A Workload-Aware, Eco-Friendly Daemon for Cluster Computing

S. Huang and W. Feng
Department of Computer Science
Virginia Tech
{huangs,feng}@cs.vt.edu

***Abstract*—This paper presents an eco-friendly daemon that reduces power consumption while better maintaining high performance via a novel behavioral quantification of workload. Specifically, our behavioral quantification achieves a more accurate workload characterization than previous approaches by inferring "processor stall cycles due to off-chip activities." This quantification, in turn, provides a foundation upon which we construct an interval-based, power-aware, run-time algorithm that is implemented within a system-wide daemon. We then evaluate our power-aware daemon in a cluster-computing environment with the NAS Parallel Benchmarks. The results indicate that our novel behavioral quantification of workload allows our power-aware daemon to more tightly control performance while delivering substantial energy savings.**

I. Introduction

According to a recent IBM report [16], the annual budget for power and cooling is fast approaching the annual budget for new server spending. This is why companies like Google located one of their new data centers in rural Oregon on the Columbia River — to take advantage of the cheap hydroelectric power generated by the nearby Grand Cooley Dam [17]. Rapidly increasing utility bills, coupled with how heat from excessive power affects reliability [5], motivates the need for power awareness in cluster computers, whether in a supercomputing center or a large-scale data center.

One way to address this growing problem is to improve the energy and power efficiency of cluster computers at different levels of abstraction: hardware [14], [4], systems integration [5], systems software [8], [7], middleware [12], and applications software [12]. In this paper, we use a *systems-software approach* that leverages *accurate workload characterization* via a *unique synthesis of hardware performance counters* in order to determine *when and how* to use dynamic voltage and frequency scaling (DVFS) to improve energy efficiency while strictly maintaining performance. Because the power consumption of a processor is proportional to its clock frequency and the square of its voltage supply, we use DVFS, available on virtually all modern processors, e.g., SpeedStep on Intel and PowerNow! on AMD, to set the voltage and frequency of the processor so as to reduce power consumption.

In a DVFS-enabled processor, a low (or high) power-consuming mode corresponds to processor that runs at a low (or high) frequency and voltage. Thus, a DVFS algorithm should reduce the frequency and voltage of a processor only when the processor is not needed to do useful work, e.g., waiting for the completion of a large block of I/O accesses. Application performance during such periods of off-chip access is insensitive to processor performance. Thus, we can reduce the processor voltage and frequency during such periods to reduce power consumption while maintaining application performance.

However, given that the time to scale voltage and frequency takes O(10,000,000) clock cycles, sophsticated use of DVFS is needed if energy savings is to be realized within a performance bound. Enabling such use requires accurate workload characterization, one of the main contributions of this work. Then, the challenge is to make use of this workload characterization to *autonomically* scale the frequency and voltage.

Thus, this paper presents a novel methodology for workload characterization on a per-node basis that is then used to enable intelligent power-aware computing across computational nodes in a cluster, datacenter, or grid. For the purposes of this paper, however, our focus will be on the former.

We refer to our power-aware, eco-friendly algorithm as `eco` and its implementation as `ecod`. The `ecod` system manages application performance and power consumption in real time based on an accurate measurement of CPU stall cycles due to off-chip activities and does *not* require application-specific information a priori. The paper will show that `ecod` controls performance with only a 4.8% impact (if our performance-bound knob is set to 5%) and less than 1% variance, both better than the current state-of-the-art, while saving up to 50% in CPU energy and 10% in overall system energy.

The remainder of the paper is organized as follows. Section II discusses related work on workload characterization and power-aware algorithms. Section III presents our novel behavioral quantification based on CPU stall cycles due to off-chip activities. In Section IV, we present our power-aware, run-time algorithm called `eco`, which is implemented as a daemon. Sections V and VI present our experimental setup and results, respectively, followed by a conclusion in Section VII.

II. Related Work

The past few years has seen significant research in power-aware cluster computing, which can be categorized into two types [7]: off-line, trace-based scheduling [1], [6], [15] and on-line, profile-based scheduling [8], [11], [7], [3]. For brevity, the related work below focuses on the latter, which is the more challenging problem.

With respect to on-line, profile-based scheduling, Lim [11] designs an MPI runtime system that dynamically reduces CPU performance during communication phases in MPI programs. Matthew [3] presents a comprehensive framework for autonomic power-performance adaptation of multi-threaded programs using thread throttling. However, these two works have limited application in that they are designed only for MPI and OpenMP applications, respectively. For power-aware research using general workload characterization, Choi and Pedram [2], Hsu and Feng [8] (β algorithm), and Ge et al. [7] possess the current state-of-the-art for general computing systems.

Choi and Pedram proposed a DVFS approach based on the ratio of off-chip access to on-chip computation time that targeted embedded systems. It uses the number of instructions and external memory accesses to compute the ratio of off-chip computation time to on-chip computation time. However, this workload characterization is *CPU-frequency dependent*. On the one hand, the off-chip access time is constant no matter what CPU frequency is used. On the other hand, on-chip computation time will decrease as CPU frequency increases. Hence, the ratio of off-chip access to on-chip computation time depends on the CPU frequency. Moreover, Choi's work only considers memory access and ignores thread synchronization in exploring energy-saving opportunities.

The β algorithm [8] of Hsu and Feng assumes that CPU boundedness is indirectly reflected via the MIPS (millions of instructions per second) rate. Since the MIPS rate only approximately reflects CPU boundedness and is dependent on CPU frequency, it cannot accurately characterize application workload nore can it effectively bound performance loss. In addition, another (arguable) drawback is that the β algorithm takes the entire history of workload into consideration when making DVFS decisions. While appropriate for some applications, it is not for many other applications. Figure 1 provides an example of significant workload variance in NPB FT benchmark. Consequently, with the accuracy of the workload characterization compromised, the β algorithm misses potential opportunities to save energy. As such, the β-adaptation algorithm does not perform well on these benchmarks.

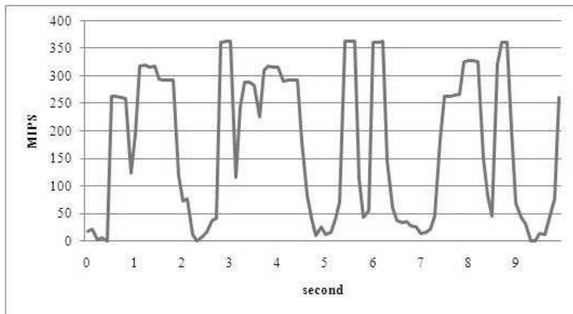

Fig. 1. MIPS of NPB FT benchmark

CPU MISER [7] relies on cache-access statistics to provide information about the workload. It also assumes that the number of instructions executed approximately equals the number of on-chip accesses based on heuristics. As such, this approach only accurately characterizes workload on average.

Finally, the Linux on-demand governor is the most widely employed across laptops, desktops and servers. The Linux on-demand governor is provided in the `CPUFreq` subsystem of a recent Linux kernel. It dynamically changes CPU frequency depending on CPU utilization [13]. Because CPU utilization is misleading in terms of characterizing a program's workload, the on-demand governor cannot efficiently deliver both power savings while controlling performance loss.

III. Novel Workload Characterization

From a power-aware perspective, the behavior of an application can create opportunities for energy savings. Execution phases with memory-intensive activities have been an attractive target for DVFS algorithms because the time for a memory access is independent of how fast the processor is running. When frequent memory or I/O accesses dominate a program's execution time, they limit how fast the program can finish executing. It is this *memory wall* that provides an opportunity to reduce power and energy consumption while maintaining performance. In cluster computing and grid environments, there are further opportunities for power and energy savings, particularly network process synchronization as well as I/O synchronization, e.g., traditional collective I/O. During the synchronization, CPUs are either waiting or idling.

Below we conduct a theoretical study on how to best control performance and how to derive a parameter λ to characterize application workloads, i.e., quantify application behavior. We then present our methodology on how to measure λ using CPU stall cycles due to off-chip activities.

A. Workload Characterization

At the systems level, any execution time of a program at CPU frequency f can be divided into two parts. One part is frequency sensitive, and the other is frequency insensitive. Correspondingly, we divide the CPU execution cycles into on-chip cycles C_{on} and off-chip cycles C_{off}.

$$C_{on} + C_{off} = T(f) \cdot f \tag{1}$$

C_{on} is the CPU cycles whose execution is affected by frequency variation while C_{off} is the CPU cycles whose execution is not affected by frequency variation.

We define T_{off} to represent the execution time that is CPU frequency insensitive.

$$T(f) = C_{on} \cdot \frac{1}{f} + T_{off} \tag{2}$$

When a program runs at maximum frequency f_{max},

$$T(f_{max}) = C_{on} \cdot \frac{1}{f_{max}} + T_{off} \tag{3}$$

T_{off} in Eq.~(3) is the same as in Eq.~(2) when executing the same amount of program instructions since T_{off} is not affected by the change of CPU frequency f.

To quantify the performance loss, we define a parameter δ that indicates the performance bound in employing DVFS,

$$\frac{T(f) - T(f_{max})}{T(f_{max})} < \delta \tag{4}$$

Substituting $T(f)$ and $T(f_{max})$ from Eq. (2) and (3), respectively, into Eq. (4), we get

$$\frac{C_{on}}{C_{on} + T_{off} \cdot f_{max}} \cdot \frac{f_{max} - f}{f} < \delta$$

The equation can be reformulated as

$$\lambda \cdot \frac{f_{max} - f}{f} < \delta \tag{5}$$

where

$$\lambda = \frac{C_{on}}{C_{on} + T_{off} \cdot f_{max}} \tag{6}$$

The workload characterization, denoted by λ in Eq. (6), can be reformulated as

$$\lambda = \frac{C_{on}}{C_{on} + C_{off} \cdot \frac{f_{max}}{f}} \tag{7}$$

Combining Eq. (1) and (7), we obtain

$$\lambda = \frac{f^2 T(f) - f C_{off}}{f^2 T(f) - f C_{off} + f_{max} C_{off}} \tag{8}$$

where $0 \leq \lambda \leq 1$. The value of λ serves two purposes:

- Intrinsic workload characterization. From Eq. (6), the workload characterization λ is a parameter that is independent of the CPU frequency that the application is running at. λ only depends on the application itself. Eq. (7) shows that λ characterizes the percentage of on-chip cycles out of the total CPU cycles at frequency f_{max}. When λ equals to 1, C_{off} is 0, which means that the program spent all its time on on-chip activities. When λ equals 0, C_{on} must be 0, which means the program spent all its time on off-chip activities. Eq. (8) gives us a method to quantify the behavior of applications, even if they are not running on frequency f_{max}.
- Frequency schedule indicator. In Eq. (5), assuming the required performance constraint δ is constant, running at frequency f is a decreasing function of λ. The larger the λ, the more opportunities that exist for saving energy within the performance constraint. So, λ can direct us to schedule the appropriate frequency for a given workload.

B. Methodology for Measuring CPU Off-Chip Stall Cycles

In this section, we present our methodology for measuring SC_{off}. In order to achieve the desired accuracy, we obtain the CPU stall cycles due to off-chip activities from two aspects: on-chip (SC_{off}^{on}) and off-chip (SC_{off}^{off}).

1) Measuring from the On-Chip Perspective:

$$SC_{off}^{on} = SC_{total} - SC_{on} \simeq SC_{total} - SC_{branch} - SC_{reorder}$$

where SC_{off}^{on} is the on-chip measurement of CPU stall cycles due to off-chip activities. For our platform, we measure SC_{total} using the CPU's decoder/dispatch stall cycles and measure SC_{on} using the sum of the CPU's decoder stall cycles due to branch misprediction (SC_{branch}) and full reorder buffer ($SC_{reorder}$). Why choose these two events? They dominate CPU stall cycles due to on-chip activities and hardly overlap with each other. There are also other stall cycles contributors, e.g. segment load, serialization, and so on. However, our empirical results show that CPU stall cycles contributed by these events are small; thus, we ignore them in our estimation.

2) Off-Chip Measurement:

$$SC_{off}^{off} = N_{mem} \cdot \tau_{mem} \cdot f + T_{io} \cdot f + T_{idle} \cdot f$$

where SC_{off}^{off} is the off-chip measurement of CPU stall cycles due to off-chip activities. N_{men} is the number of off-chip memory accesses; τ_{mem} is the memory-access latency; T_{io} is the CPU stall time for waiting on I/O completion; and T_{idle} is the CPU idle time. We use L2 cache misses to emulate the number of off-chip memory accesses and use `LMBench` [10] to measure the memory-access latency τ_{mem}. T_{io} and T_{idle} can be obtained through `/proc/stat` on Linux systems.

3) Synthetic Measurement: We obtain our final measurement by taking the minimum of on-chip and off-chip measurement of CPU stall cycles due to off-chip activities.

$$SC_{off} = min(SC_{off}^{on}, SC_{off}^{off})$$

Why take the minimum? Both measurements over-estimate the number of CPU stall cycles. On the one hand, for on-chip measurement, many events can cause CPU stalls, e.g. branch abortion, serialization, full reorder buffer [9], but there is no such hardware event that can measure CPU stall cycles due to off-chip activities directly. Moreover, most of the events involve both on-chip activities and off-chip activities. Therefore, an event cannot be simply treated as an event due to on-chip activities or off-chip activities. To exacerbate the problem, the events sometimes overlap with each other. On the other hand, off-chip measurement is also not accurate enough.

Let us take CPU stall cycles due to off-chip memory accesses as an example. Both off-chip memory accesses and memory latency are hard to determine precisely. The L2 cache misses measured by the hardware counter usually includes some due to speculative execution. Additionally, due to CPU prefetching and block transfer, some L2 cache misses will be combined and transferred together. Thus, it is not exactly accurate to measure off-chip memory accesses using L2 cache misses. The actual number of memory accesses will be smaller than the measured value.

Two facts lead us to combine on-chip and off-chip measurements. For CPU-bound applications, L2 cache misses are smaller and the opportunity for combining and overlapping cache misses is small. Thus, off-chip measurement works better for CPU-bound applications. For non-CPU-bound applications, however, CPU stall cycles due to off-chip activities dominate the total CPU stall cycles. Therefore, on-chip measurement fits non-CPU-bound applications well.

IV. eco ALGORITHM

Here we present our workload-aware, eco-friendly algorithm called eco. The algorithm consists of multiple components: (1) the high-level algorithm itself that periodically determines whether to scale the frequency and voltage, (2) workload prediction to enable the decision of what to scale the frequency (and voltage) to, and (3) once a frequency is determined, how to schedule and emulate the frequency (and voltage) if the platform does not explicitly support the frequency.

A. Overview of Algorithm

The eco algorithm is an interval-based, run-time algorithm, whose execution time is divided into intervals that span the running time of an application program. Within each interval, the algorithm performs the following:

1) *Characterizes the workload for the current interval, as noted in Section III.* As stated before, frequent memory and I/O access, network process synchronization, as well as CPU idling constitute the three main opportunities for power-aware computing. However, these three opportunities vary from application to application and change from time to time. In short, the eco algorithm quantifies the application behavior at run time for each interval.
2) *Predicts the workload characterization for the next interval.* The eco algorithm predicts the workload for the next interval based on that of previous intervals. It uses the average of a λ window of previous intervals to predict the workload, since we observe that workload tends to be constant for short periods of time.
3) *Schedules the frequency for the next interval.* The eco algorithm schedules the CPU frequency based on the predicted workload characterization in order to maintain the performance bound while saving as much energy as possible. However, we must address two problems in frequency scheduling for real systems in this step: (1) CPUs only support discrete frequencies, and (2) CPU frequencies have a lower and upper bound.

B. Workload Prediction

Though workloads may vary from application to application, the workloads can still be predictable at some level. For example, we set a window size of L and use the average across the window to predict the λ in current interval. The window size cannot be too large so that the DVFS scheduler is reactive to workload variation, but the window size cannot be too small either as it risks significant prediction error. Empirically, we set the window size to be 3 by default in ecod.

Because there will always exist some error in any workload prediction, we integrate a rectifying mechanism to monitor and control the global performance slowdown. The basic idea is to calculate the workload prediction error in each interval and make some correction in the future scheduling of frequencies to compensate for the prediction error. Initially, the performance bound δ equals a user-defined performance constraint Δ, e.g. 5%. During execution, if the predicted λ is larger than the measured λ, we increase the value of δ for the next interval and vice versa.

Consider an interval of $T(f)$ in a program execution. Assume λ_p is the predicted workload characterization of the program in an interval. The actual measured workload characterization is denoted as λ_m. Let f_p be the frequency based on λ_p, which is the frequency the program has been running on and let f_m be the frequency based on λ_m, which is the frequency the program should have been running on.

The error in execution time over the interval is

$$\zeta = T(f_p) - T(f_m) = C_{on} \cdot (\frac{1}{f_p} - \frac{1}{f_m}) \tag{9}$$

where C_{on} can be measured directly for current interval. f_p is already known in the current interval and f_m can be obtained after completing this interval via frequency scheduling, i.e., Eq. (10). To compensate for the prediction error, the performance constraint for next interval becomes

$$\delta = \Delta + \frac{\zeta}{T(f)}$$

where $T(f)$ is the time for next interval, Δ is the standard performance constraint without compensation, and ζ is calculated via Eq. (9).

C. Frequency Scheduling and Emulation

Assuming that $\bar{\lambda}$ is the predicted workload characterization for the current interval, then based on Eq. (5), the ideal frequency for the current interval is

$$f^* = \frac{\bar{\lambda} \cdot f_{max}}{\bar{\lambda} + \delta} \tag{10}$$

However, due to the physical constraints of the processor itself, the available frequencies in a real system are bounded. Thus, f^* needs to be calculated as

$$f^* = max(f_{min}, \frac{\bar{\lambda} \cdot f_{max}}{\bar{\lambda} + \delta})$$

Finally, the calculated frequency f^* may not be directly supported on a real system. So, we apply the method proposed in [8] to emulate the calculated frequency f^*.

D. The eco Algorithm

Synthesizing the steps shown above, we design our eco algorithm. Figure 2 presents the pseudocode for the eco algorithm. Steps 1 and 2 encompass workload characterization. Step 3 is workload prediction, and Steps 4 and 5 deal with frequency scheduling and emulation.

V. EXPERIMENTAL SET-UP

Here we detail the experimental set-up for evaluating our eco algorithm, including hardware and software platform, power and energy measurement, and ecod implementation.

A. Experimental Platform

The hardware platform in our experiment includes a four-node cluster for computing and an additional node for recording the power and energy consumption. Each compute node

Hardware:

n frequencies $f_1, \cdots, f_n$

Parameters:

I: time-interval size
δ: performance bound
L: prediction window size

Algorithm:

Initialize the λ window
Repeat

1. Measure CPU stall cycles due to off-chip activities for current interval C_{off}
2. Compute coefficient λ for current interval

$$\lambda = \frac{f^2 I - f C_{off}}{f^2 I - f C_{off} + f_{max} C_{off}}$$

3. Predict the workload for next interval for all λ in window [0, L]

$$\bar{\lambda} = average(\lambda)$$

4. Compute the desired frequency f^*

$$f^* = max(f_{min}, \frac{\bar{\lambda} \cdot f_{max}}{\bar{\lambda} + \delta})$$

5. Schedule next interval I at f^*

Fig. 2. Pseudocode for eco algorithm

TABLE I
POWER/PERFORMANCE MODES FOR ICE CLUSTER NODE

Frequency (GHz)	Voltage (V)
2.6	1.30
2.4	1.25
2.2	1.20
2.0	1.15
1.8	1.15
1.0	1.10

contains two dual-core AMD Opteron 2218 processors and 4-GB main memory. Each CPU core includes one 128-KB split instruction and data L1 cache. Two cores on the same die share one 1 MB of L2 cache. Each processor supports *six* power/performance modes, as shown in Table I. Finally, the nodes are interconnected with Gigabit Ethernet.

We run Red Hat Linux (kernel version 2.6.18) on each compute node. The Linux kernel `CPUFreq` subsystem is used for controlling DVFS and `PERFCTR` for hardware counter monitoring. With respect to the benchmarks, we use the latest NAS Parallel Benchmarks (NPB3.2-MPI). We use `mpich2` (version 1.0.6) to run the benchmarks.

B. Energy Measurement and Processing

We use the “Watts Up? PRO ES” power meter to measure the total system energy for each node. Energy values are recorded immediately before and after the benchmark runs. The difference of the two energy values is the energy consumed by the system when the benchmark ran. Since DVFS scheduling only affects the power consumption of CPU, it is (arguably) misleading to evaluate our `eco` algorithm based on the energy consumption of total system. So, in addition to reporting the total system energy, we also evaluate the effect of `eco` on CPU energy by applying a CPU power model used in [8] to isolate the CPU energy from the total system energy.

C. The ecod Implementation

Figure 3 illustrates the software architecture of our `ecod` implementation. We implement `ecod` as a lightweight daemon that monitors all the cores in a node and schedules appropriate frequencies for them. When `ecod` starts up, it reads the configuration file and dynamically detects processor settings, e.g. available frequencies, number of cores, etc. In each sampling interval, the master daemon fetches hardware-event information from the “Hardware Event Monitor Module.” Then, workload prediction and performance rectification are performed. In the end, the master daemon dispatches the desired frequency to “DVFS Scheduler Module,” which then takes care of frequency scheduling of the cores.

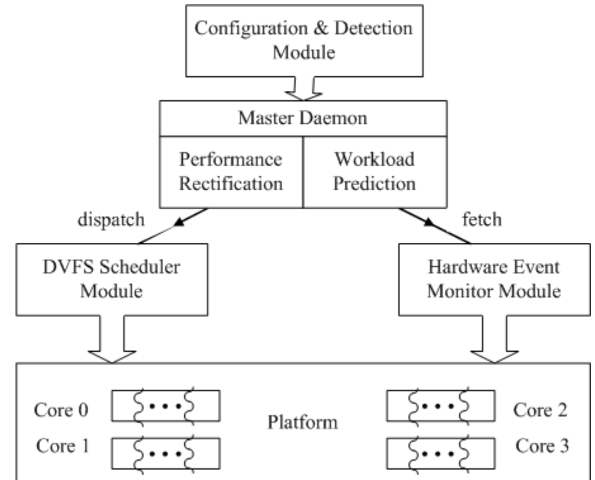

Fig. 3. Software architecture of ecod

D. Parameters and Sensitivity Analysis

`ecod` is configurable and tunable. The configuration parameters as well as their default values for experiments in this paper are shown in Table II. The user-configurable parameters are *sampling interval, performance bound,* and *prediction window size.* Below are the tradeoffs of these user-configurable parameters.

- *Sampling Interval*. As sampling intervals increase in length, the precision of workload characterization and its prediction will worsen, resulting in performance that

TABLE II
CONFIGURATION PARAMETERS AND THEIR VALUES IN ECOD

Parameter	Description	Value
I	sampling interval	1 second
δ	performance bound	5%
L	prediction window size	3

cannot be tightly controlled. Conversely, when the sampling intervals get too short, the overhead of sampling the workload and scheduling the frequency is not as easily amortized.

- *Performance Bound*. The larger the performance bound (or percentage slowdown), the more energy that will be saved. However, once the frequency reaches the system's lowest frequency, it cannot save any more energy.
- *Prediction Window Size*. If the window size is large, the algorithm will depend on a larger amount of historical information, thus making more instantaneous workload prediction inaccurate. If the window size is small, the algorithm will be too sensitive to the workload variation.

In our experiments, we compare `ecod` with the β-algorithm [8] and the Linux on-demand governor [13]. The performance constraint in the β algorithm is set to 5%. As for Linux on-demand governor, we use the default configuration with a sampling rate of 560,000 ms and up threshold of 80%.

VI. EXPERIMENTS AND ANALYSIS

In this section, we first validate the workload characterization λ obtained by measuring the CPU stall cycles due to off-chip activities against an off-line approach, described in the Appendix. Then, we evaluate the workload prediction method used in `eco` algorithm along with a sensitivity analysis of the algorithm. Finally, we demonstrate the efficacy of `ecod`, our power-aware daemon based on `eco`, on the NAS Parallel Benchmarks (NPB3.2-MPI) in a cluster environment.

A. Validation of Workload Characterization

Before evaluating `eco` on the NAS Parallel Benchmarks, we first validate our workload characterization (λ) on a representative set of 10 SPEC CPU2000 benchmarks: three CPU-bound, three memory-bound, and four in between. Specifically, by evaluating λ, we indirectly evaluate the measurement of CPU stall cycles due to off-chip activities.

Figure 4 shows our evaluation of measured λ to that of an off-line approach (see Appendix), with the benchmarks arranged in such a way that the CPU-boundedness (i.e., Y-axis) of the benchmarks decrease going left to right. The error of the measured λ to off-line value is only 3.4% on average.

B. Evaluation of Workload Prediction

Here we use the workload characterization (λ) obtained by CPU stall cycles due to off-chip activities as a baseline to evaluate the effectiveness of our workload prediction method. We chose `crafty`, `mcf`, and `bzip2` SPEC CPU2000 to

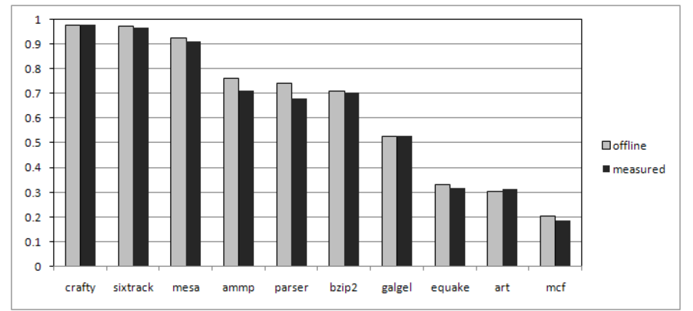

Fig. 4. Validation of measured λ against that obtained offline

illustrate the predictive performance on CPU-bound, memory-bound, and in-between benchmarks, respectively.

Figures 5, 6, and 7 show the comparison between measured λ and predicted λ for the `crafty`, `mcf`, and `bzip2` benchmarks, respectively, where the y-axis denotes the workload characterization λ. Over the execution time of the benchmarks, the difference between measured λ and predicted λ is within 2%. The figures also show that the predicted λ changes more smoothly than measured λ. This reflects the stability of our algorithm, which in turn, avoids significant DVFS scheduling overhead since the larger the frequency transition, the more overhead that is induced in DVFS scheduling [8].

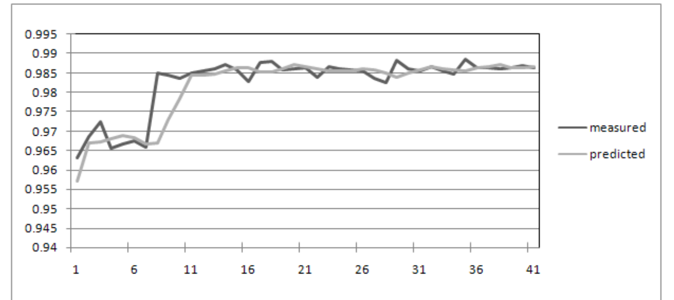

Fig. 5. Measured λ versus predicted λ for `crafty` over execution time

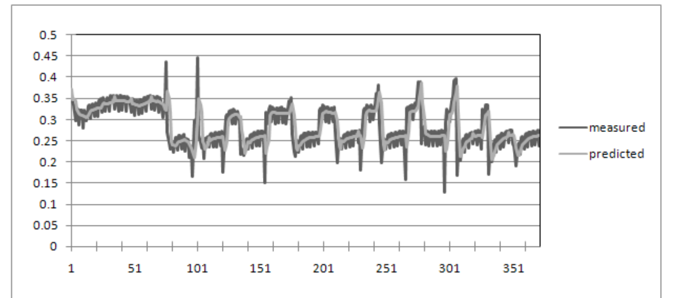

Fig. 6. Measured λ versus predicted λ for `mcf` over execution time

C. Sensitivity Analysis of Performance Bound

Because one of the contributions of this paper is that `ecod` can more tightly control performance loss, we also evaluate how `ecod` behaves to differente performance bounds. Figure 8

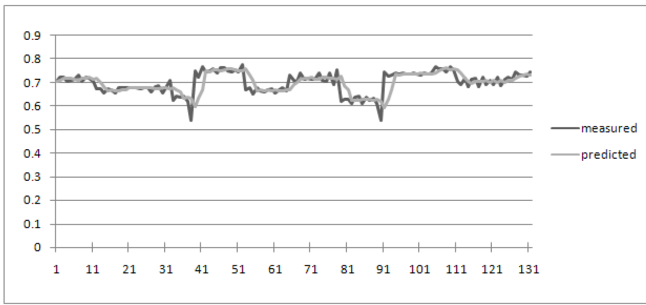

Fig. 7. Measured λ versus predicted λ for bzip2 over execution time

shows that ecod can bound the performance quite well; the performance variances for all the performance bounds are within 3%. Figure 9 shows that while maintaining performance, ecod can also achieve up to 56% in energy savings.

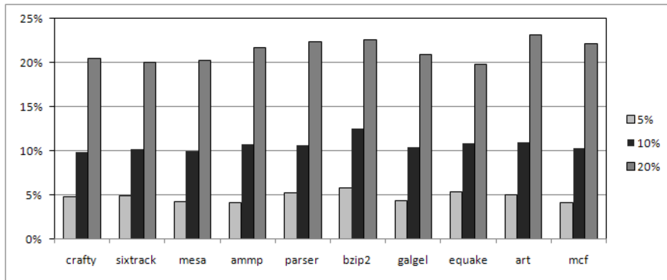

Fig. 8. The performance control of ecod for various performance bounds

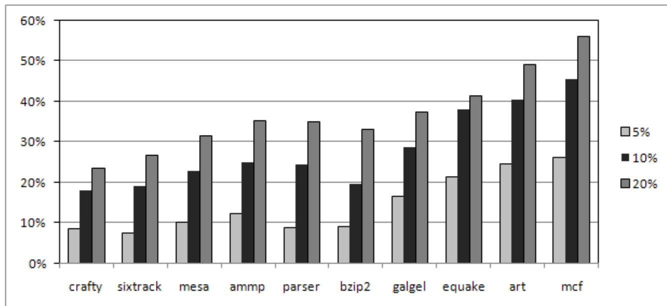

Fig. 9. The energy savings of ecod for various performance bounds

D. Parallel Experiment

With the validation of our workload characterization and workload prediction, coupled with our sensitivity analysis, all on a per-node basis as shown above, we are now ready to evaluate our eco algorithm, implemented as an eco-friendly daemon that we call ecod in a cluster environment. In such an environment, we expect the performance of our eco-friendly daemon to be quite good given the additional opportunities for energy savings due to frequent memory and I/O access, network process synchronization, as well as CPU idling.

To evaluate ecod, we use the NAS Parallel Benchmarks. We run the benchmarks with a Class C workload on 16 cores across four compute nodes, with each compute node containing four cores. Since the cores on the same die have a common power/performance mode, we schedule the core frequency according to the higher one on the same die in order to guarantee performance.

TABLE III
STATISTICS ON PARALLEL EXPERIMENT

	ecod	β	on-demand
Performance Mean	5.1%	10.6%	7.9%
Performance Standard Dev.	3.5%	10.3%	7.7%
Energy Mean	31.5%	32.9%	28.6%

Figures 10 and 11 show the performance control and energy savings of ecod in comparison with the β algorithm and Linux on-demand governor, respectively. Table III summarizes the statistics on performance loss and energy savings. The performance loss averages 5.1%, which is better than the β algorithm (10.6%) and Linux on-demand governor (7.9%). The standard deviation of ecod is also the best among the three algorithms.

The CPU energy savings are comparable between ecod (average of 31.5%), β-algorithm (average of 32.9%) and on-demand governor (average of 28.6%). Considering that ecod achieves the same energy saving by sacrificing far less performance, ecod clearly performs better than the β algorithm and Linux on-demand governor.

Finally, with respect to overall energy savings, ecod performs better than the β algorithm and the Linux on-demand governor on average, as shown in Figure 12. ecod can achieve 11% energy savings on average across the NAS Parallel Benchmarks. Both β and the Linux on-demand governor save 8% of energy for the same benchmarks on average.

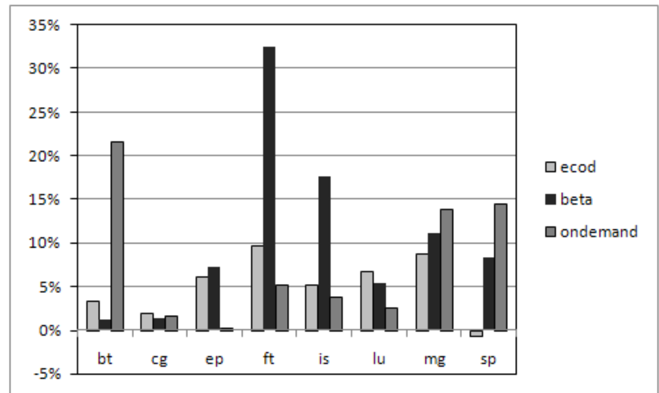

Fig. 10. Performance loss on NAS parallel benchmarks

VII. CONCLUSION

This paper presents a novel behavioral quantification of cluster workloads using CPU stall cycles due to off-chip activities. We leverage this quantification to create a power-aware, eco-friendly, run-time algorithm called eco. This algorithm dynamically monitors processor states and obtains the

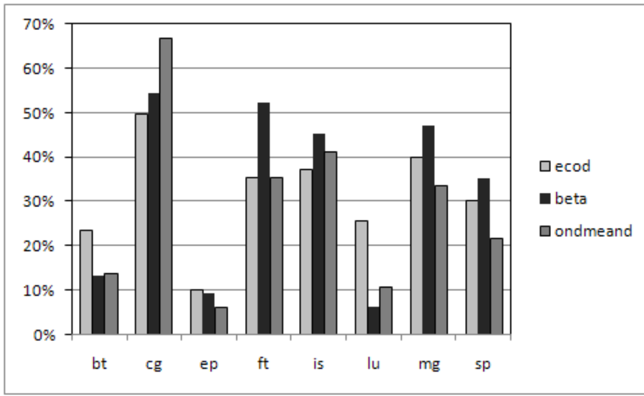

Fig. 11. CPU energy savings on NAS parallel benchmarks

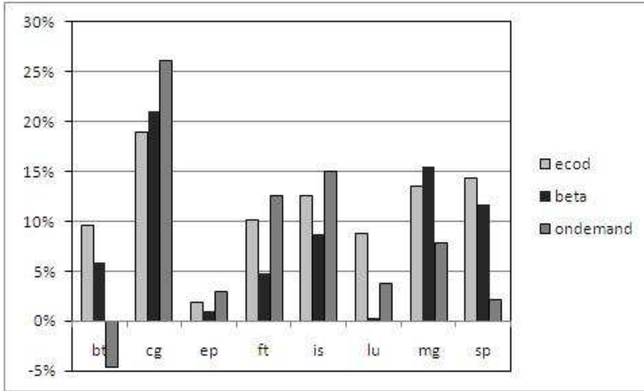

Fig. 12. Overall energy savings on NAS parallel benchmarks

workload characterization at run time in order to guide the appropriate scaling of frequencies and voltages in a parallel computing environment. Results show that our implementation `ecod` achieves the best performance control over the β-adaptation algorithm and Linux on-demand governor while delivering an overall energy savings of 11%.

References

[1] K. W. Cameron, R. Ge, and X. Feng. High-performance, power-aware distributed computing for scientific applications. In *IEEE Computer*, 2005.

[2] R. Choi and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to onchip computation times. *IEEE transactions on computer-aided design of integrated circuits and systems*, 24(1), 2005.

[3] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *International Conference on Supercomputing (ICS06), Queensland, Australia*, June 2006.

[4] J. Ebergen, J. Gainsley, and P. Cunningham. Transistor sizing: How to control the speed and energy consumption of a circuit. In the 10th International Symposium on Asynchronous Circuits and Systems, 2004.

[5] W. Feng and C. Hsu. Green destiny and its evolving parts. In *Innovative Supercomputer Architecture Award, International Supercomputer Conference*, Heidelberg, Germany, 2004.

[6] V. Freeh, D. Lowenthal, F. Pan, and N. Kappiah. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Principles and Practices of Parallel Programming (PPoPP)*, 2005.

[7] R. Ge, X. Feng, W. Feng, and K. W. Cameron. CPU MISER: A performance-directed, run-time system for power-aware clusters. In *International Conference on Parallel Processing, 2007 (ICPP07)*, 2007.

[8] C. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the ACM/IEEE Supercomputing 2005 (SC05)*, 2005.

[9] AMD Inc. BIOS and kernel developer's guide for AMD athlon 64 and AMD opteron processors. Feburary 2006.

[10] C. Staelin L. McVoy. lmbench: portable tools for performance analysis. In *Proceedings of the Annual Technical Conference on USENIX*, 1996.

[11] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *Proceedings of the ACM/IEEE Supercomputing 2006 (SC06)*, 2006.

[12] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In Proceedings of the 11th ACM International Conference, 2003.

[13] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor – past, present, and future. *Ottawa Linux Symposium Proceedings 2006*.

[14] P. Penzes, M. Nustrom, and A. Martin. Transistor sizing of energy-delay-efficient circuits. Technical Report. California Institute of Technology, 2002.

[15] B. Roundtree, D. K. Lowenthal, S. H. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *Proceedings of the ACM/IEEE Supercomputing 2007 (SC07)*, 2007.

[16] Chris Scott. Energy efficiency in the data center. 2007.

[17] New York Times. Hiding in plain sight, google seeks more power. 2006.

Appendix

VIII. Off-Line Measurement of Off-Chip Cycles

Here we describe an off-line method to calculate the CPU-boundedness for an application and use it as a baseline to evaluate our measurement of CPU stall cycles due to off-chip activities. The method is described below.

1) run the application for each available CPU frequency and record the corresponding execution time.
2) normalize the execution time for each CPU frequency to the execution time at maximum CPU frequency f_{max}.
3) draw a graph canvas in which X-axis is CPU cycle time and Y-axis is the execution time of the application.
4) draw points onto the canvas. X-coordinate of each point is the reverse of its running CPU frequency and Y-coordinate of each point is the execution time on that CPU frequency.
5) take the point of maximum frequency as the fixed point of trend line and use linear least square regression method to determine the slope of the trend line. The slope will minimize the least-square error:

$$min \sum_{i=1}^{n-1} ||T(f_i) - k(\frac{1}{f_i} - \frac{1}{f_{max}}) - T(f_{max})||^2$$

6) the slope of the line is actually the CPU execution cycle C_{on} when the application is running at maximum frequency for 1 second. In other words, the slope is the average CPU execution cycles when running on maximum frequency.
7) use the equation (1) to calculate C_{off}.
8) use the equation (8) to calculate β.