An Energy Efficient Demand-Response Model for High Performance Computing Systems

Kishwar Ahmed, Jason Liu
School of Computing and Information Sciences
Florida International University
Emails: {kahme006,liux}@cis.fiu.edu

Xingfu Wu
Mathematics and Computer Science Division
Argonne National Laboratory
Email: wuxf@tamu.edu

***Abstract*—Demand response refers to reducing energy consumption of participating systems in response to transient surge in power demand or other emergency events. Demand response is particularly important for maintaining power grid transmission stability, as well as achieving overall energy saving. High Performance Computing (HPC) systems can be considered as ideal participants for demand-response programs, due to their massive energy demand. However, the potential loss of performance must be weighed against the possible gain in power system stability and energy reduction. In this paper, we explore the opportunity of demand response on HPC systems by proposing a new HPC job scheduling and resource provisioning model. More specifically, the proposed model applies power-bound energy-conservation job scheduling during the critical demand-response events, while maintaining the traditional performance-optimized job scheduling during the normal period. We expect such a model can attract willing participation of the HPC systems in the demand response programs, as it can improve both power stability and energy saving without significantly compromising application performance. We implement the proposed method in a simulator and compare it with the traditional scheduling approach. Using trace-driven simulation, we demonstrate that the HPC demand response is a viable approach toward power stability and energy savings with only marginal increase in the jobs' execution time.**

I. Introduction

High Performance Computing (HPC) systems, such as petaflops supercomputers, can consume a tremendous amount of power. For example, as of November 2016, China's 34-petaflops Tianhe-2 supercomputer, which currently consumes the most power in the list of top 500 supercomputers [1], has been reported to consume almost 18 MWs of power, sufficient to power a small town of 20,000 residences. With the advent of exascale supercomputers in the next few years, power consumption of the HPC systems will surely increase. The massive power consumption of these HPC systems can expound significant stress for the power grid. HPC has also shown significant fluctuations in the power consumption due to the varying job execution profiles and also sporadic maintenance schedules. Effective power saving and power stabilizing methods must be seriously considered when building future HPC systems.

Demand response programs are designed to help energy service providers to stabilize the power system by reducing the energy consumption of participating systems when the power grid becomes unstable due to a sudden rise in power demand or other emergency incidents (which we refer to as *demand response events*). The U.S. Department of Energy (DoE) and the National Institute of Standards and Technology (NIST) have identified demand response as one of the important policy goals to achieve power grid efficiency [2], [3]. In addition to monetary benefits, demand response can also provide the associated environmental benefits, such as reducing carbon emission [4]. We have observed recent increase in the participation of the demand response programs in various sectors [5], [6]. Recent projection also shows that there will be substantial growth in the coming years—an anticipated doubling of the overall participation in the demand response programs in 2020 has been projected [7].

With such rising popularity of demand response programs, it is not surprising that there exists a large body of research that study demand response participation focusing on various sectors (e.g., data centers [8], smart buildings [9]). These studies exploit various job scheduling schemes (e.g., load shifting, job queuing, geographical load balancing) and leverage many resource management parameters (e.g., speed-scaling, server consolidation, power-capping) to enable demand response program participation. Such studies, however, are not confined to theoretical realm only. Some recent reports suggest that many large-scale companies have already been participating in demand response programs [10], [11]. An empirical study [12] also demonstrated feasibility of data center's participation in demand response programs. Through adopting various demand response strategies (such as server consolidation, job scheduling, workload migration), the study [12] demonstrated data center's capability of reducing energy consumption during demand response periods.

Being a massive energy consumer of the power grid, the HPC sector can contribute toward ensuring grid stability and energy reduction through its participation in the demand response programs. Recent research has studied the feasibility and identified the associated challenges in the HPC demand response [13], [14]. Patki et al. [14] suggested that supercomputing systems in the U.S. may be willing to participate in the demand response programs if tighter and more frequent communications can be established between the supercomputing centers and their energy service providers. The study also suggested that various supercomputing centers are interested to develop demand response capability through enhancing software system for resource management (e.g., power-capping, job scheduling) [13]. This study is based on a qualitative analysis of cooperative demand-management strategies. We note, however, that there is no related work on the job scheduling and resource provisioning strategies at HPC centers that can operate with demand response. Various energy-efficient HPC job scheduling algorithms (e.g., [15], [16]) and resource provisioning methods (e.g., [17], [18]) have been proposed in the literature. These studies aim at reducing the overall energy consumption of the HPC systems, but do not consider demand response.

In this paper, we explore the opportunities of the HPC centers participating in the demand response programs through a study of detailed job scheduling and resource provisioning strategies. More specifically, this paper makes the following contributions:

- We propose an HPC job scheduling and resource provisioning algorithm for demand response. For job scheduling, we assume first-come-first-serve (FCFS) with possible job eviction and restart in response to the reduced power level during the demand response periods. For resource provisioning, we dynamically scale the frequency of the processors in order to achieve optimal energy conservation and power stability during the demand response periods. During normal periods, the processors in HPC systems operate at maximum frequency for best performance.
- We developed a simulator for job scheduling and resource provisioning to study the effect of demand response. The simulator is built upon a parallel discrete-event simulation engine capable for handling large-scale models. The simulator has been validated using real-life HPC workload traces.
- We conducted extensive simulation studies to show the effectiveness of the proposed job scheduling and resource provisioning algorithm for demand response. The results demonstrate that our proposed approach is a viable solution for attracting supercomputing centers to participate in the demand response programs as it can improve power stability and energy reduction with only moderate increase in execution time for the jobs.

The rest of this paper is organized as follows. Section II describes related work and compares the existing approaches with our proposed method. Section III proposes the model for HPC demand response, including a job scheduling and resource provisioning algorithm. We also describe the energy and performance models used by the algorithm. Section IV discusses our job scheduler simulator, which has been developed to study the effectiveness of demand-response-aware job scheduling and resource provisioning methods. Section V shows a comparative study of our approach against traditional methods using real-life HPC workload traces. Section VI concludes the paper with a discussion on the assumptions and limitations of our approach, and outlines some interesting directions for future work.

II. Related Work

In this section, we discuss related work in performance and power prediction models, dynamic voltage and frequency scaling (DVFS) methods for energy saving, HPC job scheduling and resource provisioning strategies, and demand response techniques for data centers.

Many power and performance prediction models have been proposed in the literature. For example, Singh, Bhadauria, and McKee [19] proposed an analytical model for real-time prediction of processor and system power consumption. Performance monitoring counters are used to estimate the power consumption of the processors. Shoukourian et al. [20] proposed an analytical model for application-specific power and energy prediction. Based on historical energy usage by specific applications, the model predicts future power and energy usage; the model can also adapt the prediction accuracy with further execution of the applications. Wu et al. [21] also presented performance and power models based on hardware performance counters with CPU frequency. They used non-negative multivariate regression analysis to build models for application execution time, system power, CPU and memory power, using a small set of major performance counters and CPU frequency. They implemented a counter-ranking method to identify the model contribution of the measured counters. The model can be used to suggest modifications of applications to improve execution time and power consumption.

Different energy saving techniques have also been proposed for HPC systems. They include energy-efficient design for hardware components, including CPU, memory, and interconnection network. Saving energy intuitively implies a reduction in power consumption, runtime, or both. Wu et al. [21] classified the methods in this area into three categories: reduce time and power, reduce time but allow an increase in power, and reduce power while allowing an increase in time. Energy-saving methods that exploit the dynamic voltage and frequency scaling (DVFS) capabilities on processors have been introduced (e.g., [17], [22], [23]). CPU MISER [17] is an early effort that includes a runtime DVFS-based HPC power management scheme, which exploits different application phases using performance measurements (in cycles per instruction) during the execution of the applications. Freeh et al. [23] proposed an energy saving approach exploiting the energy-time trade-off of MPI programs. Adagio [22] performs runtime CPU frequency scaling and exploits the variations in the energy consumption during computation and communication phases of an application to reduce the overall energy consumption without impacting the overall execution time of the application. A more recent effort on DVFS by Bao et al. [18] automatically selects the optimal frequency and core count at compile-time to achieve lower energy.

Job scheduling and resource provisioning methods have also been proposed for HPC systems to save energy. They assume bounded energy consumption of the systems (e.g., [15], [16]). Yang et al. [24] proposed a job scheduling approach to exploit the variable electricity price and power consumption profile of the jobs. A day is divided into two parts based on the electricity price: on-peak and off-peak. Jobs are classified based on their power profiles (derived from past execution data). Low power-consuming jobs are executed preferably during the on-peak time periods, while high power-consuming jobs are executed preferably during the off-peak time periods. Two power-aware job scheduling solutions are proposed: a greedy policy and a 0-1 knapsack-based policy, where fairness is ensured through a window-based scheduling mechanism. Sarood et al. [15] proposed an online job scheduling and resource allocation approach to achieve power-efficiency in HPC systems. The resource management system leverages over-provisioning, power-capping and job malleability (i.e., dynamic shrinking and expanding the job size) to optimally allocate power and nodes. Cao, He, and Kondo [25] recently proposed a job scheduling algorithm to limit the overall system power consumption within a given power budget and improve the system throughput and resource utilization. While we also implement power-capped job scheduling in our proposed model, our goal is to improve power grid stability and energy saving for HPC demand-response participation. We schedule jobs and allocate resources to achieve optimal energy (only during demand response events), while theirs is to achieve power capping.

There also exist efforts for green HPC through reducing the use of brown energy in HPC systems. For example, GreenPar [26] adopts different job scheduling strategies (e.g., dynamic job migration, and resource allocation) to reduce brown energy consumption. ZCCloud [27] also explored possibility of increasing wasted green renewable energy in the system. The proposed model in the paper [27] can reduce carbon footprint in the system, while achieving optimal job performance (e.g., through reducing job turnaround time). However, none of the proposed work considers demand response participation specific to HPC systems, which is the goal of our study described in this paper.

Workload scheduling and resource provisioning in data center with consideration of demand response scheme have been studied quite extensively. Load shifting in time, geographical load balancing, speed-scaling, server consolidation, power-capping are some of the approaches proposed in the literature for data center's demand response [8]. Caramanis et al. have also studied demand response participation for both data centers (e.g., [28]) and smart buildings (e.g., [9]). These studies address the problem of ensuring data center's participation in demand response program through responding to regulation signal from the power grid, while considering intermittent renewable energy sources. However, these approaches are applicable for internet transaction-based data center workload, not for HPC applications. For data center workload, the service time is typically assumed to be uniform and delay intolerant (most jobs need to be serviced within the hour from which they are submitted). HPC jobs are much less uniform both in terms of service time and job size (requested resources in the number of processors). Also, most HPC jobs can tolerate some delays (given that some jobs may take hours or days to finish). As such, the data center demand-response models cannot be applied to HPC systems.

To the best of our knowledge, the solution outlined in this paper is the first demand-response-aware job scheduling and resource provisioning scheme for HPC applications.

III. HPC Demand Response Model

In this section, we present our HPC demand response model. At first, we present performance and energy models based on frequency scaling in section III-A, which we use for determining a job's runtime and energy consumption in the proposed job scheduling and resource provisioning algorithm, which we describe next in section III-B. The algorithm involves dynamically adjusting the processors' frequency for all running jobs in order to achieve optimal energy conservation, which we describe in section III-C. The algorithm possibly involves evicting running jobs during a demand-response period if the power consumption exceeds the set limit. We describe an optimal job eviction method in section III-D.

A. Power and Performance Prediction Models

Consider the set of jobs to be executed on an HPC system is $\{1, 2, \cdots, J\}$. For each job j, we denote the average power consumption running at CPU frequency f as $p(j, f)$, and the execution time as $t(j, f)$. There are a number of existing models for predicting a job's average power consumption and execution time. Here, we use a rather simple regression-based model. We derive the relationship between CPU power and runtime with respect to frequency using linear regression. To do so, we first observe the average power and runtime characteristics of each job running at different frequency values. We then determine a polynomial fitting function based on the observed data.

Similar approaches can be found in other studies (e.g., [29], [30]). The purpose of this study is to assess the feasibility of having HPC centers to participate in the demand response programs, by proposing a job scheduling and resource provisioning algorithm that can improve power stability and energy conservation while maintaining good application runtime performance. Here we choose simple prediction models, which can be later improved for more general applications.

For determining the average power consumption, we use a similar model as proposed in the paper [21]. The average power consumption of job j running on a processor at frequency f can be estimated using the following third-order polynomial function:

$$p(j, f) = a + b \cdot f + c \cdot f^2 + d \cdot f^3 \tag{1}$$

where a, b, c, and d are constants determined from empirical analysis of average power relation with different frequency values. Here, a represents the static power consumption while running the job.

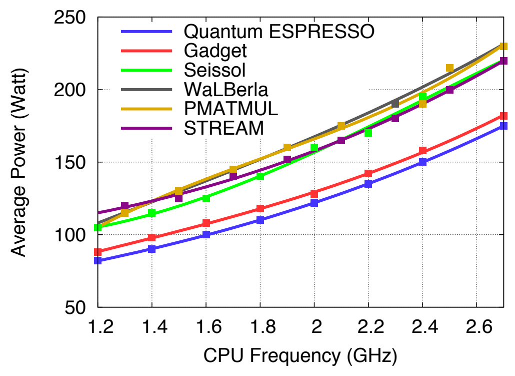
In a similar approach, we can determine the execution time of job j at frequency f using the following function:

$$t(j, f) = \alpha + \beta \cdot f + \gamma \cdot f^2 \tag{2}$$

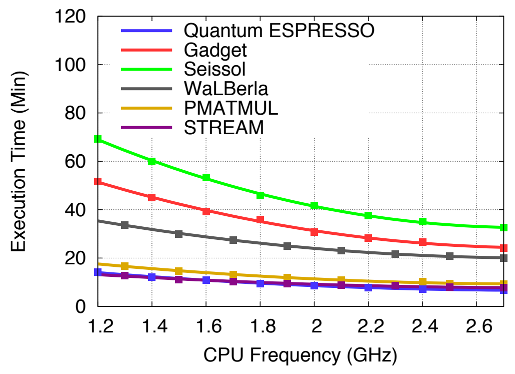
where α, β, and γ are regression coefficients determined from polynomial fitting function using empirical data.

We assume that a job j runs with the same CPU frequency f on all n_j processors. As such, the total energy consumption of the job can then be determined as follows:

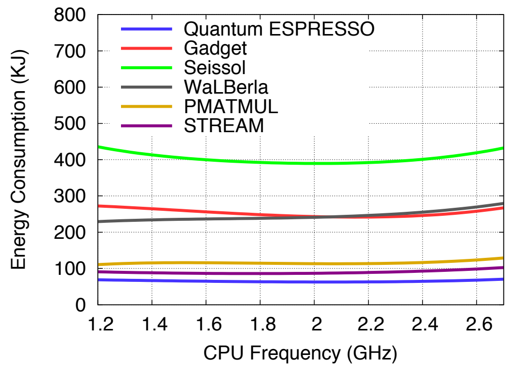
$$e(j, f) = n_j \cdot p(j, f) \cdot t(j, f) \tag{3}$$



(a) Average Power



(b) Execution Time



(c) Energy Consumption

Fig. 1. Result of the power and performance prediction models for six HPC applications.

To illustrate the power and performance prediction models, we collected frequencies and frequency-related power variations for six HPC applications from an existing study [29]. More specifically, the six applications include four scientific applications (including Quantum ESPRESSO [31], Gadget [32], Seissol [33] and WaLBerla [34]) and two synthetic benchmarks (where PMATMUL is a parallel benchmark for dense matrix multiplication, and STREAM is a benchmark for measuring sustainable memory bandwidth [35]). Measurements were collected when running these applications with different CPU frequencies, ranging from 1.2 GHz to 2.7 GHz.

We use the regression models to derive the least square polynomial fitting functions representing the relationship of average power consumption and execution time with the scaling frequency for each application. Fig. 1 shows the result from the power and performance regression models for the six HPC applications, as well as their total energy consumption. Fig. 1(a) and Fig. 1(b) show the empirical data and fitted polynomial function for average power and execution time, respectively. Fig. 1(c) shows the total energy consumption for different frequencies, derived from the average power and execution time polynomial fitting function models. The regression models we use for this study generally incur low prediction errors. For example, errors for predicting execution time are 0.69%, 1.1%, 0.68%, 1.25%, 0.95% and 1.14% for the applications Quantum ESPRESSO, Gadget, Seissol, WaLBerla, PMATMUL and STREAM, respectively. Errors for predicting power using our model are 0.14%, 0.44%, 1.01%, 0.82%, 1.32% and 0.53% for the same set of applications.

In general, the average power consumption of the applications increases as we increase the CPU frequency. The execution time decreases as we increase the CPU frequency. The total energy for running the applications is the product of the average power and execution time, which may either increase, decrease, or have its minimum somewhere in the given frequency range, depending on the applications.

B. Job Scheduling and Resource Provisioning

We describe the proposed job scheduling and resource provisioning algorithm. When a job is submitted, it is inserted in the waiting queue Q and the job scheduling algorithm is invoked. Here we use the first-come-first-serve (FCFS) policy, although other job scheduling policies can be applied as well.

The job scheduler keeps the list of running jobs R. Each job $j \in R$ runs on n_j processors as requested. The scheduler determines the frequency of the processors, f_j, that the job is run. The frequency can be changed dynamically during the job's execution depending on the current running jobs and the available power limit. (We assume all processors running the same job maintain the same frequency nevertheless.)

Let f_{min} and f_{max} denote the minimum and maximum frequency allowed by the HPC processor architecture. That is,

$$f_{min} \leq f_j \leq f_{max} \tag{4}$$

Let $\hat{p}$ be the current power limit set by the energy service provider. The power cap $\hat{p}$ can be set to a lower value during a demand response period and infinite otherwise. In any case, the scheduling algorithm needs to ensure that the average power consumption of all the running jobs, p_{run}, is bounded by the current power limit. That is,

$$p_{run} = \sum_{j \in R} p(j, f_j) \leq \hat{p} \tag{5}$$

The pseudo-code of the proposed HPC scheduler is shown in Alg. 1. When invoked, the scheduler first checks to see if there is an eligible job to run in the job waiting queue (line 1). We scan the waiting jobs from the head of the queue to the tail of the queue according to the FCFS policy. The job w

Algorithm 1 HPC Demand Response Job Scheduler

```
1: find the first eligible job $j$ in $Q$
2: if job $j$ exists then
3:     dequeue job $j$ from $Q$
4:     allocate $n_j$ processors to run job $j$
5:     $R \leftarrow R \cup \{j\}$
6:     goto line 1
7: end if
8: determine optimal frequency $\forall j \in R$ (section III-C)
9: if no optimal solution exist then
10:    evict jobs to reduce power consumption (section III-D)
11:    goto line 8
12: end if
13: reset processor frequency if changed $\forall j \in R$
```

is eligible to run, (a) if there are enough available processors, that is,

$$n_w + \sum_{j \in R} n_j \leq \hat{n} \tag{6}$$

where $\hat{n}$ is the total number of processors in the system, and (b) if the average power consumption of all running jobs remains within the power limit (assuming we run job w with the minimum allowed frequency):

$$p(w, f_{min}) + p_{run} \leq \hat{p} \tag{7}$$

If such a job is found, we remove the job from the waiting queue (line 3) and allocate the processors to run the job (line 4). The new job is placed into R, which maintains the set of all currently running jobs (line 5). This process is then repeated until we find all eligible running jobs from the waiting queue.

Before the new jobs begin, we first need to determine the optimal frequency of the processors to run them (line 8). Also, it may be necessary to adjust the frequency of existing running jobs (not just the new arrivals) to achieve the optimal energy conservation. We discuss the details of calculating optimal frequencies in section III-C.

The job scheduler is invoked when a new job is submitted or when a running job has finished execution. In the latter case, the completed job is simply removed from R and the scheduler is invoked so that other eligible jobs can be scheduled to run. Another possible case for invoking the scheduler is when the energy service provider changes the power limit $\hat{p}$ of the HPC system. This can be the start of a demand response event, in which case a lower power limit is imposed, or at the end of a demand response event, when the power limit returns to normal (e.g., infinite or some higher values for hardware overprovisioned systems [36]). If the power limit is reduced for a demand response event, it is possible that no optimal solution can be found for frequency scaling of the existing running jobs. In that case, one or more running jobs must be terminated prematurely to preserve power (line 10). We discuss this step in more detail in section III-D on how to choose the victims so that we can minimize the overall impact. Once the eviction is done, we need to calculate again the optimal frequency of the remaining running jobs.

In the last step (line 13), we change the frequency of the processors of the running jobs, as long as their newly calculated frequency is different from the previous settings. The job scheduler finishes the current invocation and will wait until it is invoked again in response to either a new job arrival, a job departure, or a demand response event.

C. Determining Optimal Frequency

This step is to calculate the frequency of the processors running the jobs. During the normal operating time, the power limit should be infinite (or set to be the peak power), in which case the jobs can run at the maximum allowed CPU frequency, i.e., f_{max}, to achieve the best performance. This is a conscious decision. HPC systems are designed for high performance. A willing participation of supercomputing centers in the demand response programs should not alter the main design purpose of these HPC systems. We want to minimize the overall impact of demand response, in this case, by recovering the potential performance loss by maximizing the application performance outside the demand response periods.

However, once a demand response event happens, we need to resort to the energy conservation mode. In this case, we want to select the proper CPU frequencies of all running jobs so that we can minimize the energy use while observing the reduced power limit set by the energy service provider. By reducing the energy demand, we can contribute to stabilizing the power grid which may encounter possible emergency situations. This frequency selection problem can be formulated as an optimization problem, as follows:

$$\text{Minimize: } \textstyle\sum_{j \in R} e_R(j, f_j)$$
$$\text{subject to constraints (4) and (5)}$$

where $e_R(j, f_j)$ denotes the remaining energy expected to be consumed if running job j at frequency f_j. It can be calculated as follows:

$$e_R(j, f_j) = (1 - \alpha_j) \cdot n_j \cdot p(j, f_j) \cdot t(j, f_j) \tag{8}$$

where α_j is the percentage of job j that has been completed thus far. This quantity can be accumulated by the job scheduler upon each time the job is updated with a new frequency.

It is commonly believed that the energy and frequency observe the convexity property under certain conditions [37], such as a job's average power consumption and execution time are monotonic functions of the frequency within a given range. This convexity property suggests the existence of an optimal frequency where energy consumption can be minimalized. We can therefore solve the optimization problem with the sequential least squares programming algorithm using the Han-Powell quasi-Newton method [38]. The optimization problem solver returns the frequencies $f_1, f_2, \cdot, f_{|R|}$ for all running jobs in $j \in R$. Note that in practice, processors can choose from a certain set of frequencies dictated by hardware. In this case, we can pick the closest allowed frequency that is no larger than the optimal frequency value.

D. Job Eviction

As mentioned earlier, with reduced power limit during a demand response event, it is possible that no optimal frequencies can be found for the existing running jobs, in which case some jobs have to be terminated to preserve power. In this section, we provide an algorithm for choosing the jobs so that we can minimize the impact.

We represent the selection of job j by a binary variable, $x_j \in \{0,1\}$, where $x_j = 1$ denotes that job j is selected to continue and $x_j = 0$ denotes that job j is selected for eviction. We formulate an optimization problem to determine the optimal subset of the running jobs such that the power bound constraint can be satisfied, with the objective of maximizing the energy that has already been spent by the running jobs. The idea is that we want to keep the jobs that have consumed more energy, because evicting them would mean this energy would be wasted as they need to rerun.

We formulate the optimization problem as follows:

$$\begin{gathered}\text{Maximize: } \textstyle\sum_{j\in R}\left(x_j \cdot e_X(j)\right)\\ \text{subject to} \textstyle\sum_{j\in R}\left(x_j \cdot p(j, f_{min})\right) \le \hat{p}\end{gathered}$$

where $e_X(j)$ is the energy that has so far been spent running job j. On the one hand, the job scheduler can accumulate $e_X(j)$ using power measurement. On the other hand, we can adopt an easier alternative, by estimating the energy cost of a job using the job's completion percentage value α_j and the projected energy used under the current frequency f_j. That is,

$$e_X(j) \approx \alpha_j \cdot n_j \cdot p(j, f_j) \cdot t(j, f_j) \tag{9}$$

We can convert this optimization problem into a 0-1 knapsack problem and solve it directly. In this case, we treat $\hat{p}$ as the knapsack capacity, $p(j, f_{min})$ as the weight associated with each job, and the spent energy $e_X(j)$ as the job's value.

IV. Job Scheduler Simulator

We use simulation to study the effect of the proposed job scheduling and resource provisioning algorithm both on performance and energy. Plenty job scheduler simulators exist. For example, PYSS (Python Scheduler Simulator) is an open-source HPC workload scheduling simulator written in Python [39]. The simulator was developed by the Experimental System Lab at the Hebrew University, and has been used to study various scheduling algorithms (e.g., [40], [41]). CQSim is another event-based simulator to study the detailed queuing behavior of job schedulers using real system workload [42]. The simulator was developed by Illinois Institute of Technology and has been used to evaluate fault-aware utility-based job scheduling [43], adaptive metric-aware job scheduling [44], and so on. Current HPC simulators provide only limited capabilities for studying job scheduling. For example, SST/Macro contains only limited support for running multiple jobs via trace replay [45]. CODES offers similar capabilities using trace replay to study multi-job workload of proxy applications and their impact on communication over different interconnection networks [46].

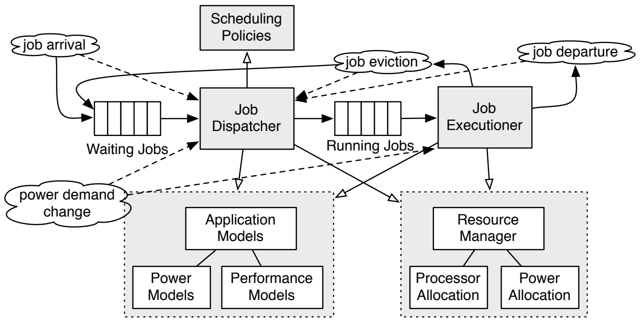


Fig. 2. The overall design of our job scheduler simulator.

In general, job scheduling is relatively straightforward to simulate and validate. We developed our own simulator with trace-driven capabilities so that we can have the flexibility to incorporate new scheduling functions, power-aware methods, as well as demand response models.

A. Simulator Design

Our job scheduler simulator is developed based on Simian, which is an open-source, process-oriented, parallel discrete-event simulation engine [47]. Simian has several unique design features that make it more attractive for us to build our scheduler simulator. First, Simian has a very simple application programming interface (API). The simulator adopts a minimalistic design with only a handful of core functions. The code base is around 500 lines at its core, which makes it easier to understand and debug the applications. Simian also supports process-oriented world view for easy model development. Second, Simian is developed using interpreted languages, including Python, LUA, and Javascript. Simian takes advantage of just-in-time (JIT) compilation and, for some models, has demonstrated capable of even outperforming simulators using compiled languages, such as C or C++. Simian is also a parallel discrete-event simulator, capable of running large-scale models on parallel platforms. Third, there has been a significant ongoing effort in developing models for HPC architectures and applications using Simian (e.g., [48], [49], [50], [51]). Our job scheduler can take advantage of these models.

The overall design of the simulator is illustrated in Fig. 2. The job scheduler simulator consists of five major components: a job dispatcher, a job executioner, scheduling policies, application models, and a resource manager. The job dispatcher takes four different types of events: job arrival, job departure, job eviction (when an executed job is interrupted and removed in the middle of the run), and power demand change (when the power service provider of the HPC center changes the current power limit either at the start or the end of a demand response event). When a job is submitted, it enters the job waiting queue and invokes the job dispatcher. The job dispatcher determines whether the job is eligible to run according to the application models (that describe the job's power and performance characteristics) and the current available resources from the resource manager.

The job dispatcher processes the jobs from the waiting queue according to the scheduling policies. For this study, we only use FCFS, although other policies, such as backfilling [52], may be incorporated as well.

When a job is scheduled to run, the job dispatcher removes the job from the waiting queue and put it in the list of running jobs. The job executioner allocates the resources using the resource manager to represent the occupied processors (at the specified frequencies) with associated power consumption for running the job. The job executioner then simulates the job's execution accordingly. For this study, it is sufficient to simulate using the job's execution time and power consumption according to the estimates from the power and performance models. Detailed job execution can also be simulated for specific computation and communication demands, in case one needs to model the application's runtime behavior. When a job completes its execution, the job executioner removes the job from the list of running jobs, reclaims the resources occupied by the job, and then invokes the job dispatcher to select new eligible jobs to run.

Our simulator has also been augmented to handle demand response. We can schedule an event to indicate the power demand change, with a lower power limit upon the arrival of a demand response event, or an another when the power limit returns to level for normal operations. In the former case, the job executioner may evict jobs if the current power level is no longer sufficient to support all running jobs. In the latter case, the job scheduler may start new jobs to run.

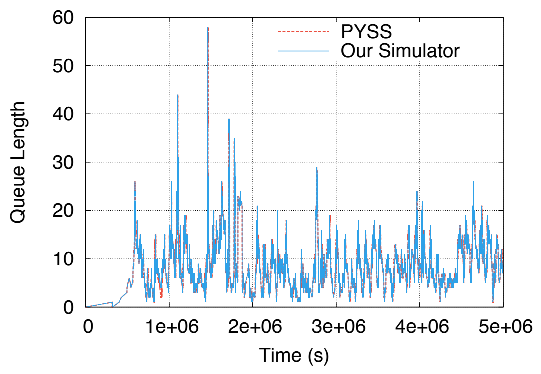
B. Simulator Validation

We conducted experiments to validate the basic functions of our job scheduler simulator. We used the real system workload traces, obtained from the Parallel Workloads Archive [53]. The workload traces contain runtime information collected at the San Diego Supercomputer Center (SDSC) during the time period from May 1998 through April 2000. The runtime information contains the job start time, the job run time, the requested number of processors, and the job wait time, etc. We use this dataset for validation by comparing the performance of our simulator with that of PYSS [39], which has been previously validated against empirical results.

The results are shown in Fig. 3. The specific workload trace we use contains 5,000 jobs running on a system with 512 processors. The top plot shows the length of the job waiting queue as it fluctuates over time. The bottom plot shows the number of available processors in the system. In both cases, we can observe that our simulator generates results that match well with those from PYSS.

V. Performance Evaluation

In this section, we present an elaborate trace-based simulation study to evaluate the effectiveness of the proposed HPC job scheduling and resource provisioning algorithm for demand response.



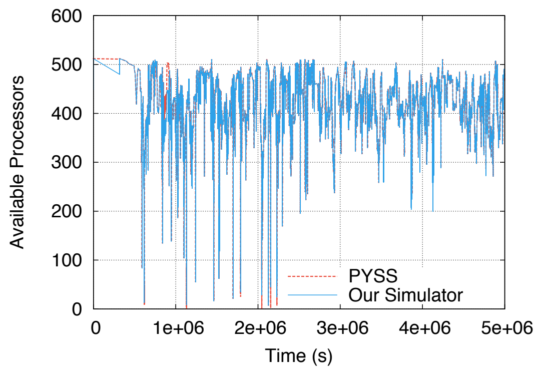


Fig. 3. Comparing results from PYSS and our simulator.

A. Data Sets for Benchmarking

We use real-life workload trace to evaluate our design. More specifically, the trace was collected at the San Diego Supercomputer Center (SDSC SP2), which contains 5,000 jobs. This trace has been used widely in the literature, and referenced in a number of studies throughout the years to generate useful workloads (e.g., [54], [55]).

The workload trace we collected does not contain any power usage information. Therefore, we collected and used power-related information from literature for this study. We use the performance and power data at different frequencies for four HPC applications (Quantum ESPRESSO, Gadget, Seissol and WaLBerla), as outlined in Section III. We use discrete frequency values for the processors, ranging from 1.2 GHz to 2.4 GHz at 0.2 GHz intervals and 2.7 GHz. The peak power of the processors was set to 220 W (determined from the power consumption of the four HPC applications when running at the maximum frequency). We target three HPC systems, which consist of 128, 256, and 512 processors, respectively. The peak power capacity for the 512-processor system can reach 112.64 KW.

To evaluate the performance of our job scheduling and resource provisioning algorithm for demand response, we compare it with two scheduling policies that do not consider demand response. *Performance-policy* is one of the CPU frequency scaling policies implemented in the Linux kernel [56]. It always chooses the maximum frequency to ensure best application runtime performance [18]. *Powersave-policy* is the opposite to the previous one, also implemented in the

Linux kernel [56]. Under this policy, the processors are run instead with the minimum frequency, to minimize the power consumption for application execution.

In the following, we show the results from our simulation study. We first present the power capping capability of our demand-response algorithm. We then compare results from the demand response algorithm with those from the two demand-response-agnostic policies, both in terms of average job turnaround time and average energy consumption. Next, we show the potential improvement in the power stability achieved by the demand response algorithm. Finally, we perform a sensitivity study, where we artificially introduce errors in the power and performance prediction models to study their effect on our proposed approach.

B. Power Capping

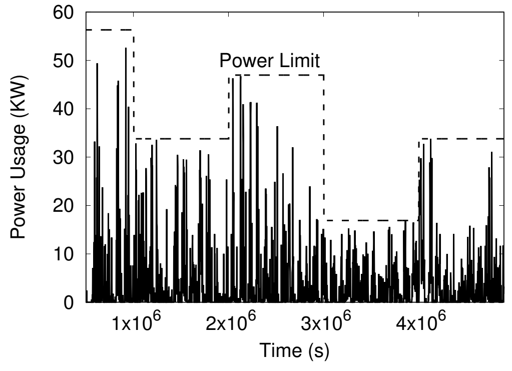
During a demand response event, the proposed job scheduling and resource provisioning algorithm switches to the energy conservation mode. In addition, the system's power consumption is also kept to be within a given power limit in order to improve power stability. To show its effect of power capping, we designed an experiment by changing the power limit at different time intervals to demonstrate that our algorithm can schedule jobs according to the set power constraint at the time.

In this experiment, we arbitrarily set different power limit over time. Fig. 4 shows the result when we set the power limit at regular intervals to be 50%, 30%, 41.7%, 15%, and 30% of the system's peak power. The figure shows the power usage of the system over time, with and without power capping. In the former case, we used our demand response algorithm. In the latter case, we used the default performance-policy, which selects the maximum CPU frequency to run the jobs. The figure shows that our demand response algorithm can adapt to the changes in the power limit and schedule jobs accordingly under the power constraint. As a result, the power usage for demand-response policy does not go beyond the dynamically changed power limit throughout the time period. However, performance-policy does not consider power capping constraint during job scheduling decision. Therefore, the power limit is often violated in such policy: power usage often goes beyond the power limit. This is evident from Fig. 4(b).

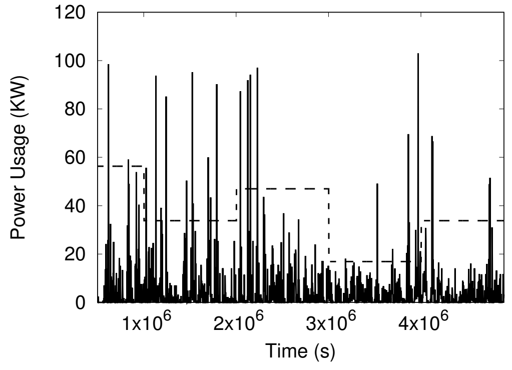
C. Energy versus Performance

We conducted second set of experiments to study the effect of the proposed scheduling algorithm for demand response on the job's energy consumption and execution time.

For this study, we vary the system size to be 128, 256, or 512 processors. We assume that a demand response event happens randomly during the system's operation and lasts for 25% of the entire duration of operation. When the demand response event happens, we expect the power limit of the system to drop to 80% from the peak power. We measure the job turnaround time to be between the time when the job is submitted and the time when the job has completed its execution. We report the average job turnaround time and the average energy among



(a) With Power Capping



(b) Without Power Capping

Fig. 4. Power usage over time with and without power capping.

all jobs. For the demand response algorithm, we also make a distinction of the average turnaround time and average energy between jobs that start inside or outside the demand response period. Finally, we compare the results of our demand response algorithm with those from using the performance-policy and the powersave-policy.

The top plot in Fig. 5 shows the average job turnaround time decreases for all scheduling policies when we increase the system size (from 128 to 256 to 512 processors). This is expected: the average job waiting time would decrease due to less contentions when more resources are available. The scheduler running performance-policy has the smallest job turnaround time among the three scheduling algorithms since it always uses the maximum CPU frequency to achieve best application runtime performance. Our demand response algorithm operates in the same way as the performance-policy during the normal operation time (non-DR event), but performs slightly worse than the performance-policy during the demand response time period (DR Event). The processors may be set to run jobs with less than the maximum frequency in order to achieve the optimal energy conservation during the demand response period.

The bottom plot of Fig. 5 shows the average energy consumption of the jobs. The per-job energy consumption is largely independent of the system size. The difference between performance-policy and powersave-policy is almost negligible,

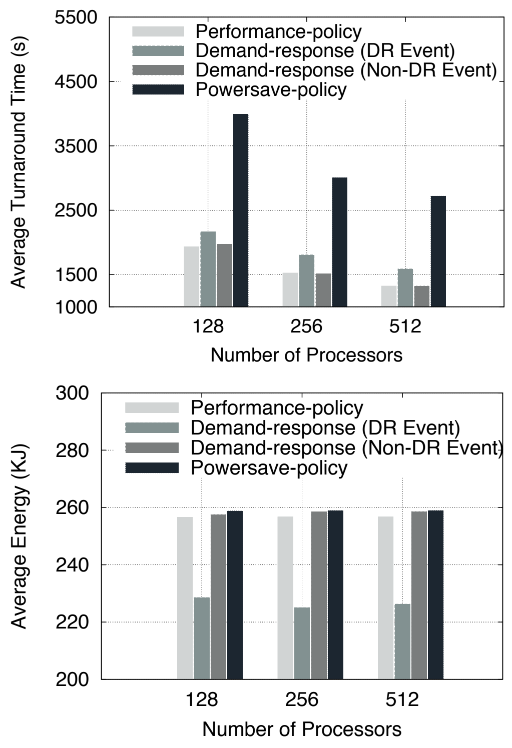


Fig. 5. Comparing performance and energy for different scheduling policies and with different system size.

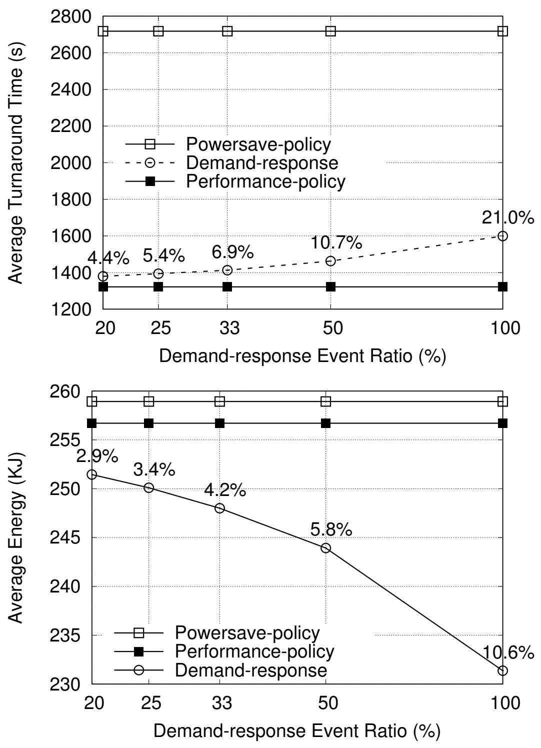


Fig. 6. Impact on the demand response event ratio.

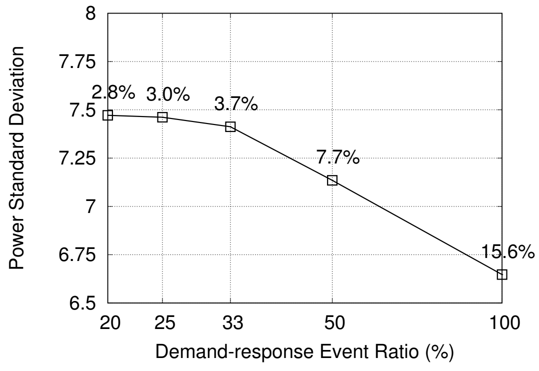


Fig. 7. Power stability during the demand response periods.

which is not unexpected. As shown previously in Fig. 1(c), the energy consumption of the four applications we choose for our study (e.g., Seissol) is at a similar level at both frequency extremes (at 1.2 GHz and 2.7 GHz). Considerable energy savings (around 15%) are achieved during the demand response period, when our algorithm finds the optimal CPU frequencies to achieve the best energy conservation for running the jobs.

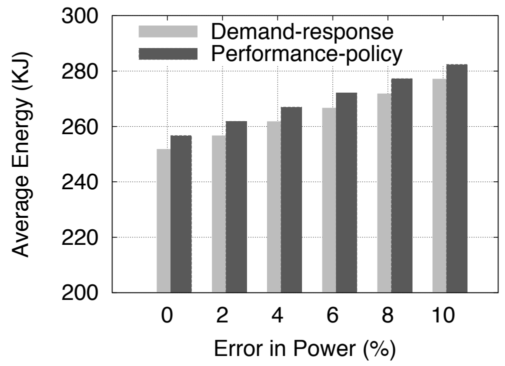
We observed that both average job turnaround time and average job energy consumption depend on the demand response event ratio (i.e., the percentage of time in the system operation that demand response happens). In the next experiment, we fixed the system size to be 512 processors and varied the demand response event ratio from 20% to 100%. Fig. 6 shows the results. The top plot shows that the average turnaround time increases only slightly for our scheduling algorithm when the demand response event lasts longer. Relative to the average job turnaround time achieved by performance-policy (which does not change with the demand response event ratio), we see that the demand response algorithm may introduce an increase between 4.4% and 21.0% in the average turnaround time.

The bottom plot shows that the average job energy consumption decreases with the longer demand response event, since our scheduling algorithm would be more likely to operate in the energy-conservation mode. Relative to the average energy achieved by powersave-policy, we see that the demand response algorithm can achieve energy savings from 2.9% to 10.6%.

D. Power Stability

An important aspect of the demand response program is that it is expected to help stabilize the power system during the demand response periods when the power grid may encounter instability, either due to the sudden rise in the demand or because of some emergency incidents. In this case, we would like to be able to minimize the fluctuations in the power demand of the HPC systems during the demand response events.

Fig. 7 shows the standard deviation of power usage of the system over time. In this experiment, we use the same simulation setup as in the previous experiment. We observe that the standard deviation of the power consumption decreases as the demand response event ratio increases. Relative to the standard deviation of the power usage under performance-



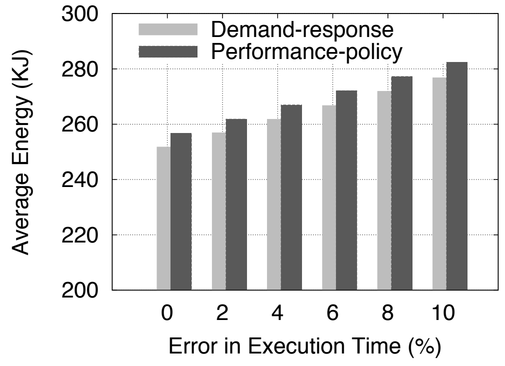


Fig. 8. The effect of prediction error in power and execution time.

policy, the demand response algorithm is shown to have achieved a reduction from 2.8% to 15.6%, and thus can contribute to the stability of the power system.

E. Sensitivity Study

In this subsection, we present the results from a sensitivity study based on inaccuracy for power and execution time prediction models. Fig. 8 shows the effect of prediction error on the average energy consumption.

In this experiment, we introduced the errors in the power and execution time of the jobs by adding a random error margin, from 0% to 10% with an increment of 2%. As can be seen from Fig. 8(a), with increase in error for power predictions, the average energy consumption for both demand-response and performance-policy increases (for as much as 10%). Our demand-response scheme achieves lower energy compared to the performance-policy scheme. Similar conclusions can be derived for introduced error in execution time prediction model; the demand-response scheme also achieves lower energy consumption consistently compared to the performance-policy scheme with increased prediction error.

VI. CONCLUSIONS

We propose a demand-response-aware job scheduling and resource provisioning algorithm for HPC systems. The job scheduling algorithm operates between the power-constrained energy-conservation mode (using DVFS) and the performance-conservation mode, depending on whether the system is in a demand response period or not. We developed a scheduler simulator to evaluate the effectiveness of our approach. The simulator has been validated by comparing with existing simulators using real-life workload traces. We performed evaluation studies and results have demonstrated that our proposed demand response method can achieve energy savings with only moderate impact to the application performance due to demand response. As such, we conclude that it is feasible for the HPC centers to participate in the demand response programs.

This current paper only provides a preliminary study in job scheduling and resource provisioning for demand response, since our proposed system is based on a simulator. However, note that we already considered modeling the most important factors: time, power cap, and energy. In the remainder of this section, we discuss the assumptions and limitations of our proposed approach, and point out some possible directions for future investigations.

Our job scheduler uses linear regression models for predicting the job's power consumption and execution time. Regression models require empirical measurements of each application's power consumption and execution time with different CPU frequencies. In practice, this may not be readily available *a priori*, especially for unknown applications, at the job's submission. More advanced models may be applied in this case. High-level application models, like the AEPCP model [20], which allows for ahead-of-time prediction and online adaptation with further execution of the application, can be useful. Inference and learning techniques (such as artificial neural networks) may also be employed to help in these scenarios [57], [58]. Our power prediction model does not consider the power consumption of memory, I/O, and other power sources, such as cooling, etc. These factors need to be considered for a more holistic demand response solution.

Our demand response job scheduler considers only processor-level frequency scaling. Some current HPC systems can support only machine-level DVFS. In future systems, frequency scaling may also be available more commonly at individual cores. We have not yet investigated these options. Moreover, frequency scaling is not the only control knob for energy saving and power capping. More adaptable power-aware scheduling policies and resource provisioning algorithms should be considered in addition to frequency scaling. There are also techniques that consider power consumption as a function of computation and communication at the system level and/or at the sub-job level. For example, Adagio [22] considers variations in the energy consumption during computation and communication phases of the applications. Also, SERA-IO [59] replaces the traditional I/O middleware with an energy-conscious mechanism that combines with DVFS for power-performance scheduling. These techniques can be considered to allow more effective demand response.

ACKNOWLEDGMENTS

This work is supported in part by NSF grants CNS-1563883, ACI-1550126, and CCF-1619236, the DOE LANL LDRD Program, and a USF/FC2 SEED grant.

References

[1] TOP500.org, "Top 500 list," https://www.top500.org/lists/2016/11/, 2016.

[2] D. G. Holmberg, S. T. Bushby, and D. B. Hardin, "Facility smart grid interface and a demand response conceptual model," NIST, Tech. Rep. NIST Technical Note 1832, 2014.

[3] Federal Energy Regulatory Commission, "Assessment of demand response and advanced metering," https://www.ferc.gov/legal/staff-reports/2016/DR-AM-Report2016.pdf, 2016.

[4] PJM Interconnect, "Demand response and why its important," https://www.pjm.com/~/media/markets-ops/dsr/end-use-customer-fact-sheet.ashx, 2014.

[5] The Energy Collective, "Demand response in the US electricity market," http://theenergycollective.com/rasika-athawale/195536/demand-response-us-electricity-market, 2013.

[6] J. McAnany, "2016 demand response operations markets activity report: April 2017," http://www.pjm.com/~/media/markets-ops/dsr/2016-demand-response-activity-report.ashx, 2017.

[7] H. Marianne and W. Eric, "Market data: Demand response. residential, commercial, and industrial demand response participation and sites, load curtailment, and spending: Global and regional market sizing and forecasts," 2013.

[8] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad, "Opportunities and challenges for data center demand response," in *Green Computing Conference (IGCC), 2014 International*. IEEE, 2014, pp. 1–10.

[9] E. Bilgin, M. C. Caramanis, I. C. Paschalidis, and C. G. Cassandras, "Provision of regulation service by smart buildings," *IEEE Transactions on Smart Grid*, vol. 7, no. 3, pp. 1683–1693, 2016.

[10] P. Fox-Penner, "Why apple is getting into the energy business," https://hbr.org/2016/11/why-apple-is-getting-into-the-energy-business, 2016.

[11] Mission Critical Power, "Equinix in r&d phase of demand response experiments," https://missioncriticalpower.uk/equinix-tests-demand-response/, 2015.

[12] G. Ghatikar, V. Ganti, N. Matson, and M. A. Piette, "Demand response opportunities and enabling technologies for data centers: Findings from field studies," 2014.

[13] N. Bates, G. Ghatikar, G. Abdulla, G. A. Koenig, S. Bhalachandra, M. Sheikhalishahi, T. Patki, B. Rountree, and S. Poole, "Electrical grid and supercomputing centers: an investigative analysis of emerging opportunities and challenges," *Informatik-Spektrum*, vol. 38, no. 2, pp. 111–127, 2015.

[14] T. Patki, N. Bates, G. Ghatikar, A. Clausen, S. Klingert, G. Abdulla, and M. Sheikhalishahi, "Supercomputing centers and electricity service providers: a geographically distributed perspective on demand management in Europe and the United States," in *International Conference on High Performance Computing*. Springer, 2016, pp. 243–260.

[15] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned HPC data centers under a strict power budget," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 807–818.

[16] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Parallel job scheduling for power constrained HPC systems," *Parallel Computing*, vol. 38, no. 12, pp. 615–630, 2012.

[17] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron, "CPU MISER: A performance-directed, run-time system for power-aware clusters," in *Parallel Processing, 2007. ICPP 2007. International Conference on*. IEEE, 2007, pp. 18–18.

[18] W. Bao, C. Hong, S. Chunduri, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, and P. Sadayappan, "Static and dynamic frequency scaling on multicore CPUs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 4, p. 51, 2016.

[19] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[20] H. Shoukourian, T. Wilde, A. Auweter, and A. Bode, "Predicting the energy and power consumption of strong and weak scaling HPC applications," *Supercomputing frontiers and innovations*, vol. 1, no. 2, pp. 20–41, 2014.

[21] X. Wu, V. Taylor, J. Cook, and P. Mucci, "Using performance-power modeling to improve energy efficiency of HPC applications," *IEEE Computer*, vol. 49, no. 10, pp. 20–29, 2016.

[22] B. Rountree, D. K. Lownenthal, B. R. De Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making dvs practical for complex HPC applications," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 460–469.

[23] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 10–pp.

[24] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 60.

[25] T. Cao, Y. He, and M. Kondo, "Demand-aware power management for power-constrained HPC systems," in *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE, 2016, pp. 21–31.

[26] M. E. Haque, I. Goiri, R. Bianchini, and T. D. Nguyen, "Greenpar: Scheduling parallel high performance applications in green datacenters," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 217–227.

[27] F. Yang and A. A. Chien, "Zccloud: Exploring wasted green power for high-performance computing," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 1051–1060.

[28] H. Chen, M. C. Caramanis, and A. K. Coskun, "Reducing the data center electricity costs through participation in smart grid programs," in *Green Computing Conference (IGCC), 2014 International*. IEEE, 2014, pp. 1–10.

[29] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, and T. Wilde, "A case study of energy aware scheduling on SuperMUC," in *International Supercomputing Conference*. Springer, 2014, pp. 394–409.

[30] B. Austin and N. J. Wright, "Measurement and interpretation of microbenchmark and application energy use on the Cray XC30," in *Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing*. IEEE Press, 2014, pp. 51–59.

[31] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.

[32] V. Springel, "The cosmological simulation code gadget-2," *Monthly notices of the royal astronomical society*, vol. 364, no. 4, pp. 1105–1134, 2005.

[33] M. Käser, J. d. a. Puente, C. Castro, V. Hermann, and M. Dumbser, "Seismic wave field modelling using high performance computing," in *SEG Technical Program Expanded Abstracts 2008*. Society of Exploration Geophysicists, 2008, pp. 2884–2888.

[34] C. Feichtinger, S. Donath, H. Köstler, J. Götz, and U. Rüde, "Walberla: HPC software design for computational engineering simulations," *Journal of Computational Science*, vol. 2, no. 2, pp. 105–112, 2011.

[35] J. D. McCalpin, "Stream benchmark," *URL: http://www. cs. virginia. edu/stream/stream2*, 2002.

[36] T. Patki, D. K. Lowenthal, B. L. Rountree, M. Schulz, and B. R. d. Supinski, "Economic viability of hardware overprovisioning in power-constrained high performance computing," in *2016 4th International Workshop on Energy Efficient Supercomputing (E2SC)*, Nov 2016, pp. 8–15.

[37] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *International Conference on Parallel Processing and Applied Mathematics*. Springer Berlin Heidelberg, 2013, pp. 793–803.

[38] M. J. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical analysis*. Springer, 1978, pp. 144–157.

[39] Parallel Systems Lab, "Python scheduler simulator," https://code.google.com/archive/p/pyss/, 2010.

[40] Y. Georgiou, D. Glesser, K. Rzadca, and D. Trystram, "A scheduler-level incentive mechanism for energy efficiency in HPC," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 617–626.

[41] F. Liu and J. B. Weissman, "Elastic job bundling: An adaptive resource request strategy for large-scale parallel applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 33.

[42] "CQsim," http://bluesky.cs.iit.edu/cqsim/, 2012.
[43] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on Blue Gene/P systems," in *2009 IEEE International Conference on Cluster Computing and Workshops*, Aug 2009, pp. 1–10.
[44] W. Tang, D. Ren, Z. Lan, and N. Desai, "Adaptive metric-aware job scheduling for production supercomputers," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*. IEEE, 2012, pp. 107–115.
[45] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A simulator for large-scale parallel computer architectures," *Technology Integration Advancements in Distributed Systems and Computing*, vol. 179, 2012.
[46] N. Jain, A. Bhatele, S. White, T. Gamblin, and L. V. Kale, "Evaluating HPC networks via simulation of parallel workloads," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 14:1–14:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3014923
[47] N. Santhi, S. Eidenzenz, and J. Liu, "The Simian concept: parallel discrete event simulation with interpreted languages," in *Proceedings of the 2015 Winter Simulation Conference*, L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, Eds., 2015.
[48] G. Chapuis, S. Eidenbenz, N. Santhi, and E. J. Park, "Simian integrated framework for parallel discrete event simulation on GPUs," in *2015 Winter Simulation Conference (WSC)*, Dec 2015, pp. 1127–1138.
[49] G. Chapuis, D. Nicholaeff, S. Eidenbenz, and R. S. Pavel, "Predicting performance of smoothed particle hydrodynamics codes at large scales," in *Winter Simulation Conference (WSC), 2016*. IEEE, 2016, pp. 1825–1835.
[50] K. Ahmed, M. Obaida, J. Liu, S. Eidenbenz, N. Santhi, and G. Chapuis, "An integrated interconnection network model for large-scale performance prediction," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 177–187.
[51] K. Ahmed, J. Liu, S. Eidenbenz, and J. Zerr, "Scalable interconnection network models for rapid performance prediction of HPC applications," in *2016 IEEE 18th International Conference on High Performance Computing and Communications (HPCC)*, Dec 2016, pp. 1069–1078.
[52] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Parallel Processing Workshops, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 514–519.
[53] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967 – 2982, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001154
[54] K. Deng, J. Song, K. Ren, and A. Iosup, "Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 55.
[55] K. Kianfar, G. Moslehi, and R. Yahyapour, "A novel metaheuristic algorithm and utility function for qos based scheduling in user-centric grid systems," *The Journal of Supercomputing*, vol. 71, no. 3, pp. 1143–1162, 2015.
[56] Arch Linux, "CPU frequency scaling," https://wiki.archlinux.org/index.php/CPU_frequency_scaling, 2017.
[57] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," *Euro-Par 2005 Parallel Processing*, pp. 627–628, 2005.
[58] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2007, pp. 249–258.
[59] R. Ge, X. Feng, and X. H. Sun, "SERA-IO: Integrating energy consciousness into parallel i/o middleware," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, May 2012, pp. 204–211.