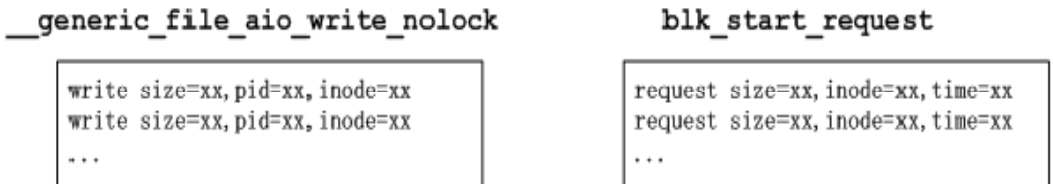

Figure 3.6. A sample log dump for two important kernel routines. I/O requests are mapped to causative processes by comparing inode numbers from the routines.

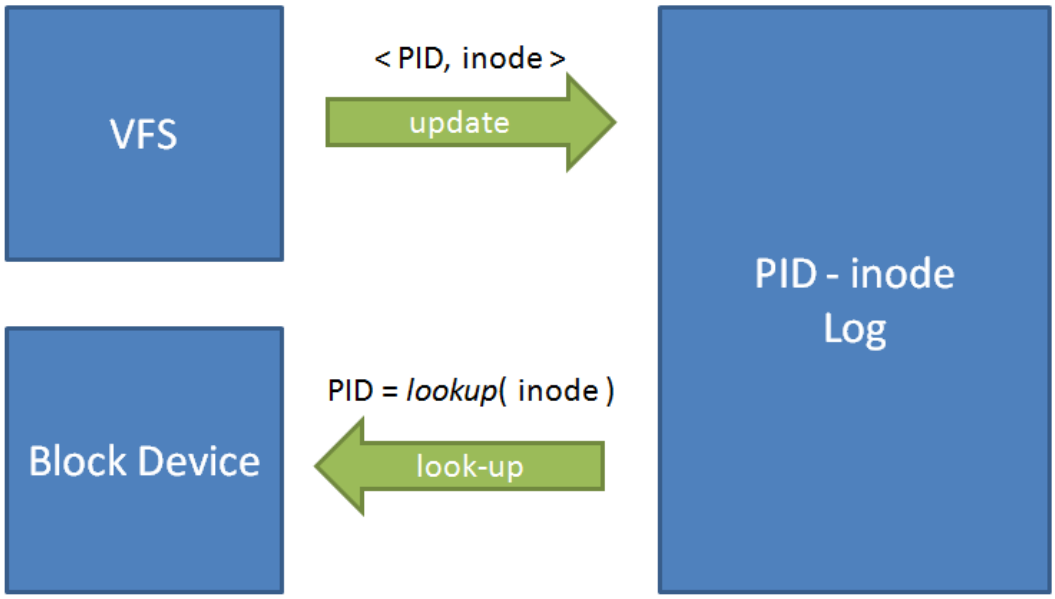

Figure 3.7. Mapping PIDs to block device I/O requests using inode numbers.

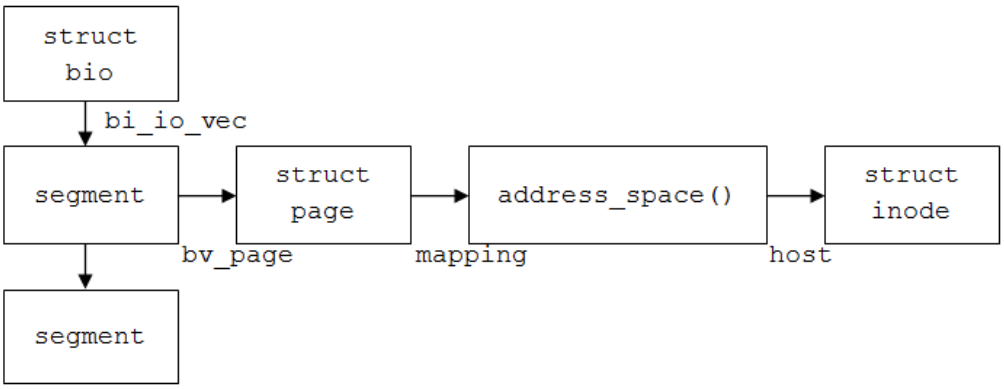

Figure 3.8. Steps involved in obtaining the inode number from a `bio` structure.

3.5. HDD Energy Model

The energy consumption of hard disk drives is influenced by several parameters. According to the results of previous work [HSR08], transferred data or file size, block size and logical block number are the important parameters that impact disk drive energy consumption. In this dissertation, the file system block size is fixed at the default value of 4kB as this is case for most Linux systems. Logical block number (LBN) is not taken into consideration in this work because its contribution to disk drive energy consumption is miniscule compared to the effect of the data file's size. Furthermore, ignoring the effect of LBN decreases the complexity of the energy model and the overhead of the kernel I/O monitoring.

3.5.1. Detecting Disk Activity

The disk drive may become active a non-deterministic time after a data write request is issued by an application. Thus, detecting disk activity is important as the energy consumption for this complete period must be used during model construction. This is accomplished using *power log analysis*, which is the analysis of the energy measurement log provided by DEEP for the test HDD. A rise in average power consumption by the HDD after issuing a file write request

indicates activity. All instances of such rise in power consumption are detected during power log analysis and the total HDD energy consumption during these periods is used as the HDD energy required to complete the file write request. An example of such a file write with a data payload or file size of 128MB is illustrated in Figure 3.9.

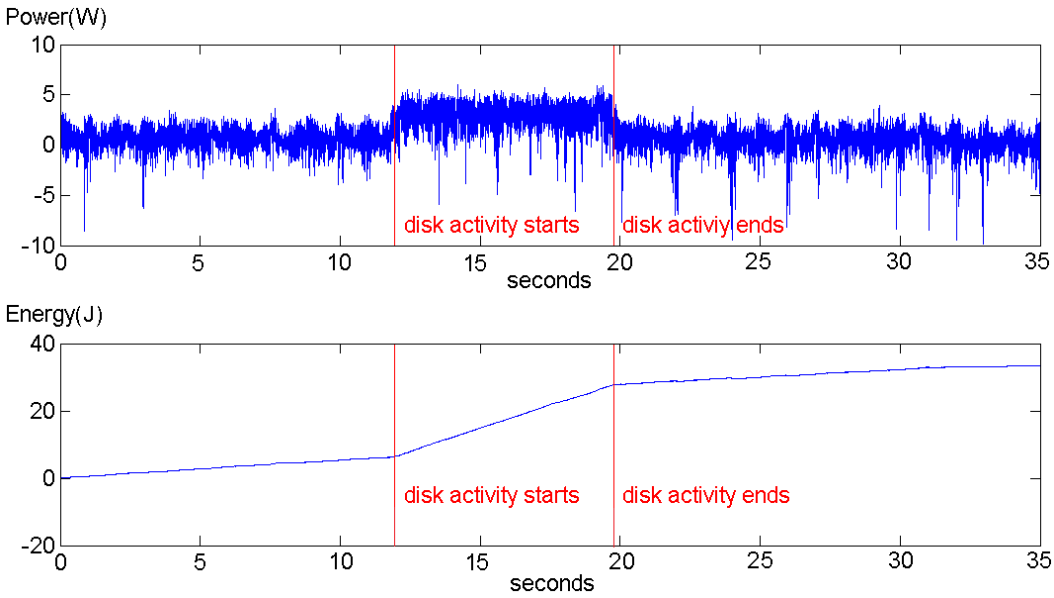

Figure 3.9. The energy measurement logs provided by DEEP are analyzed to detect the periods during which the HDD is active due to a file write request. The energy for these periods is the HDD energy consumed for the write request.

3.5.2. Model Formulation

Since the size of the I/O request or the file size is the only parameter used for the HDD energy model, energy consumption data is collected for write trials for 65 different file sizes using the power log analysis described in the previous subsection. One such write trail was illustrated in Figure 3.9. Each write trial is repeated 100 times. The histogram for HDD energy consumed for the hundred repetitions of the write trial for a particular file size is illustrated in Figure 3.10.

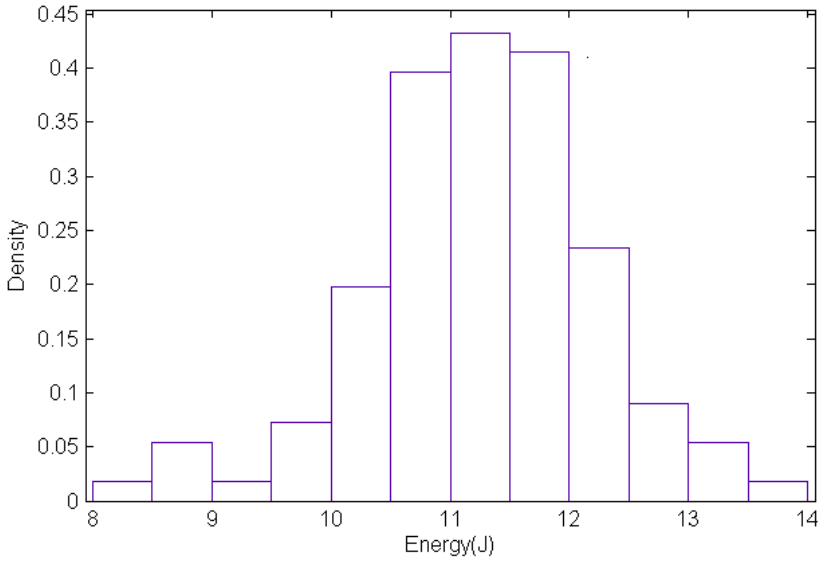

Figure 3.10. Histogram for energy consumption during 100 repetitions of an HDD write trial using the same file size.

Histograms similar to Figure 3.10 are observed for write trials of all the 65 different file sizes. The Anderson-Darling test [DS86] is used to test the goodness-of-fit of the set of hundred samples for a write trial. Each of the sets of samples passes the test for being normally distributed with 95% significance. The means for all hundred samples for each file size are plotted in Figure 3.11 (file sizes above 450MB also follow a linear trend and so are not shown) to determine the relationship between file size and HDD energy consumption for the test HDD.

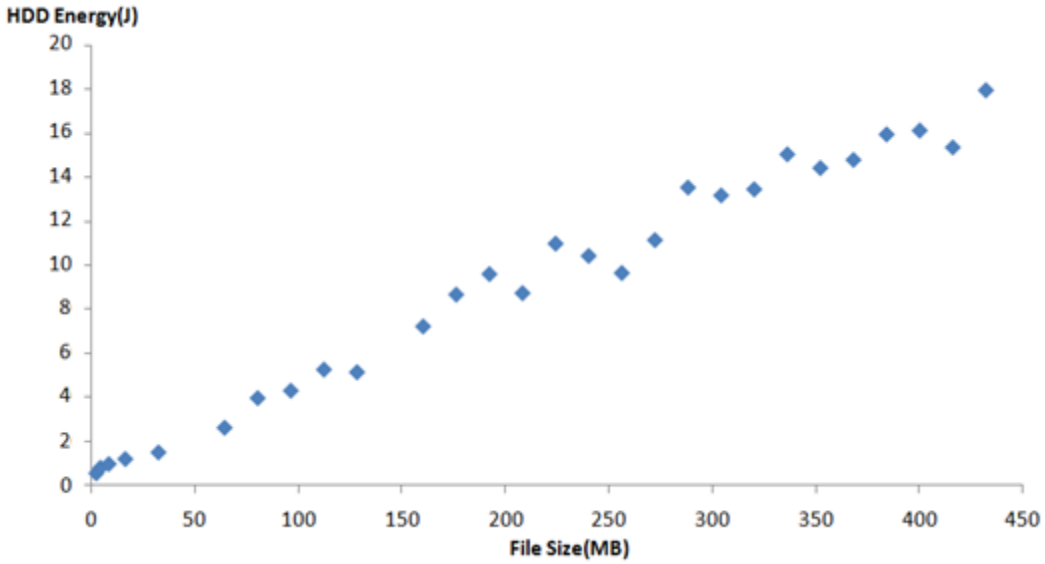

Figure 3.11. The observed relationship between file size and HDD energy consumed for the test HDD.

$$Energy = w_0 + w_1*(FileSize) \qquad (3.1)$$

The data from the write trials, shown previously in Figure 3.11, is used to fit the curve in Equation 3.1 and this equation models the energy consumption of the test HDD to service a file write request with file size equal to *FileSize*. The first term w_0 is a constant that represents the fixed energy cost due to preprocessing and setting up of data transfer [HSR08]. The second term is a linear term that captures the linear relationship between energy and file size.

3.5.3. Multiple File Writes

For simultaneous multiple write requests by one or more processes, the energy for each file write request is treated as an independent random variable with values that are normally distributed. Thus, HDD energy for multiple file writes is the sum of the estimated HDD energy for each individual file write. It is important to note that energy consumption for each file write is not strictly independent. For example, consecutive data writes following the first file write does not require disk start-up energy. However, in Section 3.7 it is demonstrated that this

assumption does not create large prediction errors while providing a model that is simple and has low overhead.

3.6. Implementation and Integration

The HDD energy estimation system is implemented on a DEEP Atom platform running Linux kernel 2.6.31-14 and with a SATA 5,400 rpm test HDD having an 8MB cache. This implementation is used for all the results presented in the next section.

The final implementation of the HDD energy estimation system consists of an integration of DEEP energy measurement, I/O monitoring and the HDD energy model. A user-space interface called *ioJoules* enables access to the system. This interface allows users to obtain a log of predicted HDD energy consumption for each of the write requests along with the process ID of the processes that are responsible for the request. This data is available in addition to the direct energy measurements that DEEP provides. A snippet of an example output log created by the ioJoules interface is illustrated in Figure 3.12 with the inode number (Inodenr) and process ID (Id) shown for each data write request to the HDD.

Command	Total	Transferred	Inodenr	Energy(J)	Id
ioelf1	67108864	19922944(30%)	1	1.78	7152
ioelf2	33554432	33554432(100%)	0	2.43	5544
ioelf3	2097152	2097152(100%)	0	0.20	5544

Figure 3.12. Sample outputs from the ioJoules interface for the HDD energy estimation system.

The HDD energy estimation system is integrated as part of the DEEP platform and a kernel module is built to create a /proc file system entry that turns the I/O request information logging ON or OFF by uninstalling all kprobe handlers for I/O monitoring in the kernel and halting data reporting through the ioJoules interface. This allows users to utilize the HDD energy estimation system only when needed while still enabling direct energy measurement from DEEP at other times.

3.7. Results

This section presents results from both the evaluation of the HDD energy estimation system and comparison to the direct energy measurements from the DEEP platforms.

3.7.1. System Evaluation

The HDD energy estimation system is evaluated in terms of both accuracy of estimation and overhead created. To evaluate the accuracy of the energy estimation system, write tests were performed using both normal I/O and direct I/O (page cache disabled) using file sizes up to 1GB. From the DEEP power log, power log analysis is used to compute the ground truth about HDD energy consumption. The predicted energy is compared to the values obtained from power log analysis to determine prediction accuracy. Figure 3.13 illustrates the results of the comparison for file sizes ranging from 128MB to 768MB.

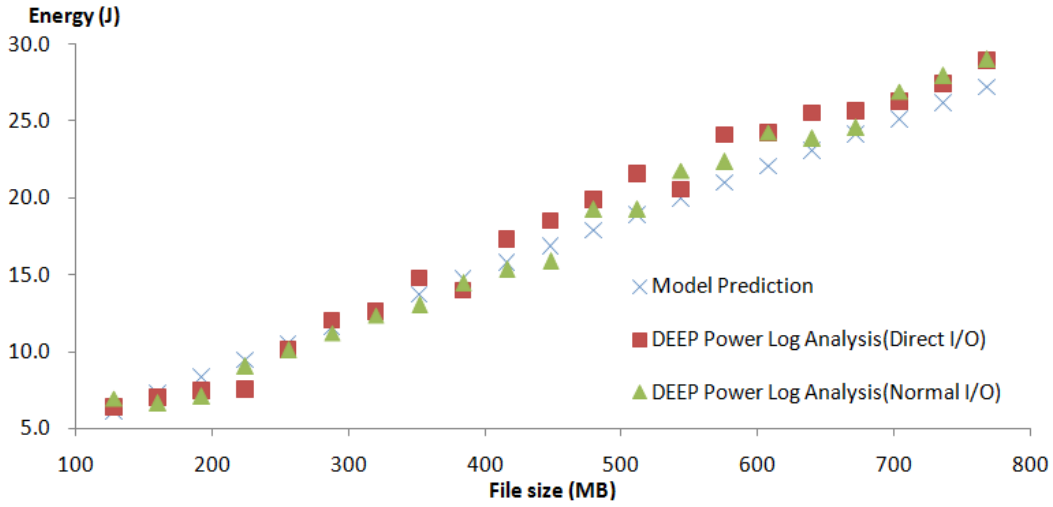

Figure 3.13. Accuracy of HDD energy prediction compared to power log analysis.

The system is also evaluated for accuracy of multiple file write energy prediction using write tests where ten processes randomly request to write files to the HDD. A combined write data payload ranging from 1.28GB to 3.52GB is used. Power log analysis is used to obtain the ground truth for actual HDD energy required to complete all the file transfers to the disk drive. Accuracy is determined by comparing the model's prediction to the ground truth from power log analysis. The comparison results are shown in Figure 3.14. As observed, the HDD energy estimation system's predictions are accurate and assumptions in Section 3.5.3 about independence of each of the multiple file writes does not degrade accuracy.

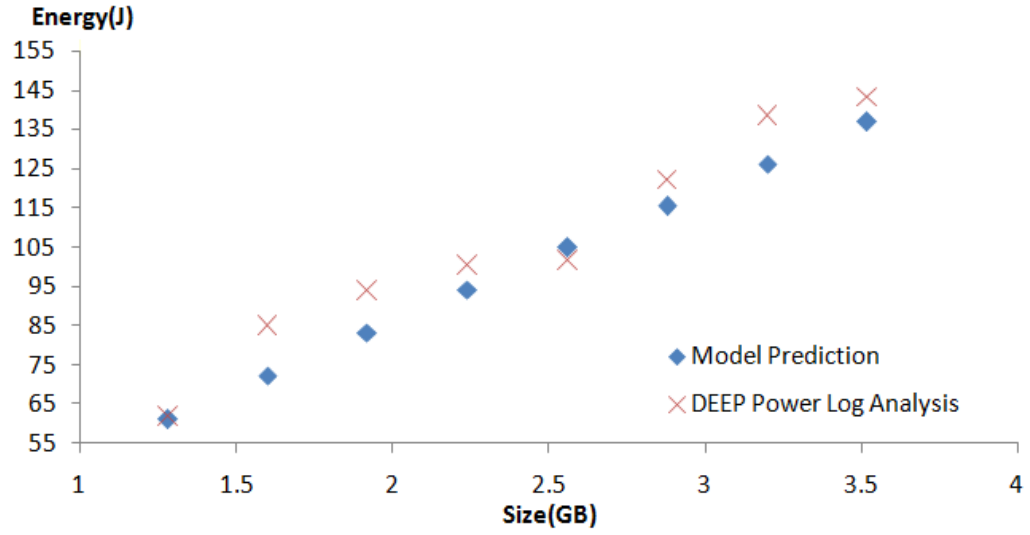

Figure 3.14. Accuracy of HDD energy prediction for multiple file write requests.

The results from evaluation of the accuracy of the HDD energy estimation system are summarized in Table 3.1. The model used is quite simple and still has excellent prediction accuracy. Accuracy is very slightly degraded for direct I/O because the model was built using data for normal I/O and for multiple file I/O due to the assumption that each of the file writes are independent.

Table 3.1. The HDD energy estimation system's prediction accuracy.

HDD Write Method	Average Prediction Accuracy (%)
Direct I/O	92.39
Normal I/O	94.09
Multiple File Normal I/O	92.98

The energy estimation system is also evaluated in terms of the energy overhead it creates on the DEEP platform. This is important because even though the I/O monitoring infrastructure uses low performance overhead kprobes for data gathering, this may still impact the energy efficiency of the DEEP platforms. Both the CPU and memory (RAM) power consumption are compared with and without kernel I/O monitoring during file writes of sizes ranging from 128 to 768MB to evaluate any power consumption overhead that the HDD energy estimation may create. Results of this comparison are illustrated in Figures 3.15 and 3.16.

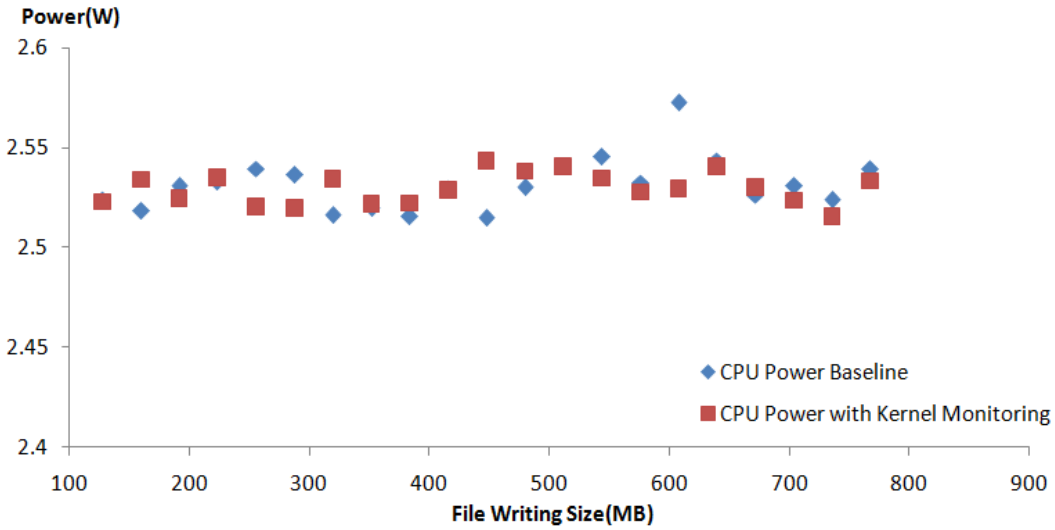

Figure 3.15. CPU power consumption with and without kernel I/O monitoring.

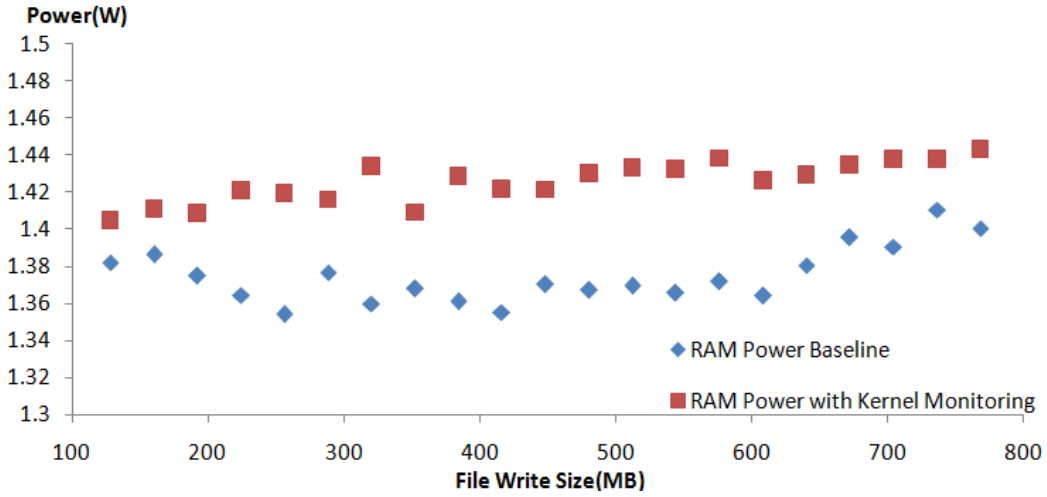

Figure 3.16. RAM power consumption with and without kernel I/O monitoring.

The power consumption overhead due to the HDD energy estimation system is summarized in Table 3.2. Miniscule power consumption overhead is observed for the CPU and a small average overhead of 3.61% was observed for the memory module (RAM).

Table 3.2. The average power consumption overhead for the CPU and RAM.

Component	Power Consumption Overhead (%)
CPU	0.95
RAM	3.61

The overhead on performance is also characterized through execution of standard applications with and without the HDD energy estimation system. Execution times for both cases are compared to derive the performance overhead. The performance overhead is illustrated in Table 3.3.

Table 3.3. The average performance overhead for some common applications.

Application	Performance Overhead (%)
firefox	1.15
gcc	1.01
grep	1.33
bzip2	1.61

3.7.2. Comparison to Direct Energy Measurement

The energy required for deferred disk write operations cannot be accurately estimated through direct measurements gathered during a benchmark's execution lifetime. To demonstrate this, the energy consumption data of DEEP direct measurements are compared with the values predicted by the energy estimation system.

Spew [ber04] is an I/O performance tool to measure block device performance or to generate disk drive workloads. It is a flexible file system workload generator and is used to generate workloads in different I/O modes with various file sizes. For the experiments in this section, single file writes were performed with file sizes ranging from 32MB to 608MB in normal I/O mode.

For every benchmark test the DEEP platform was used to directly measure HDD energy consumption. Note that this is done by DEEP's energy calipers, where the energy calipers generate time stamp values during Spew execution in order to obtain the benchmark's execution time interval and calculate energy consumption during that interval. Also, the HDD energy estimation system's model is used to

predict the HDD energy consumption. Finally, after each benchmark test is completed, power log analysis is used to analyze the DEEP energy measurement log and obtain the ground truth for HDD energy consumption.

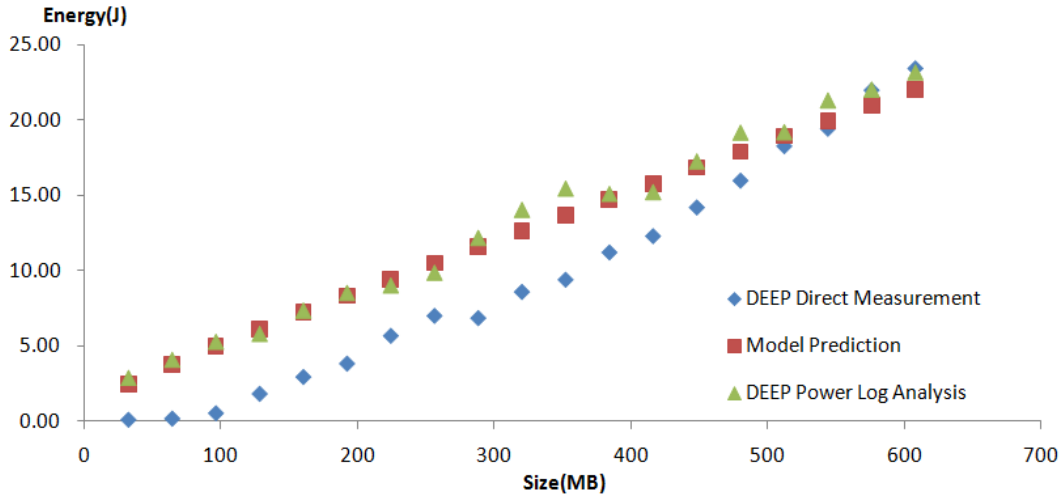

Figure 3.17. Comparison of energy using direct measurement during benchmark execution, energy estimation system's model prediction, and power log analysis.

Figure 3.17 illustrates that direct measurement cannot provide an accurate estimate of HDD energy due to the operating system kernel's buffer cache and other disk drive I/O deferring mechanisms. Due to deferring of data writes, a benchmark application's execution lifetime cannot provide the time interval for

estimating HDD energy consumption. Hence, power meter based direct measurement can be utilized for CPU or RAM energy estimation, but for the HDD this is not sufficient. However, for file sizes larger than 500MB, the errors due to direct energy measurements tend to become insignificant. This is because the memory module used in this implementation has a capacity of 512MB and the effect of kernel buffer cache is eliminated by page flushing that is activated by the Linux kernel because of larger files.

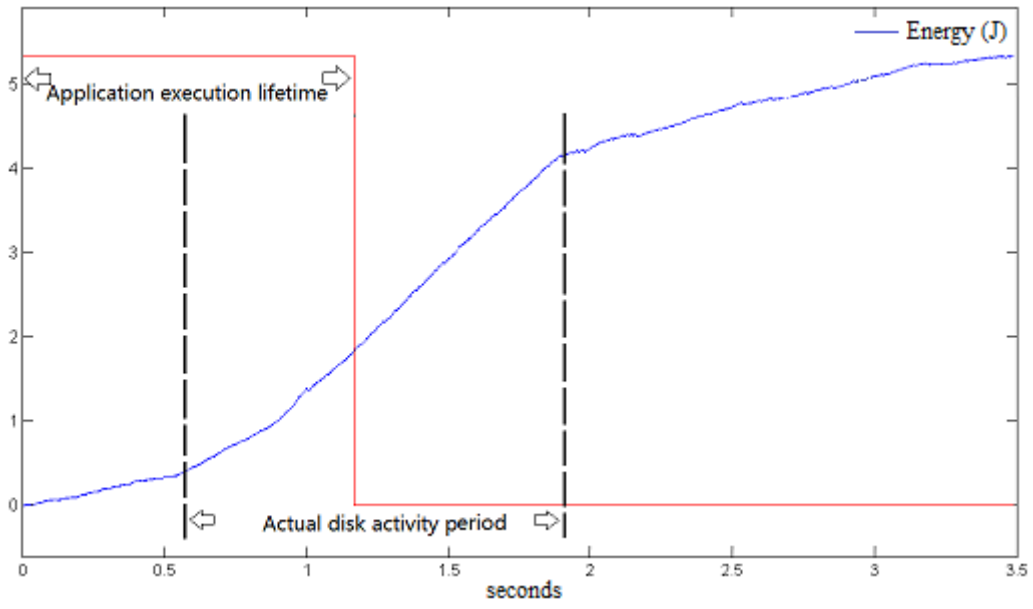

Figure 3.18. Benchmark application execution lifetime versus disk activity period.

To further demonstrate the limitations of direct energy measurement, Figure 3.18 illustrates a comparison of the benchmark application's execution lifetime for a file write with the actual HDD activity period. It is clearly observed that the direct measurements, such as ones obtained by power meters, will only aggregate the energy consumption during benchmark execution. However, more than half of the disk drive activity occurs after the benchmark has completed execution. Using the energy estimation system presented in this dissertation can greatly improve accuracy of HDD energy consumption estimates for an application. In addition, the ability to attribute the HDD energy consumption to individual processes can guide optimization of software energy efficiency.

3.8. Conclusions

There is an urgent and largely unmet need for guidance tools that assist application developers to accurately estimate the energy consumed due to their applications. Thus, systems that predict secondary storage I/O energy assist in the development of energy efficient software because modern storage hardware can be complex due to the presence of I/O rescheduling and buffering mechanisms.

The HDD energy estimation system presented is novel because of unique features such as the capability of mapping estimated HDD energy to specific causative processes while using a relatively simple energy model. Also, the energy estimation system creates very low overhead on the computing platform and requires minimal modification to the operating system kernel while providing accurate estimates of HDD energy consumption for write workloads of various payload sizes.

CHAPTER 4

ENERGY EFFICIENT ADAPTIVE DATA COMPRESSION

This chapter presents a direct application of the DEEP platforms. Direct energy measurements from DEEP are applied to create an energy efficient data compression algorithm that adapts to changing network and system conditions. This energy efficient adaptive compression algorithm is applied to the upload to data over a network link and significant energy savings are observed. The chapter begins with Section 4.1, which provides background and overviews work related to energy-aware adaptive data compression. Section 4.2 demonstrates the large impact that data compression can create on the uploading of data over a network link under different network and system conditions. Section 4.3 presents the energy efficient adaptive compression algorithm, which is called DEEPcompress. Section 4.4 details the results obtained from application of DEEPcompress to upload of data files. Section 4.5 concludes this chapter. Some of the results described in this chapter have been published previously [SK12].

4.1. Background and Related Work

While many applications are important for modern computing platforms and mobile devices, network data transport emerges as central to many of the important applications since the advent of cloud computing, proliferation of mobile devices and rapid increase in the number of networked systems.

Data compression has been explored and demonstrated as an effective method for improving the performance and energy efficiency of transporting data over a network link [Pet13]. Data compression is already used when downloading data files and has been shown to be beneficial even when uploading data [WM09]. A large number of previous studies show that dynamically varying the compression schemes can be beneficial in adapting to changing network or system conditions [MS06], [KC01], [PS05], [KS05], [KB99].

4.1.1. Computation versus Communication

The dynamic selection of compression schemes to form an adaptive compression algorithm confronts a major challenge because benefits from data compression in network data transport derive primarily from the trade-off between computation

and communication [BA06]. Thus, the resources invested in compression scheme selection and compressing the data (computation) should be surpassed by savings created due to a reduced payload being transported over the network (communication). This implies that an adaptive compression algorithm should be able to measure or estimate the resources required for both computation and communication to perform an effective selection of optimal compression schemes.

For a dynamic environment with system and network conditions varying at run-time, the estimation or measurement of required computation and communication resources becomes a difficult task. This is further compounded by the fact that the data being transported is also an important parameter that affects the computation and communication resources required. Thus, previous work on adaptive compression, in summary, attempts to find heuristics or measures that can estimate or measure both communication and computation costs to create an algorithm that dynamically selects efficient compression schemes. This dissertation presents a new adaptive compression algorithm called DEEPcompress that directly measures energy costs using DEEP's direct energy measurements

during run-time as a means to dynamically select energy efficient compression schemes. The DEEPcompress algorithm is detailed further in Section 4.3.

4.1.2. Datacomp

Even though this dissertation has summarized the basic concepts behind adaptive data compression algorithms, there are a number of issues beyond the scope of the work presented herein, such as data decompression and innovations in creating more effective compression schemes, that have not been discussed. Datacomp [Pet13] discusses such issues about data compression in much greater detail and attempts to create a comprehensive adaptive compression framework that is independent of assumptions or conditions specific to a certain type of data, network or computing platform.

4.2. Impact of Data Compression on Energy

This investigation characterizes the impact of compression on network data transport energy efficiency using the online version of the DEEP Atom platform. Energy consumption is measured during compression and then transmission of a file over a network link while varying three parameters: network data throughput,

data file type and compression scheme. The data files used are a subset of common files: 1) an English text document, 2) a Portable Document Format (PDF) document, and 3) an MP4 encoded media file. Three network interface types are used with energy measurement for each: a 10/100 Ethernet device, a local area wireless IEEE 802.11g device and a wide area wireless 3G CDMA 2000 1X EV-DO modem.

The compression schemes used in the investigations are Gzip, bzip2, and XZ [Pet13]. Gzip is based on a combination of the Lempel-Ziv algorithm [WZ91] with Huffman encoding [MP85], XZ employs a Lempel-Ziv-Markov-chain Algorithm (LZMA), and bzip2 utilizes run-length, delta and Huffman encoding with the Burrows-Wheeler Transform [FM08]. The energy cost for data transport without compression is also measured. For each parameter combination, a file is compressed and transmitted with data throughput limitation for the network interface enabled at each of four levels: 10kB/s, 100kB/s, 1MB/s, and 10MB/s. Network interface energy consumption is included in the system energy measurement. Energy calipers are utilized to monitor energy consumption during compression and transmission of the data files for each parameter combination.

The 3G interface consumed the most energy (10 percent of the total system energy) followed by IEEE 802.11g (4 percent) and then the Ethernet interface. Prior work has clearly demonstrated that the network interface forms a significant fraction of energy consumption in mobile devices [CH10]. In the Atom N330 system this fraction is not large enough to change the optimal compression choice.

Table 4.1. Impact of data compression on energy consumption for file upload.

Rate (B/s)	**XZ (J/MB)**	**bzip2 (J/MB)**	**Gzip (J/MB)**	**None (J/MB)**	**Energy Ratio**
10k	**196.8**	739.4	994.3	3397.1	**17.3**
100k	101.9	**97.2**	107.3	338.7	**3.5**
1M	92.5	33.1	**18.9**	33.9	**4.9**
10M	91.6	26.8	10.1	**3.4**	**26.9**

It also is observed that the content of the data file is an important factor in selecting the optimal compression scheme due to variation in compressibility. Network conditions like data throughput (rate) have a very large impact on the selection of energy efficient compression schemes. This is illustrated using the *energy ratio* in Table 4.1 with measurements for the text file. None of the compression schemes are optimal in all cases and the penalty for selecting sub-

optimal schemes is very large (up to 17.3x) in certain cases. This can further be impacted by the influence of the platform's computing environment, such as CPU load, on the energy required for compression. Thus, to conserve energy the selection of the optimal compression scheme should be made dynamically at run-time during the transmission of the data file based on the contents of the data file, network conditions and the computing environment.

4.3. DEEPcompress

Preceding sections demonstrate that data compression can lead to large energy savings in uploading certain types of files over a network link. Since the characteristics of the computing system (like CPU load) and network (like the energy required to transmit or compress a byte of data) can change during file transmission, the compression scheme must be dynamically selected. This section presents the DEEPcompress algorithm detailed in Figure 4.1. The file being uploaded is divided into constant-size blocks, and both the transmission energy per byte and the compression energy per byte is measured for a block. The DEEPcompress algorithm selects the most energy efficient compression scheme for a data block by utilizing both the communication and computation energy.

```
# Values of energy per byte for bzip2, Gzip, XZ and transmission
ebz = egz = exz = etrans = 0

# Compress 1st block for estimate of compression energy
block = GetNextBlock ( file )
bz = bzip2Compress ( block )
gz = GzipCompress ( block )
xz = XZCompress ( block )

repeat
      # Energy = Computation + Communication
      Ebz = ebz * Size ( block ) + etrans * Size ( bz )
      Egz = egz * Size ( block ) + etrans * Size ( gz )
      Exz = exz * Size ( block ) + etrans * Size ( xz )
      Eno = etrans * Size ( block )

      # Objective - select scheme with least energy for block.
      if ( Ebz ≤ Egz and Ebz ≤ Exz and Ebz < Eno )
      # Use bzip2 compression.
            bz = bzip2Compress ( block )
            Transmit ( bz )
      else if ( Egz < Ebz and Egz ≤ Exz and Egz < Eno )
            # Use Gzip compression.
            gz = GzipCompress ( block )
            Transmit ( gz )
      else if ( Exz < Egz and Exz < Ebz and Exz < Eno )
            # Use XZ compression.
            xz = XZCompress ( block )
            Transmit ( xz )
      else
            # Don't use compression - none.
            Transmit ( block )

      # Energy per byte of latest transmit and compress.
      UpdateEnergyValues ( )

      # Get next block from file to be uploaded.
      block = GetNextBlock ( file )

# Repeat until all blocks of the file have been transmitted.
until ( block = 0 )
```

Figure 4.1. Pseudocode for implementation of the DEEPcompress algorithm.

The transmission and compression functions in the DEEPcompress algorithm are monitored using energy calipers that record the energy consumption during their execution. The algorithm uses these measurements to dynamically select energy efficient compression schemes.. The algorithm can thus adapt to changes in the energy consumption characteristics due to variation in network conditions that can affect transmission energy required per byte, or availability of computing resources that can change the energy requirement by each of the compression schemes.

4.4. Results

The DEEPcompress adaptive compression and transport algorithm is deployed on the online version of the DEEP Atom implementation with an added IEEE 802.11g device that is the wireless network interface used for data file transmission. A measurement delay is possible between execution of the transmission or compression functions and the availability of the latest energy caliper measurements corresponding to these functions. If the latest energy values are not available, the preceding data are used without waiting for the updated values.

4.4.1. Platform Energy Efficiency

To characterize the benefits of DEEPcompress over other static compression schemes, DEEPcompress, XZ, bzip2, Gzip and no compression (none) are all compared for data compression and transmission during the upload of the following files: 1) tarball of Linux- 2.6.31 kernel sources, 2) tarball of twenty plain English texts, 3) tarball of twenty PDF e-books and 4) an MP4 encoded commercial film trailer. The transmission is performed using a TCP/IP socket connection to another system on the UCLA wireless network. A block size of 16kB is used in this deployment of DEEPcompress and each of the files is transmitted thirty times using each compression choice.

The energy consumption by the system is averaged over all the transmissions and includes the wireless device's energy. This value is normalized to each file's size and is reported in Figure 4.2. DEEPcompress consistently demonstrates superior energy efficiency for the English texts, Linux sources and PDF ebooks. Energy savings as large as 38% are observed compared to the next best compression scheme. Miniscule degradation in energy efficiency compared to none (1.2%) is observed for the MP4 file because of the inherent compression present in the MP4

format. DEEPcompress is still more energy efficient than the other static compression schemes as it dynamically determines that no compression is the best choice in this case and selects it.

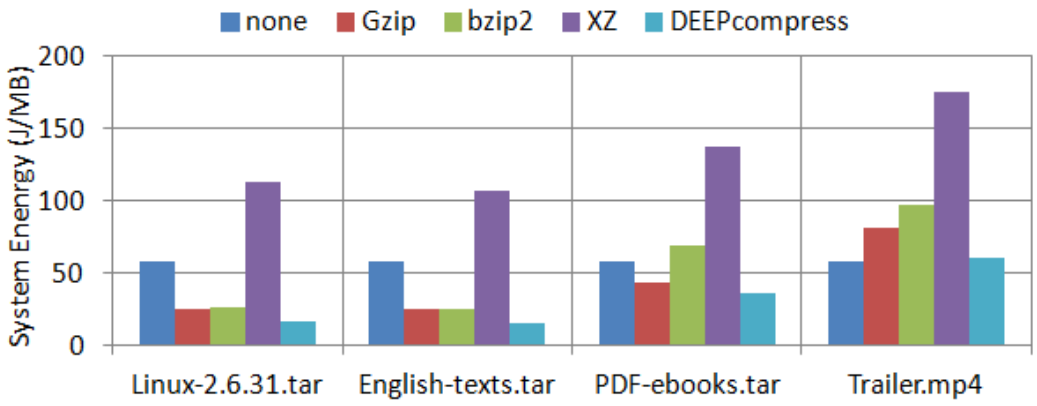

Figure 4.2. The DEEPcompress algorithm creates significant energy savings during upload of various types of data files.

DEEPcompress achieves superior energy efficiency by adapting the compression scheme selection to changing network and system conditions. This is illustrated in Figure 4.3 using a one minute interval during a randomly selected transmission of the Linux sources. The variation in energy for transmission along with the compression scheme selected by the DEEPcompress algorithm is shown. The

algorithm's use of the DEEP platform for direct energy measurement of the compression and transmission tasks enable it to adapt to change in energy consumption due to network and system conditions.

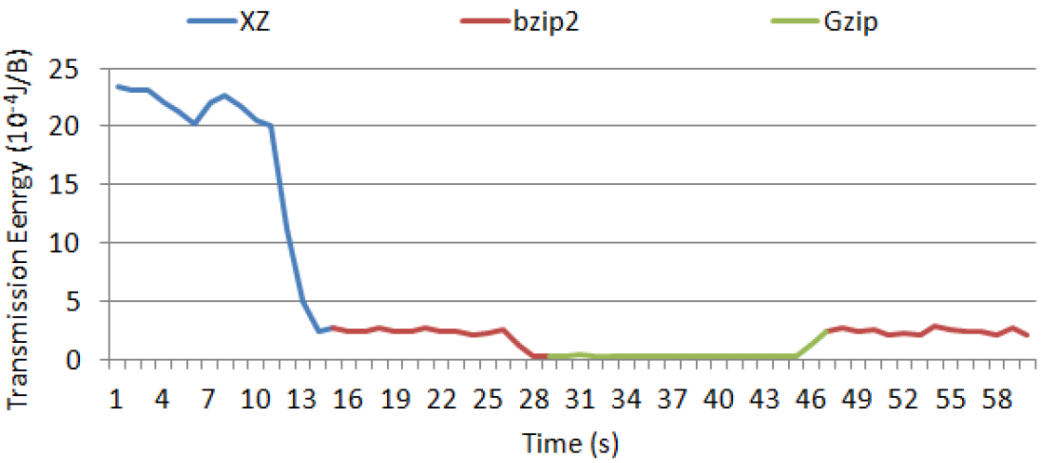

Figure 4.3. DEEPcompress adapts its compression choice as it detects variations in the wireless network and system energy consumption characteristics.

4.4.2. Component Energy Efficiency

The DEEPcompress algorithm can be extended to include a wider range of compression choices like LZO. Also, instead of total system energy, applications like thermal management may require subsystem-level power management objectives like reduction in CPU or network interface energy. As demonstrated in

Figure 4.4, attempting to reduce the IEEE 802.11g network interface's energy using DEEPcompress creates energy savings as large as 44% for the interface instead of the 37% when using total system energy as the objective. However, the total system energy savings decrease from 38% to 33%. This demonstrates the utility of component-resolved measurement capabilities while showing that subsystem energy reduction objective is not always optimal for system energy efficiency and vice-versa.

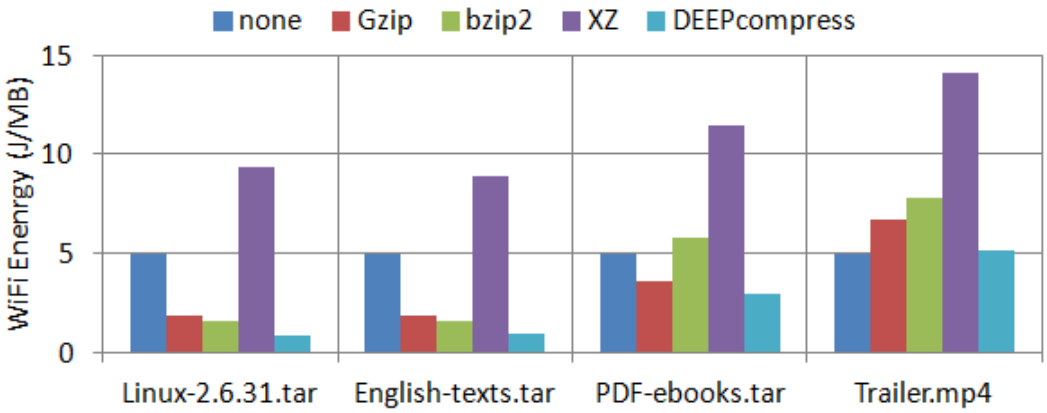

Figure 4.4. DEEPcompress can also be used for reducing energy consumption of subsystems, such as for the IEEE 802.11g (WiFi) interface here. Instead of total system energy, the WiFi interface's energy is used as the DEEPcompress algorithm's optimization objective.

DEEPcompress also supports joint objectives. A combination of power, energy consumption and processing parameters like execution time for the system or subsystems can be used as the objective for compression scheme selection.

4.5. Conclusions

The DEEP Atom is utilized for an investigation of compression in transport of data over a network link. As confirmed in previous work, it is observed that dynamically selecting compression schemes to adapt to changing network and system conditions can significantly improve the energy efficiency of network data transport.

The DEEP platform's capabilities assist in development of an energy efficient data compression and transport algorithm called DEEPcompress. The algorithm provides large energy savings (38%) for upload of data files over a wireless network link through dynamic selection of compression schemes to adapt to system and wireless network conditions. Also, DEEPcompress can be adapted to optimize for component energy savings instead of complete platform energy savings during compression and transmission of data. Furthermore, joint

optimization objectives such as a combination of peak power, energy or execution time may also be used.

CHAPTER 5

ENERGY AWARE TASK SCHEDULING

This chapter presents the Energy Aware Scheduler (EAS), which is a novel operating system task scheduler designed for multi-core computing platforms. EAS utilizes the high-resolution time-synchronized direct energy measurements provided by the DEEP platforms along with data from the CPU performance monitoring unit to improve platform energy efficiency and performance by reducing resource contention between tasks. The chapter begins with an overview of the relevant background material and related work in Section 5.1. Section 5.2 describes the EAS architecture and the concepts that have culminated in the design of EAS. Section 5.3 presents the implementation of the task scheduler using extensions to a standard Linux kernel operating on a commodity computing platform. Section 5.4 presents the results of comparison between EAS and the task scheduler adopted by Linux using a range of standard benchmark applications. Section 5.5 concludes this chapter.

5.1. Background and Related Work

The operating system's task scheduling policy has a well-known and significant impact on the performance and energy efficiency of computing platforms [ZSB13]. Development of task schedulers confront the challenges of ensuring low latency, meeting task deadlines, improving fairness in distribution of computing resources to tasks and load balancing among processors [DB11], [Shi05]. However, the introduction and large-scale deployment of multiprocessor platforms now also requires that resource contention be minimized to ensure efficient usage of resources. The demands for reducing resource contention and increasing energy efficiency add new constraints to scheduler system development. This requires a task scheduling system that responds to real-time energy and performance monitoring to optimize selection of tasks and assignment to processors to avoid degradation in energy efficiency and performance.

5.1.1. Resource Contention in Multiprocessing Computing Platforms

Multiprocessing CPU architectures are critical in the delivery of computing performance for a broad range of platforms from mobile embedded devices to

server systems. Symmetric multiprocessing (SMP) CPU architectures are widely deployed to enable advances in both energy efficiency and performance through parallel execution [Shi05]. In the case of the most widely deployed mobile systems using the Linux operating system, parallel task execution is enabled by the scheduling of task threads by the operating system on each core with the objective of maintaining fairness in support across all tasks, low latency in task execution, and load balancing across CPUs. This can lead to well-known increases in computational throughput and platform energy efficiency [WL08]. However, the potential for these benefits is not reached if multiple tasks frequently contend for resources, such as shared CPU cache, on a multi-core CPU and this causes a degradation in performance known as co-run degradation [BZF10]. Current operating system task schedulers do not detect the energy efficiency or performance penalties due to resource contention.

The impact of co-run degradation is illustrated in Figure 5.1 using four benchmarks from the UnixBench suite [SGY11] that execute in parallel on a quad-core x86 platform with the Linux 2.6.32 kernel. Execution time required during parallel execution is compared with the time required when the same four

applications are executed individually. The worst and best cases in the figure correspond to different task schedules that were observed over a thousand different repetitions of the experiment. The results of the experiment clearly demonstrate that co-run degradation can be very significant and different task schedules can influence the extent of co-run degradation.

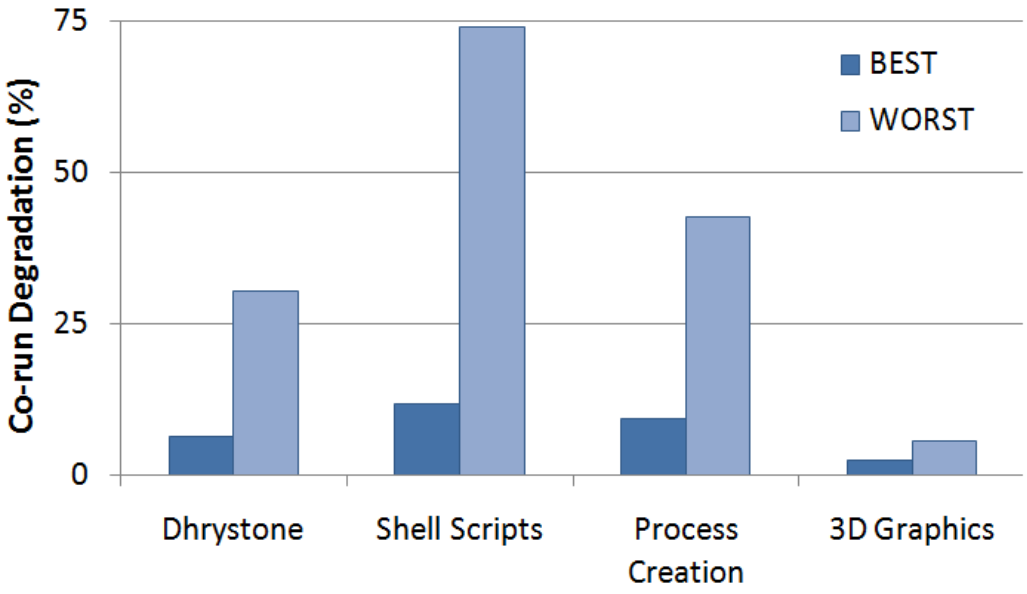

Figure 5.1. The co-run degradation problem is illustrated with four benchmark applications from the UnixBench suite on a quad-core CPU. Execution times when the four applications execute in parallel are compared to when each is executed individually.

EAS is a novel task scheduler that is broadly applicable across platforms and is directly implemented in Linux. EAS actively seeks and detects the effects of co-run degradation through a new task observation method. Then, it exploits an efficient task selection architecture to seek a task selection that reduces co-run degradation. It is important to note that co-run degradation is determined at run-time because of run-time changes in task behavior that may lead to variability in the characteristics of resource contention. EAS exploits direct real-time energy measurements along with standard CPU performance counters to identify inefficient processes suffering from resource contention. The tasks corresponding to these inefficient processes have their scheduling priority adjusted to reduce co-run degradation.

5.1.2. Completely Fair Scheduler

The Completely Fair Scheduler (CFS) has been adopted as the task scheduler in Linux since the 2.6.23 kernel [WTK08]. CFS constantly attempts to maintain fairness among tasks in terms of allotted CPU time [WCJ09]. Thus, if a task has used the least amount of CPU time then CFS will assign highest scheduling priority to this particular task.

In order to efficiently perform priority assignment and to maintain a list of executable tasks, CFS constructs a balanced binary search tree called the red-black tree [Hin99]. Each node of the red-black tree data structure represents an executable task along with the key being the CPU time used by the corresponding task. Since the tree is balanced, the scheduler can efficiently perform scheduling operations such as insertion/removal of a task and selection of the highest priority task in O(log n) time complexity for a tree with n tasks.

In multiprocessing platforms, CFS maintains a red-black tree structure for each CPU core present on the platform. Tasks are divided among the cores via the mechanism of *load-balancing* and tasks can be moved from one core to another through the use of *task migration* [WCJ09]. Thus, CFS provides each CPU core with its own set of tasks and the corresponding red-black tree is used in the scheduling of these tasks.

5.1.3. Performance Measurement Unit

Modern CPU architectures support detailed monitoring of computing platform events through the performance monitoring unit. Performance monitoring units

provide access to special-purpose CPU registers called performance counters that can count important events such as cache misses. They have been recognized as an important resource for performance monitoring and improvement [ZDF07]. Statistics obtained from performance counters have been used in a wide range of applications from power/energy modeling [SBM09] to virtualization [XJJ12] and even security [SZD08]. EAS utilizes two types of performance counters: 1) the time-stamp counter (TSC) is used as a clock for high-resolution timing of platform events, and 2) performance counters are used to measure the number of CPU operations performed during execution of a task.

5.2. Architecture

The EAS architecture is based on CFS to enable both rapid adoption of EAS into standard platforms and to also harness important CFS features. Primary features of CFS, such as load-balancing and task migration, are included without modification. EAS also uses a red-black tree, similar to CFS, to maintain the task list for each CPU core. The primary distinction between CFS and EAS is that EAS utilizes a run-time indicator of a task's energy efficiency computed directly and termed as Operations per Joule (OPJ).

OPJ values for each task are used to assign scheduling priority to the corresponding task. This provides EAS with the ability to detect tasks that induce inefficiency due to co-run degradation caused by resource contention. Furthermore, EAS does not perform complicated dynamic time slice calculations for a task, but instead uses a fixed value. Thus, this time slice decides the scheduling quantum for a task in EAS while the OPJ value for a task determines its scheduling priority.

5.2.1. OPJ

EAS uses OPJ as a run-time indicator of a task's energy efficiency. OPJ represents the number of CPU micro-operations (μops) [SCF03] executed by the task per joule of energy consumed by the platform. A performance counter is used for each CPU core to count the number of μops executed by a task each time it is scheduled. Time-synchronized energy measurements are then used to determine energy consumption of the platform during the time quantum that the task was scheduled. The number of μops is divided by the platform energy consumption to obtain the latest task OPJ value, which is then used by EAS for priority assignment to that task.

5.2.2. Priority Assignment for Efficient Co-Scheduling

EAS applies an approach for selecting tasks that may operate together without the inefficiencies introduced by resource contention. This is accomplished by establishing a scheduling priority assignment that favors co-scheduling of these tasks. and also reduces the probability of co-scheduling tasks exhibiting co-run degradation.

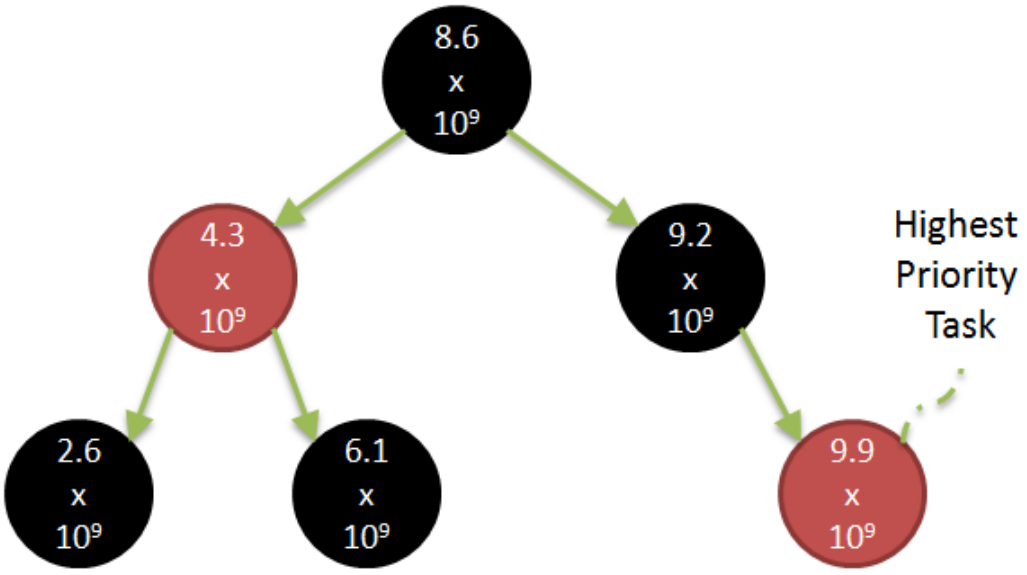

Figure 5.2. The Energy Aware Scheduler (EAS) uses the red-black tree data structure where each node represents a task. The OPJ value for a task is used as the key for its node.

Similar to CFS, EAS uses the red-black tree to maintain a list of tasks. Each node of the red-black represents a task and uses the OPJ value of the task as its key. Larger OPJ values represent a higher priority as this ensures that tasks that exhibit higher efficiency when co-scheduled are scheduled together while tasks that exhibit inefficiency when co-scheduled, are demoted in priority and are less likely to be scheduled together. Figure 5.2 illustrates an example red-black tree that is used by EAS with the OPJ of each task being used as the key. A red-black tree is maintained for each CPU core on the platform and the mechanisms of load-balancing along with task-migration, borrowed from CFS, are used to distribute tasks among the CPU cores.

5.2.3. Task Promotion

The priority assignment scheme used by EAS has the advantage of efficient task co-scheduling, but inefficient tasks with reduced OPJ values also need to be provided with opportunities for being rescheduled since task behavior varies at run-time and contention from other tasks may not be present anymore. Furthermore, complete starvation of inefficient tasks also needs to be addressed. This is accomplished in EAS through *task promotion*.

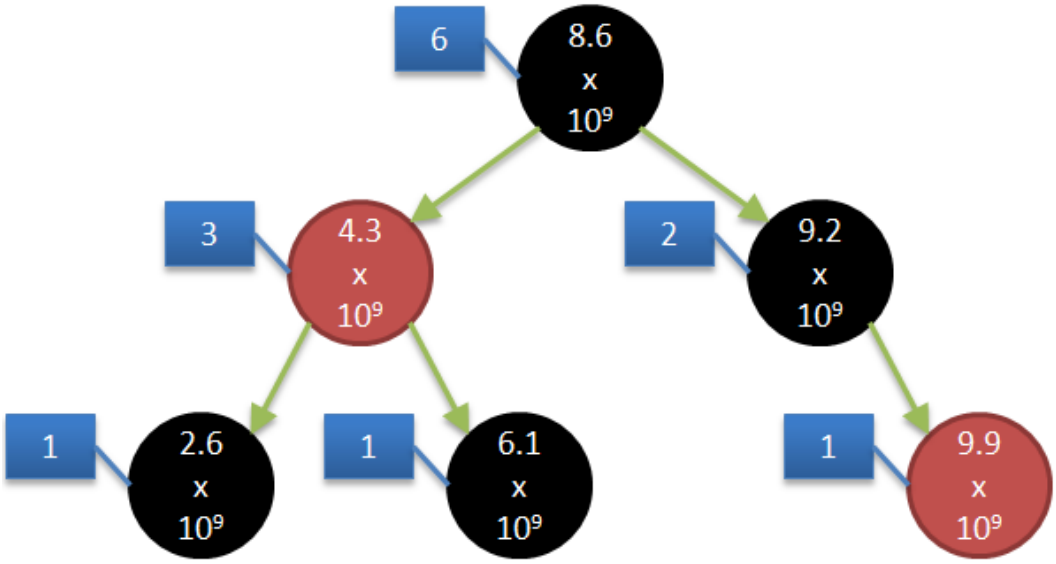

Figure 5.3. EAS uses a modified red-black tree where each node has an additional value attached to it. This value represents the size of the tree if the corresponding node was the root.

For a red-black tree with n tasks, task promotion randomly selects a natural number $k \leq n$ every n scheduler time quanta so that the k^{th} lowest priority task's OPJ value is set to one greater than the largest OPJ value in the tree. This causes the task to be promoted to highest priority for the succeeding scheduling quantum. To preserve the algorithmic efficiency of red-black tree operations, the time complexity of performing task promotion must be limited to O(log n). Hence,

EAS uses a modified red-black where each node has an additional value, called *Size*, attached to it that represents the size of the tree if that node was the root of the tree. An example of such a tree is illustrated in Figure 5.3. This enables selection and promotion of the k^{th} lowest priority task in O(log n) time.

5.2.4. Scheduling Details

At the end of a scheduler time quantum due to a timer/interrupt event or the voluntary yielding of control by a task, EAS performs several important steps on a CPU core before scheduling the next task:-

1) The time-stamp counter (TSC) value denoting the end of the time quantum is recorded.
2) The number of CPU micro-operations performed during the previous time quantum is read from a performance counter and stored.
3) The latest energy measurements are synchronized with TSC values and recorded.
4) The latest OPJ values are calculated from data obtained in previous steps.
5) The OPJ value of the previously scheduled task is updated to the latest available value.

6) The task is inserted back into the red-black tree with appropriate updates to Size values.
7) Task promotion is performed if *n* time quanta have passed since the last task promotion.
8) The TSC value marking the beginning of the next time quantum is recorded.
9) The highest priority task is selected from tree and scheduled to execute on the CPU core till the time next quantum expires or the task voluntarily yields.

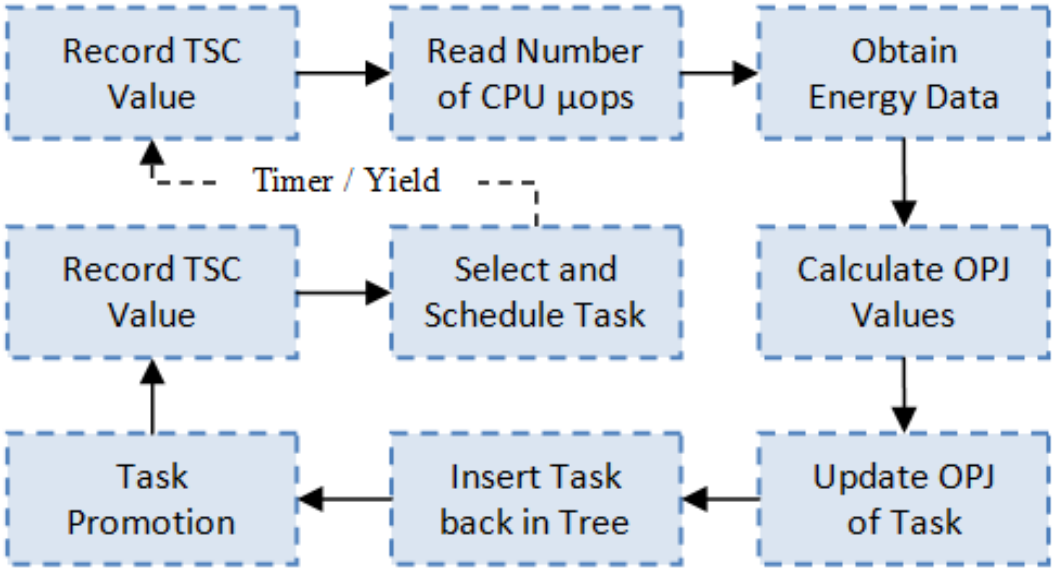

Figure 5.4. EAS performs a number of important steps between the scheduling of tasks. These steps are in addition to features borrowed from CFS.

The aforementioned scheduling steps are illustrated in Figure 5.4. Steps involving features borrowed directly from the Linux kernel or CFS infrastructure, such as timer handling, context switching logic, load balancing and task migration, are not included to preserve clarity.

5.3. Implementation

EAS is implemented on a multi-core x86 computing platform with changes to the task scheduling infrastructure of Linux kernel 2.6.32.

5.3.1. Modifications to CFS

EAS is implemented in the Linux kernel based on CFS code with important modifications:-

1) The vruntime for a task, which is used for recording the CPU time used by a task, is replaced with an OPJ variable with a default initial value of infinity.
2) An additional entry is maintained at each red-black tree node called Size to enable efficient task promotion.
3) Red-black tree task insertion and removal functions are modified to correctly update Size value at nodes.

4) While this parameter might be changed as required, this implementation applies a fixed scheduling time quantum of 4ms
5) The scheduler functions and control-flow are modified to conform to the steps in Figure 5.4.

5.3.2. Scheduling Classes

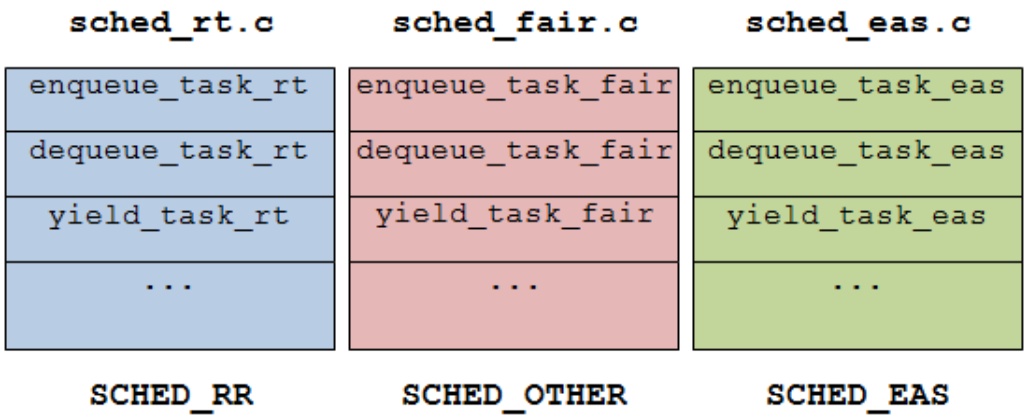

Figure 5.5. EAS creates its own scheduling class in addition to the standard scheduling classes provided by the Linux kernel.

The concept of modular schedulers, which is part of modern Linux kernels [Mol07], is utilized in the implementation of EAS. Thus, EAS co-exists with

other task schedulers in the Linux kernel where each scheduler has their important scheduling functions exposed to the kernel using standard scheduling classes that abstract away the tedious details of their implementation. EAS creates its own scheduling class as illustrated in Figure 5.5. The scheduling class based implementation enables EAS to be used for executing tasks that need to improve their energy efficiency by assigning them to the SCHED_EAS policy while still allowing a scheduler like CFS to co-exist and continue scheduling other tasks on the platform.

5.3.3. Platform Support

The implementation of EAS in this dissertation is based on modern x86 platforms and some of the features they provide [Ban04]. EAS relies on standard platform features in addition to the energy monitoring infrastructure provided by DEEP. The standard CPU performance monitoring unit is required along with the standard TSC. The performance monitoring unit provides performance counters that are needed for counting the number of μops that each CPU core executes during a task's scheduled time quantum while excluding PAUSE and NOP instructions as these don't represent actual work performed by a task. The TSC is

required for high resolution timing of platform events and for time synchronization of energy measurements.

5.4. Results

The primary objective for determining the effectiveness of EAS is to determine both energy efficiency benefits and to detect any possible degradation in performance due to introduction of EAS. Therefore, EAS is evaluated in terms of both energy efficiency and execution time compared to CFS for standard benchmarks. The investigations reported here utilize Linux kernel version 2.6.32 on an Intel x86 quad-core DEEP platform. Energy measurement infrastructure is added to the platform as described in Chapter 2. When measuring the energy consumption for CFS, measurements are not performed by the computing platform under observation. Instead, an offline version of DEEP is used with another identical platform with a DAQ to used collect energy measurements from the instrumentation of the platform under observation so that data acquisition overhead does not create errors in the data. For EAS this overhead is part of the data because the measurement infrastructure is an integral component of this scheduler.

5.4.1. Benchmark Selection

Eight applications have been selected to create a compact, but comprehensive suite of benchmarks that represent a wide range of important workloads on modern computing platforms:-

1) Apache is the most widely-deployed web server and is used in this dissertation with the TPC-W commercial multi-threaded benchmark for characterizing web servers [ACC02].
2) AES-256 is a modern federal government security standard that is used for data encryption [DR10].
3) Tpcc-mysql is a benchmark for MySQL databases based on TPC-C, which is a standard commercial benchmark for database transactions [CRF07].
4) Java LINPACK is the Java version of the standard supercomputing linear algebra benchmark called LINPACK [CRF07].
5) Bzip2 is an open-source data compression tool based on the widely-utilized Burrows-Wheeler transform [BK00].
6) BlogBench is a modern file system benchmark based on emulation of real-world file servers [ZYC13].

7) VirtualBox is the most commonly used open-source virtualization product and this dissertation utilizes it to run another instance of Linux kernel 2.6.32 on the platform as a virtual machine [Wat08].
8) Transaction Processing over XML (TPoX) is finance application based XML database benchmark based on real transactions and XML schema [NKS07].

The eight benchmarks' input workload is normalized such that each benchmark has an average execution time of 100s using CFS on the platform when no other benchmarks execute in parallel.

5.4.2. Energy and Performance Benefits

In Figures 5.6 through 5.9, SOLO-RUNS represent the case where each of the eight benchmarks is executed individually using CFS on the platform so that resource contention among benchmarks is absent. EAS represents the case where benchmarks begin execution in parallel and EAS is used to schedule them. CFS represents the case where CFS is used to schedule the benchmarks. Experiments for each case are repeated a thousand times and the mean values are reported in the figures. BEST-OBSERVED represents the minimum values for energy

consumption and execution time observed over all thousand repetitions of the CFS and EAS experiments. This value indicates the best possible schedule that is achieved in presence of resource contention among tasks. TOTAL represents the values for execution of all benchmarks while the individual values represent the decomposition of TOTAL among each benchmark.

It is important to note that the energy data for individual benchmarks may not represent the energy consumption due to that benchmark, but the energy consumption that occurs during execution of the benchmark. This is because there can be multiple benchmarks executing in parallel and each of these impact the energy consumption at an instant.

Figures 5.6 and 5.7 illustrate the results of the comparisons between EAS and CFS. EAS demonstrates a large benefit over CFS with an execution time improvement of 24.7% and energy efficiency improvement of 30.2% averaged over all benchmarks. Furthermore, energy consumption and execution time with EAS are much closer to BEST-OBSERVED values than the values obtained when using CFS with benefits being observed for each and every benchmark.

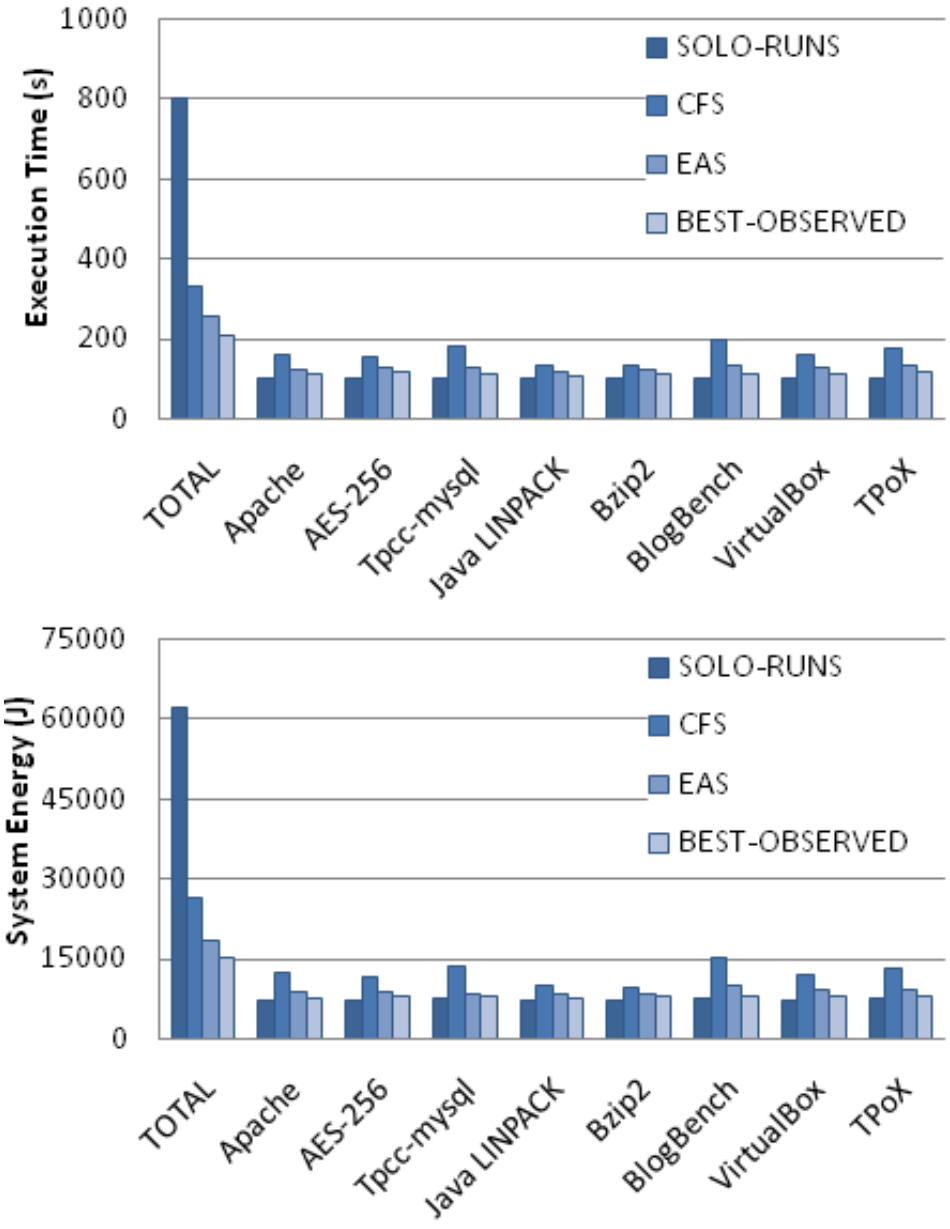

Figure 5.6. Execution time and platform energy consumption for all benchmarks.

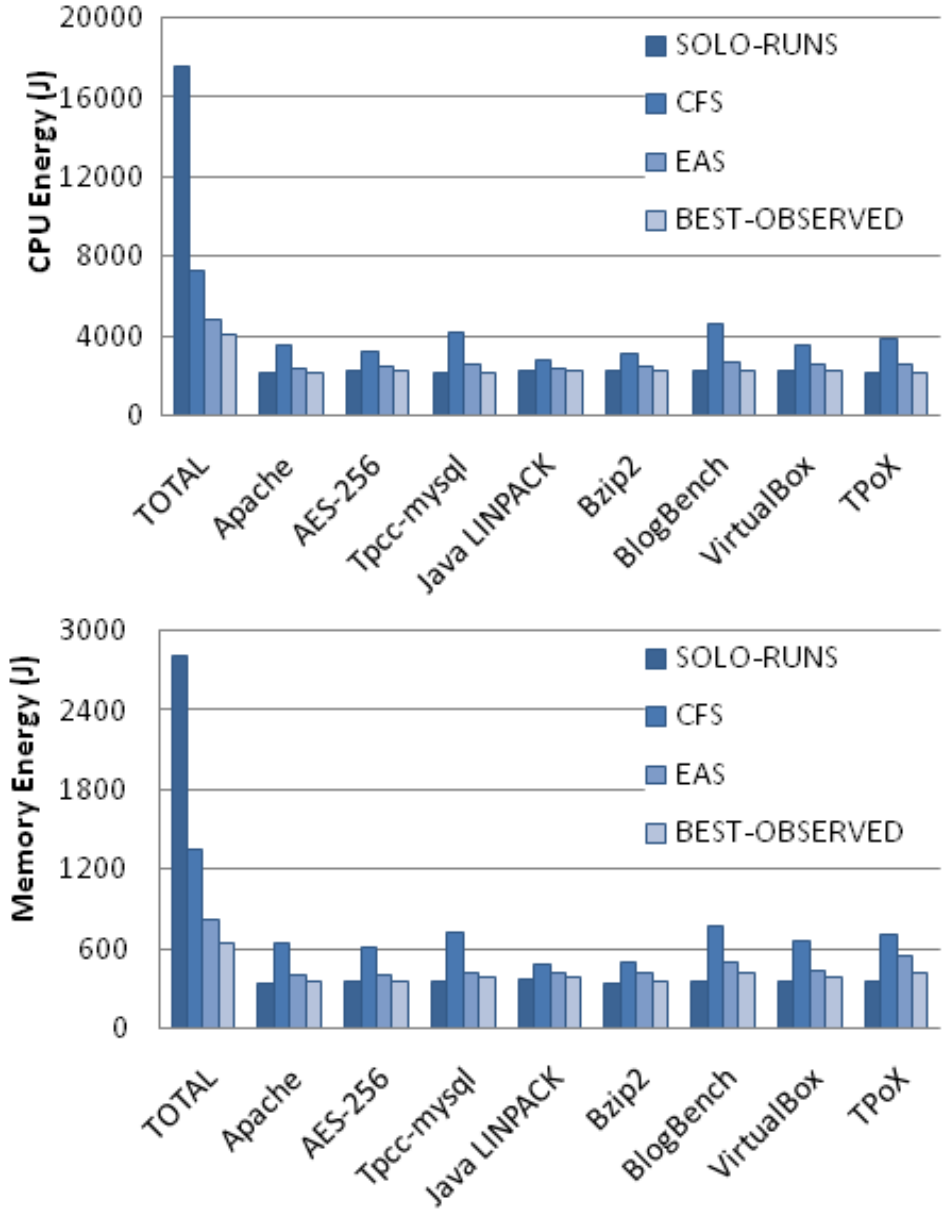

Figure 5.7. CPU and memory energy consumption for each of the benchmarks.

5.4.3. Impact of Resource Contention

The Intel quad-core platform used in this evaluation offers an additional experiment capability. Specifically, the four CPU cores are arranged in two pairs with each pair sharing a level-2 (L2) cache. Therefore, two tasks scheduled on both of the cores of a pair contend for the shared L2 cache. However, if the two tasks are scheduled on cores from different pairs then L2 cache contention is absent. This enables an evaluation where the eight benchmarks are executed using only two of the four cores on the platform. In one case the benchmarks are allowed to execute on a pair of cores sharing an L2 cache and in the other case the benchmarks are executed on only one core of each pair with separate L2 cache. Both CFS and EAS were used in the experiments and the results are illustrated in Figures 5.8 and 5.9. The execution time and platform energy consumption are larger in the case where L2 cache is shared when using CFS, but EAS effectively reduces the impact of L2 cache contention because both the energy consumption and execution time approach the values when L2 cache is not shared. This further demonstrates the co-run degradation that can occur because of resource contention and the effectiveness of EAS in mitigating this degradation in performance and energy efficiency.

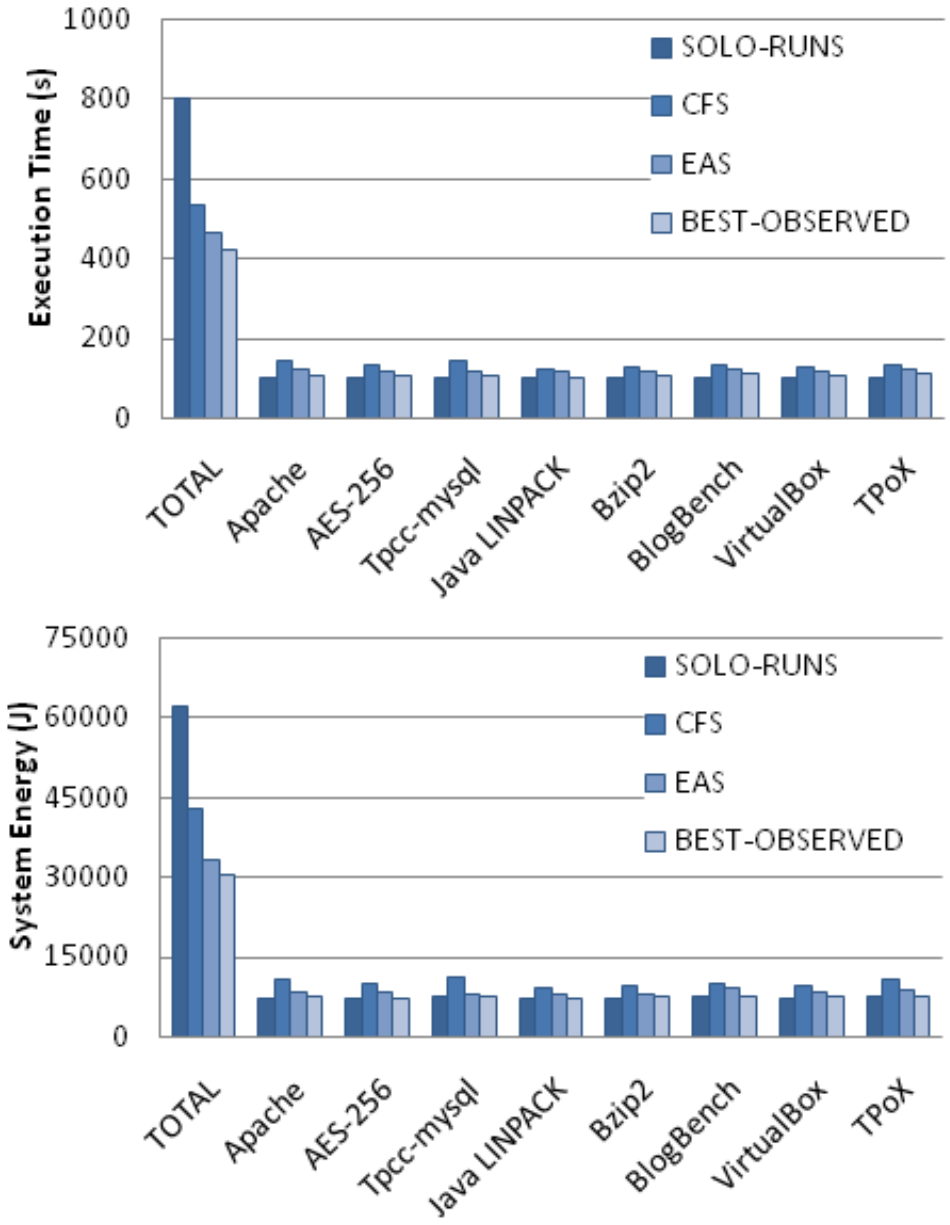

Figure 5.8. Execution time and energy consumption without L2 cache contention.

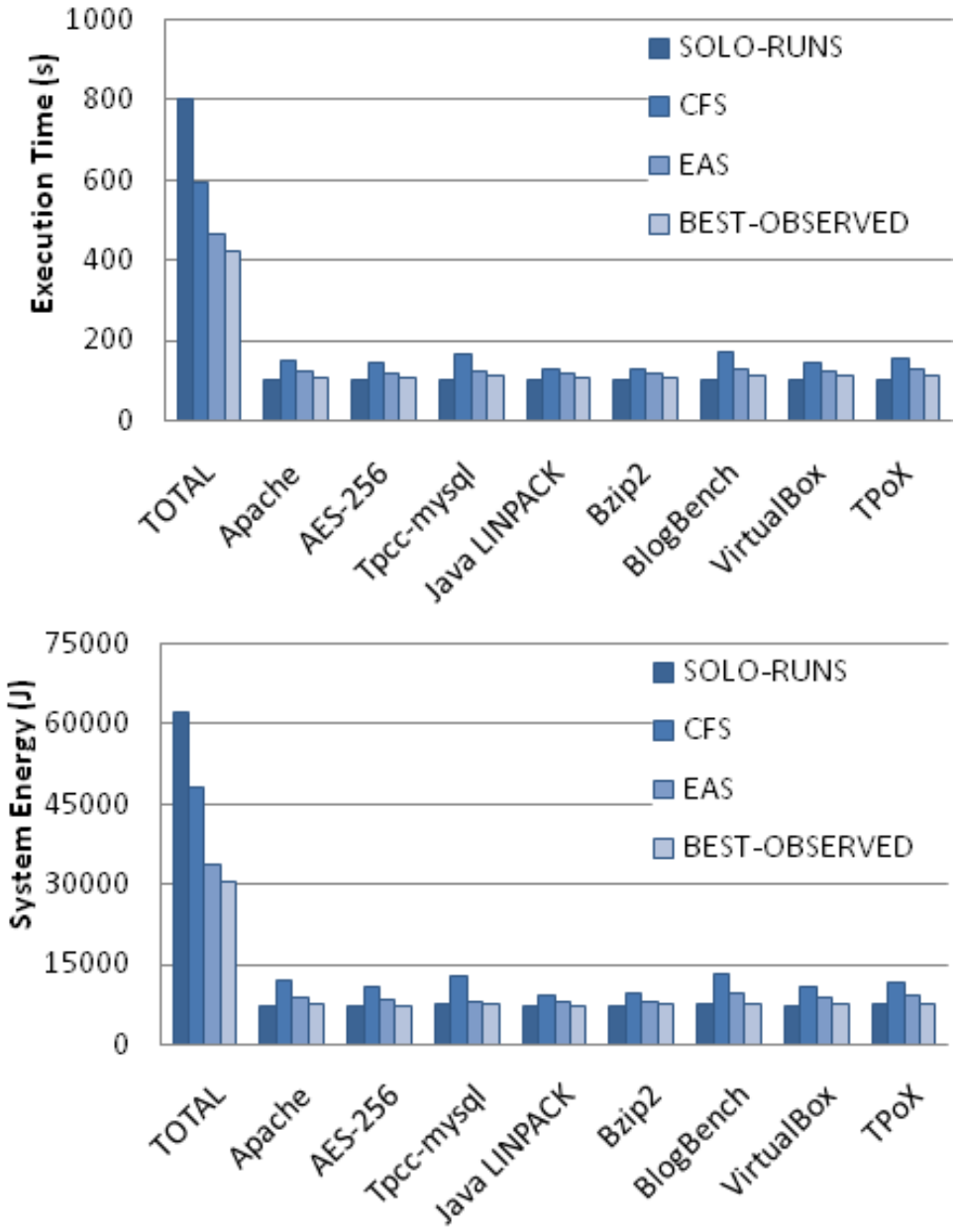

Figure 5.9. Execution time and energy consumption with L2 cache contention.

5.5. Conclusions

This chapter presented the Energy Aware Scheduler (EAS) that exploits event-synchronized energy measurements to advance the energy efficiency and performance of multi-core computing platforms through efficient co-scheduling of tasks. The DEEP platform's real-time energy measurements and CPU performance counters are used by EAS to detect and avoid resource contention among tasks at run-time. This alleviates the inefficiencies caused by co-run degradation between tasks.

EAS is implemented by extensions to existing Linux kernel and task scheduling infrastructure through the use of scheduling classes to allow EAS to co-exist with other task schedulers in the operating system. Energy measurement infrastructure for the implementation is integrated directly at the platform-level.

Evaluation of EAS on a quad-core x86 system over a wide range of commonly used benchmark applications demonstrates significant improvement in energy efficiency (30.2%) and execution time (24.7%) compared to the standard Linux task scheduler (CFS). Furthermore, both execution time and energy efficiency

benefits were observed for each and every benchmark application that was investigated.

CHAPTER 6

CONCLUSION

This chapter concludes the dissertation. The chapter begins with an overview of limitations of the work described herein and details possible solutions to these limitations as promising directions for future work in Section 6.1. Section 6.2 summarizes the dissertation's results and presents conclusions.

6.1. Limitations and Future Work

The following limitations of the work presented herein have been indentified and possible future solutions are also listed:-

1) The 100μs sampling resolution of the DEEP platforms, while sufficient for characterizing the energy consumption of most software applications and a large advance over previous work, cannot monitor the energy usage of extremely brief events such as interrupt handling. This is a fundamental limit imposed by the measurement instrumentation and synchronization signal

latency in the DEEP platforms. A possible solution is the integration of high frequency power metering as part of the computing platform's architecture itself and this is now being explored [NRA11]. Leading computing platform manufacturers, such as Intel and Qualcomm, have realized the importance of direct energy measurement in optimizing computing platform energy efficiency. Recently a limited amount of power metering is being provided using machine specific registers (MSRs) in the latest CPU architectures [NRA11]. Extending the use of these MSRs and including them in the next generation DEEP platform architecture would be a very promising direction for future work as it would reduce measurement overhead, eliminate external instrumentation and enable very low latency energy measurement while allowing DEEP's software utilities, such as energy calipers, to still be applied. Furthermore, DEEP provides a vast amount of untapped potential for measurement and optimization of software energy efficiency in multiple computing domains, such as networking protocols, operating system internals, security/privacy frameworks and other user applications.

2) The HDD energy estimation system's per-process energy attribution system is limited to cases where a single process has ownership or a file. Thus, cases

where multiple processes write to the same file or same data block would require a different mechanism of attribution as the inode number is unique for the data block and not for each of the writing processes. Hence, the energy consumption for the data block must be distributed between each of the processes using a different attribution scheme. However, such shared block writes are rare and don't create a large impact on the accuracy of the estimation system. Also, the HDD used in our investigation only supports idle and active states, but certain HDDs also have a standby mode that is activated during long periods of inactivity. This requires a more complex model that accounts for transitions to and from the standby mode of the HDD.

3) The adaptive compression algorithm DEEPcompress is suited to upload to data files when direct energy measurements are available. For platforms without power measurement capabilities, different metrics for estimating computation and communication energy costs need to be utilized. This requires a comprehensive characterization of system and network conditions. Such a framework for adaptive compression has been studied recently [Pet13] and confronts a number challenges compared to direct measurement of energy.

4) The energy aware task scheduler EAS is designed to improve the energy efficiency of applications on multiprocessing computing platforms. However, certain applications have requirements beyond performance and energy efficiency that are of primary concern, such as interactivity, response times, deadlines and fairness [WCJ09]. Extending EAS with mechanisms that can limit unfairness and loss of interactivity for processes while sacrificing some energy efficiency or performance would the task scheduler applicable to wider range of applications. An example of such a mechanism was provided in the previous chapter using task promotion, wherein starvation of inefficient processes was alleviated through a scheme for promoting lower priority tasks. Furthermore, investigation of metrics that use performance measures (such as cache misses) instead of OPJ to detect resource contention would be valuable since these can be deployed without energy measurement capabilities.

6.2. Dissertation Conclusions

This dissertation presented the architecture and implementation of the DEEP platforms. These platforms provide accurate and low-overhead time-synchronized energy measurements for commodity computing systems and important

components like the CPU, memory modules and secondary storage. With a design that enables rapid assembly using standard computing hardware and software, multiple DEEP platforms have been implemented as a networked tested that is remotely accessible to research groups and students. In addition to direct energy measurement capabilities, DEEP includes a set of innovative software utilities called energy calipers that enable the measurement of energy associated with execution of sections of software code. This makes DEEP ideal for investigation of the energy efficiency of both application and operating system code. Furthermore, DEEP provides an accurate and low overhead energy estimation system for modern secondary storage hardware to enable estimation of energy associated with deferred file system and storage operations. The energy for these deferred events is attributed directly to the causative processes making the estimation system applicable to exploration of the energy consumption by applications that perform a large amount of activity on the secondary storage device.

The DEEP platforms were also applied to the problem of network transport. An adaptive compression algorithm called DEEPcompress is developed. The

algorithm adapts to changing network and system conditions to select energy efficient compression schemes during the upload of data file over a network link. The DEEP platform's direct real-time energy measurements guide the compression scheme choice. Significant energy savings (up to 38%) are observed when using the DEEPcompress algorithm compared to other commonly utilized static compression schemes such as Gzip, bzip2 and XZ.

An energy aware task scheduler is developed using the capabilities provided by the DEEP platforms. This task scheduler uses data from the CPU's performance monitoring unit and energy measurements from DEEP to improve the energy efficiency of multi-core computing platforms. This is done through the use of a metric called OPJ for run-time detection and avoidance of the co-run degradation that takes place due to resource contention among tasks. The scheduler is implemented on a standard Linux distribution on a quad-core computing platform. Comparison of the scheduler to the standard task scheduler adopted by the Linux operating system demonstrates large energy (about 30%) and execution time (about 24%) benefits over a broad range of common benchmark applications.

BIBLIOGRAPHY

[AAF09] M. Allalouf, Y. Arbitman, and M. Factor, "Storage modeling for power estimation." ACM International Systems & Storage Conference, 2009.

[ACC02] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and implementation of dynamic web site benchmarks." IEEE International Workshop on Workload Characterization, 2002.

[BA06] K. Barr and K. Asanovic, "Energy-aware lossless data compression."ACM Transactions on Computer Systems, 2006.

[Ban04] S. Bandyopadhyay, "A study on performance monitoring counters in x86-architecture." Indian Statistical Institute, 2004.

[BBV09] N. Balasubramanian, A. Balasubramanian, and A. Venkatramani, "Energy consumption in mobile phones: a measurement study and implications for network applications." ACM SIGCOMM Internet Measurement Conference, 2009.

[ber04] berliOS, "Spew: An I/O performance measurement and load generating tool." http://spew.berlios.de

[Bha03] S. Bhattacharya, "Dynamic probes – debugging by stealth." Linux Conference Australia, 2003.

[BK00] B. Balkenhol and S. Kurtz, "Universal data compression based on the burrows-wheeler transformation: theory and practice." IEEE Transactions on Computers, 2000.

[BTM00] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations." International Symposium on Computer Architecture, 2000.

[BZF10] S. Blagodurov, S. Zhuravlev, and A. Fedorova, "Contention-aware scheduling on multicore systems." ACM Transactions on Computer Systems, 2010.

[CH10] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." USENIX Annual Technical Conference, 2010.

[CRF07] R. Cheveresan, M. Ramsay, C. Feucht, and I. Sharapov, "Characteristics of workloads used in high performance and technical computing." ACM International conference on Supercomputing, 2007.

[DB11] R. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems." ACM Computing Surveys, 2011.

[DR10] J. Daemen and V. Rijmen, "The first 10 years of advanced encryption." IEEE Security and Privacy, 2010.

[DS86] R. D'Agostino, and M. Stephens, "Goodness-of-fit techniques." New York: Marcel Dekker, 1986.

[DZ11] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems." International Conference on Mobile Systems, Applications, and Services, 2011.

[ERK06] D. Economou, S. Rivoire, and C. Kozyrakis, "Full-system power analysis and modeling for server environments." Workshop on Modeling, Benchmarking, and Simulation, 2006.

[FM08] P. Ferragina and G. Manzini, "Burrows–Wheeler transform." Encyclopedia of Algorithms, Springer US, 2008.

[FPR12] A. Fujimoto, P. Peterson, and P. Reiher, "Comparing the power of full disk encryption alternatives." International Green Computing Conference, 2012.

[FS99] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications." IEEE Workshop on Mobile Computing Systems and Applications, 1999.

[Gra08] J. Gray, "Go green, save green with Linux." Linux Journal, 2008.

[Hin99] R. Hinze, "Constructing red-black trees." Workshop on Algorithmic Aspects of Advanced Programming Languages, 1999.

[HSR08] A. Hylick, R. Sohan, A. Rice, and B. Jones, "An analysis of hard drive energy consumption." IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2008.

[IM03] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: methodology and empirical data." IEEE International Symposium on Microarchitecture, 2003.

[KB99] B. Knutsson and M. Bjorkman, "Adaptive end-to-end communication for variable-bandwidth communication." Computer Networks, 1999.

[KC01] C. Krintz and B. Calder, "Reducing delay with dynamic selection of compression formats." International Symposium on High Performance Distributed Computing, 2001.

[KOI10] Y. Kaneda, T. Okuhira, T. Ishihara, K. Hisazumi, T. Kamiyama, and M. Katagiri, "A run-time power analysis method using OS-observable parameters for mobile terminals." International

Conference on Embedded Systems and Intelligent Technology, 2010.

[KS05] C. Krintz and S. Sucu, "Adaptive on-the-fly compression." IEEE Transactions on Parallel and Distributed Systems, 2005.

[KS92] J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system." ACM Transactions on Computer Systems, 1992.

[MAC11] J. McCullough, Y. Agarwal, J. Chandrashekhar, S. Kuppuswamy, A. Snoeren, and R. Gupta, "Evaluating the effectiveness of model-based power characterization." USENIX Annual Technical Conference, 2011.

[MB05] A. Merkel and F. Bellosa, "Event-driven thermal management in SMP systems." Workshop on Temperature-Aware Computing, 2005.

[MHY06] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. Kaiser, "The low power energy aware processing (LEAP) system." International Conference on Information Processing in Sensor Networks, 2006.

[Mol07] I. Molnar, "Modular scheduler core and completely fair scheduler (cfs)." Linux Kernel Mailing List, 2007.

[MP85] D. McIntyre and M. Pechura, "Data compression using static Huffman code-decode tables." Communications of the ACM, 1985.

[MPK06] A. Mavinakayanahalli, P. Panchmukhi, J. Keniston, A. Keshavamurthy, and M. Hiramatsu, "Probing the guts of Kprobes." Linux Symposium, 2006.

[MS06] R. Maddah and S. Sharafeddine, "Energy-aware adaptive compression for mobile-to-mobile communications." IEEE Symposium on Spread Spectrum and Applications, 2006.

[NDR08] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: practical power management for enterprise storage." USENIX Conference on File and Storage Technologies, 2008.

[NF04] E. Nightingale and J. Flinn, "Energy-efficiency and storage flexibility in the blue file system." USENIX Symposium on Operating Systems Design and Implementation, 2004.

[NKS07] M. Nicola, I. Kogan, and B. Schiefer, "An XML transaction processing benchmark." ACM SIGMOD International Conference on Management of Data, 2007.

[NRA11] A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power management architecture of the 2nd generation Intel® Core™ microarchitecture, formerly codenamed Sandy Bridge." Hot Chips, 2011.

[Pet13] P. Peterson, "Datacomp: locally-independent adaptive compression for real-world systems." Doctoral Dissertation, University of California, Los Angeles, 2013.

[PFW11] G. Perrucci, F. Fitzek, and J. Widmer. "Survey on energy consumption entities on the smartphone platform." IEEE Vehicular Technology Conference, 2011.

[Phi13] M. Philips, "The cloud begins with coal. Big data, big networks, big infrastructure and big power: An overview of the electricity used by the global digital ecosystem." Digital Power Group, 2013.

[PMW09] S. Pelley, D. Meisner, T. Wenisch, and J. VanGilder. "Understanding and abstracting total data center power." Workshop on Energy-Efficient Design, 2009.

[Pra03] M. Prasad, "WattProbe software-based empirical extraction of hardware energy models." Master's Thesis, Computer Science Dept., Stony Brook University, 2003.

[PS05] C. Pu and L. Singaravelu, "Fine-grain adaptive compression in dynamically variable networks." IEEE International Conference on Distributed Computing Systems, 2005.

[PSK11] P. Peterson, D. Singh, W. Kaiser, and P. Reiher, "Investigating energy and security trade-offs in the classroom with the Atom LEAP testbed." USENIX Cyber Security Experimentation and Test, 2011.

[Ras09] N. Rasmussen, "Allocating data center energy costs and carbon to IT users." APC White Paper, 2009.

[RCJ05] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMsim2: a cycle accurate memory system simulator." IEEE Computer Architecture Letters, 2005.

[RLG08] K. Rajamani, C. Lefurgy, S. Ghiasi, J. Rubio, H. Hanson, and T. Keller, "Power management for computer systems and datacenters." International Symposium on Low Power Electronics and Design, 2008.

[RSI08] N. Ravi, J. Scott, and L. Iftode, "Context-aware battery management for mobile phones." IEEE International Conference on Pervasive Computing and Communications, 2008.

[RSM09] S. Ryffel, T. Stathopoulos, D. McIntire, W. Kaiser, and L. Thiele, "Accurate energy attribution and accounting for multi-core systems." Technical Report, Center for Embedded Network Sensing, University of California, Los Angeles, 2009.

[RSR07] S. Riviore, M. Shah, P. Ranganathan, and C. Kozyrakis, "JouleSort: a balanced energy-efficiency benchmark." ACM International Conference on Management of Data, 2007.

[SBM09] K. Singh, M. Bhadauria, and S. McKee, "Real time power estimation and thread scheduling via performance counters." ACM SIGARCH Computer Architecture News, 2009.

[Sca06] J. Scaramella, "Worldwide server power and cooling expense 2006-2010 forecast." International Data Corporation, 2006.

[SCF03] B. Slechta, D. Crowe, B. Fahs, M. Fertig, G. Muthler, J. Quek, F. Spadini, S. Patel, and S. Lumetta, "Dynamic optimization of micro-operations." IEEE International Symposium on High-Performance Computer Architecture, 2003.

[SGY11] B. Smith, R. Grehan, T. Yager, and D. Niemi, "Byte-unixbench: a unix benchmark suite." 2011.

[Shi05] S. Shiva, Advanced Computer Architectures. CRC Press, 2005.

[SK12] D. Singh and W. Kaiser. "Energy efficient network data transport through adaptive compression using the DEEP platforms." IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2012.

[SMK08] T. Stathoupoulos, D. McIntire, and W. Kaiser, "The energy endoscope: real-time detailed energy accounting for wireless sensor nodes." International Conference on Information Processing in Sensor Networks, 2008.

[SPR10] D. Singh, P. Peterson, P. Reiher, and W. Kaiser, "The Atom LEAP platform for energy-efficient embedded computing: architecture, operation, and system implementation." Technical Report, University of California, Los Angeles, 2010.

[SZD08] K. Shen, M. Zhong, S. Dwarkadas, C. Li, C. Stewart, and X. Zhang, "Hardware counter driven on-the-fly request signatures." ACM SIGARCH Computer Architecture News, 2008.

[WAB10] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava, "A case for opportunistic embedded sensing in the presence of hardware power variability." International Conference on Power Aware Computing and Systems, 2010.

[Wat08] J. Watson, "Virtualbox: bits and bytes masquerading as machines." Linux Journal, 2008.

[WCJ09] S. Wang, Y. Chen, W. Jiang, P. Li, T. Dai, and Y. Cui, "Fairness and interactivity of three CPU schedulers in Linux." IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2009.

[WL08] D. Woo and H. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era." IEEE Computer, 2008.

[WM09] L. Wang and J. Manner, "Evaluation of data compression for energy-aware communication in mobile networks." International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009.

[WTK08] C. Wong, I. Tan, R. Kumari, J. Lam, and W. Fun, "Fairness and interactive performance of o (1) and cfs linux kernel schedulers." IEEE International Symposium on Information Technology, 2008.

[WZ91] A. Wyner and J. Ziv, "Fixed data base version of the Lempel-Ziv data compression algorithm." IEEE Transactions on Information Theory, 1991.

[XJJ12] X. Xie, H. Jiang, H. Jin, W. Cao, P. Yuan, and L. Yang, "Metis: a profiling toolkit based on the virtualization of hardware performance counters." Human-centric Computing and Information Sciences, 2012.

[YKJ12] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: application energy metering framework for android smartphone using kernel activity monitoring." USENIX Annual Technical Conference, 2012.

[YLV13] J. Yan, C. Lonappan, A. Vajid, D. Singh, and W. Kaiser, "Accurate and low-overhead process-level energy estimation for modern hard disk drives." IEEE International Conference on Green Computing and Communications, 2013.

[YVK00] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "The design and use of SimplePower: a cycle-accurate energy estimation tool." Design Automation Conference, 2000.

[ZDD04] Q. Zhu, F. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing energy consumption of disk storage using power-aware cache management." IEEE International Symposium on High-Performance Computer Architecture, 2004.

[ZDF07] X. Zhang, S. Dwarkadas, G. Folkmanis, and K. Shen, "Processor hardware counter statistics as a first-class system resource." HotOS, 2007.

[ZEL02] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat, "ECOSystem: managing energy as a first class operating system resource." ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.

[ZSB13] S. Zhuravlev, J. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques." IEEE Transactions on Parallel and Distributed Systems, 2013.

[ZSG03] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang., "Modeling hard-disk power consumption." USENIX Conference on File and Storage Technologies, 2003.

[ZYC13] X. Zhao, J. Yin, Z. Chen, and S. He, "Workload classification model for specializing virtual machine operating system." IEEE International Conference on Cloud Computing, 2013.