TracSim: Simulating and Scheduling Trapped Power Capacity to Maximize Machine Room Throughput

Ziming Zhang^a, Michael Lang^b, Scott Pakin^c, Song Fu^a

^a*Department of Computer Science and Engineering, University of North Texas, Denton, TX 76203. Email: zimingzhang@my.unt.edu, song.fu@unt.edu.*

^b*UltraScale Systems Research Center, Los Alamos National Laboratory, Los Alamos, NM 87544. Email: mlang@lanl.gov.*

^c*Applied Computer Science Group, Los Alamos National Laboratory, Los Alamos, NM 87544. Email: pakin@lanl.gov.*

Abstract

The power supplied to machine rooms tends to be over-provisioned because it is specified in practice not by workload demands but rather by high energy LINPACK runs or nameplate power estimates. This results in a considerable amount of *trapped power capacity*—excess power infrastructure. Instead of being wasted, this trapped power capacity should be reclaimed to accommodate more compute nodes in the machine room and thereby increase system throughput. But to do this we need the ability to enforce a system-wide power cap. In this paper, we present *TracSim*, a full-system simulator that enables users to measure trapped power capacity and evaluate the performance of different policies for scheduling parallel tasks under a power cap. *TracSim* simulates the execution environment of a production HPC cluster at Los Alamos National Laboratory (LANL). *TracSim* enables users to specify the system topology, hardware configuration, power cap, and task workload and to develop resource configuration and task scheduling policies aimed at maximizing machine-room throughput while keeping power consumption under a power cap by exploiting CPU throttling techniques. We use real measurements from the LANL cluster to set *TracSim*'s configuration parameters. We leverage *TracSim* to implement and evaluate four resource scheduling policies. Simulation results indicate the performance of those policies and quantify the amount of trapped capacity that can effectively be reclaimed.

1. Introduction

Each of today’s 10 fastest high-performance computing (HPC) systems consumes between 1.5 and 17.8 MW of power during heavy computing [1]. Because of the recurring and nonrecurring costs in delivering so much power, industry and government sectors demand efficient allocation and effective utilization of power and energy in machine rooms. A direct way to control the power consumption of machine rooms is to set an upper bound on power, called a *power cap*, and enforce that a machine room cannot use more power than the assigned power cap [2, 3]. Currently some types of individual compute nodes can be statically capped via BIOS settings at boot time or in newer systems at run time. However, the idea of setting power caps on a per job basis and managing them from a system-wide rather than a per-node perspective has not yet been investigated thoroughly.

Meanwhile, HPC systems in production environments do not make full utilization of the provided power. In particular, the power supplied to machine rooms tends to be over-provisioned because it is specified in practice not by workload demands but rather by high-energy LINPACK runs or nameplate power estimates [4], i.e., the worst-case power usage. This implies the existence of *trapped power capacity*, which is the amount of power supplied to a machine room but not used to yield useful cycles. Instead of being wasted, *the trapped power capacity should be reclaimed to accommodate more compute nodes in the machine room and thereby increase system throughput*. Our goal is not to reduce power consumption but rather to maximize the use of the existing power infrastructure, for example the power distribution units (PDUs).

Maximizing the utilization of the trapped power capacity to complete as many parallel jobs as possible per unit time without exceeding the power cap is challenging, given how dynamic the system is. Determining the most efficient way to utilize available power requires running various job and resource scheduling strategies under different workload and computer architectures. Tests on production HPC systems are costly and often infeasible. To make things worse, there is no practical and widely accepted simulator that can reflect the power consumption under different job and resource scheduling policies in real HPC environments.

In this paper, we present *TracSim*, a high-level macro-scale simulator that enables users to evaluate the performance of different policies for scheduling parallel tasks under a power cap. *TracSim* simulates the execution environment of an HPC system at Los Alamos National Laboratory (LANL). We use real measurements from the LANL cluster to set the config-

uration parameters of *TracSim*. *TracSim* enables users to specify the system topology, hardware configuration, power cap, and task workload and to develop job- and resource-scheduling policies aiming to maximize the machine-room throughput while keeping power consumption under the power cap by exploiting CPU throttling techniques. To demonstrate *TracSim*'s feasibility we leverage *TracSim* to implement and evaluate four task scheduling policies: three power-aware, and one performance-dominant. The simulation results present the performance of the policies in terms of the trapped capacity utilization and system throughput. To the best of our knowledge, *TracSim* is the first simulator of its kind to estimate the power usage of HPC systems by simulating a production execution environment to test job- and resource-scheduling policies under a power cap.

Though this work focuses on HPC use cases, i.e., tightly synchronized multi-node jobs, it is also useful in data centers where multi-node jobs are not tightly coupled but could still be managed dynamically by priority rather than with static node power caps. The following are the main contributions of this paper:

- the presentation of *TracSim*, a simulation framework informed by application power and performance measurements taken on a real, 150-node (2400-core) cluster and capable of predicting both workload performance and node and system power consumption,
- the introduction of a new, power-aware job-scheduling algorithm (PDRC) that executes workloads with high throughput while honoring a given, system-wide power cap, and
- an analysis of the performance and power consumption of multiple job-scheduling algorithms (including PDRC) and a quantification of the amount of trapped power capacity these leave in the system.

The rest of paper is organized as follows. The related research is presented and discussed in Section 2. Section 3 presents in detail the architecture and major components of *TracSim*. The design and implementation of *TracSim* is described in Section 4. We implement four power-aware job scheduling policies and use *TracSim* to analyze their performance in Section 5. Results from *TracSim* are validated in Section 6. Power and performance modeling is discussed in Section 7. Section 8 draws conclusions from our findings and notes some avenues for future work.

2. Related Work

Power consumption and energy efficiency have become imperative issues for researchers and engineers in HPC community [5, 6]. To control the power usage of HPC systems, power capping [2, 3] is a promising and effective approach. System operators can balance the performance and power consumption of clusters by adjusting the maximum amount of power (a.k.a. power budget) that can be consumed by the clusters. Pelly et al. present a dynamic power provisioning and capping method at the power distribution unit (PDU) level [7]. They propose to shift the slack power capacity to servers with growing power demand by using a heuristics policy. For HPC jobs, many factors affect the power usage including hardware configurations and resource utilization. Femal et al. developed a hierarchical management policy to distribute power budget among clusters [8]. Kim et al. investigate the relation between CPU voltages and system performance and power efficiency [9]. By exploiting the dynamic voltage scaling (DVS) technologies, they propose a task scheduling policy that aims to minimize the energy consumption while satisfying the specified performance requirements.

Power-aware task scheduling has been extensively researched and many scheduling policies have been proposed [10, 11, 12, 13]. The performance of these scheduling policies is usually evaluated either on small scale testbeds or by simulations. Testing the policies on large production systems can be time-consuming and costly. Simulating HPC clusters also provides a flexible way to evaluate power-aware task scheduling policies. In order to simulate the execution of parallel jobs, a simulator needs to model both the performance and power consumption of a system. Several simulators are available. For example, Grobelny et al. presented the FASE framework to predict the performance and power consumption of HPC applications based on an application characterization scheme [14]. FASE enables users to test different architectural designs. Economou et al. propose a server power usage estimation framework that explores hardware utilization to achieve accurate estimation of the server’s power usage [15]. Jackson et al. have developed an HPC cluster simulator named Maui, which allows users to test scheduling policies with regard to jobs’ performance and resource utilization [16]. However, Maui cannot provide information about the cluster’s power consumption.

Power profiling in production computer systems provides valuable data and knowledge for developing power simulators. Fine-grained power profiling techniques measure the power usage of individual hardware components, such as CPU [17], memory [18], disk [19] and other devices [20]. In contrast,

coarse-grained power profiling aims to characterize system-wide power dynamics, such as the macropower framework [21]. Kamil et al. profiled HPC applications on several test platforms and projected the power profiling results from a single node to a full system [22]. Ge et al. studied the influence of software and hardware configurations on the system-wide power consumption [23]. They found that characteristics of HPC applications affect the power usage of a system.

Characterizing power usage is an important research topic. Existing approaches can be broadly classified into three categories: power measurement from real hardware [24, 25, 26], power estimation by simulations [27, 28, 29], and power characterization by quasi-analytical performance modeling [30]. Direct power measurements are accurate but costly and applicable only to existing systems. Power estimation by simulations works for both existing and future computer systems and can provide detailed power usage breakdown. Quasi-analytical performance modeling can be accurate but requires a deep understanding of each individual application running on the system.

The novelty of this work is that we analyze multiple power-cap-aware job scheduling strategies using a simulator informed by actual sub-rack-level power and performance measurement of HPC applications running on a cluster. Using this framework we are able to design and analyze scheduling policies to achieve high CPU utilization while keeping the power usage under a given power cap.

3. *TracSim*: A Full-System Power and Performance Simulator

We aim at developing a simulator that enables users to evaluate the performance of different policies for scheduling parallel jobs under a power cap. It should simulate the execution environment of a production HPC system and be sufficiently flexible and scalable to enable users to define different system topology, hardware configuration, power cap, and task workload. It also needs to provide a well-defined API for users to implement different resource allocation and reconfiguration policies.

Figure 1 depicts the architecture of the *TracSim* simulator. *TracSim* consists of four major components: *workload*, *job*, *node*, and *cluster*. A *workload* represents one type of application with a given problem size and a number of tasks (its *concurrency*). A *job* is an instance of a workload type with user-specified performance requirements and a priority level. As is common in HPC systems, a parallel job contains a number of tasks that are scheduled to run on multiple nodes, where a task is usually a rank in an MPI [31] job. A *node* includes the configuration information of the major

Figure 1: The architecture and major components of *TracSim*.

hardware devices including CPU, memory, disk, and network, the identity of a running job task, and a power monitor that calculates the power usage of executing the task on this node. There are a number of nodes in a *cluster*. In addition, a *job scheduler* in a cluster decides which job will run on which node. Users can define a power cap as the upper bound of power usage of the cluster and implement policies that adjust the resource configurations of nodes to use the trapped capacity in the cluster.

We now present the design details of the four major components in *TracSim*: *workloads*, *jobs*, *nodes*, and *clusters*.

3.1. Workloads and Jobs

TracSim defines a *workload* class for each category of application it simulates. HPC applications display varying load which is manifested as the dynamic utilization of system resources, including CPU, memory, disk and network when run on a real-world system. The corresponding power usage also changes at run time. Different types of applications have different run-time dynamics. To model these dynamics we could directly use the time series of the continuous measurements of the resource utilization and power usage from a production system to represent an application’s workload. However, the time and space complexity of this approach makes it intractable. An efficient method is to leverage statistical analysis of the time series, such as data distributions or statistics, to model the workload dynamics [32, 33]. Data distributions are relatively more accurate. However, finding good fits for various time series is challenging. In *TracSim*, we use statistics of the measured performance and power usage data while running different applications on a production cluster to model the applications’ workloads.

Table 1: Major attributes of components in *TracSim*.

<i>Attribute</i>	Description
<i>Workload.Type</i>	Type of a <i>workload</i> .
<i>Workload.ProblemSize</i>	Problem size of a <i>workload</i> .
<i>Workload.Concurrency</i>	Number of tasks in a job of a <i>workload</i> .
<i>Job.Workload</i>	Workload type of a <i>job</i> .
<i>Job.Priority</i>	Priority value of a <i>job</i> .
<i>Job.Nodes</i>	A set of nodes scheduled to run tasks of a <i>job</i> .
<i>Job.Perf</i>	Performance statistics of a <i>job</i> .
<i>Job.Power</i>	Power usage of a <i>job</i> .
<i>Job.State</i>	Execution state of a <i>job</i> .
<i>Job.SubTime</i>	Timestamp of <i>job</i> 's submission.
<i>Job.StartTime</i>	Timestamp of starting <i>job</i> 's execution.
<i>Job.FinishTime</i>	Timestamp of <i>job</i> 's completion.
<i>Node.Config</i>	Hardware configuration of a <i>node</i> .
<i>Node.Utilization</i>	Resource utilization of a <i>node</i> .
<i>Node.Task</i>	Running task on a <i>node</i> .
<i>Node.Power</i>	Power consumption of a <i>node</i> .
<i>Cluster.DirtyNodes</i>	A set of nodes assigned to run jobs in a <i>cluster</i> .
<i>Cluster.IdleNodes</i>	A set of idle nodes in a <i>cluster</i> .
<i>Cluster.JobList</i>	A set of jobs running on a <i>cluster</i> .
<i>Cluster.PowerCap</i>	Power cap of a <i>cluster</i> .

Three important attributes of a workload are the *type*, *concurrency* and *problem size* as listed in Table 1. They determine the type of an application (It is associated with the power usage and performance models as described in Section 4.2), the number of tasks (in turn the number of nodes) and the amount of computation performed by each task.

A *job* is an instance of a *workload* with user specified performance requirements and system recorded execution information. A user provides the priority of a job during the job submission. The *job scheduler* then selects a set of qualified and available nodes to run the job's tasks and schedules the tasks' execution based on the job's priority. The number of selected nodes is the concurrency of the job's workload. The scheduler also updates the job's execution state, execution time, and completion time. Based on a power usage model (described in Section 4) that uses measurements of power consumption from a production HPC cluster at LANL, *TracSim* calculates the power usage of running the job as follows:

$$job.Power = \sum_{node \in job.Nodes} node.Power. \quad (1)$$

Table 2: Major programming functions provided by *TracSim*.

<i>Method</i>	Description
<i>Workload.setConcurrency</i>	Set the concurrency of a <i>workload</i> .
<i>Workload.setProblemSize</i>	Set the problem size of a <i>workload</i> .
<i>Job.setWorkload</i>	Set the workload type of a <i>job</i> .
<i>Job.getWorkload</i>	Get the workload type of a <i>job</i> .
<i>Job.allocNode</i>	Assign a set of nodes to run a <i>job</i> .
<i>Job.suspend</i>	Suspend the execution of a <i>job</i> .
<i>Job.resume</i>	Resume the execution of a <i>job</i> .
<i>Job.setPriority</i>	Set the priority of a <i>job</i> .
<i>Job.setSubmissionTime</i>	Set the submission time of a <i>job</i> .
<i>Node.setFreq</i>	Set the CPU frequency of a <i>node</i> .
<i>Node.getPower</i>	Get the power consumption value of a <i>node</i> .
<i>Node.assignTask</i>	Assign a task of a job to run on a <i>node</i> .
<i>Cluster.setPowerCap</i>	Set the power cap of a <i>cluster</i> .
<i>Cluster.setTracManager</i>	Set the trapped capacity management policy of a <i>cluster</i> .
<i>Cluster.setJobScheduler</i>	Set the job scheduling policy of a <i>cluster</i> .

In many production HPC systems, a node is the minimum scheduling unit. That is, nodes are not shared across jobs. Hence, by aggregating the power consumption of those nodes executing tasks of a job, *TracSim* can simulate the job’s total power usage. In addition, the statistics of the job’s execution performance are also recorded.

3.2. Nodes and Clusters

Node is a representation of a computer or a server in an HPC system. From the hardware’s perspective, a node comprises multiple hardware devices. *TracSim* simulates the hardware composition of a node by providing a set of configuration parameters to allow users to define the node’s hardware specification. We include those configuration parameters that affect the power consumption of the node, including the number of processor cores, CPU frequency, and various fixed costs.

From the software’s perspective, a node provides computing/storage services and runs application tasks. One or multiple tasks belonging to the same job can be run on the node at a time. The *Node.Task* attribute keeps tracking the currently running task on the node. The *power monitor* exploits the measurements of resource utilization from the LANL cluster to simulate the dynamics of resource utilization which determines the node’s power consumption according to the power usage model as described in Section 4.

A *cluster* consists of a number of *nodes* which are connected by a network. For each cluster, *TracSim* maintains a list of nodes belonging to that cluster. When the *job scheduler* dispatches a job’s tasks to run on the nodes of a cluster, the cluster’s job list is updated. The *PowerCap* of the cluster determines the maximum amount of power that the cluster can use at any time. The *job scheduler* and *trapped capacity manager* are two important components that control the overall power consumption and the utilization of the trapped capacity in a cluster. The former decides which job tasks will run on which cluster nodes at what time and the latter changes the hardware configurations, such as the number of active processor cores and the CPU frequency, according to user-defined policies. Although the cooling system is an important power consumer, its detailed simulation is beyond the scope of this paper. In *TracSim*, a fixed amount representing the power usage of a cooling system based on the measurements from real systems can be added to the cluster’s power consumption.

3.3. Job Scheduler

The *job scheduler* maps tasks of the submitted jobs to the available nodes in a cluster or across clusters. Consequentially, it is also responsible for updating the execution states of jobs and the resource utilization of nodes and for calculating the power usage of jobs and nodes.

In production computer systems, job scheduling tools such as SLURM, PBS, and MOAB [34, 35] decide the execution sequence of jobs and node assignments based on certain properties of jobs including their priorities and/or deadlines. Following this common practice, *TracSim* provides a priority-based job scheduling policy by default. Algorithm 1 outlines this policy. It is implemented by using the attributes and functions provided by *TracSim*, as listed in Tables 1 and 2 respectively.

The algorithm is a basic backfill with no dynamic priority and no specific job placement. At every scheduling time point, the *job scheduler* selects the job with the highest priority to schedule. The scheduler searches for a set of idle nodes whose number matches the *concurrency* value of the job’s workload. If successful, the selected nodes are marked as “dirty nodes” which indicates that those nodes are not available for scheduling until the currently running job terminates. In this way, the *job scheduler* keeps allocating the available nodes to those jobs that have been submitted but not yet started until there are not enough nodes available for scheduling a new job.

When a job finishes execution or is terminated, the nodes allocated to the job are released. The *job scheduler* is then invoked in an attempt to allocate the available nodes to the next job with the highest priority. Even

Algorithm 1 Default Priority-based job scheduling policy in *TracSim*.

```

for  $priority = 1$  to  $MAXPRIORITY$  do
  for all  $job \in cluster.JobList$  and  $job$  is not started do
    if  $job.priority = priority$  and  $job.Workload.Concurrency \leq$ 
       $||cluster.IdleNodes||$  then
       $list = \emptyset$ ;
      Job scheduler assigns nodes requested by  $job$  to  $list$ ;
       $cluster.IdleNodes \leftarrow cluster.IdleNodes - list$ ;
       $cluster.DirtyNodes \leftarrow cluster.DirtyNodes \cup list$ ;
      for all  $node \in list$  do
         $node.assignTask(job)$ ;
      end for
    end if
  end for
end for

```

when there are sufficient available nodes for a lower-priority job but not enough for a higher-priority job, the *job scheduler* reserves those available nodes for the higher-priority job until enough nodes are released in order to avoid starving the higher-priority job. This is only one of many possible methods of scheduling priority jobs. Other job scheduling policies can be implemented and added to *TracSim*.

In addition to the default priority-based job scheduling policy, other scheduling policies can also be implemented and simulated in *TracSim*. As an example, we include the Knapsack job scheduling policy [36] in *TracSim*. This policy draws an analogy between job scheduling and the 0-1 knapsack problem. The requested number of nodes and the total power consumption are two key attributes that represent the underlying characteristics of an HPC job. Inside a queue full of pending jobs there are numbers of different combinations of jobs that can run simultaneously, which yield different scenarios of systemwide power consumption and resource utilization. The method associates one job with a gain value and a weight in terms of the basic definition of a 0-1 knapsack problem. Essentially, the gain value denotes the number of nodes a job requests, and the weight represents the estimated power consumption of the job. For a given system-wide power cap and a set of pending jobs, the scheduler attempts to select a set of jobs maximizing the system utilization. The 0-1 knapsack problem can be solved in pseudo-polynomial time, which ensures the relatively low overhead of enforcing the knapsack policy on a production HPC infrastructure. All the pending Jobs

out of the scheduling window will not be considered and scheduled by the policy. After all the jobs inside the scheduling window are either started or completed, the scheduler will select a new set of jobs and locate them inside the scheduling window.

Algorithm 2 Knapsack job scheduling policy in *TracSim*.

```

window  $\leftarrow \emptyset$ ;
while not all jobs are started do
  Select jobs with certain priority into window
  while window is not empty do
    Perform Knapsack on all jobs  $\in$  window and attain a set of jobs
    Adjust CPU frequencies if needed
    Run selected jobs set
    Remove selected jobs from window
  end while
end while

```

Algorithm 2 sketches the Knapsack job scheduling policy. The original policy [36] does not consider CPU throttling in job scheduling, we enhance the policy by practicing CPU frequency adjustment and evaluate the potential of this policy in terms of trapped capacity. There are two specific cases that need additional discussion. First, it is possible that an optimal set of selected jobs requires more than the total number of available nodes. In this case, the scheduler raises the CPU frequency for power estimation and re-runs the knapsack process until the optimal set of selected jobs can be allocated to the available resources. Second, when the power cap is altered, the scheduler concurrently adjusts the CPU frequencies of all engaged nodes to adapt to the new power cap. In general, the knapsack policy is a utilization-dominant method.

TracSim facilitates the development and evaluation of job scheduling policies. The main contribution of this paper is not to propose new job scheduling strategies. We use the preceding two examples to illustrate how job scheduling policies can be implemented by *TracSim*. We develop *TracSim* with the intention of exploiting trapped capacity and maximizing machine-room throughput.

Table 3: Measurements of application performance and power consumption from running xRAGE on a LANL HPC cluster.

Frequency		Time		Power	
GHz	norm.	s	norm.	W	norm.
1.2	0.40	1,525.61	2.38	24,442.19	0.49
1.5	0.50	1,239.06	1.93	27,170.10	0.54
1.8	0.60	1,007.50	1.57	30,123.53	0.60
2.1	0.70	867.55	1.35	34,981.00	0.70
2.4	0.80	772.18	1.20	37,486.77	0.75
2.7	0.90	706.77	1.10	44,434.47	0.89
3.0	1.00	640.97	1.00	49,890.20	1.00

4. Implementation Issues of *TracSim* and Trapped Capacity Management Policies

In this section, we focus on the implementation issues, including the setting of parameters and the power usage model used in *TracSim*. We also present two trapped-capacity management policies to demonstrate how to use *TracSim* to manage resource and power.

4.1. Performance Profiling and Parameter Setting

TracSim simulates the executing environment of a production HPC system at Los Alamos National Laboratory (LANL). We use real measurements from the LANL cluster to set the configuration parameters of *TracSim*. The cluster contains 150 nodes. Each node is equipped with two 8-core Intel E5-2670 Sandy Bridge processor. The total number of CPU cores in the cluster is 2400. The frequency of each core can be set from 1.2 GHz to 3.0 GHz. (3.0 GHz is in fact the effective speed achieved when Turbo Boost is active and all cores are busy.) The nodes are connected by a QLogic InfiniBand QDR network.

We selected two micro-benchmarks and four parallel scientific benchmarks as applications to run on the cluster. The micro-benchmarks that we tested are STREAM [37] which performs four vector operations on long vectors, and DGEMM (double-precision general matrix-matrix multiplication) from the Intel’s Math Kernel Library [38], optimized for the processors in the cluster, run on a set of matrices small enough to fit in the level 3 cache. The scientific applications that we tested are xRAGE [39], which simulates radiation hydrodynamics, POP [40], which simulates global ocean

circulation, VPIC [41], which simulates magnetic reconnection and laser-plasma interactions, and PARTISN [42], which solves the neutral-particle Boltzmann transport equation. Each benchmark application was run on all 2400 cores in the cluster. We profiled the execution time and system power consumption under different hardware configurations.

These programs were selected to represent a gamut of scientific applications from completely compute-bound (DGEMM) to memory-bound (STREAM), with the more realistic scientific applications falling in between the two extremes. Table 3 lists the execution time and cluster-wide power consumption profiled from running xRAGE with different CPU frequencies on the cluster. From the table, it is clear that higher CPU frequency leads to better application performance but also higher power consumption. The profiling results of the other five benchmarks display a similar trend. The details are covered in a related work [4] and are not repeated here.

Based on these measurements, we set the values of some configuration parameters in *TracSim*. For instance, six types of *workload* correspond to the six types of applications tested on the production cluster. The CPU frequency in *Node.Hardware* has seven possible values 1-7 corresponding to the seven frequency levels from 1.2 GHz to 3.3 GHz. The users can specify the cluster scale (i.e., the number of nodes), the problem size and number of tasks of *workload*, *job*'s priority ("low", "medium", and "high") and submission time, the number of processor cores of *node*, and the power cap of *cluster*. *TracSim* calculates a *job*'s execution time and power consumption by using the models that are presented in the next section.

4.2. Power Usage and Performance Models

Figure 2 plots the normalized power consumption and execution time when the benchmark application xRAGE was run on the LANL cluster. With the increase of CPU frequency, the power consumption increases while the execution time decreases.

Assume a *job* of a workload type w is scheduled to run on n nodes in a cluster. There are N nodes in total in the cluster and the highest level of CPU frequency is F while the CPU frequency level for running the *job* is f . Then the power usage of the *job* can be calculated as

$$Power(w, n, f) = Power(w, N, F) \times \frac{n}{N} \times norm_{power}(w, f), \quad (2)$$

where $norm_{power}$ is a scaling term of the power consumption which either comes from empirical data such as the last column of Table 3 or is approximated by a relation such as $norm_{power}(w, f) = f/F$. (The latter has an

Figure 2: Execution time and power consumption of running xRAGE on the LANL production cluster.

average error of 5.59% relative to Figure 2.) Our scaling method is feasible because in the production HPC systems the CPU utilization of active nodes are always very high and hence the power level of a single node with the highest CPU frequency can be scaled to any CPU frequency level [10].

For HPC applications, we assume that higher concurrency leads to shorter execution time. Following the preceding notation, the execution time of the job is calculated as

$$Exectime(w, n, f) = Exectime(w, N, F) \times \frac{N}{n} \times norm_{exectime}(w, f). \quad (3)$$

Similarly, the normalization term, $norm_{exectime}$, can be determined from the profiled execution time, such as the 4th column in Table 3. According to Figure 2, it can also be calculated as $norm_{exectime}(w, f) = F/f$ whose average error rate is 5.37%.

The cluster-wide power consumption and application execution time have been measured from running DGEMM, POP and PARTISN on the LANL cluster with different CPU frequencies. Due to the space limit, the measurements are not listed in this paper. The dynamics of the normalized power consumption and execution time of these applications are shown in Figure 3. By using $norm_{power}(w, f) = f/F$ to model the normalized power consumption, the average error is quite low, as 4.08%, 5.76% and 4.52% for DGEMM, POP and PARTISN respectively. The accuracy of using $norm_{exectime}(w, f) = F/f$ to model the application's execution time is

Figure 3: Measured power consumption and execution time of DGEMM, POP and PARTISN on the LANL cluster.

also good. The modeling error for the three applications are 0.43%, 5.58% and 6.17% respectively. We discuss a more accurate method for modeling the execution time and power consumption in Section 7. However that method is more complex and needs application-dependent parameters. For efficiency and accuracy, we currently implement the power and performance models presented in Equations (2) and (3) in *TracSim*. Users of *TracSim* can specify the power and performance models for their systems and *TracSim* will use them in simulations.

We design the power usage and performance models to be generic and practical to ensure the simulator is applicable to different systems and applications. We have tested six benchmark applications on a production cluster at LANL and the proposed models are accurate in characterizing the execution time and power consumption. The simulator is calibrated directly to the power measured for these applications, inaccuracies may be enlarged in predicting at greater scale. Prior work [5] has shown that LANL workloads run at 80% of a LINPACK workload in terms of power usage. Controlling DVFS settings on nodes is not expensive. Detailed application-specific behavior is valuable for building accurate workload models. However, this may compromise the efficiency and applicability of the simulator. Since most applications running on LANL clusters are CPU-intensive/memory intensive, CPU throttling is the most effective way to control the power consumption. The models are a balance between complexity and speed to estimate the power for scheduling jobs. Additionally, we focus on the trapped capacity that exists in current machine rooms and by power capping we can utilize that power for additional capacity. In the current design, the change of the communication and computation patterns as a system scales up or scales down is not included in the power and performance models. We will measure and model the change and its influence on the power usage and

execution time of different types of workloads and enhance our models by incorporating them.

Algorithm 3 Power-level increment (PI) in the PDRC policy.

```

list  $\leftarrow \emptyset$ ;
for all job  $\in$  the cluster.JobList such that job is started do
    list  $\leftarrow$  list  $\cup$  {job};
end for
for all priority = 1 to k do
    while all job not running at the highest CPU freq do
        for all job that job  $\in$  list do
            if job.Priority = priority and Power(cluster) <
                cluster.PowerCap then
                for all node that node  $\in$  job.Nodes do
                    if Power(cluster) > cluster.PowerCap then
                        break;
                    end if
                    node.setFreq(node.Freq + 1);
                end for
            end if
        end for
        if Power(cluster) > cluster.PowerCap then
            for all job that job  $\in$  list do
                if job.Priority = priority then
                    for all node  $\in$  job.Nodes do
                        node.setFreq(node.Freq - 1);
                    end for
                end if
            end for
        end if
    end while
end for

```

4.3. Trapped Capacity Manager

TracSim allows users to specify the system topology, hardware configuration, power cap and task workload. It also provides a set of functions, as listed in Table 2, that facilitate users to develop and evaluate resource management policies. The *trapped capacity manager* (TCM) enforces the policies aiming to maximize the machine room throughput while keeping

power consumption under a power cap by exploiting CPU throttling techniques.

As an illustration, we have designed and implemented a *power-aware dynamic resource configuration (PDRC)* policy. The policy consists of two mechanisms. One is power-level increment (PI) which aims to utilize the trapped capacity of a cluster by increasing the nodes' CPU frequencies (and therefore performance), which drives up their power usage. The other is power-level decrement (PD) which intends to avoid power cap violations by reducing the nodes' power usage. The pseudocode of adjusting the resource configurations for PI is presented in Algorithm 3. The PI mechanism keeps the power usage level close to but less than the power cap in order to maximize the system performance. It allocates the trapped capacity to reduce the execution time of higher-priority jobs by increasing the CPU frequency of the nodes running those jobs until the power cap is reached. Scheduling is determined strictly by priority; we currently do not take into account the application's CPU utilization. The function *Power(cluster)* in Algorithm 3 applies the power usage model described in Section 4.2 to calculate the power consumption of a cluster.

Algorithm 4 Power-level decrement (PD) in the PDRC policy.

```

list ← ∅;
for all job ∈ cluster.JobList such that job is started do
    list ← list ∪ {job};
end for
for priority = k to 1 do
    while all job not running at the lowest CPU freq and Power(cluster) >
cluster.PowerCap do
        for all job that job ∈ list do
            if job.Priority = priority then
                for all node that node ∈ job.Nodes do
                    node.setFreq(node.Freq − 1);
                end for
            end if
        end for
    end while
end for

```

When the power cap is to be violated, the PD mechanism sacrifices jobs' performance for reduced power consumption. Algorithm 4 shows the pseudocode of PD. To reduce the power usage, PD first selects nodes that run

lower-priority jobs to decrease their CPU frequency. If changing the CPU frequency of those nodes to the lowest level still cannot solve the power cap violation, PD tries to degrade the performance of jobs with higher priorities until the cluster’s power usage is no greater than the power cap.

The PDRC policy is invoked and enforced by TCM in the following three cases. 1) When the system is initiated, the *job scheduler* starts to schedule jobs to run. Since the nodes in the cluster are in an idle state, the power usage of the cluster is at the lowest level. As the scheduled jobs begin execution using the default resource configurations, an abrupt jump in the cluster’s power usage occurs. If there still exists certain trapped capacity after the job scheduling, TCM performs PI to exploit this extra trapped capacity for better job performance. Otherwise, PD is invoked to handle a possible power cap violation. 2) When a job is completed or terminated, some nodes are released. When the *job scheduler* allocates the newly available nodes, TCM executes PI and/or PD to utilize the trapped capacity and/or to enforce the power cap. In addition, as soon as some nodes are freed, TCM runs PI to exploit the additional trapped capacity. 3) When the power cap of a cluster changes (by applying a dynamic power capping strategy), PI or PD is invoked by TCM to remedy an increase or decrease of the power cap respectively.

5. Performance Evaluation

We have implemented *TracSim* in Java. The prototype has about ten thousand SLOCs. We have employed it to simulate an HPC cluster and conducted experiments to evaluate its performance. In this section, we present the experimental results.

5.1. Experimental Setup

The cluster that we simulate contains 150 nodes each of which has 16 processor cores. Each core provides seven frequency levels from 1.2 GHz to 3.0 GHz. To simulate jobs of different sizes in real systems, three types of jobs are tested: 1) a large-size job list in which every submitted job in the list needs 50–80 nodes; 2) a small-size job list where each job needs fewer than 50 nodes; and 3) a mixed-size job list where one half of the jobs are small-size and the other half are large-size. Each job is assigned a priority from one of the three values: 1 (high), 2 (medium) and 3 (low). There are six types of workloads in the system. At the start of a simulation, 30 jobs with randomly selected workload types are submitted to the cluster. The

Figure 4: Average CPU utilization with three types of jobs.

base time (with maximum frequency and concurrency) for each workload is determined by empirical data.

5.2. Experimental Results

5.2.1. CPU utilization

The resource utilization, especially the CPU utilization, is an important performance metric of the job scheduler. Figure 4 depicts the average CPU utilization with the three types of jobs by applying the priority-based job scheduling policy (as described in Section 3.3). The three figures show the job scheduler can effectively utilize the processing capacity as the average CPU utilization is no less than 75.0% during 72.3%, 72.4% and 87.4% of time when running large-size jobs, mixed-size jobs and small-size jobs, respectively. The overall average CPU utilization reaches 77.5% (large-size jobs), 78.3% (mixed-size jobs) and 83.8% (small-size jobs).

Figure 5: Cluster-wide power consumption in running mixed-size jobs.

5.2.2. Trapped power capacity utilization

In the next set of experiments, we focus on the utilization of the trapped capacity by testing four policies. In addition to the *power-aware dynamic resource configuration (PDRC)* policy (described in Section 4.3) and *Knapsack* policy, we have implemented two static resource configuration policies in which the CPU frequency of a node can only be at one of the two levels. When the node is scheduled to run a job task, TCM sets the CPU frequency of the node’s processor cores to the highest level (3.0 GHz in the experiments), while the CPU frequency is reduced to the lowest level (1.2 GHz in the experiments) when the node becomes idle. The static resource configuration policy reflects a common practice in production HPC systems. When there is a possible power cap violation, TCM adjusts the resource configuration by giving bias towards either power saving or system performance. In the former, which is called a *static power-precedence* policy, the job scheduler selects some running jobs to suspend so that TCM can change the CPU frequency of the corresponding nodes to the lowest level until the power cap is complied with. In the latter, called a *static performance-precedence* policy, no running job is suspended which aims to

Figure 6: Utilization of the trapped capacity by employing four resource configuration policies to run 30 jobs in about 3 hours on a 150-node cluster.

Figure 7: CPU utilization by running three types of jobs with resource configuration policies.

maintain the system performance.

To evaluate the adaptability of these policies to changes of the power cap, the power cap is increased or decreased every 3000 seconds to mimic the cases where the power budget is cut/regained or the electricity rate changes. Figure 5 presents the cluster-wide power consumption by applying the resource configuration policies. From the figures, we can see the PDRC policy achieves the best utilization of the trapped capacity because it dynamically adjusts the CPU frequency according to changes of jobs and the power cap. Only 5.1% trapped capacity is unused by this policy and there is no power cap violation. Knapsack policy shows that 19.2% of the trapped capacity is unused. Among the two tested static resource configuration policies, the power-precedence policy outperforms the performance-

Figure 8: Average turnaround time of jobs with three types of workload and three priority levels.

precedence policy in terms of the trapped capacity utilization with 16.4% and -28.1% (negative means power cap violation) unused trapped capacity respectively. Moreover, the former experiences no power cap violation, while the latter causes the cluster’s power usage to exceed the power cap 74.3% of the time. In total, the amount of unused trapped capacity of the cluster is 7.21 kWh (PDRC), 26.12 kWh (Knapsack), 21.94 kWh (static power-precedence) and -29.85 kWh (static performance-precedence) for running only 30 jobs in around 3 hours as shown in Figure 6. If we scale the execution to the period of one year, PDRC can reduce energy consumption by over 43,000 kWh compared with the static power-precedence policy on one cluster. Given that xRAGE draws a maximum of 49,890W on 150 nodes (Table 3), this improvement from PDRC enables an increase in cluster size of 16 nodes (11%) without requiring any additional power infrastructure. When the power cap is enforced, i.e., xRAGE draws 34,981W on 150 nodes (Table 3), PDRC enables 23 more nodes (16%) to be added to the cluster. Thus, TracSim allows system operators and architects to deploy more

Figure 9: Spatial and temporal distribution of mixed-size jobs with different priorities on nodes.

nodes in order to achieve a full utilization of the trapped power capacity in a machine room.

The impact of resource configuration policy on the resource utilization is also studied. Figure 7 plots the average CPU utilization by employing the policies. The figure shows PDRC is slightly better than the static performance-precedence policy as their achieved CPU utilization can reach 80.3% and 78.9% respectively. The static power-precedence policy yields the lowest CPU utilization as 65.7% because of its aggressive job suspensions in face of power cap violations. In addition, the resource utilization differs when the cluster runs different types of jobs. Small-size jobs lead to the highest CPU utilization followed by the mix-size jobs and then the large-size jobs as their CPU utilization can reach 87.8%, 80.3% and 77.6% respectively. This is reasonable because as the job size decreases it is more likely to find a small job task to fill an idle slot of a node.

5.2.3. Job turnaround time

The turnaround time of a job, which refers to the total time taken between the submission and the completion of the job, is an important metric to evaluate the performance of jobs. Figure 8 depicts the average turnaround time of three types of jobs with three levels of priority. From the figures we can see the priority of a job imposes the most significant impact on its turnaround time. For example, by applying the PDRC policy to run-

Figure 10: CPU frequency of nodes running mixed-size jobs with the PDRC policy.

ning small-size jobs, the turnaround time of jobs with the highest priority is only 41.9% and 25.6% of that of jobs with the medium and lowest priorities respectively. Among the tested resource configuration policies, the static performance-precedence policy achieves the best job turnaround time as it sacrifices power usage for performance. Between the other two policies which comply with the power cap requirement, PDRC outperforms the static power-precedence policy in terms of the job performance. The job turnaround time with PDRC is at best 69.1% and at worst 91.8% of that achieved by the latter policy. Moreover, the type of jobs also affects the turnaround time. Large-size jobs are completed earlier than mixed-size and small-size jobs. In the cases where PDRC is applied, the turnaround time of large-size jobs is 78.7% (priority level 1), 88.3% (priority level 2) and 86.1% (priority level 3) of that of small-size jobs. This is counter-intuitive. Apparently, the performance gained by strong scaling the workloads outweighs the increase in waiting time needed to schedule the larger jobs.

To gain a deeper understanding of the execution of jobs on the cluster, we plot the distribution of jobs with three priority levels on the nodes as shown in Figure 9. The TCM employs the PDRC policy while running mixed-size jobs. A vertical view of the figure presents the spatial distribution of jobs on the nodes and a horizontal view depicts the jobs' execution in time. Three grayscale colors represent the three priority levels of jobs, while a blank area means the corresponding nodes are idle. The figure shows the job scheduler schedules jobs according to their priorities and it reserves nodes that are not

Figure 11: Validation results from a LANL HPC machine. Three applications have been tested with varying scale and CPU frequency. The accuracy of the simulated power consumption is above 90% in most cases.

enough to run a job for higher-priority jobs in order to avoid blocking high-priority jobs by low-priority ones. Figure 10 displays the CPU frequency of nodes set by the TCM. It is clear that the PDRC policy dynamically adjusts the CPU frequency according to the workload of jobs and the changes of the power cap.

6. Validation Experiments on a Production Machine

To validate the accuracy of the simulated results, we have conducted a series of experiments on a production HPC machine at LANL. Each node in the machine has two Intel E5-2670 Sandy Bridge processors. The CPU frequency ranges from 1.2 GHz to 2.6 GHz with an increment of 0.1 GHz plus “Turbo Mode” [43] at 3.0 GHz. We tested three programs—PARTISN, STREAM, and DGEMM—at different node counts, from 10 to 80 nodes. We report the power for the cages (groups of 10 nodes) used.

Figure 11 shows the real power consumption measured from a number of cages in the machine and the results from *TracSim*. In the figure, we can see the simulated power consumptions are very close to the real measurements. For PARTISN and DGEMM, the accuracy of the simulated results is 90.6—96.8% and 89–91.1%, respectively. However, the results of STREAM show a relatively large variance. When STREAM is run on 10 nodes with 1.5 GHz CPU frequency, the difference between the real and simulated power consumption is within 4.0%. As the node count increases to 30 and 40, the difference jumps to 23.2% and 28.5%. Increasing the CPU frequency also affects the accuracy of simulation, as the error rate reaches 33.7% and 26.4% when the CPU frequency is set to 2.7 GHz and 3.0 GHz, respectively. A

reason for the relatively low accuracy for STREAM is that we have observed a lot of variability in the power readings from multiple cages when running STREAM in the last four settings on the production machine. This variability in the overall power measurements affects the accuracy with *TracSim*. Another noteworthy observation is that the real power consumption remains unchanged when the CPU frequency goes from 2.7 GHz to 3.0 GHz due to the Turbo Mode, while the simulated results, which do not take Turbo Mode into consideration, increase proportionally. The discrepancy demonstrates the nonlinearity in power consumption induced by Turbo Mode. Overall, the results from *TracSim* are accurate—above 90% in most cases.

7. Discussion

In Section 4.2, we use linear scaling to model the power consumption and execution time of applications on an HPC machine. We show that it achieves a reasonable trade-off between modeling complexity and accuracy. A more accurate method is to explore compute-boundedness [6] to quantify the applications’ power consumption and performance.

Table 3 and Figures 2 and 3 show that applications’ power consumption is sensitive to processor frequency to different extents. We refer to frequency sensitivity as compute boundedness (denoted by c), ranging from 0 to 1. When an application is frequency insensitive, the power consumption is constant, drawing the baseline power (b). At the other extreme, the power consumption of a 100% frequency-sensitive application is quadratic with frequency. Thus, the power consumption can be quantified as

$$Power = c\gamma f^2 + c\delta f + b, \quad (4)$$

where f is the normalized processor frequency ($=f_{any}/f_{max}$), $Power$ is the normalized average power consumption which is 1 when $f=1$, and γ and δ are regression parameters. Parameters γ , δ and c are application and platform specific. The value of c can be determined by the expected behavior (e.g., the proportion of computations) of the application. It can also be calculated based on the power measurements of the application as $c = (\text{mean power} - \text{idle power})/(\text{maximal power} - \text{idle power})$.

The execution time of an application is also a function of compute boundedness. For a frequency-insensitive application, its execution time is independent of frequency. If the application is entirely frequency sensitive, its performance (i.e., inverse time) is linear in frequency. Therefore, the execution time can be modeled as

$$Exectime = \frac{c}{\alpha f + \beta} + 1 - c, \quad (5)$$

where α and β are regression parameters with $\alpha + \beta = 1$ and $Exectime$ is the normalized execution time which is 1 when $f = 1$.

The models in Equations (4) and (5) are more accurate in characterizing the power consumption and execution time of applications run on HPC machines. However, they are not intuitively interpretable. The application and platform specific parameters and computation complexity make them not easy to implement. Users of *TracSim* simulator can provide the parameters of these models. Thus *TracSim* can apply the power and performance models in simulations.

Currently, *TracSim* is implemented to simulate homogeneous clusters, which requires a relatively small amount of profiled performance and power usage data. To enhance *TracSim* we plan to explore measurements from multiple, heterogeneous clusters to extend the power usage and performance models and leverage the framework of *TracSim* to simulate heterogeneous HPC systems.

In addition, *TracSim* currently exploits CPU-throttling techniques to change the power usage of nodes. We will add more attributes and functions which will allow users to adjust the configuration of disks and other devices to develop and evaluate more comprehensive resource scheduling policies.

8. Conclusions

Power consumption and energy-aware resource management are increasingly important in the design and operation of large-scale HPC systems. A deep understanding of the correlation between the various design factors and the system’s power usage is valuable for developing power-saving and energy-efficient resource policies while meeting the expected performance requirements. In this work, we introduce the *TracSim* simulator, which enables users to simulate a high-performance computing environment and to evaluate the performance of different policies for adjusting resource configurations and scheduling parallel jobs under a power cap. The novelty of our approach is that we enable users to develop and test resource scheduling policies that are aware of both power caps and job priorities and that we use real measurements from a production HPC system to set configuration parameters. The simulation results presented provide insights on how to efficiently utilize the trapped capacity to maximize the machine room throughput.

In this work, we proposed a *power-aware dynamic resource configuration* (PDRC) scheduling policy, which schedules jobs based not only on their priority but also with regard to a system-wide power cap, which can be

changed dynamically. Among the tested resource scheduling policies evaluated on *TracSim* PDRC achieves the highest utilization (94.9%) of trapped capacity, outperforming the power-precedence policy by 11.3%, under a dynamic power cap. This improvement can be used to power an additional 16% more nodes in the cluster. The performance-precedence policy violates the power cap by 28.1%. Moreover, PDRC achieves the highest CPU utilization reaching 80.3% and a low job turnaround time without power cap violation in the experiments. The results of *TracSim* have been validated on a production HPC machine and the accuracy of the simulated power consumption is above 90% in most cases.

Our work further demonstrates that thoughtful management of power using active power capping enables a substantial amount of trapped capacity to be reclaimed. For example, PDRC enables 16% more nodes to be added to our cluster without increasing the power infrastructure associated with that system. On a major supercomputer, 16% can represent a substantial amount of performance.

Acknowledgement

This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-AC52-06NA25396. The publication has been assigned the LANL identifier LA-UR-14-20294.

A preliminary version of this paper was presented at International Workshop on Energy Efficient Supercomputing (E2SC) in conjunction with IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14), November 2014 [44].

References

- [1] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, Top500 supercomputer sites: November 2014 (Nov. 18, 2014).
URL <http://www.top500.org/lists/2014/11>
- [2] J. Choi, S. Govindan, B. Urgaonkar, A. Sivasubramaniam, Profiling, prediction, and capping of power consumption in consolidated environments, in: Proceedings of IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS), 2008.

- [3] C. Lefurgy, X. Wang, M. Ware, Power capping: a prelude to power shifting, *Cluster Computing* 11 (2) (2008) 183–195.
- [4] S. Pakin, M. Lang, Energy modeling of supercomputers and large-scale scientific applications, in: *Proceedings of IEEE International Green Computing Conference (IGCC)*, 2013.
- [5] S. Pakin, C. Storlie, M. Lang, et al., Power usage of production supercomputers and production workloads, *Concurrency and Computation: Practice and Experience*. To appear.
- [6] C.-H. Hsu, W.-C. Feng, A power-aware run-time system for high-performance computing, in: *Proceedings of ACM/IEEE Supercomputing Conference (SC)*, 2005.
- [7] S. Pelley, D. Meisner, P. Zandevakili, et al., Power routing: dynamic power provisioning in the data center, in: *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.
- [8] M. Femal, V. W. Freeh, Boosting data center performance through non-uniform power allocation, in: *Proceedings of IEEE International Conference on Autonomic Computing (ICAC)*, 2005.
- [9] K. H. Kim, R. Buyya, J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters., in: *Proceedings of IEEE International Symposium on Cluster Computing and Grid (CCGrid)*, 2007.
- [10] R. Ge, X. Feng, K. W. Cameron, Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters, in: *Proceedings of ACM/IEEE Supercomputing Conference (SC)*, 2005.
- [11] H. F. Sheikh, H. Tan, I. Ahmad, S. Ranka, P. Bv, Energy- and performance-aware scheduling of tasks on parallel and distributed systems, *Journal on Emerging Technologies in Computing Systems* 8 (4) (2012) 32:1–37.
- [12] Z. Zhang, S. Fu, An adaptive power management framework for autonomic resource configuration in cloud computing infrastructures, in: *Proceedings of IEEE International Performance Computing and Communications Conference (IPCCC)*, 2012.

- [13] A. Verma, P. Ahuja, A. Neogi, Power-aware dynamic placement of HPC applications, in: Proceedings of ACM International Conference on Supercomputing (ICS), 2008.
- [14] E. Grobelny, D. Bueno, I. Troxel, A. D. George, J. S. Vetter, Fase: A framework for scalable performance prediction of HPC systems and applications, *Simulation* 83 (10) (2007) 721–745.
- [15] D. Economou, S. Rivoire, C. Kozyrakis, P. Ranganathan, Full-system power analysis and modeling for server environments, in: Proceedings of Workshop on Modeling, Benchmarking, and Simulation in conjunction with ACM Symposium on Computer Architecture (ISCA), 2006.
- [16] D. Jackson, H. Jackson, Q. Snell, Simulation based HPC workload analysis, in: Proceedings of IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2001.
- [17] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, S. Dropsho, Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor, in: Proceedings of ACM ISCA, 2003.
- [18] W. Ye, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, The design and use of SimplePower: A cycle-accurate energy estimation tool, in: Proceedings of ACM Design Automation Conference (DAC), 2000.
- [19] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, R. Wang, Modeling hard-disk power consumption, in: Proceedings of USENIX Conference on File and Storage Technologies (FAST), 2003.
- [20] T. Ye, L. Benini, G. De Micheli, Analysis of power consumption on switch fabrics in network routers, in: Proceedings of ACM Design Automation Conference (DAC), 2002.
- [21] Z. Zhang, S. Fu, macropower: A coarse-grain power profiling framework for energy-efficient cloud computing, in: Proceedings of IEEE International Performance Computing and Communications Conference (IPCCC), 2011.
- [22] S. Kamil, J. Shalf, E. Strohmaier, Power efficiency in high performance computing, in: Proceedings of IEEE Parallel & Distributed Processing Symposium (IPDPS), 2008.

- [23] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K. Cameron, Powerpack: Energy profiling and analysis of high-performance systems and applications, *IEEE Transactions on Parallel and Distributed Systems* 21 (5) (2010) 658–671.
- [24] V. Jiménez, F. J. Cazorla, R. Gioiosa, M. Valero, C. Boneti, E. Kursun, C.-Y. Cher, C. Isci, A. Buyuktosunoglu, P. Bose, Power and thermal characterization of POWER6 system, in: *Proce. of ACM International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010.
- [25] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, C. Le, RAPL: memory power estimation and capping, in: *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2010.
- [26] M. Y. Lim, A. Porterfield, R. Fowler, SoftPower: fine-grain power estimations using performance counters, in: *Proceedings of ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.
- [27] M. Brown, J. Renau, ReRack: power simulation for data centers with renewable energy generation, *SIGMETRICS Performance Evaluation Review* 39 (3) (2011) 77–81.
- [28] D. Brooks, V. Tiwari, M. Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, in: *Proceedings of ACM International Symposium on Computer Architecture (ISCA)*, 2000.
- [29] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, L. K. John, Using complete machine simulation for software power estimation: The SoftWatt approach, in: *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2002.
- [30] S. L. Song, K. Barker, D. Kerbyson, Unified performance and power modeling of scientific workloads, in: *Proceedings of International Workshop on Energy Efficient Supercomputing (E2SC) in conjunction with ACM/IEEE Supercomputing Conference (SC)*, 2013.

- [31] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, 2nd Edition, Vol. 1, The MPI Core, The MIT Press, Cambridge, Massachusetts, 1998.
- [32] R. Cheveresan, M. Ramsay, C. Feucht, I. Sharapov, Characteristics of workloads used in high performance and technical computing, in: Proceedings of ACM International Conference on Supercomputing (ICS), 2007.
- [33] H. Zhang, H. You, Comprehensive workload analysis and modeling of a petascale supercomputer, in: Proceedings of Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) in conjunction with IEEE Parallel & Distributed Processing Symposium (IPDPS), 2012.
- [34] Y. Etsion, D. Tsafir, A short survey of commercial cluster batch schedulers, Tech. Rep. 2005-13, School of Computer Science and Engineering, The Hebrew University of Jerusalem, available from <http://leibniz.cs.huji.ac.il/tr/742.pdf> (May 2005).
- [35] A. B. Yoo, M. A. Jette, M. Grondona, SLURM: Simple Linux utility for resource management, in: D. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, 12th IEEE International Symposium on High-Performance Distributed Computing, Vol. 2862 of Lecture Notes in Computer Science, Springer, Seattle, Washington, 2003, pp. 44–60.
- [36] Z. Zhou, Z. Lan, W. Tang, N. Desai, Reducing energy costs for ibm blue gene/p via power-aware job scheduling, in: Proceedings of Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) in conjunction with IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2013.
- [37] J. D. McCalpin, Memory bandwidth and machine balance in current high performance computers, IEEE Technical Committee on Computer Architecture (TCCA) Newsletter (1995) 19–25.
- [38] P. Gepner, V. Gamayunov, D. L. Fraser, Effective implementation of DGEMM on modern multicore CPU, *Procedia Computer Science* 9 (2012) 126–135.
- [39] M. Gittings, R. Weaver, M. Clover, et al., The RAGE radiation-hydrodynamic code, *Computational Science & Discovery* 1 (1).

- [40] P. W. Jones, P. H. Worley, Y. Yoshida, J. White, J. Levesque, Practical performance portability in the Parallel Ocean Program (POP), *Concurrency and Computation: Practice and Experience* 17 (10) (2005) 1317–1327.
- [41] K. J. Bowers, B. Albright, L. Yin, B. Bergen, T. Kwan, Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation, *Physics of Plasmas* 15 (5).
- [42] R. S. Baker, A block adaptive mesh refinement algorithm for the neutral particle transport equation, *Nuclear science and engineering* 141 (1) (2002) 1–12.
- [43] P. Gepner, M. F. Kowalik, D. L. Fraser, K. Waćkowski, Early performance evaluation of new six-core Intel Xeon 5600 family processors for HPC, in: *Proceedings of the 9th International Symposium on Parallel and Distributed Computing*, Istanbul, Turkey, 2012, pp. 117–124.
- [44] Z. Zhang, M. Lang, S. Pakin, S. Fu, Trapped capacity: Scheduling under a power cap to maximize machine-room throughput, in: *International Workshop on Energy Efficient Supercomputing (E2SC) in conjunction with IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.