# A Data Driven Scheduling Approach for Power Management on HPC Systems

Sean Wallace*, Xu Yang*, Venkatram Vishwanath†, William E. Allcock†,
Susan Coghlan†, Michael E. Papka†‡, Zhiling Lan*
*Illinois Institute of Technology, Chicago, IL, USA
†Argonne National Laboratory, Argonne, IL, USA
‡Northern Illinois University, DeKalb, IL, USA
swallac6@iit.edu, xyang56@hawk.iit.edu, venkat@anl.gov, allcock@anl.gov,
smc@anl.gov, papka@anl.gov, lan@iit.edu

***Abstract*—Modern schedulers running on HPC systems traditionally consider the number of resources and the time requested for each job that is to be executed when making scheduling decisions. Until recently this has been sufficient, however as systems get larger, other metrics like power consumption become necessary to ensure system stability.**

**In this paper, we propose a data driven scheduling approach for controlling the power consumption of the entire system under any user defined budget. Here, "data driven" means that our approach actively observes, analyzes, and assesses power behaviors of the system and user jobs to guide scheduling decisions for power management. This design is based on the key observation that HPC jobs have distinct power profiles. Our work contains an empirical analysis of workload power characteristics on a production system, dynamic learner to estimate the job power profile for scheduling, and an online power-aware scheduler for managing the overall system power. Using real workload traces, we demonstrate that our design effectively controls system power consumption while minimizing the impact on system utilization.**

## I. Introduction

There has been significant research into the topic of energy efficiency. Existing studies include energy-efficient or energy proportional hardware, dynamic voltage and frequency scaling (DVFS), shutting down unused or unnecessary hardware components when system utilization is low, thermal management, and power aware scheduling of special applications [1], [2], [3], [4], [5], [6].

Complementing the above studies, this work proposes a data driven scheduling design for controlling the power consumption of the entire system under any user defined cap. The design combines continuous monitoring of power behaviors of the system and user jobs, dynamic learning and prediction of job power profiles, and timely power-aware scheduling to realize power management. The design is *data driven* as it emphasizes the dynamic monitoring and learning of power data to guide decision making. Unlike the existing studies targeting moldable jobs (i.e., the jobs whose running sizes are dynamically decided by the scheduler at job allocation time [5], [6]), this work doesn't put any extra requirement on workload and *is applicable to general HPC jobs*. Moreover, HPC systems require a significant capital investment, hence making efficient use of expensive resources is of paramount importance [7]. Distinguishing from the existing studies lowering system utilization for power management [4], [1], this work *enforces a power cap while maintaining high system utilization of HPC systems*.

Our design leverages an important observation — *HPC jobs have distinct power profiles and these profiles can vary significantly from job to job with the difference being as high as* 4.4 *times*. Built on this observation, we present three new techniques enabled by our design. First is *a dynamic learner*. Unlike public clouds and data centers, HPC resources are provided to multiple users through allocations. Each allocation is associated with a group of users working on a common project. As such, HPC jobs are typically repetitive. As jobs enter, execute, and exit the system, our dynamic learner records power data associated with each project and job, and estimates job power behavior by applying inferential statistics on online and offline power data of projects and jobs. Later, we will show that our dynamic learner is highly accurate. Second is *a window-based scheduling mechanism*. In an effort to mitigate the impact to system utilization, rather than assigning user jobs to available resources in a one-by-one manner as adopted by conventional job scheduling, our design makes job allocation decisions by checking a window of jobs. Finally, *a 0-1 knapsack based policy* is developed for selecting jobs in the window, with the objective of controlling the overall power consumption while maximizing resource utilization.

We evaluate our design via extensive trace-based simulations with one full year of production traces collected from the 48-rack IBM Blue Gene/Q system, Mira, at Argonne Leadership Computing Facility [8]. The empirical evaluation indicates that our dynamic learner is capable of delivering and maintaining a highly accurate estimate of job power profiles with a short period of training (e.g., as short as 26 days). Our trace-based experiments also indicate that the proposed power-aware scheduling can effectively control the system wide power consumption even when the power budget varies dynamically. We find that when the power cap is set to 83% of the maximum, the relative degradation caused by our design is less than 1%. Interestingly, we notice that our power aware scheduling is able to perform better than an unlimited power

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00 ©2016 IEEE

cap in four months out of the year's worth of data with regards to wait times.

The organization of the rest of the paper is a follows. We begin by looking at related work in Section II. Section III presents an empirical analysis of job power profiles present in the our newly acquired workload data. Section IV gives an overview of our design and discusses some of the challenges associated with power aware scheduling. Section V describes the learning process to obtain job and group power profiles for power-aware scheduling. Section VI describes scheduling policies. Section VII presents our methodology for evaluation, followed by the experimental results in Section VIII. Finally, our conclusions are discussed in Section IX.

## II. Related Work

The field of energy efficient computing is a diverse and popular one. We discuss some of the more closely related studies and point out the key differences.

As the processor accounts for a significant portion of total power consumption (around 50% when under load [4]), DVFS is a technique which has been widely presented for use in controlling CPU power consumption [9]. The idea is very simple. By running processors at a lower frequency/voltage energy, is saved potentially at the expense of increased job execution times. DVFS is typically applied at a period of low system activity to meet user service level agreements (SLAs). Additional research in this field can be found in [1], [2], [3].

Taking DVFS one step further, a number of studies have looked at shutting down or suspending all idle nodes during low system utilization [4], [1]. The goal is to optimize the number of active nodes such that it exactly meets the demand of incoming application requests. Since this approach is highly dependent on system workload, the key challenge with this approach is determining when a node or its components should be shut down.

Thermal management techniques are another approach frequently found in the literature. The reasoning is that higher temperatures have a large impact on the system reliability and can also increase cooling costs. By using thermal management, system workload is adjusted according to a predefined temperature threshold: if the temperature on a given node rises beyond that threshold, its workload is reduced. The disadvantages of thermal management are delayed response, high risk of overheating, and excessive cooling and recursive cycling [10].

Sarood et al. [11] leveraged the notion that fault rates are known to double for every 10°C rise in core temperature to experimentally demonstrate the potential of restraining core temperatures and load balancing. This was done to improve the reliability of parallel machines and reduce the total execution time of applications. Results showed they were able to improve the reliability of the machine by a factor of 2.3 and reduce the execution time by 12%.

Many data centers employ power capping or power budgeting to reduce the total power consumption. Etinski et al. [3] proposed a parallel job scheduling policy based on integer linear programming under a given power profile. The implementation of this work was realized by utilizing DVFS at the node level to keep power consumption under a given power cap.

This work differs from our previous work [12] in several critical ways. First of all, our prior study focuses on reducing the electricity bill of HPC systems, whereas this work intends to dynamically control the overall power consumption of HPC systems within a user-provided power budget. Second, our previous work made an implicit assumption that job power profiles were known a priori when making scheduling decisions. This work provides a more practical solution that dynamically learns job power profiles by using historical data and online profiling of power data.

## III. Workload Power Analysis

In this work we collect and analyze a year's worth of workload trace and power data in 2014 from the 48-rack IBM Blue Gene/Q machine Mira at Argonne. A rack of BG/Q consists of two midplanes with each midplane containing 16 node boards. Each node board holds 32 compute cards, for a total of 1,204 nodes per rack. Each compute card has a single 18-core PowerAC A2 processor [13] (16 cores for applications, one core for system software, and one core inactive) with four hardware threads per core, and DDR3 memory. Mira consists of a total of 49,152 compute nodes and 786,432 cores, with the peak power consumption of 4.8MW. It is ranked 5th on the November 2015 Top500 list [14].

Blue Gene systems have environmental monitoring capabilities that periodically sample and gather environmental data from various senses and store this information together with the timestamp and location information in an IBM DB2 relational database, commonly referred to as the environmental database. In addition to this environmental data collection, there are other mechanisms which collect data from the scheduler about jobs running on the system such as job start, end, queue time, etc. Abstractly, the system which governs the collection of all this data is referred to as Core Monitoring and Control System (CMCS) [15].

By pulling data from the CMCS system and analyzing it, we are able to make observations about jobs running on the system. We found out that *not only do jobs have distinct power profiles, so do the projects they belong to*. Table I provides the detailed information about all of the jobs in the log. This data shows that job power profiles per rack differ greatly with the difference being as high as 4.4 times.

Figure 1 provides a graphical glimpse into this observation featuring job power per rack of a random sampling of six projects. From the box plot it's clear that just about every statistical aspect of power consumption varies from project to project. Conversely, Figure 2 displays a collection of users belonging to the same project and shows that while the jobs that users run within a project do vary slightly, overall they offer a pretty good representation of the power consumption of the project on the whole. This new observation is a step

TABLE I: Statistics of job power profiles in Kilowatts per rack on Mira

| | |
|---|---|
| Minimum | 36.48 |
| Mean | 65.76 |
| Maximum | 160.60 |
| Percentile 05 | 56.60 |
| Percentile 25 | 61.20 |
| Percentile 75 | 71.03 |
| Percentile 95 | 74.57 |
| Percentile 99 | 77.99 |
| Standard Deviation | 6.18 |

forward from our prior study [16]. It enables us to predict a job power profile based on its project information.

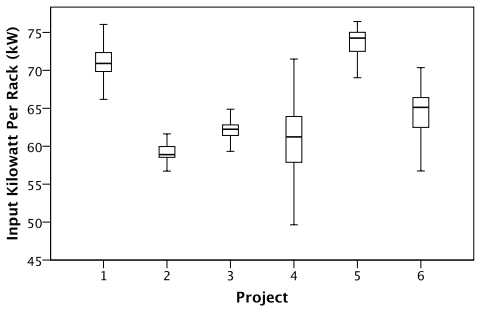



Fig. 1: Box plot of a random sample of projects and their power consumption at the rack range. It clearly shows, different projects have very different medians, maximums, and minimums.

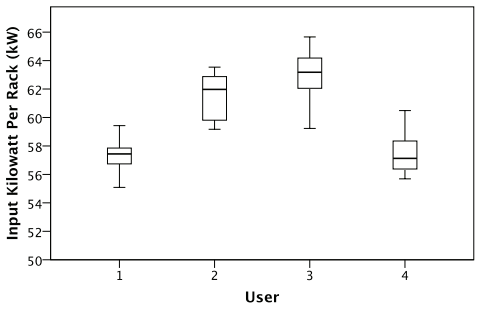



Fig. 2: Box plot of power consumption by users in a random project.

## IV. Design Overview

This section provides an overview of our design, along with a motivational example. Our design leverages the key observation presented in the previous section.

As shown in Figure 3 (top), HPC systems typically utilize a batch scheduler to allocate jobs to resources. Users interact with the batch scheduler by submitting their job into the queue or queues (which may have different priorities) and subsequently wait for the requested resources to become available for the requested amount of time. Jobs are selected by the scheduler in a one-by-one fashion from the head of the waiting queue. The exact methodology by which jobs are selected out of the queue varies between schedulers; one commonly used policy is FCFS (first-come, first-serve) with EASY-backfilling [17]. On Mira, a policy called WFP with EASY-backfilling is adopted for batch scheduling [18]. WFP favors large and old jobs, adjusting their priorities based on the ratio of their wait times to their requested runtimes. The common goal of batch scheduling is to increase resource utilization while increasing user's satisfaction.

Similar to the conventional batch scheduling, our design intends to meet the same scheduling objective under a constraint — a power budget for the entire system. *There are a number of technical challenges in such a power aware scheduling at the system level*. The first hurdle is the *acquisition of accurate power data*. Fortunately, hardware manufacturers have begun deploying various sensors on HPC systems to monitor power usage of their components, and a number of software libraries have been developed for users to access these data [19], [16], [20], [21]. For instance, there are two power monitoring capabilities deployed on Mira: one is an environmental database and the other is a user-level power profiling library called MonEQ which is built on the vendor-supplied application programming interfaces [16]. They provide information about the power consumption at two different scales: the data collected by the control system is coarse grained at a sampling rate of every 4-6 minutes, while the user-level power profiling library MonEQ allows users to profile jobs every 560ms. While our design will work with any power monitoring facility, we advocate the use of fine-grained power monitoring, if available. The second hurdle is to estimate job power profile *before* the job is actually run on the system and to capture job power variation *during* the job execution. The third hurdle is to *dynamically make an optimized scheduling decision that meets power budget constraint*. Moreover, the decision making has to adapt to variation in workload power profile and change in the available power budget.

Figure 3 (bottom) depicts a high level overview of our design. With our approach, we maintain the same job ordering in the waiting queue but add two critical elements to address the aforementioned challenges. The first is a *dynamic learner* which takes power data from a power monitoring facility to estimate job power profiles. At each scheduling instance, the learner has two tasks: one is to estimate power profiles of the jobs in the queue, and the other is to calculate the available power budget for incoming jobs by estimating the power requirements of the running jobs on the system. As demonstrated in Section III, not only do HPC jobs have distinct power profiles, but groups of jobs also have distinct power profiles. Leveraging this observation, our dynamic learner actively records power data collected by the power monitoring tool. Any job that runs on the system will be profiled from start to finish and the sampling data will be stored and used to both better understand this particular job as well as the jobs in the project it belongs to. In this sense, our design "learns" over time and the more jobs that are run on the system the better it gets. Furthermore, because of the continuous streaming nature of the power data, our design can adapt to a job's power change

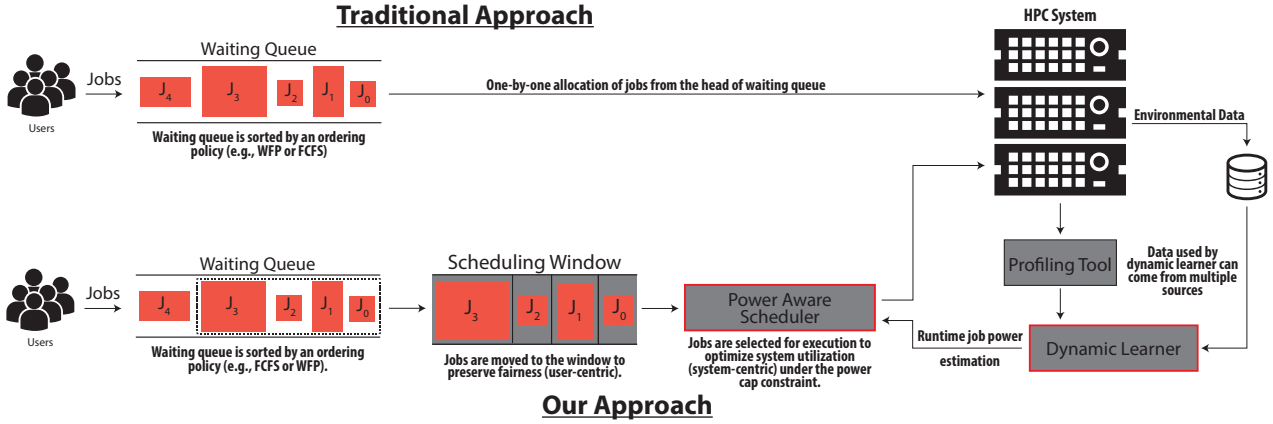



Fig. 3: Comparison of the traditional batch scheduling (top half) and our power-aware scheduling (bottom half).

on-the-fly. The detailed design of the dynamic learner will be presented in Section V.

The second key component is the *power aware scheduler* which selects jobs in the waiting queue for execution to meet the scheduling goal under the power constraint. We propose a window-based optimization method. In contrast to the conventional scheduling approach that allocates jobs in a one-by-one manner, our design examines a window of jobs in the queue for decision making. The window is adopted to preserve job fairness [12]. Furthermore, a 0-1 knapsack problem is formulated to describe the power aware scheduling problem. It enables us to select jobs in the window to meet the scheduling goal under the power budget constraint. The detailed design of the job scheduler will be presented in Section VI.

### A. A Motivational Example

In order to better illustrate the key idea of our design, suppose four jobs $J_0, J_1, J_2, J_3$ are currently in the queue and ready to be allocated onto the system. The parameters associated with each job are given as:

| Job | Job Size (Racks) | Power Profile (kW/rack) |
|---|---|---|
| $J_0$ | 3 | 60 |
| $J_1$ | 1 | 50 |
| $J_2$ | 5 | 30 |
| $J_3$ | 4 | 40 |

Now, suppose these jobs are to be allocated onto a 6-rack system with a total power cap of 230 kW. Using the conventional FCFS policy, it will always select $< J_0, J_1 >$ as the total power consumption of $J_0 + J_1 = (3 \cdot 60) + (1 \cdot 50) = 230$ kW, despite the fact there are 2 unused racks. Our scheduling mechanism on the other hand will select $< J_1, J_2 >$ as that is the combination of jobs which maximizes the utilization while staying under the power cap.

## V. Dynamic Learner

The dynamic learner constantly monitors power consumption of user jobs, and is continually invoked to estimate job power profiles. The jobs include those in the waiting queue as well as those running on the system. For each job, considering that a job's power consumption may have temporal variation, our learner estimates the *high* power requirement of the job over a window.

For a job in the waiting queue, there are two possibilities when determining its power profile: (1) there is no power information for this job, or (2) there are previous power data from this job. For case 1, leveraging the key observation presented in Section III, our learner works as follows: if the group's power profile is known, we assume the group profile is representative of this job and use the group profile for the job; otherwise, we assume the maximum power for the job (here, the maximum is defined as the peak power value of the underlying device). For case 2, we use the job's previous profile as its current power profile.

For a job running on the system, as it executes, more power data are generated, collected, and statistically compared to its old values. Our learner constantly updates power profiles of all the jobs and the groups by applying a *two-sample T-test* for runtime power learning and adaptation. The T-test is a commonly used method for assessing whether the means of two populations are statistically different from each other [22]. Mathematically, assume that $Y_1$ and $Y_2$ are two lists of power measurements collected, $N_1$ and $N_2$ are the sample sizes, $\bar{Y}_1$ and $\bar{Y}_2$ are the sample means, and $s_p$ is the sample variances (equal variances assumed). The T-value measuring the difference between group means is calculated as $T = \frac{\bar{Y}_1 - \bar{Y}_2}{s_p \sqrt{\frac{1}{N_1} + \frac{1}{N_2}}}$ where $s_p^2 = \frac{(N_1-1)s_1^2 + (N_2-1)s_2^2}{N_1+N_2-2}$.

The significance level $\alpha$ is used to reject the null hypothesis that two means are equal if $|T| > t_{1-\alpha/2,v}$, where $t_{1-\alpha/2,v}$ is the critical value of the t distribution with $v = N_1 + N_2 - 2$ degrees of freedom. In our tests, the significance level is set to 99%.

As jobs execute, more power data are generated and collected and statistically compared to any old values. The comparison of new data to old can have two outcomes: 1) there isn't a statistically significant difference between the two samples so the power profile does not change or, 2) there is a statistically significant difference between the two samples

and the power profile needs to change. In the case of the latter, there are two possible values for the profile to change to. First, the new data is compared to the group's to see if there is a significant difference between it and the group's on the whole. If there isn't a significant difference between a job's new data and the group's data, the power profile for the job is updated to the group's power profile. If there is a significant difference between the job's data and the group's, neither is representative of this job so the maximum possible must then be assumed.

To safe guard against such an outcome, there are a number of precautions taken. First, no job can even begin to have a significant profile until there are a satisfactory number of samples collected for it. Since our simulation is based on control system data which is collected infrequently, we have deduced that a minimum of 20 sample points are necessary before any job can even have a significance test run on its data.

Based on our experience with this data, after 20 samples are collected it takes a *very* drastic change in application behavior to violate a previously computed profile and even when violation does occur, it's maximally bounded by the system's architecture. To better put this problem into perspective, 20 sample points equates to almost 80 minutes of runtime. Consequently, a job would have to do a relatively mundane task for 80 minutes only to ramp up computation thereafter, not to mention completely going against the group data in the first place. Even those applications which have stages of data generation where power consumption is relatively low and switch to intense computation where power consumption is relatively high, they never do this in intervals of 80 minutes.

## VI. Power Aware Scheduler

Our power aware scheduler is designed around two key components: a window-based mechanism for improving scheduling efficiency without violating job ordering in the waiting queue, and a power aware scheduling policy for meeting the scheduling objective under the power budget constraint.

In this work, we make a distinction between the ordering of the waiting queue and the selection of jobs from a window of jobs in this queue to the system. In a traditional system, jobs are allocated directly onto the system in a *one-by-one manner* from the head of the waiting queue where the job ordering is managed by a policy like FCFS or WFP. Our implementation doesn't modify this ordering, rather it selects some of these jobs from the head of the waiting queue into a *window*. Once in the window, our scheduler then treats this window of jobs as an instance of the 0-1 Knapsack problem. The "solution" to this Knapsack problem are the jobs to be dispatched to the system for running, with the purpose of optimizing certain scheduling objective under a power budget. This process is fully explained in the following subsections.

### A. Window-Based Scheduling

One of the most challenging aspects of any scheduling algorithm is striking the balance between job fairness (e.g., first-come, first-serve or large-job-first) and scheduling performance (e.g., job wait time, system utilization, etc.). In this work while we are still trying to maximize fairness, rather than allocating jobs one-by-one from the front of the wait queue, we use a window-based mechanism which allocates a window of jobs. The jobs that are selected into the window are done so using certain *user-centric* metrics such as fairness. The selection of jobs from the window to system resources is done using *system-centric* metrics such as utilization and power consumption. By maintaining this window and always drawing jobs directly from it, we are able to best balance fairness and performance. Regardless of the scheduling policy used to select jobs from the window, the window is always a fixed size collection of jobs at the head of the queue. When a user submits a job it first enters the queue and subsequently the window when there is enough room. In this way, even a scheduling policy which does not allocate jobs in the absolute order they are received can still guarantee there is job fairness.

### B. Scheduling Policy

Our scheduling policy selects jobs from the scheduling window to optimize a scheduling objective. As our primary objective is to select those jobs from the window such that utilization is maximal, we formulate a 0-1 Knapsack problem based policy where both the system utilization and power consumption are taken into consideration for scheduling decisions. We now explain how the 0-1 Knapsack based policy works to maximize utilization while staying under a given power cap.

Suppose at the current scheduling instance, there are $N$ nodes in the system with $N_{used}$ nodes being used by the running jobs, and the power cap is set at $PB$ with $P_{used}$ representing the power used by the running jobs. Let $w$ represent the size of the scheduling window, and there are a set of jobs in the window: job $J_i$ requiring $n_i$ nodes and $p_i$ power profile. The scheduling problem is then formalized as follows: to select a subset of jobs $\{J_i | 1 \leq i \leq k\}$ such that: $\sum_{1 \leq i \leq k} p_i \leq PB - P_{used}$ with the objective to maximize: $\sum_{i \leq i \leq k} n_i \leq N - N_{used}$.

The above problem can be formalized as a 0-1 Knapsack problem. By setting the sum of the individual power profiles as the knapsack's weight and the sum of the associated job's node requirements as the value, our scheduling problem can be transformed into an instance of the 0-1 Knapsack problem. Formally stated, the objective is to find a binary vector $X = \{x_i | 1 \leq i \leq k\}$ such that:

$$\begin{aligned} &\text{maximize} \sum_{1 \leq i \leq k} x_i \cdot n_i \leq N - N_{used}, \quad x_i = 0 \text{ or } 1 \\ &\text{subject to} \sum_{1 \leq i \leq k} x_i \cdot p_i \leq PB - P_{used} \end{aligned} \tag{1}$$

The values of $P_{used}$ and $p_i$ are calculated by the learner presented in Section V. This 0-1 Knapsack problem can be solved in pseudo-polynomial time by using dynamic programming [23]. To avoid redundant computation, the memoization technique can be used by building a 2D table $T$, where $T[k, w]$

denotes the maximum gain value that can be achieved by scheduling jobs $\{j_i | 1 \leq i \leq k\}$ which require no more than $PB - P_{used}$ power and no more than the number of remaining free nodes $N - N_{used}$. $T[k, w]$ then has the following recursive feature:

$$T[k,w] = \begin{cases} 0 & kw = 0 \\ T[k-1,w] & w_i > w \\ \max(T[k-1,w], \mathcal{W}) & w_i \leq w \end{cases} \quad (2)$$

where:

$$\mathcal{W} = \begin{cases} -\infty & N_{pot} > N_{free} \\ N_{pot} & \text{otherwise} \end{cases}$$

and $N_{pot} = n_i + T[k-1, w-w_i]$. The solution to the problem after computation is then $T[J, N_{tot}]$ and its corresponding binary vector $X$ determines the selection of jobs scheduled to be run. The complexity of Equation 2 is $\mathrm{O}(J \cdot N_{tot})$.

## VII. Evaluation Methodology

To validate our design, we conduct a series of experiments using trace-based simulations.

### A. StreamQSim: Trace-Based Scheduling Simulation

We developed a simulator named *StreamQSim* to evaluate our design. StreamQSim is an extension of the open-source simulator CQSim [24] with the primary difference being that unlike CQSim which requires all data for simulation be provided at execution time in a flat-file format, StreamQSim "streams" data as it is necessary. This additional development was vital as the overall size of the data we had to process in order to simulate our job traces was far too large to fit into main memory and impractical to process as a flat file. For this reason, it aggressively prefetches relevant power data from a database instance into memory as it is necessary and removes it when it is not. In this way, no more data than what is absolutely vital to make a scheduling decision with the current subset of jobs is contained in memory.

StreamQSim uses data obtained entirely from a database we created as a result of analysis of the control system data. This data includes the job events such as job submission, job start, job end, etc. Another primary difference from CQSim, StreamQSim has to work in "real time" fetching data just as if it were actually a scheduler making decisions for a production system at a given time. This means that StreamQSim cannot skip from event to event (such as from job start to job end) as it must gather data throughout execution of jobs as this could have an impact on scheduling decisions. As such, it works incrementally–by a given interval (scheduling quantum)–to schedule jobs onto the system. StreamQSim is available at [25].

### B. System Traces

System traces of production systems are not easily accessible. In order to assess power aware scheduling, we need both workload traces and corresponding power data for the traces. While many production systems collect and archive their workload traces [26], far fewer systems gather and store their power data. Making matters worse, even if workload traces and power data are obtainable for a given system, there needs to be some way to link the trace to the power data. In this study, we have collected a year long system trace from Mira, including both workload (i.e., jobs) log and the corresponding power data (i.e., watts in use for the whole system broken down by component), for the experiments.

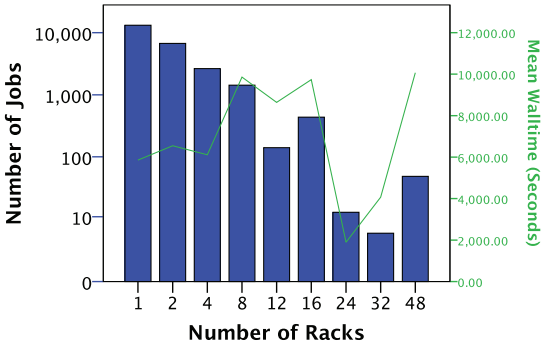



Fig. 4: Distribution of job sizes (log scale) and average runtimes at each scale.

There are 24,565 valid jobs in the job log, and Figure 4 summarizes the distributions of job sizes and runtimes. As shown in Section III, the corresponding power data is obtained from the environmental log on Mira. Since the job log and the power data are collected by different tools, a critical challenge is the linking of power data to jobs. The process of accurately assigning power data to jobs is a multi-step process.

To get power data for a job, one must join the Cobalt (the job scheduler on Mira [27]) database to the control system database to find the control system job ID's, use these control system job IDs to gather the complete list of compute nodes the job is actually using, and finally join this list of nodes to the control system's power database bounded by the time during which the job was actually run. Since there is no notion of sharing of hardware on Mira, it is guaranteed that this power data is for a single job. In this way, given just a Cobalt job ID, one can acquire the complete list of power data for that job.

### C. Evaluation Metrics

In this study, we evaluate our design in three aspects. First, we quantify **learning accuracy** achieved by the dynamic learner. Next, we define a metric **capping success rate (CSR)** measure the proportion of scheduling intervals which the total power consumption is controlled within the cap. Specifically, CSR is defined as the ratio of the number of scheduling intervals within the power cap to the total number of scheduling intervals. A score of 1.0 indicates an ideal case

where the power cap is never violated. Finally, we use two well established metrics for comparison analysis of job scheduling: (1) **resource utilization rate** which measures the ratio of the node-hours that are used for useful computation to the elapsed system node-hours, and (2) **average job wait time** which measures the average time between the moment a job is submitted to the queue to the moment it is actually running on the system across all the jobs.

## VIII. Experimental Results

### A. Learning Accuracy

One of the interesting questions that can be asked of a policy such as ours is, *how does the ratio of known power profiles to unknown power profiles (referred to here as "learning rate") change over time?* As mentioned previously, jobs submitted to HPC systems are typically repetitious and, at least within the project, tend to use similar code bases. Consequently, our policy, which starts out not knowing any profiles, is capable of achieving and sustaining a very high learning rate very quickly. Figure 5 shows our learning rate over the course of the year of simulation. After just 26 days of execution our policy has learned 94% of the power profiles.

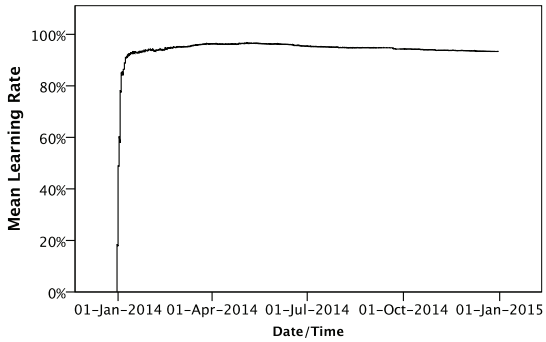



Fig. 5: Learning rate over time. 94% after just 26 days of execution.

The learning rate is entirely proportional to the diversity rate in jobs. As time progresses, it's reasonable to expect that the workload on the system would change or evolve. This happens because new projects are added thereby introducing entirely unknown power profiles to the mix, or, existing projects make modifications to their codes which alters their power profiles. Regardless of the reason, our dynamic learning is capable of maintaining as good of a learning rate as the diversity of the workload on the system allows it to be. For other systems where repeat job running is rare, our learning scheme would certainly struggle to achieve a high learning rate. As mentioned earlier, resource access in HPC is provided through allocation [28], [29]. Each allocation is associated with a group of users working on a common project for a fixed period of time. As such, jobs are highly repetitive and have distinct power profiles. Hence our learner is able to perform exceedingly well.

### B. Capping Success Rate

The peak power consumption of Mira is 4,800 kW. We conduct two sets of experiments to evaluate whether our design is capable of controlling the system-wide power consumption. The first set of experiments is to assess whether our design can control the overall power consumption under a fixed cap. Figure 6 shows the power consumption of the workload with the power cap set to 3,000 kW or about 62.5% of the peak power. There are two curves in the figure: one without power management (light gray), and the other using our design (red). As can be seen, the majority of the power consumption when our policy is used is well under the power cap. Our capping success rate is maintained over 99% for the entire year.

There are a few exceptions shown in the figure. As the power cap is set to a lower value (like this case), none of the jobs in the scheduling window can be allocated onto the system because each of them individually would violate the power cap and the scheduler is stuck in a state of deadlock and unable to schedule any jobs to the system. For example, a job requesting the full system of 48 racks would absolutely consume more than 3,000 kW of power, so this job is held until the power cap is raised to a level which would be greater than or equal to that required by the job. If the window is entirely full of jobs which would all violate the power cap, the first job which entered the window is allowed to temporarily violate the cap and run. There are other alternatives to handle this deadlock situation. One is to let the system be idle until the power cap is raised. Or we could have chosen to increase the size of the window allowing other jobs with lower power profiles to run, but this would have unnecessarily hurt average job wait time. Obviously, there are trade-offs with each approach. In this study, we chose to temporarily allow an explicit violation of the power cap by allowing the jobs in the queue to run.

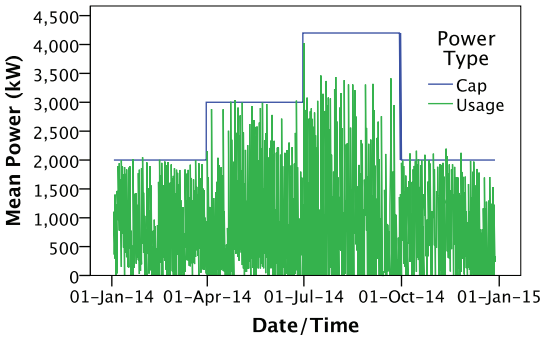



Fig. 7: Variable power cap scenario. The power cap is changed four times over the course of the year.

In the second set of experiments, we simulated a scenario where the power cap is changed several times throughout execution (in our case, over the year). In this scenario the power cap is changed four times during the year (from 2000 kW to 3000 kW to 4000 kW back to 2000 kW), but the number of changes could be many more. Figure 7 shows the result of this simulation. As can be clearly seen, with the exception of several jobs which had to be run to prevent deadlock, the scheduler is capable of keeping jobs under the power cap when

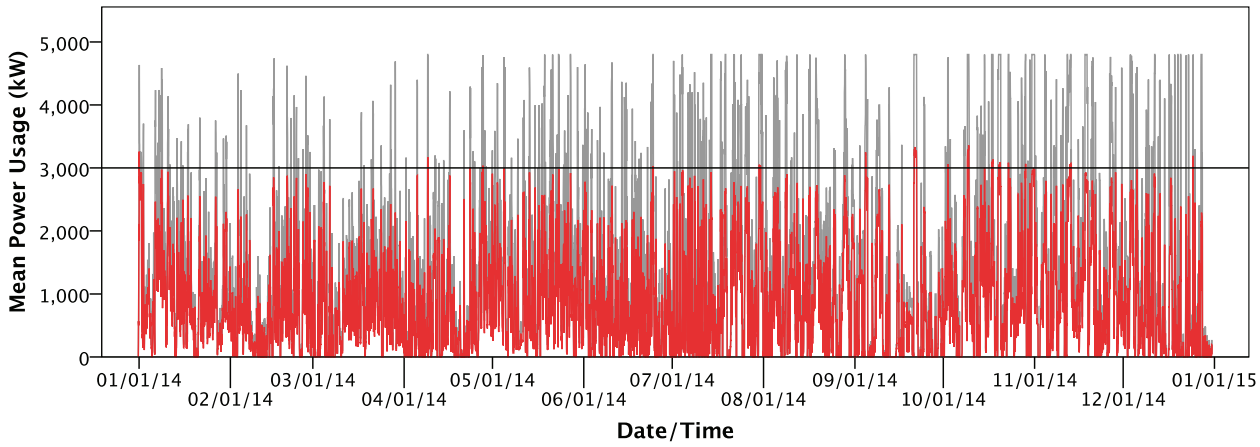



Fig. 6: Power consumption using the original Mira scheduler (light gray) without power capping and our power-aware scheduler (red) with the power cap set to 3,000 kW and window size of 20. Those points violating the power cap with our scheduler denote jobs which had to be run in order to free the window from deadlock. This is discussed in more detail in Section VIII-B.

the cap is dynamically changed. Our capping success rate for this experiment was 99.2%.

### C. Scheduling Performance

It's impossible to say that scheduling performance will never be impacted with our design or any power cap based policy simply because if the power cap is set to a low enough level, few (if any) jobs will be allowed to run. Since there is no known related scheduling studies, in our experiments, we compare our design against *a naive power capping method*. This naive method is a simple augmentation of the traditional one-to-one scheduling design such that it is able to know what the current power consumption of the system is. It assumes that the power consumption of a job is the theoretical maximum. In the experiments, we compare the performance impact brought by our design as against the naive capping. We evaluate the impact on both system utilization and average job wait time. In addition, we examine both FCFS and WFP based job ordering mechanisms to evaluate whether our design is influenced by different job orderings.

Figures 8 and 9 compare the average job wait time with power caps of 41.7% of maximal power (2,000 kW), 62.5% of maximal power (3,000 kW), and 80% of maximal power (3,000 kW) for both the WFP and FCFS job ordering strategies. For each of the figures presented, we show the change relative to an un-power capped system with either FCFS or WFP job ordering (depending on the experiment). Therefore, the further away from 0, the greater the relative change to an un-capped system.

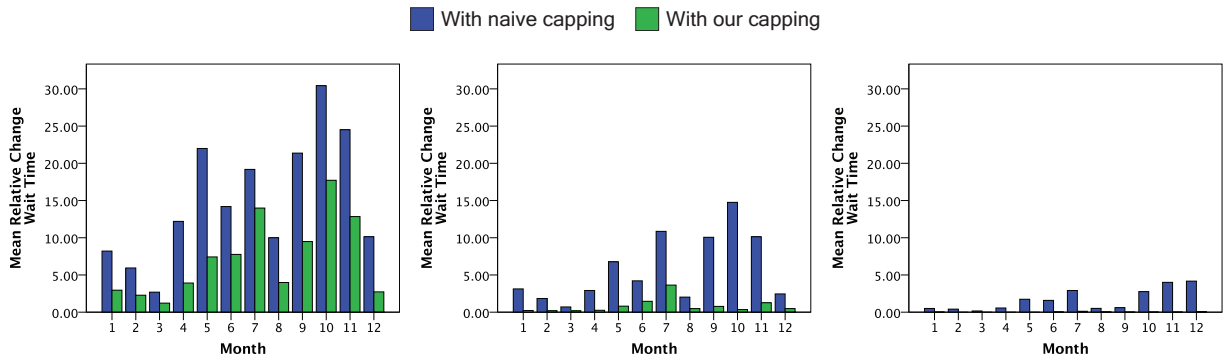
Looking at Figures 8a and 9a, it's clear that the difference between naive power capping and our capping approach is very pronounced on a heavily power-capped system. The naive approach is always significantly worse on average than our approach. Similarly, in Figures 8b and 9b, we see this is also the case. Most interestingly, on a modestly power-capped system (83.3% of maximal power), Figures 9c and 8c show that our approach has very little negative impact on wait times as compared to an un-capped system. The same cannot be said for naive capping however as it still has a pronounced impact on wait times for the majority of months. In Figures 9a and 9b we see that our approach is able to actually provide better average wait times compared to an un-capped system.

This most likely has to do with the fact our knapsack based solution is free to pick from any job in the window. An un-capped system on the other hand enforces strict job ordering, which means it could get stuck in a situation where there was a job which would not fit in the available resources at the head of the queue. Thus, on average, it's possible our capping solution will perform better by reducing the wait times of jobs other than those at the head of the queue.

The results clearly show our capping always outperforms the naive approach, with the potential average relative change in wait time being as high as 15 which was the case in month 5 of Figure 9a. The absolute improvement of job wait time introduced by using our capping over naive capping ranges from 42% to 188% with FCFS, and from 36% to 177% with WFP. With a power budget, our design can make intelligent decisions based on not only job size and runtime, but also job power profile. Job power profiles vary greatly, which enables our design to meet the budget while minimizing the impact on wait time.

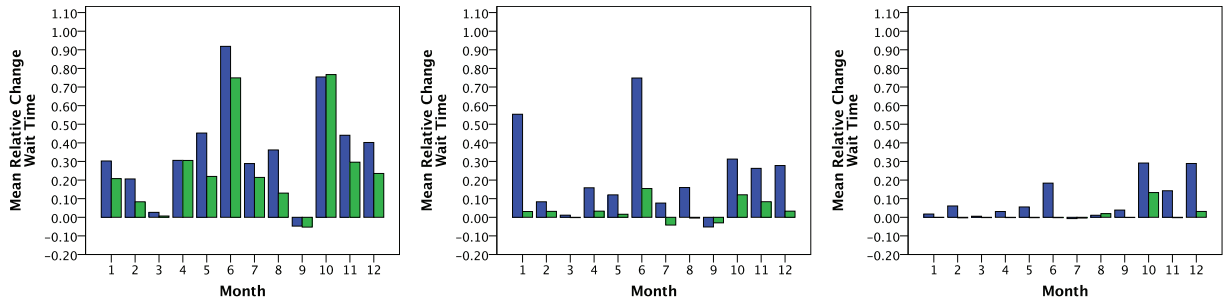
It is extremely important to remember that we are not discarding any jobs simply because they will never fit under the power cap. As such, 48 rack jobs, in particular, constitute the vast majority of outliers with respect to wait time. Remember that the only time we allow those jobs to run and violate the power cap is when we have no other options. As such, they will incur extremely unrealistic delays driving the average much higher than normal. Therefore, these wait time results should be thought of as a sort of "worst case" scenario.

When comparing the FCFS and WFP job orderings, we find that for the entire year WFP performs better than FCFS by almost 100 seconds on average with month-by-month variations having the potential to be much more. Surprisingly




(a) 2,000 kW (41.7% of maximal power) (b) 3,000 kW (62.5% of maximal power) (c) 4,000 (83.3% of maximal power)

Fig. 8: Average relative change from an un-capped system of job wait times for different scheduling policies using WFP job ordering.




(a) 2,000 kW (41.7% of maximal power) (b) 3,000 kW (62.5% of maximal power) (c) 4,000 (83.3% of maximal power)

Fig. 9: Average relative change from an un-capped system of job wait times for different scheduling policies using FCFS job ordering.

however, when compared to an un-capped system, we see that the relative change in wait time has the ability to be significantly greater with the WFP job ordering. It turns out this has to do with those large jobs which are forced to wait around until they simply have to be run and violate the power cap. This also has another more subtle effect: because WFP prefers larger and older jobs, those jobs which violate the power cap just sit occupying precious window space just waiting for their turn to run. This means that the window can get plugged up with mostly big jobs thereby hurting smaller jobs which haven't even had the opportunity to enter the window yet. This is also why the relative change in average wait time decreases dramatically in Figures 8b and 8c, those bigger jobs are actually able to run without having to violate the power cap thereby making room in the window for smaller jobs.

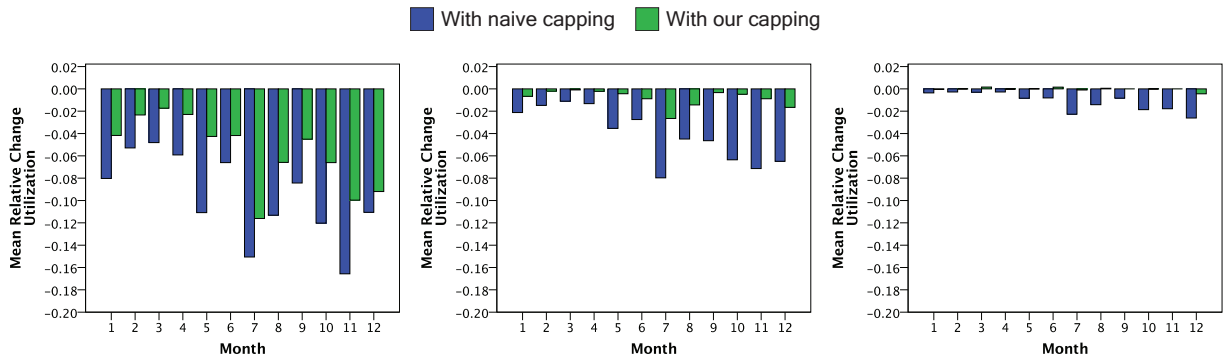
The average relative change in utilization to an un-capped system per month is shown in Figures 10 and 11. Clearly, our capping greatly outperforms naive capping. The absolute improvement of utilization introduced by our capping ranges from 3% to 21% with FCFS, and from 8% to 21% with WFP. The reasoning for this comes down to the learning aspect of our solution. Since we are able to learn from power data to form power profiles, we are usually able to use less of the cap than if we made the naive assumption that a job would use as much power as it possibly could.

Considering our Knapsack based policy is designed to select any job in the scheduling window to maximize system utilization without violating the power cap, it is expected that this solution would have better average utilization than *with naive capping* regardless of the job ordering in the waiting queue. Looking at both of the figures, it's clear that this is in fact always the case. Also evident, and as expected, utilization improves relative to an un-capped system as the power cap is increased. These figures also show that utilization for capping of any sort increases with respect to the cap.

The relative differences between FCFS and WFP job ordering are much less pronounced when it comes to utilization as they were with wait times. This is mainly due to the fact that sizes of jobs fit a fairly normal distribution. Therefore, which jobs are available in the scheduling window make less of a difference to utilization on the whole as the available "holes" in the system where jobs could be scheduled are very likely plugged with what is in the window.

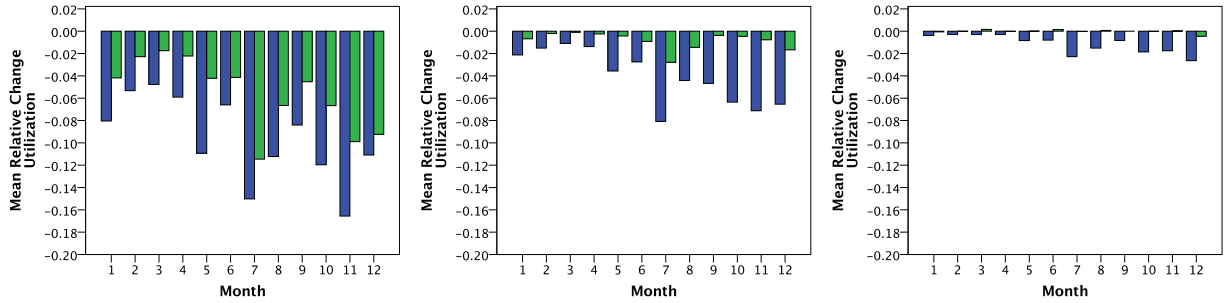
## IX. Conclusions

In this paper, we have proposed a practical power-aware scheduler which is capable of dynamically learning job power profiles and intelligently allocating user jobs to enforce a power cap without degrading resource utilization unnecessar-




(a) 2,000 kW (41.7% of maximal power) (b) 3,000 kW (62.5% of maximal power) (c) 4,000 kW (83.3% of maximal power)

Fig. 10: Average relative change from an un-capped system of system utilization rates for different scheduling policies using WFP job ordering.




(a) 2,000 kW (41.7% of maximal power) (b) 3,000 kW (62.5% of maximal power) (c) 4,000 kW (83.3% of maximal power)

Fig. 11: Average relative change from an un-capped system of system utilization rates for different scheduling policies using FCFS job ordering.

ily. Our design is based on the observation that not only do HPC jobs have distinct power profiles, so do the groups to which they belong. As our design is on-line, it can react to changes in power profiles *as jobs execute* and schedule subsequent jobs accordingly.

To the best of our knowledge, this is the first runtime power aware scheduling design which leverages individual job power profiles as well as group power profiles to make runtime scheduling decisions. In contrast to existing power-aware scheduling studies, our design doesn't put any extra requirement on workload and is applicable to general HPC applications/jobs. Our key contributions and findings are:

- Empirical analysis of a full year of production workload from a leadership supercomputer. This analysis led to the discovery that HPC jobs have distinct power profiles and these power profiles can vary significantly from job to job.
- Dynamic learning of the power profiles of jobs and groups for making scheduling decisions. Use of this dynamic learning resulted in a 94% learned job power profile rate after just 26 days.
- A job power aware scheduler to keep the total system power under a given power cap while minimizing impact on system utilization. Our design doesn't degrade scheduling performance when the power cap is above 83% of maximum and was surprisingly able to perform better than no capping in 4 out of the 12 months studied in terms of wait time.

This design is generally applicable to other systems as long as the underlying platform has some power monitoring facility for which data are available for specific portions of hardware. So long as this is the case, one could use this existing monitoring facility to get power data for a particular job and feed it into our design. Further, the repetitious nature of jobs we have demonstrated in this work isn't just limited to Mira, most HPC jobs at DOE and XSEDE facilities are repetitive and therefore predictable. Our design works with various job ordering policies, hence being applicable at most if not all of the supercomputing facilities.

## ACKNOWLEDGMENTS

The work at the Argonne Leadership Computing Facility at Argonne National Laboratory is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

Zhiling Lan is supported in part by US National Science Foundation grants CNS-1320125 and CCF-1422009.

The authors thank the ALCF staff for their help. They are especially thankful to Bowen Goletz and Eric Pershey for their assistance in obtaining the data critical for this work.

## REFERENCES


[1] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," 2001.

[2] Y. Liu and H. Zhu, "A survey of the research on power management techniques for high-performance systems," *Softw. Pract. Exper.*, vol. 40, no. 11, pp. 943–964, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1002/spe.v40:11

[3] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Parallel job scheduling for power constrained hpc systems," *Parallel Comput.*, vol. 38, no. 12, pp. 615–630, Dec. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2012.08.001

[4] J. Hikita, A. Hirano, and H. Nakashima, "Saving 200kw and $200 k/year by power-aware job/machine scheduling," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–8.

[5] 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*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 807–818. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.71

[6] 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*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 460–469. [Online]. Available: http://doi.acm.org/10.1145/1542275.1542340

[7] Z. Zhou, Z. Lan, W. Tang, and N. Desai, *Job Scheduling Strategies for Parallel Processing: 17th International Workshop, JSSPP 2013, Boston, MA, USA, May 24, 2013 Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ch. Reducing Energy Costs for IBM Blue Gene/P via Power-Aware Job Scheduling, pp. 96–115. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43779-7_6

[8] "Argonne leadership computing facility." [Online]. Available: https://www.alcf.anl.gov

[9] Z. Cao, L. T. Watson, K. W. Cameron, and R. Ge, "A power aware study for vtdirect95 using dvfs," in *Proceedings of the 2009 Spring Simulation Multiconference*, ser. SpringSim '09. San Diego, CA, USA: Society for Computer Simulation International, 2009, pp. 107:1–107:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1639809.1639922

[10] E. K. Lee, I. Kulkarni, D. Pompili, and M. Parashar, "Proactive thermal management in green datacenters," *J. Supercomput.*, vol. 60, no. 2, pp. 165–195, May 2012. [Online]. Available: http://dx.doi.org/10.1007/s11227-010-0453-8

[11] O. Sarood, E. Meneses, and L. V. Kale, "A 'Cool' Way of Improving the Reliability of HPC Machines," in *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, November 2013.

[12] 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 SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 60:1–60:11. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503264

[13] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, G.-T. Chiu, P. Boyle, N. Chist, and C. Kim, "The IBM Blue Gene/Q compute chip," *Micro, IEEE*, vol. 32, no. 2, pp. 48 –60, march-april 2012.

[14] "The Top500 List," November 2012. [Online]. Available: http://www.top500.org/list/2012/11/

[15] IBM Redbooks, *IBM System Blue Gene Solution: Blue Gene/Q System Administration*. Vervante, 2013.

[16] S. Wallace, V. Vishwanath, S. Coghlan, J. Tramm, Z. Lan, and M. E. Papka, "Application power profiling on IBM Blue Gene/Q," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, 2013, pp. 1–8.

[17] D. Feitelson and A. Weil, "Utilization and predictability in scheduling the ibm sp2 with backfilling," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, Mar 1998, pp. 542–546.

[18] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, Aug 2009, pp. 1–10.

[19] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 189–204, Aug. 2000. [Online]. Available: http://dx.doi.org/10.1177/109434200001400303

[20] M. Rashti, G. Sabin, D. Vansickle, and B. Norris, "Wattprof: A flexible platform for fine-grained hpc power profiling," in *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, Sept 2015, pp. 698–705.

[21] J. H. L. III, D. DeBonis, R. E. Grant, S. M. Kelly, M. Levenhagen, S. Olivier, and K. Pedretti, "High performance computing - power application programming interface specification version 1.2," Feb 2016.

[22] Engineering Statistics Handbook, "Two-sample t-test for equal means," http://www.itl.nist.gov/div898/handbook/eda/section3/eda353.htm.

[23] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[24] "CQSim: An event-driven simulator," http://bluesky.cs.iit.edu/cqsim.

[25] "StreamQSim." [Online]. Available: https://bitbucket.org/SeanWallace/streamqsim

[26] "Parallel Workloads Archive." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[27] "Cobalt scheduler." [Online]. Available: https://www.alcf.anl.gov/cobalt-scheduler

[28] "XSEDE allocations overview." [Online]. Available: https://www.xsede.org/allocations

[29] "DOE INCITE program." [Online]. Available: http://www.doeleadershipcomputing.org