A Case of System-Wide Power Management for Scientific Applications

Zhuo Liu† Jay Lofstead‡ Teng Wang† Weikuan Yu†
Auburn University† Sandia National Laboratories‡
{zhuoliu,tzw0019,wkyu}@auburn.edu gflofst@sandia.gov

***Abstract*—The advance of high-performance computing systems towards exascale will be constrained by the systems' energy consumption levels. Large numbers of processing components, memory, interconnects, and storage components must all be considered to achieve exascale performance within a targeted energy bound. While application-aware power allocation schemes for computing resources are well studied, a portable and scalable budget-constrained power management scheme for scientific applications on exascale systems is still required. Execution activities within scientific applications can be categorized as CPU-bound, I/O-bound and communication-bound. Such activities tend to be clustered into 'phases', offering opportunities to manage their power consumption separately. Our experiments have demonstrated that their performance and energy consumption are affected differently by CPU frequency, an opportunity to fine tune CPU frequency for a minimal impact on the total execution time but significant savings on the energy consumption. By exploiting this opportunity, we present a phase-aware hierarchical power management framework that can opportunistically deliver good tradeoffs between system power consumption and application performance under a power budget. Our hierarchical power management framework consists of two main techniques: Phase-Aware CPU Frequency Scaling (PAFS) and opportunistic provisioning for power-constrained performance optimization. We have performed a systematic evaluation using both simulations and representative scientific applications on real systems. Our results show that our techniques can achieve 4.3%-17% better energy efficiency for large-scale scientific applications.**

I. Introduction

Exascale computing systems are projected to arrive around 2018. Such systems will be roughly 50 times faster than the current top computer system, e.g., Titan at Oak Ridge National Laboratory [4]. A real challenge with developing an exascale system is that its energy consumption must be capped within a budget, e.g., 20 Megawatts according to a report from the U.S. Department of Energy [22]. Such power consumption objectives require exascale systems to be 20 times more energy efficient than the most efficient system today in terms of GFLOPS/Watt [2, 10]. To address this issue, lots of efforts have been undertaken to bring faster and more energy efficient computing [3], memory [12] and storage [14] hardware to build large-scale supercomputers.

In addition to developing energy-efficient computing hardware, managing system power within a given budget while maintaining an acceptable performance goal will become an increasingly important issue for the applications on the platform. However, there is a lack of a portable and scalable, energy-efficient computation and data management scheme for scientific applications running on HPC platforms.

Recently, heuristic feedback-based power management schemes have been developed to control power within a given cap while delivering performance [19, 24]. However, these techniques did not consider different runtime characteristics of the phases of an application's runtime. A typical scientific application performs computation over a data set in a periodic manner. Each period is called a *timestep* or iteration. Each time step consists of several phases: computation, data exchange (communication) and possibly I/O, each with different CPU, memory and power usage attributes. Application I/O phases may consume a large amount of time and energy for storing restart/checkpoint files and analysis results while performing very little computation. Varying the CPU frequency and voltage based on these distinct needs can reduce the power consumption while having very little impact on the overall wall clock performance. While traditional power-aware techniques focus either on the computation part or on the I/O and communication part, little research has been conducted to take all phases into account for managing energy consumption. For example, in some applications, the I/O and communication phases may take considerable time with a relatively low demand on computation power. A poor selection of CPU frequency may lead to some combination of wasted CPU cycles, degraded performance, or unnecessary energy expenditure.

Therefore, in order to achieve a better trade-off between energy usage and application execution efficiency for a computing system with a power budget, we propose a Hierarchical Power Management framework (HPM) which consists of two main techniques: Phase-Aware CPU Frequency Scaling (PAFS) and opportunistic provisioning for power-constrained performance optimization. PAFS carefully monitors an application's working phase and orchestrates the CPU frequency and voltage to achieve nearly identical performance with reduced energy consumption. Furthermore, built on top of PAFS, to cap the computing cabinet's power consumption within a given budget while optimizing its job throughput, we devise an opportunistic provisioning technique for power-constrained performance optimization. The HPM is implemented through three main components: Job Phase Monitor (JPM), Local Power Managers (LPM), and Global Power Manager (GPM).

978-1-4799-0898-1/13/$31.00 ©2013 IEEE

These components monitor each running job's working phase and dynamically assign a certain power level to each compute node according to every job's working phase and the global power budget.

Our contributions are three-fold. (1) We introduce the PAFS technique that enhances scientific applications' energy-efficiency without undue increase of wall clock time. (2) We provide an opportunistic provisioning scheme for power-constrained optimization, which offers an efficient and application-transparent power management solution for large-scale computing systems. (3) We evaluate our techniques through experiments on both real cluster and simulated large-scale systems. Results for scientific applications and production cluster logs show that our system-wide management framework can improve system energy efficiency by 4.3%-17% compared to baseline techniques, which results from both the wall clock time and energy consumption improvement.

II. Related Work

Freech et al. comprehensively analyzed the energy-time tradeoff for a wide range of applications on a power-scalable cluster [7]. They studied how application performance and energy use increases with the number of nodes and introduced metrics to categorize applications on the relationship of time to CPU frequency. Liu et al. provided a set of tools and metrics for evaluating energy consumptions and I/O performance of storage systems [15, 16]. Sherwood et al. proposed an automatic tool called SimPoint [20] to exploit program phase behavior for guiding program analysis, simulation and optimization. They first conducted an off-line code execution to create code behavior signatures and used clustering analysis to group similar parts of program's execution into phases. The evaluation's granularity was at the basic block level and focused on single node programs. Huang et al. provided a subroutine-based positional adaptation [8] for saving energy by static or dynamic instrumentation. The static instrumentation required many off-line profiling runs on target platforms while the dynamic instrumentation caused run-time overhead at start-up. Different from off-line code analysis and binary instrumentation, our techniques determine a running job's working phase by dynamically analyzing its node's working status and performs a phase-aware and application-transparent CPU frequency scaling for conserving system energy.

Raghavendra et al. provided a coordinated multilevel power management [19] for data centers. A control-theoretic core was introduced in the system model and multiple-level control channels were built for different power management solutions and requirements. Similarly, Wang et al. introduced a cluster-level feedback power control for performance optimization [24]. A model predictive control theory was employed to scale the servers' CPU frequencies to achieve better performance under a power budget limit. These techniques worked well for uni-processor systems but did not consider multi-core nodes with per-core DFS enabled. In addition, the distinctions between computation phases and I/O phases for scientific application jobs are not addressed.

Tiwari et al. presented a per-loop frequency selecting technique [11]. It built up an optimal benchmark-frequency library from tests on a wide range of benchmarks, then acquired application signatures by lightweight static analysis and runtime tracing. After that, a library was queried to choose an optimal frequency and frequency scaling procedures were inserted into compiled executables by binary instrumentation. This scheme worked well for small systems with a limited number of applications but was not applicable for large-scale time-critical batch processing systems.

Lim et al. implemented an MPI runtime system [13] to dynamically reduce CPU frequencies during communication phases in MPI programs for better energy-efficiency. Wang et al. provided scheduling heuristics for frequency scaling in precedence-constrained parallel tasks [23]. Kappiah et al. exploited inter-node slackness [9] in MPI programs and reduced processor frequency on those less-burdened nodes to save energy. Differently, our work provides hierarchical power management at the system level while meeting a power budget cap at the same time. Also, power gating was exploited for better core balances in CMPs [17], which is orthogonal to our DVFS based power management and can complement it to further reduce power consumptions for CPU cores' idle periods.

III. Phase-Aware CPU Frequency Scaling (PAFS)

Analyzing the behavior of many scientific simulations yields a broad generalization describing the activities performed by these applications. Each of these activities can be distilled down to different phases each performed periodically progressing the simulation. Using a simple model of these phases makes changing the CPU frequency a less common and less intrusive operation while maintaining most of the advantages of scaling the frequency at finer granularity. In the case of massively parallel simulations, this reduced granularity is sufficient for achieving most of the energy efficiency gains with low overhead. This section first explores such phases and the impact of CPU frequency scaling has on the wall clock time and the performance of these phases, then the Phase Aware Frequency Scaling (PAFS) technique is presented to exploit such different impacts for energy efficiency improvement.

A. Different working phases

Different applications can be categorized to be CPU-bound, memory-bound or communication-bound. Inside one scientific application, its running progress can be divided into repetitive periods of computation, I/O and communication phases. Based on our experimental observations, different types of applications and different phases within these applications have varying sensitivity to CPU frequency changes. For example, the run time of computation-bound working phases can vary linearly with CPU frequency while the performance of I/O and network bounded phases are almost not affected when scaling down or up CPU frequencies. Some applications may have working phases that consist of mixture of many short computation phases and I/O phases that are hard to detect.

Such working phases, termed *undistinguished phase*, display a medium performance sensitivity to CPU frequency level.

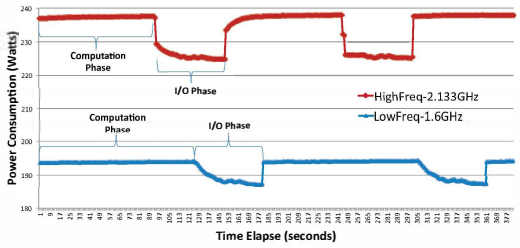


Fig. 1: Computation and I/O Phases for Low and High CPU Frequencies during Application Execution

Fig. 1 illustrates a typical execution of the NASA GEOS5 climate simulation [1]. It consists of multiple time steps with each time step divided into two distinct phases. The first phase is a computation phase during which the application is performing computation for the climate and geographic simulation and is CPU-bound. The second phase is an I/O phase when the application writes simulation status to storage where CPU frequency is not a bound. The computation phase causes 100% utilization of each CPU core leading to high power consumption while the I/O phase mainly focuses on network transfer and memory access leading to the same 100% CPU utilization but consuming less power. The red and blue curves show power consumptions for CPU frequency of 2.133 GHz and 1.6 GHz, respectively. As we can observe, when scaling the CPU frequency from high to low, the power consumption can be reduced by about 20% while the execution time may be prolonged. For different running phases in a time step, the extend degree of execution time can vary. Specifically, for the computation phase, the performance degradation ratio is proportional to the reduction ratio of CPU frequency. However, the I/O phase is only prolonged very slightly. These disparate impacts motivate us to develop a phase-aware technique for energy efficient frequency scaling.

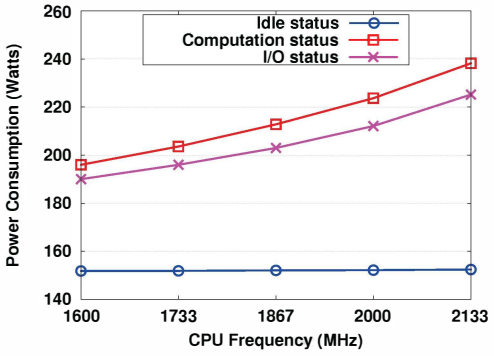


Fig. 2: Power consumption of a compute node in different statues with varying CPU frequency levels

Our compute nodes are equipped with an Intel® Xeon® E5405 8-core CPU whose cores can be independently clocked to five frequency levels: 1.6 GHz, 1.733 GHz, 1.866 GHz, 2.0 GHz and 2.133 GHz. As a job can be running in different working phases, thus each compute node may experience one of the three types of different statues: serving a job in computation phase, serving a job in I/O or communication phases and idle (not serving any application). In different statues, the compute node will have different resource utilization and power consumption. We can observe from Fig. 2 for different working statues of GEOS-5 application how the node's power consumptions vary with CPU frequency levels.

To keep the model simple, we adopt a linear regression for node power consumption and its CPU frequency (the model's residual standard error is 2.436). When CPU cores are idle, the power consumption changes very slightly (variations less than 0.4%) with the CPU frequency varying from the lowest level to the highest level. Therefore, we reasonably assume the power consumption P_{idle} is constant for various CPU frequency levels when the cores are idle.

B. PAFS vs default DVFS schemes

PAFS works by carefully monitoring a running application's working phase and orchestrates the CPU frequency and voltage to achieve nearly identical performance with reduced energy consumption. First, we categorize the job's working phases into three types: I/O or communication phase, computation phase and undistinguished phase (consisting of short periods of I/O, communication or computation work and its execution time is moderately sensitive to frequency). Then, for the I/O and computation phases that are not CPU-bound, as the execution time is not sensitive to CPU frequency, the CPU frequency is reduced to the lowest level consuming less energy even with a slight increase on the phase time. For the computation phase, the higher the frequency, the shorter the execution time and net energy consumption. Thus we can achieve better energy-efficiency if the power budget is not more limiting. For the undistinguished phase, its frequency is scheduled according to the dynamic power provisioning.

Cpufreq is a popular DVFS tool integrated in many state-of-the-art computing platforms. It scales each CPU core's clock frequency and voltage to different levels conserving power when there is less demand for compute cycles. Cpufreq provides several representative scaling schemes such as 'on-demand', 'performance', 'userspace', 'powersave' and 'conservative' with each offering different characteristics. 'Ondemand' shifts to a minimum frequency when there is no or minimal load and shifts to maximum frequency immediately under high load to minimize elapsed time; 'performance' always scales the maximum frequency to achieve best performance; 'userspace' offers dynamic, manual control of the frequency; 'powersave' always chooses the minimum frequency; and 'conservative' chooses the minimum frequency if there is no load and gradually changes the frequency according to load intensity [21].

Fig. 3 illustrates the behaviors of PAFS and three default DVFS schemes when there are two processes running on a quad-core CPU compute node assuming that two-cores are used for each process. The green squares represent a low frequency state while the red squares are for high frequency. When the application switches from a computation phase

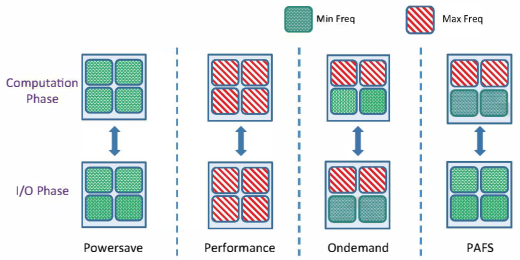


Fig. 3: PAFS and other DVFS modes in a four-core CPU.

to an I/O (or communication) phase, the 'powersave' and 'performance' modes will keep all cores in lowest power/frequency level and highest power/frequency level respectively. For 'ondemand', as only two cores are used, it will scale the active cores in high frequency while scale the idle cores in low frequency. Different from the modes above, the PAFS we propose will scale all cores to lower frequency during I/O (or communication) phases while scale the active cores to high frequency during computation phases. For undistinguished phases, CPU frequency is decided with respect to real-time power budget status. In doing so, the scheme can conserve significant energy with minimal performance loss.

IV. Hierarchical Power Management

In this section, we first present the problem of power constrained performance optimization. Then the hierarchical power management framework is proposed to address this problem. In this framework, a novel opportunistic provisioning for performance optimization is devised together with the PAFS to optimize system energy efficiency.

A. Performance Optimization with a Power Cap

Given an arbitrary eight-node computing cabinet, it will receive scientific computation jobs from the system job manager. Fig. 4 illustrates the utilization of the cabinet over time. Our main target is to maximize system's energy efficiency without exceeding the cabinet's power capping.

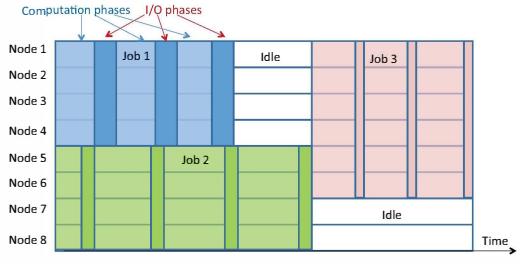


Fig. 4: Jobs in different phases on system nodes

A system cabinet's power consumption is summed up with each compute node's power consumption. For simplicity in this initial exploration of this work, we solely focus on the contribution of the compute node's CPU to energy consumption for the entire machine. Future work will expand this model to incorporate other factors such as cooling, memory, network, and storage. A node's power consumption $P_{node\,i}$ is determined by its working status, CPU frequency ($freq$) and CPU utilization (u). Different working status has varied linear coefficients - α^{status} and β^{status} in the power model. On one hand, we should cap the system's total power consumption P_{system} below a given budget P_{cap}; on the other hand, we want to minimize the average of all jobs' EDP (Energy Delay Product) to achieve a best trade-off between energy and application performance. Thus we can normalize our model and problem as follows:

$$\begin{aligned} P_{node\,i} &= f(status, freq, u) \\ &= (\alpha^{status} * freq + \beta^{status}) * u + P_{idle} \end{aligned} \tag{1}$$

$$P_{system} = \sum P_{node\,i} \tag{2}$$

$$EDP_{job\,j} = E_{job\,j} * t_{job\,j} \tag{3}$$

$$\begin{aligned} &\text{Minimize} \quad Avg\ EDP_{job} \\ &\text{Subject to} \quad P_{system} <= P_{cap} \end{aligned} \tag{4}$$

B. Hierarchical Power Management Framework

Fig. 5 illustrates how the power management framework operates at the system level by showing the global power management and each of the compute nodes housed in each cabinet. The PDU (Power Distribution Unit) takes charge of distributing power to server racks in the system. To implement system-wide power management for scientific application jobs, we design a hierarchical framework to manage high performance computing systems' power, which is implemented through three main components: **GPM**, **LPM** and **JPM**.

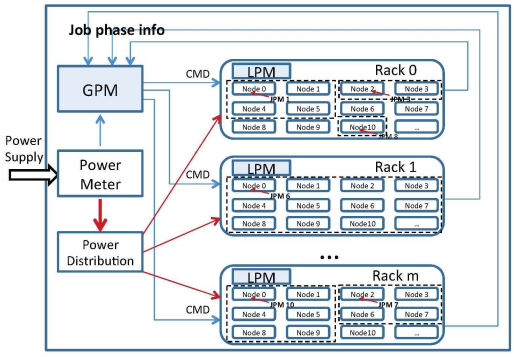
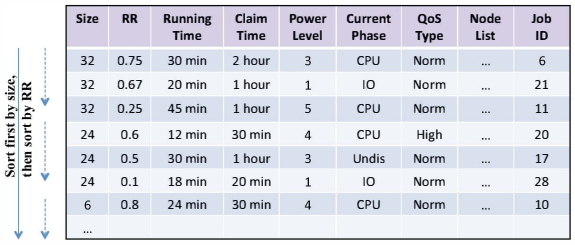


Fig. 5: Hierarchical Power Management

1) **GPM (Global Power Manager)**: A system contains a GPM to manage power for all of the computing racks. It maintains a sorted list of all running jobs. It periodically receives jobs' working phase information from each rack and updates the job list. To cap the total system power consumption within the given budget, it executes the power provisioning algorithm as described in Section IV-C. Specifically, inside a job, nodes working for this job will be assigned with the same CPU frequency level to avoid the performance throttling effects. Real-time global power consumption measured by the power meter will be sent to GPM as feedback information. For an exascale system that contains a huge number of cabinets or racks, a higher-order GPM with global

job allocation information and partitioned power budgets can be maintained to manage power capping in the whole system.

2) **LPM (Local Power Manager)**: Every compute rack has an LPM that collects jobs' phase information from the JPMs in this rack and send the collected information to GPM periodically. After receiving the power management commands from GPM, the LPM will conduct corresponding frequency scaling operations to the affected jobs' nodes.
3) **JPM (Job Phase Monitor)**: A JPM is a daemon launched with each job on one of the job's assigned computing nodes. The JPM determines the running phase of the job by monitoring the node's system statistics information which includes CPU utilization, I/O throughput and network throughput and send the phase information to the rack's LPM periodically. As the number of JPM daemons is equal to the number of running jobs (rather than the number of compute nodes) in the system, such overheads are modest and thus help our power management framework to scale well.



Size	RR	Running Time	Claim Time	Power Level	Current Phase	QoS Type	Node List	Job ID
32	0.75	30 min	2 hour	3	CPU	Norm	...	6
32	0.67	20 min	1 hour	1	IO	Norm	...	21
32	0.25	45 min	1 hour	5	CPU	Norm	...	11
24	0.6	12 min	30 min	4	CPU	High	...	20
24	0.5	30 min	1 hour	3	Undis	Norm	...	17
24	0.1	18 min	20 min	1	IO	Norm	...	28
6	0.8	24 min	30 min	4	CPU	Norm	...	10
...								

Fig. 6: Job list maintained by GPM

C. Opportunistic Power Provisioning

For every control period, to minimize jobs' execution time and power consumption and fulfill the system power budget requirement, GPM will adopt an opportunistic provisioning for power constrained performance optimization to distribute power budget to all nodes for multiple jobs running on the system. As shown in Fig. 6, a sorted running jobs' list is maintained by GPM. The job list maintains each running job's information which contains job ID, size (number of nodes assigned), RR (Remaining Ratio), running time, claim time, assigned node list, QoS requirement type and current job phase. QoS requirement is categorized into two types: high and normal (most jobs adopt the normal QoS type). And the job list is sorted firstly by the job size and then further sorted inside every sublist for jobs with the same size by their RR values. **RR** is defined to characterize the completeness degree of one job and it is calculated by $RR = \frac{ClaimTime - RunningTime}{ClaimTime}$.

As depicted in Algorithm 1, the GPM conducts phase-aware frequency scaling when certain conditions holds. Two types of scanners are used to search for candidate jobs: a global scanner termed $scanner^{size}$ is used to choose a target sublist containing jobs with a certain size; each sublist has a local scanner termed $scanner^{RR}$, which is used for designating a job inside the sublist. As we can see, when the system power P_{sys} is larger than the scaling-down threshold - TH^{down} or smaller than the scaling-up threshold - TH^{up} times of the power budget - P_{cap}, the Power Provisioning Algorithm will search for candidate jobs for frequency scaling. By setting the $scanner^{size}$, GPM will choose the sublist pointed by $scanner^{size}$ from the job list in a round-robin fashion. By setting the $scanner^{RR}$ for each sub-list, GPM will further choose the job pointed by the $scanner^{RR}$. Differently, for scaling down, GPM initializes the $scanner^{size}$ and every

Algorithm 1 Opportunistic Power Provisioning Algorithm

while True **do**
  Sleep(interval)
  Update job list
  Scale all nodes of jobs in I/O phase down to L_{min}
  if New jobs come or $P_{sys} > TH^{down} * P_{cap}$ **then**
    Add jobs to the list as needed and update list
    $Scanner^{size} \leftarrow 0$
    Every $Scanner^{RR} \leftarrow 0$
    while $P_{sys} > TH^{down} * P_{cap}$ **do**
      Scale down one job according to Algorithm 2
    end while
    Continue
  end if
  if Jobs finished/terminated or $P_{sys} < TH^{up} * P_{cap}$ **then**
    Delete jobs from the list as needed and update list
    $Scanner^{size} \leftarrow Num_{sublists}$
    Every $Scanner^{RR} \leftarrow its\ sublist's\ size$
    while $P_{sys} < TH^{up} * P_{cap}$ **do**
      if All jobs in undes or CPU phases are at L_{max} **then**
        break;
      end if
      Scale up one job according to Algorithm 3
    end while
  end if
end while

Algorithm 2 Scale down one job

$Scanner^{size}$ to choose the sub-list i
while $Scanner_i^{RR} < size^{sublist\ i}$ and !($Job.PL > L_{min}$ and $Job.QosType == normal$) **do**
  $Scanner_i^{RR}++$
end while
if Find a candidate job **then**
  if The job is in CPU or Undes phase **then**
    Scale down all nodes of this job by one level
  else {/*I/O phase*/}
    Scale all nodes of this job down to L_{min}
  end if
  $Scanner_i^{RR}++$
end if
$Scanner^{size}++$

Algorithm 3 Scale up one job

1: **while** $Scanner_i^{RR} > 0$ and !($Job.PL < L_{max}$ and $Job.Phase == CPU\ or\ Undes$) **do**
2: $Scanner_i^{RR} - -$
3: **end while**
4: **if** Find a candidate job **then**
5: **if** The job is in CPU phase **then**
6: Scale all nodes of this job up to L_{max}
7: **else** {/*Undistinguished phase*/}
8: Scale all nodes of this job up by one level
9: **end if**
10: $Scanner_i^{RR} - -$
11: **end if**
12: $Scanner^{size} - -$

$scanner^{RR}$ to point to the heads of the job list and each sublist to firstly scale down jobs with larger size and more remaining work according to Algorithm 2; while for scaling up, it initializes the scanners to the tails so as to firstly scale up jobs with smaller size and less remaining work according to Algorithm 3.

As described in Algorithm 2, at first, $scanner^{RR}$ is incrementally moved in the sublist to find a qualified job whose power level (PL) is larger than minimum level - L_{min} and QoS type is normal. With a job found, all its nodes' CPU frequency will be scaled down by one level for jobs in computation or undistinguished phase and scaled down to L_{min} for jobs in I/O or communication phase respectively.

In Algorithm 3, similarly, $scanner^{RR}$ is decrementally moved in the sublist to find a qualified job whose power level is smaller than minimum level - L_{max} and current phase is either undistinguished or computation. Then all the found job's nodes' CPU frequency will be scaled up by one level for jobs in undistinguished phase or scaled up to maximum level - L_{max} for jobs in computation phase.

V. Experimental Evaluation

In this section, we describe the setup of experimental environment and our evaluation results on both a cluster system and a large-scale simulator. First, single-application results are shown for PAFS. Then, multiple-application results are presented for the hierarchical power management framework which integrates both the PAFS and opportunistic power provisioning techniques.

A. Experimental Setup

Our experiments are conducted on both a in-house cluster and a large-scale cluster simulator. Our cluster contains 12 nodes, each of which is equipped with dual-socket quad-core 2.13GHz Intel Xeon processors, 8GB of DDR2 800 MHz memory and Linux 2.6.18-164.el5 kernels. Eight of the nodes are used for application jobs while the remaining four host a 1442 GB Lustre File system with three OSTs. We only report the power consumption results collected from the eight compute nodes.

On the cluster, we evaluate our techniques by running two applications (GTC2 [18] and GEOS5 [1]) and a microbenchmark called MADbench [5]. GTC2 is a PIC-based fusion simulation application for studying plasma microturbulance in tokamak devices. GEOS-5, the Goddard Earth Observing System model, is another scientific application being widely used by NASA for observing system modeling, conducting climate, weather prediction and other scientific research. MADbench is a benchmark for testing the overall integrated performance of I/O, communication and calculation subsystems of large-scale parallel architectures under the stresses of real scientific applications workload.

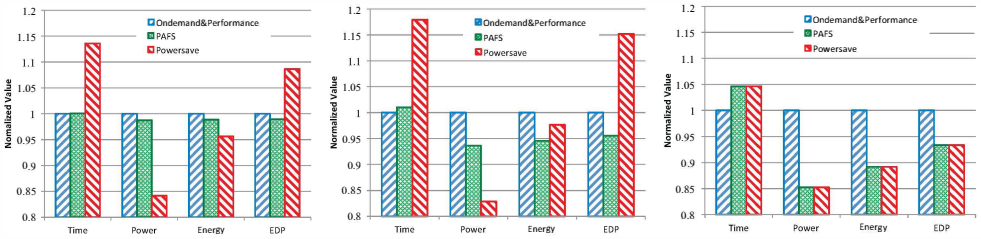
To project our techniques on large-scale clusters, we design a trace-driven simulator that simulates the job scheduling, execution, node allocation and DVFS power management across thousands of compute nodes. We adopt two real-world job traces that are based on workload logs from production clusters [6]. The RICC trace contains a running log consisting of 447794 jobs collected on Fujitsu RX200S5 Cluster with 1024 quad-core nodes from May 1st to Sep 30th 2010. The META log contains 103656 jobs collected on Czech MetaCentrum grid with 806 compute nodes for six months. To represent applications with different working phases, we randomly assign a job type to each job entry with a time ratio of undistinguished phase: I/O phase: computation phase as 1:1:3 (phases are switched midst each job's progress). Our simulation results are collected by running two 4000-job sequential sections from the RICC log and from the META log, respectively.

B. Frequency scaling and power capping for a cluster

In this section, results for running standalone applications and multiple applications at the same time with our techniques compared to with baseline schemes are described.

1) Standalone execution results: We firstly assess the performance of PAFS via running each benchmark (MADbench, GEOS5 and GTC2) individually on 4 compute nodes and measure their job execution time, consumed power, energy, and EDP (Energy-Delay Product) when power budget limitation is not issued. We compare the results of using PAFS with that of using another three DVFS schemes, which are *performance*, *ondemand* and *powersave*, respectively. During the experiments, we observe very close or nearly identical results when using *performance* and *ondemand* across all the tests. For clarity in the graphs, we report the average of those results and name it as *ondemand & performance*.

Fig. 7 shows normalized values of time, power, energy and EDP when using different power management schemes for MADbench, GEOS5 and GTC. Overall, PAFS outperforms the other alternatives in terms of EDP by from 5% to as much as 17%. Compared to *ondemand & performance* PAFS causes minimal or no performance degradation (execution time delay) but can save significant power and energy because of appropriate CPU frequency scaled down during CPU incritical phases; compared to *powersave*, PAFS achieves much



(a) MADbench (b) GEOS5 (c) GTC2

Fig. 7: Normalized values of time, power, energy and EDP of different DVFS modes for MADbench, GEOS5 and GTC.

better performance. That is why PAFS leads to better energy and performance trade-off over other DVFS schemes.

For MADbench, we observe that PAFS reduces the EDP by 2% and 8%, when compared to *ondemand & performance* and *powersave* respectively. This is because there is a data creation process in the beginning, PAFS correctly recognizes it as I/O phase and scales down the CPU frequency.

For GEOS5, PAFS performs 5% and 17% better than *ondemand & performance* and *powersave* in terms of EDP. GEOS5 consists of multiple iterations of long I/O and computation phases that provide PAFS more optimization opportunities. This explains why PAFS achieves a higher improvement ratio for the GEOS5. Fig. 8 provides detailed per-node average power and execution time of running GEOS5 application with different DVFS modes. As we see from the figure, although PAFS delays the execution time (by 1%) when compared to *ondemand & performance* modes and increases the power consumption when compared to the *powersave* mode, it achieves a much better balance between the execution time and power consumption when compared to both.

GTC2 is a communication-bounded application. PAFS and *powersave* both keep CPU at the lowest frequency level and achieve 7% better energy efficiency than *ondemand & performance*.

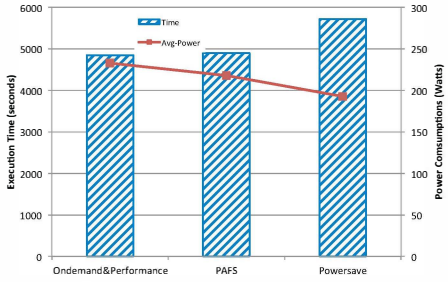


Fig. 8: GEOS5 with different frequency scaling modes

2) Multiple applications with a cluster-wide power budget: We then evaluate the HPM when multiple applications are running together in the cluster with a specific power budget and compare the result with that of using the baseline DVFS, which sequentially selects nodes to scale its CPU frequency so that the budget can be satisfied. In the experiments, GTC2 and MADbench run on the four nodes sequentially with an idle period between them, while GEOS5 runs on the another 4 nodes throughout the entire period. A global power budget of 1760 Watts is issued for the entire cluster.

TABLE I: Baseline DVFS results for multiple applications

	Time(s)	Avg_pow(W)	Energy(MJ)	EDP
GEOS(Node 4-7)	5378	842.28	4.530	24361
MAD(Node 0-3)	1618	743.56	1.203	1946.6
GTC2(Node 0-3)	1448	925.50	1.340	1940.5
Total	5378	1610.7	8.662	28248

TABLE II: HPM results for multiple applications

	Time(s)	Avg_pow(W)	Energy(MJ)	EDP
GEOS(Node 4-7)	4893	870.95	4.262	20852
MAD(Node 0-3)	1493	794.96	1.187	1772.0
GTC(Node 0-3)	1458	788.99	1.150	1677.2
Total	4893	1591.47	7.787	24301

Table I and II show the experiment results for GTC, GEOS5, and MADbench when running with the baseline DVFS and HPM, respectively. As we can see from the *Total* rows of both tables, PAFS outperforms the baseline DVFS scheme regarding to job execution time, average power consumption and total energy consumption, HPM achieves up to 13.6% improvement in terms of EDP.

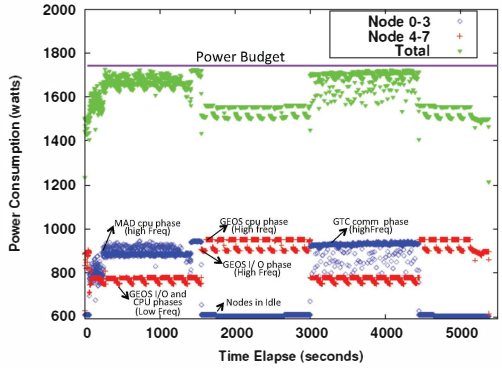
Fig. 9 (a) and (b) illustrate their detailed power footprints respectively. Green, red and blue points represent the power consumptions on node 0-3, node 4-7, and all nodes respectively. And a limitation line is provided for capping the total power budget. We can see that HPM implements a phase-aware, more flexible and efficient power management scheme leading to a better energy-efficiency than the baseline scheme.

C. Simulation results for large-scale system traces

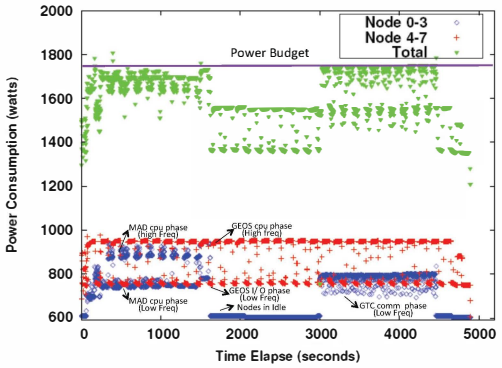
To investigate the effectiveness of HPM in the large-scale production cluster, we replay two production traces introduced in Section V-A using our simulator and compare the results with that of using the baseline DVFS scheme which is unaware of jobs and scales CPU frequency of each node as needed. Table III summarizes the comparison results between HPM and the baseline DVFS scheme, in terms of average job execution time, power consumption on each node, total power consumption, and EDP when running RICC and META workload traces. Overall, compared to the baseline scheme, HPM improves the EDP by 4.26% and 4.59% for META and RICC traces, respectively. More importantly, we notice that PAFS reduces not only the power consumption but also the job execution time over the baseline scheme.

TABLE III: Results of executing production traces in simulated large clusters

	Avg_Job_Time(s)	Avg_Node_Pow(W)	Avg_Job_E(MJ)	Avg_Job_EDP	EDP_Improve
META-baseline	46593	165.59	27.69	1291	–
META-HPM	45768	164.63	27.01	1236	4.26%
RICC-baseline	11669.6	163.72	24.764	288.98	–
RICC-HPM	11566	161.60	23.838	275.73	4.59%



(a) Baseline DVFS



(b) HPM

Fig. 9: Power Consumption Comparison

VI. Conclusions

In this work, we propose an application-transparent dynamic phase detection strategy through job status monitoring and combine this with a cluster-wide hierarchical phase-aware power management framework to achieve better energy-efficiency for active computing nodes in a power constrained system. Evaluation results on an in-house cluster and a cluster simulator show that our phase-aware power management scheme PAFS and HPM outperform baseline DVFS schemes by 4.3%-17% in terms of EDP for important scientific applications and real production cluster job logs.

Acknowledgements

This work is funded in part by a NASA grant NNX11AR20G and enabled by the U.S. National Science Foundation award CNS-1059376 to Auburn University. Sandia National Laboratories is a multi-program laboratory managed and operated under the contract DE-AC04-94AL85000.

References

[1] GEOS5. http://gmao.gsfc.nasa.gov/systems/geos5.

[2] Green500. http://www.green500.org.

[3] NVIDIA Tesla. http://www.nvidia.com/object/tesla-supercomputing\ newline-solutions.html.

[4] Top500. http://www.top500.org.

[5] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas. Integrated performance monitoring of a cosmology application on leading hec platforms. In *ICPP'05*. IEEE, 2005.

[6] D.G. Feitelson and D. Tsafrir. Workload sanitation for performance evaluation. In *International Symposium on Performance Analysis of Systems and Software*, pages 221–230. IEEE, 2006.

[7] V.W. Freeh, D.K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B.L. Rountree, and M.E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835–848, 2007.

[8] M.C. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: application to energy reduction. In *30th annual International Symposium on Computer Architecture*, pages 157–168. IEEE, 2003.

[9] N. Kappiah, V.W. Freeh, and D.K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 33. IEEE, 2005.

[10] J. H. Laros, III, K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan. Energy based performance tuning for large scale high performance computing systems. In *HPC'12*, San Diego, CA, USA, 2012.

[11] M. Laurenzano, M. Meswani, L. Carrington, A. Snavely, M. Tikir, and S. Poole. Reducing energy usage with memory and computation-aware dynamic frequency scaling. *Euro-Par 2011*, pages 79–90, 2011.

[12] D. Li, J. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu. An analysis of scientific applications for using non-volatile memory in high performance computing. In *IPDPS*. IEEE, 2012.

[13] M.Y. Lim, V.W. Freeh, and D.K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *SC 2006 Conference, Proceedings of the ACM/IEEE*. IEEE, 2006.

[14] Z. Liu, B. Wang, P. Carpenter, D. Li, J. Vetter, and W. Yu. PCM-based durable write cache for fast disk I/O. In *MASCOTS'13*, pages 451–458. IEEE, 2012.

[15] Z. Liu, F. Wu, X. Qin, C. Xie, J. Zhou, and J. Wang. TRACER: A trace replay tool to evaluate energy-efficiency of mass storage systems. In *CLUSTER'13*, pages 68–77. IEEE, 2010.

[16] Z. Liu, J. Zhou, W. Yu, F. Wu, X. Qin, and C. Xie. MIND: A black-box energy consumption model for disk arrays. In *International Green Computing Conference and Workshops (IGCC)*. IEEE, 2011.

[17] K. Ma and X. Wang. PGCapping: Exploiting power gating for power capping and core lifetime balancing in cmps. In *International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2012.

[18] L. Oliker, J. Carter, M. Wehner, A. Canning, S. Ethier, A. Mirin, D. Parks, P. Worley, S. Kitawaki, and Y. Tsuda. Leading computational methods on scalar and vector hec platforms. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 62. IEEE Computer Society, 2005.

[19] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 48–59. ACM, 2008.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ACM SIGARCH Computer Architecture News*, volume 30, pages 45–57. ACM, 2002.

[21] Y. Tawara, A. Idehara, and H. Yamamoto. DVFS and power-off controls on a multicore operating system. In *12th International Forum on Embedded MPSoC and Multicore*, Jifu, Japan, 2010.

[22] J. Vetter. On the path to exascale: Deploying an emerging hpc architecture. In *DOE Exascale Tools Workshop*, Annapolis, MD, 2011.

[23] L. Wang, G. von Laszewski, J. Dayal, and F. Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In *CCGRID*. IEEE, 2010.

[24] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *14th International Symposium on High Performance Computer Architecture*, pages 101–110. IEEE, 2008.