Adaptive Power Capping for Servers with Multithreaded Workloads

Power capping in computer clusters enables energy budgeting, efficient power delivery, and management of operational and cooling costs. Pack and Cap is a novel, practical methodology to select thread packing and dynamic voltage and frequency scaling (DVFS) configurations by learning multithreaded workload characteristics and adapting to dynamic-power caps. Pack and Cap improves energy efficiency and achievable range of power caps.

Sherief Reda
Ryan Cochran
Brown University
Ayse K. Coskun
Boston University

One of the greatest challenges for today's cluster operators is the increasing energy cost as a fraction of the total cost of ownership. In fact, power and cooling costs have risen as much as 400 percent in the past decade.[1] Modern data-center energy consumption results in millions of dollars in annual electricity costs in the US alone. Power capping, in which the cluster's average or peak power is constrained, has become a popular technique for ensuring energy budgets, planning cluster power delivery, and managing operational and cooling costs.

Wholesale energy markets introduce a new incentive for cluster power capping. Independent System Operators (ISOs) that coordinate power transmission must match supply and demand in the grid. This challenge grows with the fluctuations in loads and sources as a larger portion of highly variable green energy sources are introduced into the grid.[2] As a result, many ISOs are looking into creating flexible reserves at the demand side. Large computing clusters are candidates for demand-side regulation, owing to their load flexibility and power management features. ISOs offer credit to the demand side for regulating their power at several second intervals; thus, fine-grained modulation of cluster power could provide significant monetary savings.

Individual server power capping is an essential prerequisite to cluster-level power capping. Existing methods for server power capping include sleep modes, dynamic voltage and frequency settings (DVFS),[3,4] and throttling by idle cycle insertion.[5] At a larger scale, it's possible to devise control techniques to coordinate multiple levels of capping in a data center[6] and to automatically adjust server caps on the basis of utilization.[7]

This article proposes Pack and Cap, a novel technique for maximizing performance

Published by the IEEE Computer Society

within dynamically set power caps for multithreaded workloads. Pack and Cap builds on the observation that workloads on clusters are increasingly employing thread-level parallelism to capitalize on the hardware parallelism in multicore processors. Parallel applications offer a new control knob for power capping—namely, selecting the number of active threads. Pack and Cap leverages thread packing, in which multiple threads of an application are packed onto a variable number of cores, as a low-cost proxy to mimic dynamic selection of the number of active threads.

Pack and Cap brings several important innovations. First, it is designed to meet instantaneous server power caps, whereas most prior techniques (such as throttling and DVFS) focus on maintaining an average power consumption value. Second, by controlling the number of active cores via thread packing in addition to DVFS, we can decrease the lower bound on achievable power caps and thus achieve more dynamic-control flexibility. These two innovations enable a fine-grained power-capping strategy and higher throughput, making this technique attractive for use in adaptive power regulation. We propose a set of techniques for effective adaptive power capping. These techniques offer different tradeoffs in terms of effectiveness, hardware requirements, offline characterization efforts, and implementation costs. We conducted all experiments on a server with two quad-core processors, making our method attractive for deployment on real systems. We demonstrate that Pack and Cap meets the power caps with high accuracy, while minimizing the application runtime.

DVFS and thread packing

DVFS is a standard technique for dynamically enforcing power caps.[6,8] A major contribution of our work lies in our use of a variable number of active cores as a control knob in conjunction with DVFS while running multithreaded workloads. This additional control knob, which we call *thread packing*, lets us achieve more desirable power-performance tradeoffs compared to using DVFS alone, and decrease the lower bound of dynamically achievable power caps, thus providing more flexibility for cluster-level capping strategies.

The inspiration behind thread packing comes from the power-performance tradeoffs observed while varying the number of threads in a parallel application, which we refer to as *thread reduction*. In our initial experiments, we investigate the effects of performing thread reduction, which we evaluate in terms of the achievable power levels with experiments on a multicore server. Our server is equipped with two quad-core Intel Xeon E5520 processor chips (each core with six DVFS settings) and 12 Gbytes of memory. In Xeon processors, idle cores automatically switch to low-power states through clock gating to save energy. We run each Parsec benchmark[9] under all the DVFS settings with one, two, four, and eight threads. We then measure the power range, which is the difference between the server's highest and lowest power level observed across all DVFS settings and thread counts. Without thread reduction, we achieve the lowest power level by setting the lowest DVFS setting. With thread reduction, however, we can achieve lower power by setting the lowest DVFS setting and reducing the number of threads. Idle cores entering low-power states yield significant power reductions.

Thread reduction is useful when the lowest DVFS setting is insufficient for meeting a power cap. In our prior work, we showed that jointly controlling DVFS and the number of threads on a single-chip quad-core system increases the power range by 21 percent compared to using only DVFS.[10] Further improvements in the power range are attainable in multiprocessor systems, which are more typical configurations in today's clusters. The power range increases to 41 percent on average across all Parsec benchmarks running on our dual-processor server. In addition to the increased power range, integrating the use of the two knobs (that is, DVFS and the number of threads) enables finer-grained capping, in which power and performance can be tuned more precisely to meet desired constraints.

Although thread reduction is an effective power management tool, it can't be applied dynamically during execution without substantial modifications to the application code.

Thread packing, on the other hand, is a practical alternative, which we apply by modifying thread-core affinities in the operating system, thus eliminating the need for application-specific modification. We constrain each multithreaded workload to execute on a subset of available cores by setting thread-core affinities in the operating-system (via the `sched_setaffinity` system call interface in Linux), while the remaining idle cores enter low-power states. Allocation and scheduling of threads among the active cores on each processor are then performed transparently using the default operating system algorithms. To further motivate thread packing, we verify experimentally that each thread-packing configuration has almost identical runtime and power characteristics to the thread-reduction scenario with the matching number of active cores. We perform all experiments with the Parsec benchmark suite[9] on the multichip eight-core server, with static settings across each workload's execution. We disable hyper-threading in order to isolate the effects of thread packing. We run each benchmark using the native input set at every DVFS setting with the following thread scenarios:

- *Eight cores.* We measure power and runtime for eight threads executing on eight cores. This case serves as the baseline for both thread packing and thread reduction in the following three comparisons.
- *Four cores.* We measure power and runtime for executing eight threads packed on four cores against executing four threads on four cores.
- *Two cores.* We measure power and runtime for executing eight threads packed on two cores against executing two threads on two cores.
- *One core.* We measure power and runtime for executing eight threads packed on one core against executing one thread on one core.

We perform a comparison of runtime and average power between thread packing and thread reduction for the cases of one, two, four, and eight active cores for all Parsec benchmarks. For each comparison, the number of active cores is the same for both thread packing and thread reduction (for example, eight threads packed onto two cores compared to two threads running on two cores). Packing and reduction perform identically in the eight-core case. For the four-core case, thread packing increases runtime by 3.6 percent but decreases average power consumption by 0.1 percent. For the two-core case, thread packing decreases runtime by 4.5 percent and decreases average power by 0.9 percent, and for the one-core case, packing increases runtime by 0.7 percent and decreases average power by 1.5 percent. These results show that the number of active cores is the primary determiner of power and runtime for a multithreaded application. Although the runtime and power values are comparable, thread packing allows for dynamic adjustment during workload execution without workload-specific modifications. In addition, the allowable number of active cores in thread packing is not constrictive. Many Parsec benchmarks can be launched only with thread counts that are a power of 2. Thus, thread packing is a more practical and flexible solution to adaptive power capping.

When minimizing runtime in the absence of power caps, the optimal operating point is the one with the maximum number of active cores and the maximum DVFS setting. In the presence of power caps, however, the optimal setting that produces the best performance within a fixed power cap varies depending on workload and environmental conditions. Figure 1 provides the peak power and runtime at all possible settings for the first 100 billion retired micro-operations of four Parsec benchmarks on our server. To simplify the task of finding the optimal setting, we mark the power-runtime Pareto frontier with a dashed line in Figure 1, given power and runtime measurements for each setting. For each point along the frontier, there is no alternative point that achieves lower peak power and shorter runtime. Any point not on the frontier can't be optimal, because there exists a setting on the Pareto frontier that produces both lower runtime and lower peak power. Thus, the points on the frontier dominate the nonfrontier points, and the point along the frontier with the

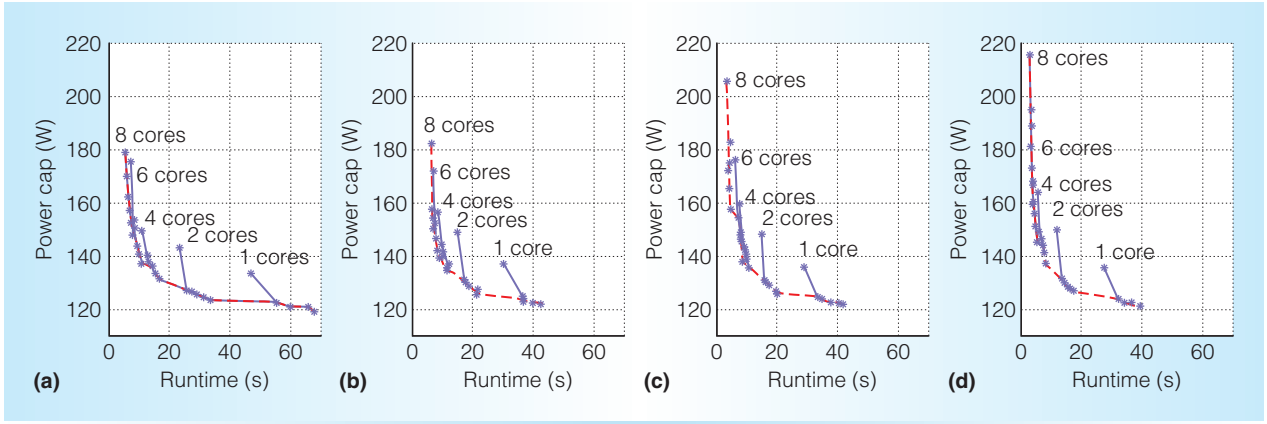

Figure 1. Plots show the impact of dynamic voltage and frequency settings (DVFS) and thread-packing settings on runtime and power consumption. The dashed line gives the Pareto frontier of optimal settings at various power caps. Each solid line gives the power and runtime results when we change the DVFS under a fixed number of cores. We include only the eight-, six-, four-, two-, and one-core cases. We leave out the three-, five-, and seven-core cases for clarity.

least runtime within the power cap is optimal. We observe that for all frontiers, within the subset of points for which the power cap is met, the optimal point that minimizes runtime always maximizes the number of active cores. Thus, any control policy should first select the highest number of cores for which the power cap can be met, and then select the highest DVFS setting within the power cap.

Our experiments with thread packing SPEC CPU2006[11] workloads show that this observation applies to single-threaded applications as well. The combined throughput of a set of single-threaded applications scales well with the number of active cores, which is intuitive, because single-threaded applications aren't limited by the synchronization overheads encountered in multithreaded applications. We have also characterized the Parsec workloads' behavior with hyper-threading enabled, and our observations about the Pareto frontier still hold. That is, the optimal operating point always maximizes the number of active virtual cores within the power constraint.

Pack and Cap methodology

We propose three techniques for minimizing application runtime within a power cap (see Figure 2). These techniques offer different tradeoffs in terms of effectiveness, hardware requirements, and implementation costs. We propose a feedback technique with a multigain controller that uses runtime power measurements without requiring any server-state measurements (such as performance counters, temperatures, or utilization). We then propose a projective-feedback technique that improves capping accuracy and performance by leveraging measurements of the server's state and extensive offline characterization. The third technique, projective modeling without a meter, is similar to the second technique, but it doesn't use a power meter. Projective modeling avoids the additional equipment cost of power telemetry at the expense of power-capping accuracy.

Proposed feedback technique

Feedback techniques assume that a server has access to power measurements through either an integrated or external power meter. On the basis of the measured power slack—that is, the difference between the required power cap and the actual power consumption—the feedback controller chooses appropriate settings to reduce the slack. Classical feedback-based capping methods solely use DVFS for feedback control, in which the DVFS setting is adjusted using a P (proportional) or PI (proportional–integral) controller based on the power slack.[3,4] In addition to DVFS, our technique adjusts the number of active cores (that is, thread-packing configuration). The main challenges are choosing between the two

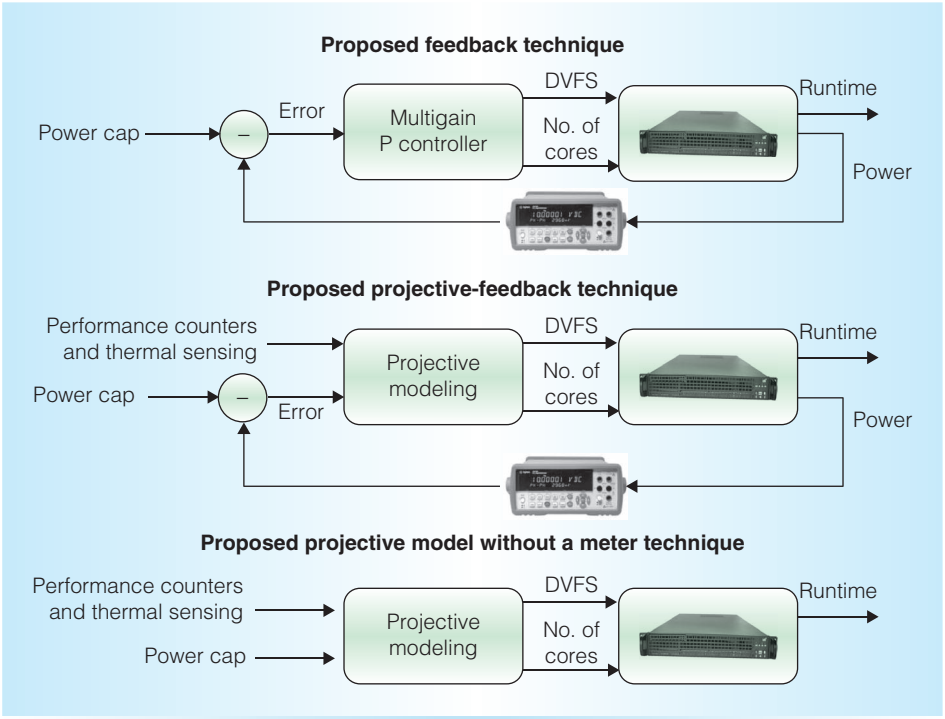

Figure 2. Proposed power-capping methods. The proposed feedback technique uses runtime power measurements without requiring any server-state measurements. The proposed projective-feedback technique improves capping accuracy and performance by leveraging measurements of the server's state and extensive offline characterization. Using the proposed projective model without a meter is similar to the proposed projective-feedback technique, except that it doesn't use a power meter.

control knobs to meet the power cap and tuning the controller to achieve the desired balance between optimizing response time and meeting the cap.

To choose between the two settings, we propose a heuristic based on the Pareto frontier characterization results, in which we observe that within the subset of points that adhere to the power cap, the optimal point along the Pareto frontier maximizes the number of active cores. Hence, it's best to avoid decreasing the number of active cores when eliminating a negative power slack, and to increase the number of cores whenever possible to eliminate positive slack. Our heuristic uses experimentally calculated proportional power gains to estimate the effect that DVFS and the number of active cores have on power. If we observe a positive power slack, the controller increases the number of active cores to the maximum number estimated to be within the power cap. We then reduce the remaining slack by increasing the DVFS setting in the same manner. If the power slack is negative, the controller selects the highest DVFS setting estimated within the power cap. If the minimum DVFS setting is estimated to be above the power cap, however, then the controller decreases the number of active cores. In this way, the controller prioritizes thread packing (by increasing the number of cores) over DVFS when there is a positive power slack, and prioritizes DVFS over packing when there is a negative power slack. In our implementation, we measure the power slack and apply feedback at each activation (for example, every 1 second) of the controller.

During offline power and runtime characterization, DVFS proportional gain depends on the number of active cores, and similarly, the thread-packing proportional gain depends on the DVFS setting. Thus, we

propose the use of a multigain controller in which the proportional gains for both control knobs are selected according to the current setting. We use a set of offline characterization data gathered across multiple workloads to calculate the average change in power per unit change in each control knob (for example, DVFS setting or number of cores), while the other control knob (for example, number of cores or DVFS) is held constant. We calculate a different gain for each static control-knob value. Thus, we calculate a thread-packing proportional gain for each DVFS setting, and a DVFS proportional gain for each thread-packing configuration.

Proposed projective-feedback technique

The feedback technique can successfully maintain a power cap. However, it doesn't always yield the minimum application runtime within the given cap, because it uses a heuristic to explore the Pareto frontier. In addition, although the feedback technique's multigain controller differentiates gains on the basis of the current DVFS and the number of active cores, it doesn't explicitly model different workload characteristics. We improve the feedback controller by incorporating a workload-sensitive projective-modeling technique that estimates the projected power of all possible DVFS and thread-packing settings on the power slack as a function of state measurements (that is, performance counter and core temperature measurements), and then selects the setting projected to be within the power cap that gives the least runtime. This modeling-based approach for adaptive power capping is divided into an offline and online phase. The offline phase is computationally demanding and is only performed once for a particular server configuration. We store the offline phase's results in lookup tables, accessed by the online phase. The online phase is computationally lightweight and can perform adaptive power capping on any server that has the same hardware configuration as the training server.

In the offline phase, we use an extensive set of data collected for multithreaded parallel workloads (for example, the Parsec benchmark suite) to train separate power estimation models for each DVFS and packing setting. Individualizing the models in this way emphasizes the salient characteristics at each control setting. The training data consists of per-core temperature measurements, system power measurements, and performance counter measurements, which include the number of micro-operations retired, floating-point operations, load locks, resource stalls, branch prediction misses, Level-2 (L2) cache misses, and Level-3 (L3) cache misses (to capture main memory activity). Because these metrics are gathered per core and not per workload, any model that takes them as input is globally defined and won't change depending on the workload. Our observations about performance counters indicate that much of the variation in power and delay among workloads can be attributed to memory boundedness (that is, the ratio of memory accesses to instructions executed).[12] For more memory-bounded applications, the sensitivity of power and runtime to DVFS and thread packing is far lower, as the workload incurs fixed latencies while stalling for cache misses and memory accesses. For less memory-bounded applications, the change in both power and runtime is much higher. For a frequency and thread-packing setting (f, t), a power estimate $\hat{P}_{f,t}[k]$ at time instance k is given by

$$\hat{P}_{f,t}[k] = \mathbf{c}_{f,t} \cdot \mathbf{x}_{f,t}[k]$$

where $\mathbf{x}_{f,t}[k]$ denotes the input vector of state measurements, $\mathbf{c}_{f,t}$ is the vector containing model coefficients, and the operator "$\cdot$" denotes the dot product operation. Note that the input vector includes a constant term in addition to the state measurements (16 terms total). Similar models have been successful in the past.[13]

To learn the model coefficients, we use robust regression to reduce the impact of spurious outlier measurements (for example, from operating-system calls) on the learned models. Robust regression seeks to minimize a weighted total square error—that is, $\Sigma_k w_k e_k^2$, where $e_k = \hat{P}_{f,t}[k] - P_{f,t}[k]$, in which $P_{f,t}[k]$ is the true power consumption.[14] To discard outliers, the weights w_k should increase as the error residual $|e_k|$ decreases in value. A popular weighting

function we use is the bi-square function. In this function, $w_k = 0$ when $|e_k| > r$, and $w_k = \left(1 - \left(\frac{e_k}{r}\right)^2\right)^2$ when $|e_k| \leq r$, where r is a constant that determines the extent of outlier rejection.

Because the input metrics in $\mathbf{x}$ are themselves dependent on the control setting (f, t), we must multiply the inputs by mapping ratios. These ratios project or map the measurements of the performance counters at the current setting (i, j) to any other candidate setting (f, t)—that is, $\mathbf{x}_{i,j} \xrightarrow{m} \mathbf{x}_{f,t}$. We learn ratios from the offline characterization data by computing the expected measurement values of performance counters at every setting combination. If $E[\mathbf{x}_{i,j}]$ denotes a vector of expected input values at setting (i, j), then the mapping ratios for projecting the input state from setting (i, j) to (f, t) is equal to $E[\mathbf{x}_{f,t}]/E[\mathbf{x}_{i,j}]$ (element-wise division). Lookup tables store these mapping ratios along with the model coefficients and error standard deviations for the model learned at each setting for use at runtime.

At runtime, our online power-capping system logs performance counter and temperature data periodically and identifies the optimal operating settings. If $P_{i,j}[k]$ denotes measured power at time k and the current setting (i, j), then $\delta[k] = P_{\text{cap}}[k] - P_{i,j}[k]$ denotes the power slack. Given the current input measurement vector $\mathbf{x}_{i,j}[k]$, the capping policy first identifies a set S of {DVFS, number of cores} settings such that the projected power consumption from any setting $(f, t) \in S$ is the within the power cap. That is,

$$S = \{(f, t) | \text{where } \hat{P}_{f,t}(\mathbf{x}_{i,j}[k] \xrightarrow{m} \mathbf{x}_{f,t}[k+1]) \leq \hat{P}_{i,j}[k] + \delta[k]\} \quad (1)$$

where $\hat{P}_{i,j}[k]$ is the current power estimate. Note that $\hat{P}_{i,j}[k] + \delta[k]$ is equal to $\hat{P}_{i,j}[k] + P_{\text{cap}}[k] - P_{i,j}[k] = P_{\text{cap}}[k] + (\hat{P}_{i,j}[k] - P_{i,j}[k])$. The term $(\hat{P}_{i,j}[k] - P_{i,j}[k])$ measures the projective-model error and converges to 0 when the model power matches the measured power. In this way, we use power measurement feedback to increase robustness against constant offset modeling errors. Constant deviations between the model estimates and the measured power can arise when the test system differs from the offline characterization system (for example, due to process variations, slightly different hardware configuration, or ambient-temperature variation). In Equation 1, the state measurements $\mathbf{x}_{i,j}[k]$ at the current setting (i, j) are first mapped into each potential new setting (f, t), and then each mapped measurement's power consumption is estimated using the power regression model $\hat{P}_{f,t}(\cdot)$. To pick the optimal setting, $(\hat{f}, \hat{t}) \in S$, the controller first filters S to retain the settings that use the largest number of cores $\hat{t}$ and then chooses the highest DVFS setting $\hat{f}$ within the filtered set. This selection process follows the Pareto frontier observations for minimizing runtime, as we discussed earlier. We then apply the projected optimal setting to the server, as shown in Figure 2. The overall runtime complexity of online power capping with this approach is linear with the number of control settings, which promises to scale well to larger numbers of settings in many-core architectures.

Proposed projective model without a meter

In this technique, we assume that the server doesn't have the ability to measure power consumption. This situation can arise with old servers or more economical servers that don't include power-sensing equipment. For such cases, we only use the model estimate $\hat{P}_{f,t}(\cdot)$ from the projective-feedback technique to guide setting selection. That is,

$$S = \{(f, t) | \text{where } \hat{P}_{f,t}(\mathbf{x}_{i,j}[k] \xrightarrow{m} \mathbf{x}_{f,t}[k+1]) \leq P_{cap}[k]\}$$

and the optimal setting is selected from S as we discussed earlier. In this case, power measurements are only used for offline learning and are not incorporated into runtime control. This approach's main drawback is that constant offset modeling errors translate directly to power cap violations. However, our experimental results demonstrate that this method can achieve relatively good results despite lacking hardware power telemetry. In addition, if power cap violations are normally distributed with no bias for positive or negative power slack, we can successfully leverage the technique for capping average power.

Experimental results

We evaluated the Pack and Cap methodology in terms of its accuracy in adhering to dynamic-power caps and application performance. To quantify the accuracy, we measured the power cap error, which is the average negative power slack magnitude normalized by the power cap value. We measured the performance by simply using workload runtimes. We performed all experiments on a dual Intel quad-core Xeon E5520 system. Each processor has six DVFS settings, ranging from 1.60 GHz to 2.27 GHz in 0.13-GHz increments. We disabled the hyper-threading feature. The server ran a Linux kernel 2.6.10.8 operating system. We used pfmon to collect performance counter data and lm-sensors (Linux-monitoring sensors) to poll the on-chip thermal sensors. We measured the server's total power consumption using an Agilent 34410A digital multimeter.

For comparison, we implemented a baseline feedback technique that incrementally adjusts both the DVFS setting and the number of active cores (that is, reduces the setting by one increment for negative power slack, and increases it by one increment for positive slack). This technique engages DVFS first to meet the power cap, and adjusts only the number of active cores when limited by the maximum or minimum frequency. This policy is a natural extension of the DVFS feedback techniques used in previous work for meeting power caps.[3] By comparing to the baseline technique, we quantify the benefits of incorporating our Pareto frontier observations into our proposed control techniques.

For all experiments, control is activated once per second. The time overhead for calculating control decisions for all Pack and Cap techniques is on the order of 10 ms, which is less than 1 percent of the control activation period. The time overhead for performing DVFS control in the hardware is on the order of tens of microseconds. Although the overhead for shifting threads among the cores in thread packing is potentially higher, our workload runtime results automatically account for all performance overheads. For all feedback techniques, we avoid any actuation if the power consumption is within 2 W below the power cap, because our offline characterization shows that setting changes trigger power changes larger than 2 W. We apply the same rule for the technique that uses projective modeling without a meter, except that we determine the power consumption through modeling rather than measurement.

To evaluate the proposed techniques' effectiveness in a realistic situation in which data center nodes must meet dynamic-power caps requested by ISOs, we set an aggressive dynamic cap that changes every 10 seconds to a random value in the range of 120 W to 170 W for each Parsec workload running with eight threads. Figure 3 compares each workload's runtime and capping accuracy. We normalized the runtime values to those observed for the baseline feedback technique. The three proposed Pack and Cap techniques outperform the baseline technique significantly in both runtime and power cap accuracy. The baseline feedback controller delivers larger power cap errors (an average of 7.5 percent) with larger runtimes, because it doesn't use power slack magnitude or knowledge of the Pareto frontier. Our feedback and projective-feedback techniques deliver the best results in terms of accuracy, with average cap errors of 3.5 and 3.6 percent, respectively. The projective-feedback technique outperforms the feedback technique in runtime with 0.90× normalized runtime compared to 0.94×. This improvement is a result of the projective-feedback technique's superior Pareto frontier-tracking ability.

Figure 3b also shows that, as expected, the technique involving a projective model without a meter delivers worse power-capping accuracy, with an average error of 4.73 percent. This approach's runtime (0.91×) is comparable to the projective-feedback technique. Without using measurement feedback, the technique involving a projective model without a meter has no way to correct for modeling errors. These errors can become significant if the system differs between online testing and offline characterization, or if the model poorly captures a particular workload. For instance, bodytrack shows larger runtime and worse power-capping accuracy compared to the other proposed techniques, indicating power

Figure 3. Runtime and average power cap tracking accuracy of proposed techniques: runtime comparison (a) and average cap error comparison (b). We normalized the runtime values to those observed for the baseline feedback technique. The three proposed techniques outperform the baseline technique in both runtime and power cap accuracy.

overestimation and underestimation. Nevertheless, the reasonable accuracy result indicates that this technique is a viable option for power capping without power measurement equipment. Feedback-based techniques eliminate such deviations at the expense of incorporating hardware power meters.

Figure 4 illustrates the measured power consumption and the control decisions (DVFS, number of cores) for the `blackscholes` benchmark over time. The baseline, proposed feedback, proposed projective feedback, and proposed projective model without a meter have an average setting of (1.85 GHz, 6.0 cores), (1.93 GHz, 6.3 cores), (1.92 GHz, 6.6 cores), and (1.92 GHz, 7.1 cores), respectively. The projective-feedback technique meets the power cap using a higher number of cores and lower average DVFS setting than the feedback technique, which clearly indicates superior Pareto frontier tracking. All the proposed techniques track the power cap tightly, eliminating power slack and reducing runtime relative to the baseline feedback technique. The disadvantage of using the projective model without a meter can be clearly seen in the 60- to 70-second interval. The power projected by the model deviates from the measured power, leading to a significant power-capping error. However, the projective-feedback technique's measurement feedback successfully eliminates this modeling error.

Table 1 summarizes the tradeoffs between our proposed techniques. We also implemented

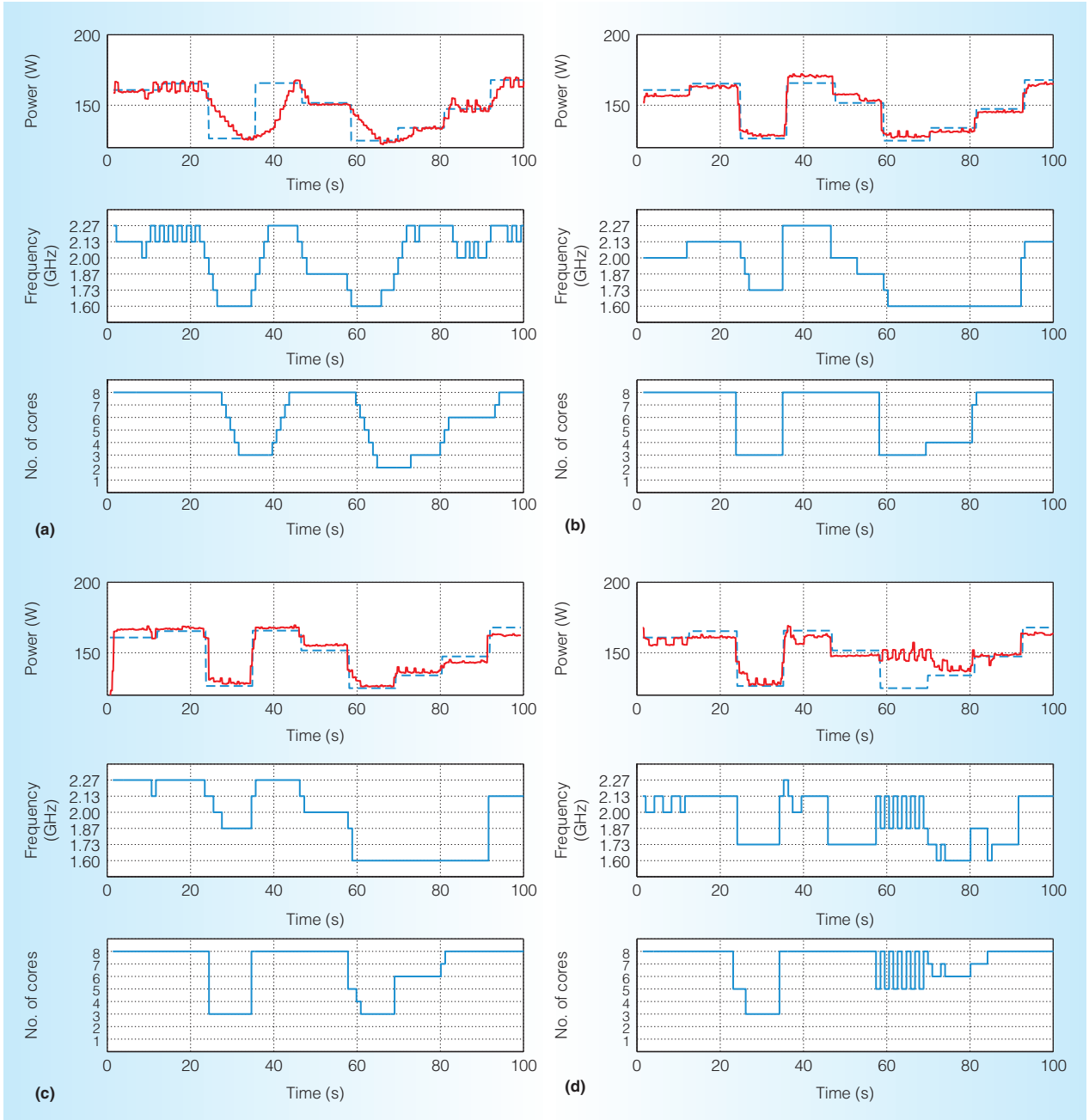

Figure 4. A detailed exploration into the first 100 seconds of the `blackscholes` application, demonstrating the selected DVFS and thread-packing settings along with system power consumption. The figure gives the power consumption and frequency of operation for the baseline feedback (a), proposed feedback (b), proposed projective feedback (d), and proposed projective model without a meter (d).

our modeling-without-meter approach on a single-processor server and compared it to our earlier capping work, which uses multinomial logistic regression (MLR) for classification and doesn't use power meters.[10] Projective modeling without a meter shows a consistent improvement of 20 to 30 percent in runtime while still meeting the power caps with the same accuracy as the MLR classifier.

As part of the Pack and Cap methodology, we demonstrated that, through

Table 1. Comparison between the proposed Pack and Cap methods.

Method	Hardware requirements	Offline characterization	Power cap error (% above the cap)	Application runtime
Baseline feedback	Power meter	None	7.52	1.00×
Proposed feedback	Power meter	Simple	3.51	0.94×
Proposed projective feedback	Power meter, performance counters, and thermal sensors	Intensive	3.69	0.90×
Proposed projective model without a meter	Performance counters and thermal sensors	Intensive	4.73	0.91×

improved feedback and workload characterization techniques, we can substantially improve power-capping accuracy. Our next steps include extending the Pack and Cap methodology to heterogeneous architectures and integrating it within group power-capping schemes in which sets of nodes are managed together under a common power constraint. We will also expand our techniques to differentiate between critical and noncritical threads when performing thread packing.[15]

Many open research problems remain in the emerging area of power capping. Virtualization is becoming more widely used in clusters for resource-consolidation purposes, bringing new challenges in system telemetry and energy management. Group power capping, where a set of nodes are managed together under a power constraint, is an essential next step in enabling efficient cluster-level management. For designing higher-level capping policies for clusters, message-passing interface (MPI) among nodes and network delays contribute to the overall power-performance tradeoffs. Another closely related area to power capping is cooling control, as cooling costs reach nearly half of the overall cost at clusters today. Intelligent cooling design and management strategies, coupled with server power capping methods, are expected to provide significant additional gains in energy efficiency. MICRO

Acknowledgments

Sherief Reda and Ryan Cochran are partially supported by National Science Foundation grants 0952866 and 1115424. Ayse K. Coskun has been funded in part by VMware and Oracle.

References

1. D. Filani et al., ''Dynamic Data Center Power Management: Trends, Issues, and Solutions,'' *Intel Technology J.,* vol. 12, no. 1, 2008, pp. 59-68.
2. I. Paschalidis, B. Li, and M. Caramanis, ''A Market-Based Mechanism for Providing Demand-Side Regulation Service Reserves,'' *Proc. 50th IEEE Conf. Decision and Control,* IEEE CS, 2011, pp. 21-26.
3. C. Lefurgy, X. Wang, and M. Ware, ''Power Capping: A Prelude to Power Shifting,'' *Cluster Computing,* vol. 11, no. 2, 2008, pp. 183-195.
4. X. Wang and M. Chen, ''Cluster-Level Feedback Power Control for Performance Optimization,'' *Proc. IEEE 14th Int'l Symp. High Performance Computer Architecture,* IEEE CS, 2008, pp. 101-110.
5. A. Gandhi et al., ''Power Capping via Forced Idleness,'' *Proc. Workshop Energy-Efficient Design,* 2009; http://repository.cmu.edu/compsci/868.
6. X. Wang et al., ''SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers,'' *IEEE Trans. Parallel and Distributed Systems,* vol. 23, no. 1, 2012, pp. 168-176.
7. L.A. Barroso and U. Hölzle, *The Datacenter as a Computer,* Morgan and Claypool, 2009.
8. M. Floyd et al., ''Introducing the Adaptive Energy Management Features of the Power7 Chip,'' *IEEE Micro,* vol. 31, no. 2, 2011, pp. 60-75.
9. C. Bienia, ''Benchmarking Modern Multiprocessors,'' doctoral dissertation, Dept. Computer Science, Princeton Univ., 2011.
10. R. Cochran et al., ''Pack & Cap: Adaptive DVFS and Thread Packing under Power Caps,'' *Proc. 44th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* ACM, 2011, pp. 175-185.

11. C.D. Spradling, ''SPEC CPU2006 Benchmark Tools,'' *SIGARCH Computer Architecture News,* vol. 35, no. 1, 2007, pp. 130-134.
12. C. Isci, G. Contreras, and M. Martonosi, ''Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management,'' *Proc. 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* IEEE CS, 2006, pp. 359-370.
13. B. Lee and D. Brooks, ''Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction,'' *Proc. 12th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM, 2006, pp. 185-194.
14. E. Alpaydin, *Introduction to Machine Learning,* MIT Press, 2004.
15. A. Bhattacharjee and M. Martonosi, ''Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors,'' *Proc. 36th Ann. Int'l Symp. Computer Architecture,* ACM, 2009, pp. 290-301.

Sherief Reda is an assistant professor in the School of Engineering at Brown University. His research interests include thermal and power modeling, management, and physical design. Reda has a PhD in computer science and engineering from the University of California, San Diego. He is a member of IEEE and the ACM.

Ryan Cochran is a PhD student at Brown University. His research areas include thermal modeling, management, and energy-proportional computing for large-scale data centers. Cochran has a BSC from Brown University. He is a member of IEEE.

Ayse K. Coskun is an assistant professor in the Electrical and Computer Engineering Department at Boston University. Her research interests include energy-efficient computing, multicore architectures, and 3D stacked architectures. Coskun has a PhD in computer science and engineering from the University of California, San Diego. She is a member of IEEE and the ACM.