Dynamic Classification of Repetitive Jobs In Linux For Energy-Aware Scheduling: A Feasibility Study

Shane Case, Kanad Ghose
SUNY Binghamton
Binghamton, NY, USA
{shane, ghose}@cs.binghamton.edu

***Abstract*—The workload offered to a typical server consists of many repeated tasks. We present and evaluate a feasibility study that shows how repetitive server workloads can be exploited to enhance the server and CPU energy savings realized by state-of-the-art linux power governors. To minimize dramatic modifications to the web server and the core kernel scheduler we exploit the forensic logging capabilities of the Apache server to collect workload specific information and to schedule requests. We use a daemon to collect the classification statistics and control the dynamic voltage frequency scaling (DVFS) setting of the kernel to batch schedule requests with similar characteristics and thus amortize the energy and performance overhead of making changes to the DVFS settings. Our experimental results show that an energy savings of up to ten percent can be realized for the server on the workloads generated by the SPECweb2005 benchmarks.**

Keywords-Energy Conservation; Power Savings; Process Management;Web Server; Performance Experiment; Scheduling; Measurement, Power Measurements, Energy Reduction

I. INTRODUCTION

Data centers are an essential part of a modern nation's infrastructure. Historically, the US has led the rest of the world in the development, deployment and widespread use of data centers in supporting a plethora of services and activities. A 2007 report places the energy expenditures associated with US data centers in 2006 at upwards of 1.5% of the total electricity expenditures of the nation [1]. These figures are likely to grow significantly as data centers are deployed widely to support a growing and wider variety of cyberservices. The hardware devices used in a data center continue to be more energy efficient but because of the annual increase in utility costs (8% to 11% annually), the cooling costs remain relatively stagnant [2]. This situation is therefore a crisis in the making and has the potential of impeding the growth rate of data centers and their increasing use in everyday life for the betterment of humanity.

The primary technique used in Linux servers is a power-performance governor that exercises the *DVFS* logic within the processor. At a low level of core utilization, the CPU clock frequency can be slowed down and the core supply voltage is simultaneously reduced. Just reducing the clock frequency does not result in a savings of the total energy consumption of the CPU, as the application still needs the same number of clock cycles to complete. Thus, the energy expended per clock cycle is lowered by reducing the core supply voltage. The *DVFS* mechanism in a modern processor has a time overhead as well as energy overhead, as the CPUs voltage and frequency cannot be changed instantaneously.

The technique presented in this paper examines the feasibility of classifying repetitive server jobs into classes (two such classes are used in the present study), with each class requiring a specific *DVFS* setting. One would expect such repetitive jobs to be abundant in a typical server workload: the same scripts or transactions are often executed repeatedly. For example, in a server supporting banking transactions, the same scripts are executed on queries that determine account balances, for queries that post a charge to an account etc.

In the proposed technique, as new server jobs come in, they are classified into one of the two possible classes based on their observed execution characteristics. An already classified job, identified using the address of the executable, is batch scheduled on a core that is set at the matching *DVFS* step, to amortize the overhead of changing the *DVFS* settings. Appropriate timeout mechanisms are used to avoid undue prolongation of the service times for jobs. Energy savings of 5% to 10% over Linux on-demand governor are realized in running the SPECweb benchmarks in a prototype implementation running the Apache server that supports job classification (in conjunction with a daemon) and passes on the classification to the kernel. This 5% to 10% energy savings at the level of an individual server translates to an energy savings of 10% to 20% for an entire data center, where the utility costs of operating the cooling equipment equals the utility costs of operating the IT equipment [1] [3] [4].

The rest of the paper describes related work in Section 2, some background information on power governors and the Apache web server in Section 3. In Section 4, we describe the proposed scheme and some relevant implementation details, Section 5 presents the experimental setup and the measured results. Section 6 represents our conclusions.

II. RELATED WORK

There is a plethora of work that has examined data center job scheduling strategies in general using simulators, synthetic job characteristics or actual server workload

 ISBN: 978-1-61208-259-2

trace data. Most of this body of work, exemplified by [5] focuses on scheduling jobs to servers. We address the problem of scheduling jobs in an energy-efficient manner within a server. However, very few server-local energy-aware scheduling techniques have targeted Linux or have actually described real implementations, validated with actual power measurements, and these are the work that we focus on in this section. In particular, these techniques do not explicitly exploit the characteristics of repetitive workloads.

Another way of managing the energy consumption of jobs allocated to a server is to limit the energy expended by the virtual machines running on a physical host. For example, the work of [6] describes an approach for allocating energy budgets to virtual machines. Such VM energy budgets are not easy to implement, as energy expended by a VM is not easy to track and control; energy dissipation in many related components are ignored in simplifications that are used. Allocation of jobs is also studied in [7], but this study focuses primarily on performance increases in a cluster environment, not on energy savings for a single server.

In general, emerging solutions have a number of potential limitations:

- The energy and performance overhead associated with job rescheduling and VM management and server-local scheduling overhead are ignored. The communication infrastructures within a data center are heavily utilized and are prone to congestion, resulting in significant added energy dissipation if jobs are rescheduled.
- A simple rescheduling of the jobs may not make the most energy-efficient use of the servers and racks - the operating configurations of such servers have to be continuously adapted to fit the characteristics of the workload.

The work of [8] classifies workload for long-lived connection-oriented services and shows, using trace data, how the information collected can be exploited for server provisioning and scheduling to save energy, minimizing the loss in live sessions in the process. A similar study is clearly needed to validate the repetitive nature of server workloads, as exploited in the proposed technique, but that study is beyond the scope of this paper. Another approach for exploiting workload characteristics in server provisioning and scheduling is presented in [9]. Global system monitoring facilities, such as IBM's Tivoli [10], [11] can collect energy-performance characteristics of jobs, including repetitive jobs, and can, in theory, use that information to schedule jobs globally and locally within a server. However, any automated scheduling facility that uses the collected information for local scheduling is missing at this point. The proposed technique not only collects the job's classification data but also integrates it with scheduling.

In [12], a framework called Koala for a model for predicting the energy and performance characteristics of a program for different *DVFS* settings and makes use of the predictions for energy-aware job scheduling. Koala requires a recalibration of the power model on any changes in the physical host configuration. In contrast, we use higher level job classification parameters to classify repetitive jobs and thus our approach does not depend on host-specific attributes. Koala can be extended to account for any *DVFS* overhead, in terms of performance and energy, repetitive jobs can in fact use the Koala framework as well.

A job classification scheme that characterizes map-reduce type jobs into compute-bound and I/O bound is introduced in [13], using a single instance of a map task and a single instance of a reduce task for classifying the set of map and reduce tasks that make up a single workload. However, results, experimental or otherwise nor any quantitative analysis is presented to assess the benefits of the proposed scheduling strategy.

In [14], the authors present a methodology for characterizing and classifying known background workload for Google's back end server in terms of their CPU and memory demand. The authors of [14] also describe very briefly the use of this classification in background job scheduling. The integration of the scheduling technique into an existing framework and its benefits are unclear.

III. BACKGROUND

A. CPU Frequency Governor

In the Linux Kernel, the most commonly used CPU frequency governor is the *Ondemand* governor. This governor gradually increases or decreases the frequency according to the current system load. The system load is measured from the scheduler, not the "loadavg" present in /proc. System load is sampled at 1000 times the frequency latency change value. Scaling is performed when the system load exceeds 95 percent and the CPU frequency is changed to the highest value. When system load decreases, the frequency is gradually stepped down.

B. The Apache Web Server

The main focus of this study will be on the pre-forking architecture of the Apache Web server [15]. Our main reason for focusing on the Apache web server are as follows. First, using the Apache web server's request scheduling capabilities and the ability to use its forensic logging mechanism to capture request specific statistics, removes the need to design any modified request scheduling and statistics collection features. Second, our approach permits us to accomadate independent changes made to both Apache and the core kernel very easily.

The performing architecture consists of the Apache parent process creating multiple processes called children to handle the incoming requests on the web server. Upon startup, the parent process will fork off a configurable number of children that will handle tasks assigned to them by the parent

 ISBN: 978-1-61208-259-2

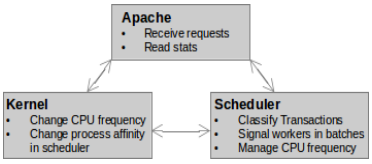

Figure 1. Software Components in the Proposed Technique

process. When an HTTP request is received, each element of the requested document is a transaction from the view of the web server. If the ForensicLog module is enabled, each transaction will be double logged by the web server, once upon reception of the request and once upon its completion.

IV. The Proposed Technique

In order to exploit the repetitive nature of transactions serviced by the web server, two components are required. The first component is a modified ForensicLog module that is loaded into Apache web server at runtime, and the second is a system daemon that will process the data gathered by the logging mechanism.

The system daemon, or power manager must be an extremely lightweight process while executing in order to avoid skewing the statistics of the hardware that is being monitored, and consuming overhead power. The overhead of any type of busy waiting, such as spin locks or sleeping, is not tolerable, as a process utilizing either of these will consume more power in busy waiting. Busy waiting does not render the CPU as idle, and will show up as CPU usage, which can affect the decision of the power manager to enter into a sleep state. Therefore, the synchronization method used to communicate between Apache web server and our power manager will be signals. The benefit of signals is twofold, as signals are a lightweight inter-process communication method, and adding signals to the Apache interface will require very little code change. The bulk of the code change will be required in the power manager itself, by simply adding a signal mask to the power management process. The power manager will then block until a signal is received from Apache.

There will only be two signals required to achieve our goal of communication. In this case the signals SIGUSR1 and SIGUSR2 will be used. SIGUSR1 will serve as the notification that a transaction has been received by Apache and processing will begin. SIGUSR2 will serve as notification that the transaction has been completed. The power manager can also read an integer value that is passed along with the signal which will serve to identify the current entry in the shared memory segment that it will be profiling. On the power manager side, SIGUSR1 will be the notification to begin profiling and search our statistic table to see if the occurring transaction has already been previously profiled. If the transaction has already been profiled, profiling can be terminated, squashing any statistics that have been gathered. The power manager can then apply the appropriate power profile, according to the transaction categorization in the statistic table. If the transaction has not been previously profiled, the counters have already begun before the search was conducted, and can then be stopped when SIGUSR2 is received. Once the statistics have been gathered, the numbers must be analyzed and a category assigned to that transaction before the transaction may be added to the statistic table.

A. Job Classification using the ForensicLog

When the web server initially receives a transaction, an entry is made in the ForensicLog via the *log_before()* function in ForensicLog. It is at this point that we modify the Log mechanism to not only continue its original logging duties, but to classify the task. At the time of the log entry, the parent Apache process has received the header of the server request and delegated processing to the active listening child. The log entry is made in the log file by the child process that is processing the request. It is at this point the ForensicLog is modified to monitor the current PID, which is the active child.

The statistics that are monitored for the purpose of system resource usage discovery are found in /proc/PID/schedstat, where PID is the currently running child's PIDİn these statistics, the CPU time spent and the I/O wait time spent is discovered about the currently running transaction. These metrics are reported on a per PID basis by the kernel's CFS scheduler. It is these two metrics that are used to profile a particular transaction. These metrics will be stored in a shared memory segment for communication to the classification daemon. This memory segment is locked using a mutex while the child process is acquiring the data for its transaction. When the metrics are gathered, the shared memory segment is unlocked, and the classification daemon is sent a *SIGUSR1* signal containing the index of the transaction of the signaling child process in the statistics table (Fig. 2). The child process will then enter a sleep state for a short period of time, via *sigtimedwait()*. The child sets its maximum timeout value to 2 seconds. The reason for this maximum timeout value is explained in the following section. Experiments were performed on different values and showed that the lower the timeout value, the lower the power savings percentage.

Upon either expiration of either the child's sigtimedwait() timeout value, or scheduling of the child by the classification daemon, the child will then process its assigned transaction. Upon completion of its transaction, the child will return to the ForensicLog module to log the transaction completion via the *log_after()* function. This function is altered to repeat what was done in the *log_begin()* function, except it will calculate its actual system resource metrics by using the values logged at the beginning of the transaction as a zero value. The CSD is again signalled, this time using a

 ISBN: 978-1-61208-259-2

SIGUSR2 value. The child process is then returned to the idle wait queue by the internal Apache scheduling mechanism.

B. Classification and Scheduling Daemon (CSD)

The CSD serves as the scheduler for the Web Server by scheduling child processes with a similar workload back to back. Their purposes of scheduling the children are to execute transactions of similar workload in batches, rather than as the transactions are received. Upon startup of the CSD, it will change the system power management policy to be in its lowest active power state.

The CSD is signalled upon every transaction beginning and ending. When a transaction is encountered that is not in the statistics table, the transaction is immediately signalled to complete. The signal that is received from the Apache child is the same signal that is returned. An entry for the unknown entry can then be established upon transaction completion.

The power manager will implement two work queues for transactions that have already been profiled. The queue will consist of PIDs to be signalled for transaction completion. One queue will be for transactions that have been categorized as CPU bound, and another queue will be for I/O bound transactions. The method used to implement these queues will require that a transaction be looked up in the statistics table to find its categorization. When a transaction has been found to be in the table, then that httpd worker child will be placed in a blocked state. The PID of the child worker will be entered into respective queue based on its classification type. While the child transaction is blocked, the power state of the system will still be at its lowest point. The queues will be emptied when a user defined number of transactions have entered a queue. In the worst case, a queue will not exceed the interval for an extended period of time, this is where the time-out value on the child waiting for a signal to continue comes in.

The queues containing the PIDs of blocked child processes are emptied when they reach a certain number of PIDs contained in each queue. This value by default was set at 3, this is a configurable number. Experiments performed with different values as a queue empty value showed that a smaller number would decrease the amount of power savings.

A different power scheme is employed based on the queue that is emptied. When the queue containing tasks primarily performing I/O based tasks, the CPU frequency is set to its lowest value. There is no loss in performance here, as the processor is making requests to the I/O subsystem and waiting for data to be returned. When the queue containing primarily CPU bound transactions is emptied, the CPU frequency is stepped up based on how many transactions are in the queue. If after initially emptying the queue, the queue becomes full again before the frequency is lowered, the frequency is stepped up again. This frequency stepping value is hardware dependent.

C. CPU Frequency Stepping

CPU frequency stepping is performed by the CSD using ioctls. To enable this ability, a new CPU frequency governor was created. The frequency governor is almost identical to the existing userspace frequency governor. The governor used by the CSD is implemented as a device driver that has ioctl() ability. The ioctls passed by the CSD contain the CPU number and a flag of 1 or 0, 1 being to increase the frequency of the specified CPU, 0 being to decrease the frequency. Individual CPUs cannot change their clock frequency in the hardware in our web server. To exploit CPU stepping, we migrated child processes to different CPUs by setting their scheduler affinity in the kernel scheduler.

In addition to handling the scheduling of the Apache child processes and the CPU frequency, the CSD will manage the CPU(s) that a child process will execute on. When the hardware that a Web Server is running on has multiple CPU sockets, the sockets can be configured to handle a specific task type. This hardware setup allows a CPU bound task to be scheduled on one socket and an I/O bound task to be scheduled on another socket. We will call this process *socket pinning*. In this case, each socket, can have different managed CPU frequencies.

V. RESULTS

We compare the results of our proposed Apache scheduler with an unmodified Apache Web Server. We use SPECweb2005, which contains three workload Banking, E-commerce, and Support, to compare the energy dissipation of the base web server compared to a web server with our modified scheduler. All of the SPECweb workloads consist of executing PHP scripts to perform specific transactions based on the workload. The data that is manipulated, whether it be text or images, is generated randomly based upon the number of concurrent connections. For our experiments, the number of clients is the same across all workloads. We used 5 clients, each generating 150 concurrent connections to place the web server under a load.

A. SPECweb Workloads

1) Banking: The banking workload of the SPECweb2005 benchmark is designed to simulate an online banking website. This workload primarily uses SSL connections to perform tasks. The makeup of the requests made on the server consist of images of checks and data to be entered into a bank ledger.

2) Ecommerce: The ecommerce workload is designed to simulate an online storefront, specifically a customizable computer system storefront. The workload will consist of a mix of SSL and non-SSL connections, due to the nature of the workload. Clients will benchmark the web server based on three phases, browsing the website, customizing a product, and ordering a product.

 ISBN: 978-1-61208-259-2

TABLE I
OVERALL SERVER ENERGY CONSUMPTIONS, TRANSACTIONS COMPLETED, AND ENERGY PER TRANSACTION

Power Management Technique	Parameter	Banking	Ecommerce	Support
Performance	Energy (J)	200285	193460	187830
	Total Transactions	40961	25285	26447
	Energy(J) per Transaction	4.889651132	7.651176587	7.102128786
Ondemand	Energy (J)	189637	190813	187713
	Total Transactions	40855	25471	26845
	Energy (J) per Transaction	4.641708481	7.491382356	6.992475321
Proposed Technique	Energy (J)	180515	184252	186730
	Total Transactions	40840	25261	26474
	Energy (J) per Transaction	4.420053869	7.293931357	7.053335348

TABLE II
CPU CORE ENERGY CONSUMPTIONS, TRANSACTIONS COMPLETED, AND ENERGY PER TRANSACTION

Power Management Technique	Parameter	Banking	Ecommerce	Support
Performance	Energy (J)	60218.6	57146.3	50328.8
	Total Transactions	40961	25285	26447
	Energy(J) per Transaction	1.470144772	2.260087008	1.903006
Ondemand	Energy (J)	56176	55402.2	50917
	Total Transactions	40855	25471	26845
	Energy (J) per Transaction	1.375009179	2.175108947	1.896703
Proposed Technique	Energy (J)	52822.3	49218.7	49771.8
	Total Transactions	40840	25261	26474
	Energy (J) per Transaction	1.29339618	1.948406635	1.880026

3) Support: The support workload is designed to simulate an online vendor's support website, whether it be a computer system vendor or a software package vendor. The workload will consist of searching and browsing a website for available downloads. The client will then download a file from the web server.

B. Experimental Setup

The hardware in our experiments consists of dual quad core Xeon E5520 CPUs with a max clock frequency of 2.27 GHz and 12GB RAMṪhe power supply is an 850W, 80 plus certified unit, with four 12V rails. The network interface card is an Intel 82574L gigabit Ethernet card using the e1000e driver. For CPU core energy measurements, we used two Fluke Y8100 current probes to collect and log data using a National Instruments Data Acquisition Unit. The overall server power consumption was measured using a Summit Technologies PS2500 Data Logging AC power meter measuring the AC power consumption for the server.

C. Overall Energy Consumption Characteristics

Table I shows the overall energy consumption, the number of transactions completed, energy consumed per completed transaction, averaged across multiple runs for a 30 minute duration for the Banking, E-commerce, and Support components of the SPECweb2005 benchmark. It is seen from Table I that the number of transactions processed for each of these benchmark components is roughly same across all of the techniques used (the performance governor, the ondemand governor, and the proposed technique). The energy per completed transaction is therefore a good indicator of the energy savings achieved by the proposed scheme. For the banking benchmark the energy per completed transaction in the proposed scheme is 4.42 Joules vs. 4.64 Joules using the ondemand governor and 4.90 Joules for the performance governor. Our scheme thus realizes roughly 9.8% energy savings on a per transaction basis compared to the performance governor and a 4.74% savings over the ondemand governor. The energy savings per transactions for the ecommerce benchmark is somewhat lower and there is hardly any energy savings on a per transaction basis for the support benchmark component. The reasons for the lower energy savings, particularly for support stem from the fact that these two components have a significantly higher fraction of I/O compared to both of the other components and both governors appear to be doing a good job in managing the *DVFS* settings for the I/O intensive benchmarks.

D. CPU Energy Consumptions Characteristics

Table II shows the energy consumption characteristics for the CPU cores for the three benchmark components of the SPECweb2005 benchmarks for each of the three power management schemes studied. The core power consumptions and the server power consumptions pretty much track each other but the percentage of energy savings realized by our scheme over the two governors are higher when one just looks at the CPU core energy consumption. This is simply because of energy consumptions elsewhere in the server and energy conversion inefficiencies within the power supply and voltage regulators on the motherboard. These other components dissipate energy none proportionately with the CPU cores.

 ISBN: 978-1-61208-259-2

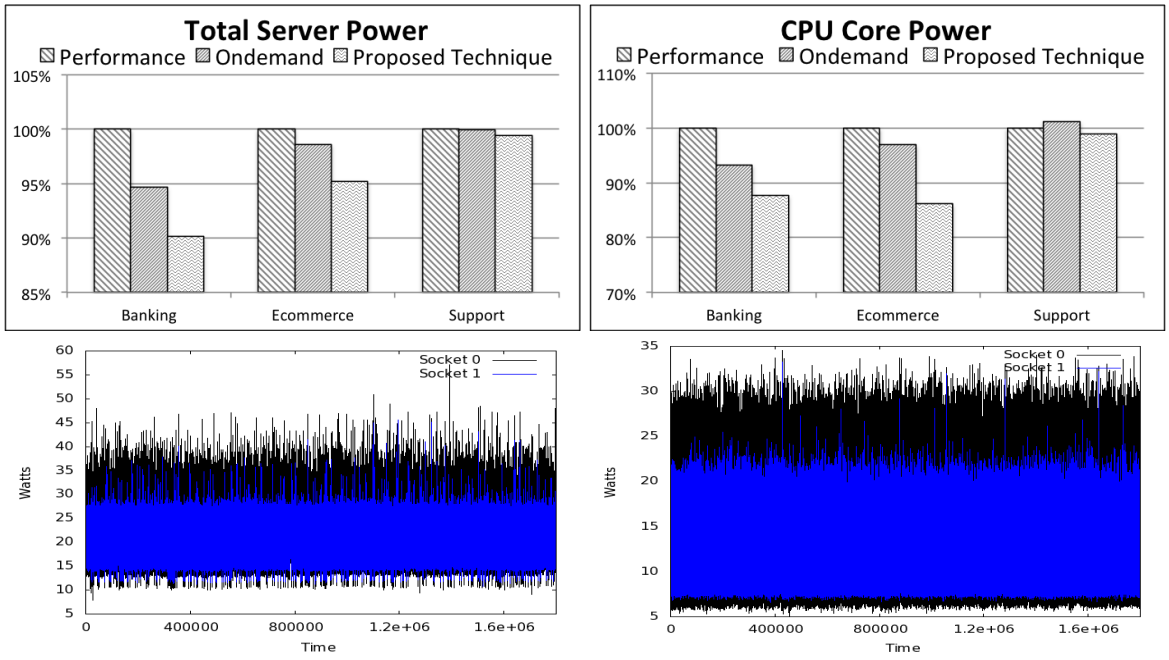

Figure 2. Reduction in Energy Consumption (a) AC Total Server Power (b) DC CPU Core Power (c) DC CPU Power Ondemand for SPECWeb2005 Banking (d) DC CPU Power for Proposed Technique for SPECWeb2005 Banking

E. Total Server Energy Consumption

In Fig. 2(a), the total energy consumption of the entire server is displayed. We expect these results to look similar to the results from the CPU Core Power experiments. There is some overhead involved with total box power however, in that the total box power is taking into account the power consumed by all components of the server plus any power lost in the power supply alone.

The left most bar is the performance governor, which is keeping the CPU frequency at it's highest setting, and therefore should consume the most power in each of the test cases. The banking benchmark displays a good mix of both I/O and CPU bound transactions, this is verified by the Ondemand governor being about midway between the performance governor and the scheduler scheme at 94.69% power of the performance governor. The CSD must keep both sockets active due to the types of transactions being performed being almost equally distributed between CPU bound and I/O bound.

The next workload, e-commerce, has a workload that executes more CPU bound transactions. The nature of the workload involves searching the product lines by the client process, and switching between SSL and non-SSL connections for checkout, which generates CPU bound transactions. This is shown as the amount of power savings between the performance governor and the ondemand governor is less than half the power savings shown between the ondemand governor and the scheduler. Since the amount of transactions that are CPU bound is greater than I/O bound, the scheduler is setting a majority of the child processes CPU affinity values to one socket, leaving the remaining socket idle. In this case, batch processing can be exploited by maximizing the efficiency of the processor when it is placed in the high frequency, and leaving the CPU in a lower frequency for a longer period than the ondemand governor does.

The last workload, support, is mostly I/O bound transactions. In this workload, the clients are primarily downloading files from the web server and this is reflected in the total power, and in the power savings. When the workload executes I/O bound transactions, the CPU will spend most of its time waiting for I/O operations to complete. As the CPU spends most of it's time in a waiting state, the load on the CPU is minimal and therefore the power consumed is minimal. The difference between the performance governor and the ondemand governor is almost non existent at 0.06%. The scheduler will still see a small power savings due to socket pinning in this case, as the CPU bound socket will be left idle during I/O bound transactions, and since the frequency of the I/O bound socket is kept to a minimum, we observe a small amount of power savings.

F. CPU Core Power

Fig. 2(b) shows the percentage of CPU core energy savings realized by our scheme against the performance governor (base case) and the ondemand governor. The percentage of energy savings realized by our scheme against the two governors is higher compared to the percentage

 ISBN: 978-1-61208-259-2

savings realized on the total server energy consumption. This is again because other components within the server do not dissipate energy proportional to the CPU energy consumption. It is also worth noting that for the support benchmark the ondemand governor actually leads to more energy energy consumption compared to the performance governor as the performance governor does a better job at handling I/O intensive workloads.

In Fig. 2(c) and (d) the power consumption of the CPU is examined on a per socket basis for the banking workload using the ondemand governor in (c) and the proposed technique in (d). When the ondemand governor is used the power consumption can range from 10 W to 50 W, compared to the proposed technique's range of 6 W to 34 W. This shows that batch scheduling transactions is effective at balancing CPU usage and idle time, lowering the range of power dissiptaion of the CPU.

VI. Conclusion

The ondemand and performance governors used in the Linux kernel for managing the *DVFS* settings have been refined over the years and do a very good job at reducing the power consumptions of servers. The goal of this effort was to exploit the repetitive characteristics of server workloads and exploit these characteristics in managing the *DVFS* settings of the processing cores, going beyond what is realized by the state of the art governors. Our experimental results clearly demonstrate that the proposed techniques realize additional energy savings beyond what is realized by the governors, up to about 10% for the Banking component of the SPECweb2005 benchmark. The energy savings realized over the governors over the other two specweb benchmark components are somewhat lower because the existing governors do a very good job at managing the *DVFS* settings for long periods of I/O where the CPU remains fairly idle.

The approach described in this paper minimized the need for modifications to the Apache server and the core kernel scheduler by using classification information collected by the Apache server and returning it to its job scheduling component and by using a separate daemon to collect the classification information and exercise the *DVFS* interface in the core kernel. Our ongoing work will implement the classification and scheduling components entirely within the core kernel, to realize a higher level of energy savings.

References

[1] "US Environmental Protection Agency Energy Star Program, Final Report Congress on Server and Data Center Energy Efficiency", Public Law 109-431, August 2007.

[2] J. G. Koomey, "Estimating Total Power Consumption By Servers in the U.S. and the World", Analytics Press. Feb. 2007.

[3] L. A. Barroso and U. Holzle, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines", Morgan-Claypool Publishers, 2009 (ISBN No. 9781598295566).

[4] J. Hamilton, "Where Does the Power Go In A Data Center?", keynote presentation at SIGMETRICS/Performance 2009.

[5] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos, "Energy-Efficient, Thermal-Aware Task Scheduling for Homogeneous, High Performance Computing Data Centers: A Cyber-Physical Approach", in IEEE Trans. On Parallel and Distributed Systems, vol. 19, no. 11), pp. 1458-1472, 2008.

[6] R. Nathuji and K. Schwan, "VPM Tokens: Virtual Machine-Aware Power Budgeting in Datacenters", in Proc. of the ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC), pp. 119-128, 2008.

[7] Z. Liu, "Load Balancing and Task Partitioning Strategy for Mixed Nature of Tasks", 2011 International Conference on Computer and Software Modeling, pp. 48-52, 2011

[8] Chen et al, "Energy-aware server provisioning and load dispatching for connection-intensive internet services", in Proc. 5th USENIX Symposium on Networked Systems Design and Implementation, pp. 337-350, 2008.

[9] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers", in Proc. EuroSys '09, pp. 317-330, 2009.

[10] IBM Corporation, "IBM Tivoli Usage Accounting Manager V7.1 Handbook", IBM Redbook, March 2008.

[11] IBM Corporation, "Value Proposition for IBM Systems Director: Challenges of Operational Management for Enterprise Server Installations", IBM ITG Group, Mangement Brief (34 pages), Nov. 2008.

[12] D.C. Snowdon, E. Le Sueur, S. Petters, and G. Heiser, "Koala: a platform for OS-level power management", Proceedings of the 4th ACM European conference on Computer systems, pp. 289-302, 2009.

[13] P. Visalakshi and T.U. Karthik, "MapReduce Scheduler Using Classifiers for Heterogeneous Workloads", IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.4, pp.68-73, April 2011,.

[14] A. K. Mishra, J. Hellerstein, W. Cirne, and C. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters", ACM SIGMETRICS Performance Evaluation Review on Industrial Research (SIGMETRICS PER, 2010), Volume 37 Issue 4, pp. 34-41, March 2010

[15] B. Grone, A. Knopfel, R. Kugel, and O. Schmidt, "The Apache Modeling Project", available at http://www.fmc-modeling.org/projects/apache 2008

 ISBN: 978-1-61208-259-2