2015 IEEE 22nd International Conference on High Performance Computing

Metascheduling of HPC Jobs in Day-Ahead Electricity Markets

Prakash Murali
IBM Research, Bangalore, India
Email: sercprakash@ssl.serc.iisc.in

Sathish Vadhiyar
Supercomputing Education and Research Center
Indian Institute of Science, India
Email: vss@serc.iisc.in

***Abstract*—High performance grid computing is a key enabler of large scale collaborative computational science. With the promise of exascale computing, high performance grid systems are expected to incur electricity bills that grow super-linearly over time. In order to achieve cost effectiveness in these systems, it is essential for the scheduling algorithms to exploit electricity price variations, both in space and time, that are prevalent in the dynamic electricity price markets. In this paper, we present a metascheduling algorithm to optimize the placement of jobs in a compute grid which consumes electricity from the day-ahead wholesale market. We formulate the scheduling problem as a Minimum Cost Maximum Flow problem and leverage queue waiting time and electricity price predictions to accurately estimate the cost of job execution at a system. Using trace based simulation with real and synthetic workload traces, and real electricity price data sets, we demonstrate our approach on two currently operational grids, XSEDE and NorduGrid. Our experimental setup collectively constitute more than 433K processors spread across 58 compute systems in 17 geographically distributed locations. Experiments show that our approach simultaneously optimizes the total electricity cost and the average response time of the grid, without being unfair to users of the local batch systems.**

I. Introduction

High performance grid computing involving supercomputer systems at distributed sites plays an important role in accelerating scientific advancement and facilitating multi-institutional and multi-disciplinary collaborations. Extreme Science and Engineering Discovery Environment (XSEDE), Open Science Grid and European Grid Infrastructure are some examples of computational grid infrastructure that support science gateways to enable communities to use HPC systems. The operational costs of these systems have become comparable to the cost of hardware acquisition, and service providers regularly budget millions of dollars annually for electricity bills [1]. Hence it is imperative to include power and electricity cost minimization in job scheduling decisions in high performance computational grids.

A large body of work has been developed to reduce the power consumption of data center servers, by switching off unused nodes [2], using voltage and frequency scaling to run servers at low power [3], and using renewable energy sources to reduce the carbon footprint of computation [4]. However, such approaches which reduce the power consumption by lowering CPU frequency or voltage, slow down HPC applications and result in unacceptable loss of performance and system utilization. Deregulation of the electricity power markets and the creation of power trading zones in many countries offer opportunities to purchase wholesale power under various dynamic pricing schemes. Dynamic electricity price markets are popular and cater to large industries and manufacturing units. In such markets, scheduling algorithms can exploit spatial and temporal electricity price differentials and schedule workloads at servers which have cheap power.

Typically, the wholesale energy market consists of a day-ahead market and a real time market. In the day-ahead market, consumers of electricity submit bids with their expected power requirements for the following day (demand), and suppliers of electricity submit bids with their expected generation and supply volumes for the coming day (supply). The trading agency which accepts these bids, sets a clearing price for each hour of the coming day, based on the supply and demand bids. In contrast, real-time markets operate at a faster rate, and the prices can fluctuate, say every 5 minutes, based on the actual supply and demand scenario in the market. Real time markets have been considered for supplying electricity for Internet data centers [5]–[8]. With the increasing power requirements in HPC, we anticipate that in the near future, HPC system operators will consider these markets as a potential source of cheap power. HPC sites like Argonne National Lab (ANL) are already considering using electricity according to time-of-use pricing [1]. We use the day-ahead hourly electricity prices because the day-ahead markets are suitable for HPC workloads. The loads on these systems are predictable at a coarse level and can be used by administrators to submit accurate demand bids for procuring power supply the following day. Moreover, the prices in the day-ahead market fluctuate smoothly and can be predicted using time series forecasting techniques. These predictions can be used for intelligent scheduling decisions.

For a scheduler to estimate the total electricity price for a job execution before allocating the job to a system with hourly price variations, it is important to know the period of execution in the system. Production parallel systems in many supercomputing sites are batch systems that provide space sharing of available processors among multiple parallel applications or jobs. Well known parallel job management frameworks including PBS are used to provide job queuing

This work is supported by Department of Science and Technology (DST), India via the grant SR/S3/EECE/0095/2012.

978-1-4673-8488-9/15 $31.00 © 2015 IEEE
DOI 10.1109/HiPC.2015.10

and execution services for users on these supercomputers. With multiple users contending for the compute resources, a batch queue submission incurs time due to waiting in the queue before the resources necessary for its execution are allocated. The queue waiting time ranges from a few seconds to even a few days on production systems, and is dependent on the load of the system, the batch scheduling policy and the number of processors requested by the user. Thus, the queue waiting time and hence the starting time of the job on the system is not known in advance. For the execution time and the ending time of the job, we use the estimated run time (ERT) provided by the user in the job script. The ERT of the job is required for system schedulers which employ backfilling to increase system utilization, and is thus supported by many of the job management frameworks including PBS. When the user does not specify the ERT, the maximum runtime limit is assumed.

In this paper, we have developed a metascheduling strategy that considers hourly electricity price variations in a day-ahead market and predicted response times to schedule HPC parallel jobs to geographically distributed HPC systems of a grid. Our metascheduler simultaneously minimizes electricity cost and response times by exploiting electricity price differences across states and countries to schedule jobs at systems where the cost of servicing the job is minimized while ensuring that the users do not suffer degradation in system response time. Our metascheduler uses a framework that we have developed for prediction of queue waiting times. We formulate the job scheduling problem in our metascheduler as a minimum cost maximum flow computation in a suitable flow network and use the network simplex algorithm for optimization. We evaluated our algorithm with trace based simulations using synthetic and real workload traces of two production grids: XSEDE [9] and NorduGrid [10], and real electricity price data sets. Our approach can potentially save \$167K in annual electricity cost while obtaining 25% reduction in average response time compared to a baseline strategy. We found that even users who do not use our metascheduler, can sometimes obtain improvements in response time when our algorithm migrates jobs away from their local systems.

To our knowledge, ours is the first work on metascheduling HPC workloads across grid systems considering actual or predicted hourly electricity prices at a predicted period of job execution.

In Section II, we motivate and describe the problem definition. We discuss our methodology including the network flow formulation in Section III. The experimental setup is detailed in Section IV. We present the results and some practical considerations in Section V. We describe the related work in Section VI and conclude in Section VII.

II. Background

Popular metaschedulers like Condor-G [11] use the concept of periodic scheduling cycles to efficiently manage job submission and dispatch decisions. When a job is submitted by a user, the metascheduler marks the job as pending for scheduling. During the subsequent scheduling cycle, the scheduling algorithm assigns a subset of the pending jobs for processing at a subset of the systems in the grid. In many currently operational grids, administrators impose restrictions on the maximum number of jobs that can be submitted to a particular system in a single scheduling cycle to prevent the middleware at these systems from being flooded by job submissions [11]. We denote this maximum number as $MaxQ$.

Given n geographically distributed grid systems with day-ahead hourly electricity prices and a meta scheduling portal for accepting job submissions, the metascheduling problem is to assign jobs in a scheduling cycle to systems while simultaneously minimizing the response time and electricity cost of the job executions.

While our metascheduler may increase the local electricity cost at a system due to job migrations from submitting to execution systems, it attempts to reduce the overall operational cost of the grid. We also claim that the variations in workload at a particular system due to our metascheduler cannot significantly alter the day-ahead hourly electricity prices at the system's location. This is because the day-ahead market trading volume is typically many orders of magnitude higher than the power consumption of a single computing system.

III. Methodology

We formulate the grid scheduling problem as a minimum cost maximum flow computation and use the network simplex algorithm to find the optimal flow. To compute the cost of scheduling a job on a system, we require predictions of the response time of the job at the system and the electricity cost required to execute the job. We describe our approach for prediction of response time in Section III-A, and prediction of electricity price in Section III-B. In Section III-C, we define the cost function and the flow network used in our approach.

A. Response time prediction

In batch queue systems, similar jobs which arrive during similar system queue and processor states, experience similar queue waiting times. We have developed an adaptive algorithm for prediction of queue waiting times on a parallel system based on spatial clustering of the history of job submissions at the system [12]. To obtain the prediction for a job J on a system S, J is represented as a point in a feature space using the job characteristics (request size, estimated run time) specified by the user, the queue state at the system at the current time (sums of request sizes of queued jobs, estimated run times of queued jobs, elapsed waiting time of queued jobs) and the state of the compute nodes at the current time (number of occupied nodes, total elapsed running time of the jobs, total estimated run times of the jobs). We compute the Manhattan distance of each history job with the target job, and consider history jobs with small distance values as being similar to the target job. Then, we use Density Based Spatial Clustering of Applications with Noise (DBSCAN) to find clusters of similar jobs. DBSCAN also allows us to eliminate outliers among the history jobs. If we find clusters which are very similar to the target job, i.e., clusters with low average distance, we use the

weighted average of waiting times of jobs in the cluster as the prediction for the job, J. If we do not find clusters which are very similar to the job, the job features of the history jobs and the queue waiting times experienced by these jobs are used to train a ridge regression model. The features of the target job are supplied as input to this model to obtain the predicted queue waiting time.

To test our predictions, we evaluated our framework using production supercomputer workload traces with varying site and job characteristics, including two Top500 systems, obtained from Parallel Workloads Archive [13]. Across workloads, our predictions results in up to 22% reduction in the average absolute error and up to 56% reduction in the percentage prediction errors over existing strategies including QBETS [14] and IBL [15]. Our prediction system also gave accurate predictions for most of the jobs. For example, for the workload of ANL's Intrepid system our predictor gave highly accurate predictions with less than 15 minutes absolute error for more than 70% of the jobs. We also observed prediction errors of less than one hour for 88-98% of the jobs for large-scale systems including CEA Curie of France which is a Top500 system, DAS2 of Netherlands, and SDSC Paragon of USA. Our prediction system is also practically applicable to a real environment. In all cases, our predictor consumes less than a second for a prediction. Our predictor is currently deployed on an 800 core system in our home department, delivering queue waiting time predictions to users with less than 30% error. In a real implementation, all the information required by our predictor can be obtained using a single command (e.g., 'qstat -a' in PBS).

To find the response time of a job on a target system, we invoke our queue waiting time predictor to find the predicted start time of the job, t_s. Then, we use the estimated run time (ERT) supplied by the user to predict the end time of the job. While the user estimates are known to be inaccurate [16], the estimates serve as strict upper bounds on the runtimes since job schedulers used in HPC systems terminate a job when its runtime exceeds the user estimated runtime. In this work, we use these estimates to demonstrate the benefits that can be obtained for the grid systems from participation in dynamic electricity markets. We show in our experiments that using these estimates results in improved scheduling decisions over a strategy that does not use predictions, but only considers the loads at the time of the submission. We expect that using improved runtime prediction strategies can lead to additional benefits.

Since the ERT supplied by the user is relevant only for the submission system, we use a scaling factor to adjust the ERT for the target system. We assume that the applications in our workload have similar scalability characteristics as HPL (High Performance Linpack) benchmark. We compute the scaling factor by obtaining the ratio of the HPL performance per core (in GFlops) of the target system and the submission system. For a job which is submitted at a system S_i, for which we require an estimate of the runtime at system S_j, we obtain the performance per core of both systems, and scale the ERT of the job as $ERT_{S_j} = ERT_{S_i} \times ppc_{S_i}/ppc_{S_j}$ where ERT_{S_i} is the estimated run time of the job provided by the user on system S_i, ppc_{S_i} is the performance per core of system S_i. The predicted end time of the job on the system S_j is $t_e = t_s + ERT_{S_j}$. We describe our approach for estimating the power per core of a system in Section IV.

Migration of jobs from submission to execution sites involves transfer of data and executables. In practice, the data size parameter can be given as input by the user, and the cost of data movement can be computed using the data size and the properties of the link (latency and bandwidth) between the submission and the execution sites. However, in this study, we do not consider data transfer times because our workloads do not include the necessary information about job file transfers and network state, and the current workload models [13], [17], [18] for synthetic logs do not generate data sizes. We assume that the executable binaries and data needed for a job are set up at multiple systems prior to the job submission and hence the cost of job migration between the systems is negligible.

B. Electricity price prediction

To obtain the electricity prices during the job's execution period at a target system, we find the predicted start and end time of the job using our response time predictor. Given the predicted start and end times of the job on the system, we check whether the job's predicted execution duration is within the end of the day (midnight). In this case, the corresponding electricity prices during the execution period in the day-ahead electricity market are known. When the execution period does not fully lie within the hours of the current day, i.e., t_e is after midnight on the submission day, we predict the prices for the duration that lies beyond midnight. We use a Seasonal Autoregressive Integrated Moving Average (SARIMA) model to model the electricity prices of the previous days. SARIMA models are commonly used to obtain forecasts for time series data which exhibit seasonal trends across days and months. Since we observed that the prices in the day-ahead market have high lag-24 autocorrelation, we use the SARIMA model with a seasonal period of 24 hours. The various parameters required for the SARIMA model were tuned using a training set of the electricity price data. Our predictions for electricity prices were found to be highly accurate with the datasets used in our experiments, yielding average percentage prediction errors of less than 10%.

C. MCMF: Minimum Cost Maximum Flow

Minimum cost maximum flow (MCMF) is a fundamental network flow model which aims to maximize the amount of shipment of a single commodity through a network while minimizing the cost of the shipment. MCMF can be solved using a number of approaches including cycle canceling, linear programming and network simplex algorithms.

To schedule a set of jobs to a set of systems, we represent the jobs and systems as nodes in a flow network. We consider a system to be *compatible* for a job, if the total cores in the system is more than the request size of the job and the

maximum wall time permitted in the system is more than the user estimated run time of the job. For each job, we add an arc of unit capacity from the job to each compatible system. A flow of unit value along an arc from J to S represents scheduling J on S. We assign the cost of each job-system arc as a weighted linear combination of the predicted response time of the job and the electricity price required to execute the job on the system.

To compute the cost of assigning a job J to a system S, we predict the start time and end time of J on S as t_s and t_e, respectively. Assuming that the job is submitted at time t, the response time of the job is $T_J = t_e - t$. In each scheduling cycle, the metascheduler polls each system in the grid to obtain its current queue and processor state in order to invoke the response time predictor for each job on each compatible system. Using these predictions for each job on each system, we find the maximum and minimum predicted response times in this scheduling cycle as T_{max} and T_{min}. The cost of scheduling the job J at the system S, in terms of response time, is defined as

$$C_T(J,S) = (T_J - T_{min})/(T_{max} - T_{min}). \quad (1)$$

We model the electricity prices at the location of the system S to obtain a function $\hat{\phi}_S(t)$ which gives the predicted electricity price during the time t. The cost of scheduling the job at this system, in terms of electricity price is defined as

$$C_E(J,S) = \frac{\sum_{t=t_s}^{t_e} P_{J,S} \times \Delta \times \hat{\phi}_S(t) - E_{min}}{E_{max} - E_{min}} \quad (2)$$

where $P_{J,S}$ denotes the power consumption of J on S, Δ denotes the period of the day-ahead electricity market, and E_{max} and E_{min} denote the maximum and minimum predicted electricity cost observed in the current scheduling cycle. Since we use prices from the day-ahead hourly market, $\Delta = 1$ hour in all our experiments. We describe our approach to calculate $P_{J,S}$ in Section IV.

We define the cost of scheduling J on S as

$$C(J,S) = w_t \times C_T(J,S) + (100 - w_t) \times C_E(J,S) \quad (3)$$

where w_t is the relevance of the response time in the cost function. $C_T(J,S)$ and $C_E(J,S)$ are normalized by the corresponding minimum and maximum values to unitless quantities so that w_t can be used for weightage of the two terms irrespective of their absolute value. Our formulation shown in Equation 3 is based on the nadir-utopia normalization method by Kim and Weck [19]. In every scheduling cycle, the objective function is re-normalized to adapt it to the current predictions of electricity price and queue waiting time.

We connect an arc of unit capacity from the source node s to each job and an arc of capacity $MaxQ$ from each system to the sink node t. The costs of these edges are set to 0. For a set of m jobs and n systems, we illustrate this network in Figure 1 where each arc is labeled with two parameters, namely, the capacity of the edge and the cost of unit flow through the edge. We scale the costs of the network edges by multiplying with a large constant (100), and round off the

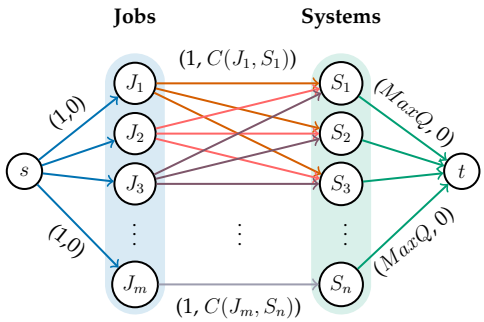

Fig. 1: Flow network used for scheduling. The edges are labelled as (edge capacity, cost of unit flow).

values to integers. In such a network, the Integrality Theorem of maximum flow networks [20] guarantees that the maximum flow is integral and each unit capacity edge in our network has a flow value of either 0 or 1. We compute a maximum flow of minimum cost in this network using the network simplex algorithm available in the Python package, *NetworkX*. After computing the minimum cost flow, we inspect the job-system arcs and select the arcs which have non-zero flow. For each arc from J to S which has non zero flow, we schedule the job J on system S. By the flow conservation principles, we are guaranteed that a) not more than one system is selected for a job and b) no system receives more than $MaxQ$ jobs during one scheduling cycle. While our current maximum flow network does not consider the limits imposed on the number of jobs executed on a system per user or community at any given time, that are imposed in resources like XSEDE, our network can be trivially extended to consider these limits.

IV. Experimental Setup

We performed trace based simulation of real and synthetic grid workloads using real electricity price data sets and power consumption profiles of compute systems to test the effectiveness of our approach. In this section we explain each component of the experimental setup.

A. Workload and Scheduler Simulator

We conduct simulations using grid traces which are in Standard Workload Format (SWF) [13] or Grid Workload Format (GWF) [21]. Each line in the SWF/GWF trace denotes a job and records the arrival time, run time, number of cores, user estimated runtime and other job parameters. GWF traces also record the system where the job was originally submitted. While using SWF traces during synthetic trace generation, we appended each line in the trace with the submission system. To conduct trace driven simulations, we used an extended version of the Python Scheduler Simulator (pyss) developed by the Parallel Systems Lab in Hebrew University [22]. pyss accepts a workload trace, system size and scheduling algorithm as inputs and replays the job arrival events, start and end of job execution events to simulate the state of the system with the input workload. Since pyss simulates only one system, we

TABLE I: The XSEDE Grid

System	Location	Cores	Power (watts/core)	Performance (Gflops/core)
Blacklight	Pittsburgh	4096	87.89	9.00
Darter	Tennessee	11968	30.58	20.79
Gordon	San Diego	16160	22.17	21.10
Trestles	San Diego	10368	42.66	9.64
Mason	Indiana	576	39.95	7.43
Lonestar	Texas	22656	15.83	13.32
Queenbee	Louisiana	5440	16.25	9.37
Steele	Purdue	4992	83.75	13.33

extended it to support multiple systems. We implemented a metascheduler class that acts as a common interface between the job submissions and the various execution systems. We configured pyss to use the EASY backfilling algorithm [23] to schedule jobs at the individual systems.

B. Grid systems

We simulate two currently operational grids which collectively span 58 individual compute systems, 17 countries and states, 10 electricity transmission operators, 7 time zones and more than 250k job submissions. For each system, we obtained the number of cores and maximum wall time of a job from publicly advertised system information.

1) XSEDE: The Extreme Science and Engineering Discovery Environment (XSEDE) project is a large scale compute grid which connects many universities and research centers in the US. For high performance computing, XSEDE connects eight supercomputing systems situated across different states in the US. For our simulation experiments, we used eight CPU-only systems of XSEDE and its previous incarnation, TeraGrid. The XSEDE system configuration ia shown in Table I. Each individual XSEDE system uses the Portable Batch System (PBS) or Sun Grid Engine (SGE) for job management, and grid submissions are processed through Condor-G metascheduler [11]. We model the jobs in the production workload.

2) NorduGrid: NorduGrid is a very large grid with 80 systems spread across 12 countries with a majority of the systems located in the Nordic countries. We simulated 50 selected systems of NorduGrid which constitute over 90% of the total CPU cores available in the grid. The grid configuration was obtained from [10].

C. Workload

For simulating the jobs at a system, we used a synthetic workload for XSEDE[1] and real workload traces for NorduGrid.

1) XSEDE: We generated synthetic workload traces for each system using the workload models available in Parallel Workloads Archive [13]. For generating the job arrival time, request size and run time, we use the model proposed by Lublin and Feitelson [17]. To generate the user estimates of runtime, we used the model proposed by Tsafrir et.al. [18].

[1]We were not able to obtain real workloads for XSEDE.

In [24], Hart provides various summary statistics about the run times, job sizes and inter-arrival times of the production jobs in XSEDE/TeraGrid. We manually adjusted the parameters of Lublin and Tsafrir models to match the aggregate statistics of the synthetic workload with the reported XSEDE/TeraGrid statistics. Statistics of our synthetic workload match the characteristics reported by Hart. The average job runtime in our workload is 8.8 hours while the actual average runtime in TeraGrid is 9 hours. The average number of job arrivals at a system per hour is 3.22 in our workload, while the actual value is 3.27.

2) NorduGrid: In NorduGrid, we used a real workload available in Grid Workloads Archive [21]. The archive records each job's submission system, submission time, requested processors and runtime. We used Tsafrir model with the maximum observed runtime as input parameter to assign user estimated runtimes for each job.

For queue waiting time and electricity price predictions, we used a subset of jobs and electricity price data as training information. Queue waiting time is predicted for a job at a system by considering the previous 2000 job submissions at the system as the training input. For predicting the electricity prices, we use the prices of the previous three days as training input for the SARIMA model. In Table II, we show our simulation configuration including the number of jobs and the duration that is simulated for each grid.

TABLE II: Simulation Configuration

			Test configuration	
Grid	Systems	Cores	Jobs	Days
XSEDE	8	76256	10000	15
NorduGrid	50	356856	126344	90

D. Variable electricity prices

For different systems, we obtained the hourly electricity prices in the day-ahead market from the electricity operator in the respective power market. For regions without variable electricity pricing, corresponding to one system in XSEDE and 24 systems in NorduGrid, we used the applicable fixed industrial electricity price. For systems in XSEDE, we used historical market prices for June 2014 available from the regional energy operators of the Federal Energy Regulatory Commission [25]. In NorduGrid, we obtained the market prices for Denmark, Sweden, Norway, Finland, Latvia and Lithuania from Nord Pool Spot [26] and for Slovenia from BSP SouthPool [27]. For systems in United Kingdom, Ukraine, and Switzerland, we used the applicable fixed industrial prices. Overall, in NorduGrid, our electricity price data set spans three months from January-March 2014 and includes variable electricity prices for 26 systems.

E. Job power consumption and execution characteristics

1) Job power consumption: To estimate the power consumption of a job at a system, we assume that the job

has the same power consumption characteristics as High Performance Linpack (HPL). The work by Kamil et.al. [28] experimentally demonstrates that the HPL power consumption can be used to closely approximate the power consumption of production scientific applications. For each system in XSEDE, we obtained the peak power consumption from Top500 and Green500 datasets, computed the HPL power consumption per core, and scaled it by the number of cores used by a job to find the power consumption of a job. Thus, if a job requires n cores on a machine which has a total of N cores and advertised HPL power is P_{HPL}, the job power consumption is considered as $P_J = n \times (P_{HPL}/N)$. Table I shows the values of HPL power consumption per core for each system. For NorduGrid, we were unable to obtain HPL benchmark data on each system. Hence, we resorted to white papers published by the chip manufacturers to obtain the power consumption per core. Similar to XSEDE, we scaled these numbers with the requested number of cores to find the power consumption of each job.

2) Runtime scaling: For XSEDE, we obtained the HPL peak performance (Rmax in TFlops) of a system using Top500 data and normalized it by the number of cores in the system to find the performance per core, and obtained scaling factor for a pair of systems as mentioned before. We computed such scaling factors for every pair of systems using Top500 data. For NorduGrid, we obtained the theoretical peak performance of a core in the system (in GFlops) from architecture whitepapers.

F. Evaluation metrics

To evaluate the benefits of our approach, we employ a number of metrics as described below.

• **Average response time**: For each job, the response time is the sum of the queue waiting and running times at a system.

• **Total electricity cost**: For a job J_i which executes on system S_j, the electricity cost is computed as:
$e(J_i, S_j) = \sum_{t=T_{W_{ij}}}^{T_{W_{ij}}+T_{R_{ij}}} P_{ij} \times \Delta \times \phi_{S_j}(t)$
where P_{ij} is the power consumption of J_i on S_j in watts, Δ denotes the period of the day-ahead electricity market, $T_{W_{ij}}$ and $T_{R_{ij}}$ are the waiting time and running incurred for J_i on S_j in hours, and $\phi_{S_j}(t)$ is the hourly price variation function for S_j expressed in currency per watts. For the day-ahead hourly market, Δ is set to one hour.

• **System utilization**: Utilization at a particular system is computed by dividing the sum of the CPU hours of jobs scheduled at the system by the product of the makespan and total processors available in the system. Thus, utilization aims to measure the fraction of the system core hours which delivered useful work.

• **System instantaneous load**: Instantaneous load is defined as the sum of the CPU hours of both the running and queued jobs divided by the total processors available in the system at a particular instant.

• **Fairness to System**: A system or site's participation in grid should not affect the quality of service provided to the jobs that are submitted to the system. Specifically, a high speed system after joining the grid may become highly loaded due to migration of jobs submitted at low speed systems. To evaluate the quality of service, we compute the speedup obtained due to the use of our metascheduling algorithm, compared to the baseline. Specifically, for a job J, which is submitted at system S_i, which has response time R_J^{local} when metascheduling is not used and R_J^{grid} in the presence of metascheduling, we compute the quality of service offered to the job as: $qosScore(S_i, J) = \frac{R_J^{local}}{R_J^{grid}}$. We then compute the fairness score for a system as the average of the QoS scores of all the jobs submitted at the system. If a system has high fairness score, it indicates that the users of the system are benefitted by the system's participation in the grid.

V. Results and Discussion

During our experiments, we observed that our Python implementation running on an Intel Core i7 3.4Ghz processor with 16GB RAM takes 8.4 seconds on average for computing the scheduling cost and constructing the flow network, and 16.3 seconds on average for computing the minimum cost flow and the subsequent job submissions to individual systems. We refer to our approach based on the Minimum Cost Maximum Flow algorithm as *MCMF*. We compare our strategy with a baseline strategy *BS*, in which the jobs are executed at the submission system.

Our strategy is primarily different from existing efforts [1], [5] in terms of using waiting time predictions to estimate the benefits in response time and electricity cost for the execution period of a job, and in terms of using the hourly electricity prices during the execution to estimate the total cost. Hence we compare our approach with two strategies, the first strategy called *INST* which does not consider predictions but makes decisions based on instantaneous loads of the systems at the time of the job submissions, and the second strategy called *TWOPRICE* which does not consider hourly prices but considers only two prices per day, namely, on-peak and off-peak. The INST strategy which assumes immediate execution start of a job is implemented by feeding the waiting times as zeros to our MCMF strategy. The TWOPRICE strategy is implemented by considering the on-peak hours as 12pm to midnight and calculating the off-peak and on-peak prices as the 10th and 90th percentile of the day-ahead market prices for the simulation period. Note that the INST and TWOPRICE strategies are grid scheduling strategies since they allow sharing and migration of jobs across the grid systems.

A. Overall Results

In this section, we analyze the overall reductions in response times and electricity cost by our algorithm and compare with the other approaches. Table III shows the comparison results. The table shows the results of our MCMF algorithm with different values of w_t. Recall that w_t denotes the weight of the response time term in the cost function minimized by MCMF.

For XSEDE, we observe that with $w_t = 25\%$, our MCMF strategy simultaneously achieves 24.6% reduction in average

TABLE III: Overall Simulation Results

Grid	Strategy	Average response time (minutes)	Total electricity cost ($ or €)
XSEDE	MCMF ($w_t = 25\%$)	477.5	$224021.6
	TWOPRICE ($w_t = 25\%$)	473.3	$232557.3
	MCMF ($w_t = 0\%$)	1095.4	$187298.9
	INST	3460.8	$205876.5
	Baseline	633.7	$230985.6
NorduGrid	MCMF ($w_t = 92.5\%$)	1678.6	€12819.6
	TWOPRICE ($w_t = 92.5\%$)	1724.4	€12991.7
	INST	5210.2	€14613.6
	Baseline	1900.3	€16608.3

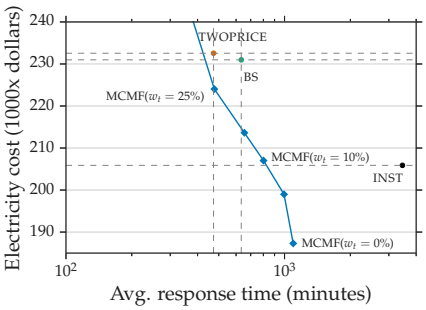

Fig. 2: Overall simulation results in XSEDE

response time and $6964 savings in total electricity cost, compared to the baseline, for the 15-day period. This reduction in electricity cost can potentially translate to a projected savings of $167K dollars per year for the whole grid. Figure 2 shows the trade off between response time and cost for our MCMF algorithm for different values of w_t compared to the other strategies. We see that when response time is not considered for optimization ($w_t = 0$), we obtain up to $43686 reduction in in electricity cost with a 1.7x increase in average response time over the baseline. Thus, for one year of operation, we can potentially save $1.04M for the whole grid. Considering that the annual electricity budget of Argonne National Laboratory's primary supercomputer is $1M [1], the savings obtained by our approach are significant.

From Table III and Figure 2, we can also see that our MCMF algorithm ($w_t = 25\%$) outperforms TWOPRICE by $8535.7 in terms of cost. This is because TWOPRICE is unaware of fine grained price fluctuations every hour. INST degrades the baseline response time by 5.5x although it achieves better cost. The reason for INST achieving lower cost and high response times in most of the cases is because at the beginning of the simulation, INST migrates jobs to good systems with low electricity cost and low response times. But soon enough, when the systems become loaded, INST continues to keep pushing jobs to the same systems without being aware of the queue waiting times caused by the high loads on the systems. So being aware of electricity cost helps INST to achieve low cost, while not being aware of waiting

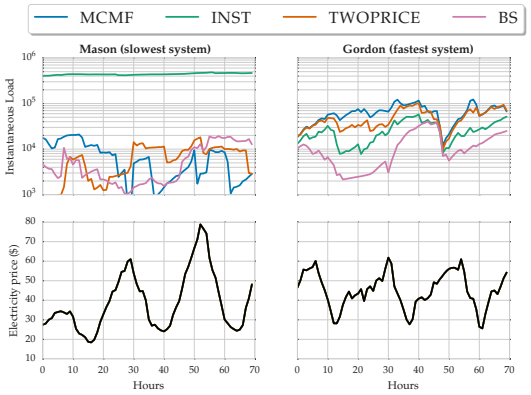

Fig. 3: Instantaneous Load Variation in XSEDE

time results in high load imbalance across systems, and hence high response times. Compared to INST, MCMF ($w_t = 10\%$) obtains 3.1x reduction in response time for the same cost. We also see that our MCMF algorithm ($w_t = 0\%$) outperforms INST in both response time and electricity cost.

For NorduGrid, we observed improvements in both response time and electricity cost when $w_t = 92.5\%$. For this workload, the response time improves by 11.7% and electricity cost reduced by €3788.7 over the baseline. Thus, in NorduGrid, our projected electricity cost savings are €15.1K per year. Similar to XSEDE, our MCMF algorithm has lesser response time and electricity cost than TWOPRICE and INST.

B. *Load, Utilization and Power Variations*

We looked at hourly instantaneous load at each system to understand the hourly behavior of our scheduling policy and compared with the other policies. We used $w_t = 25\%$ for these experiments. In Figure 3, we contrast the instantaneous load of Mason, the slowest and smallest system in the grid with Gordon, one of the largest and fastest systems. We see that INST achieves very poor load balancing because it is oblivious to response time. We also see that during the peak hours of electricity pricing at Mason, our MCMF algorithm minimizes the instantaneous load among the considered strategies. In Gordon, we see that our approach utilizes the system heavily even during a price peak at hour 30. This is because Gordon has the highest performance and the 3rd lowest service cost among the systems in XSEDE. We can see that by moving jobs to fast systems which have less service cost, our algorithm is able to simultaneously optimize electricity cost and response time better than the other strategies.

We also studied the variation of overall utilization in different systems due to our metascheduler, and show the results for XSEDE. We define service unit cost (SUC) as the product of the average electricity price at the system's location and the power per core. SUC represents the average cost in dollars (or euros) required to deliver one CPU hour of computation at the system. In XSEDE, we arranged the systems in increasing order of SUC and labelled the first four systems as *cheap*

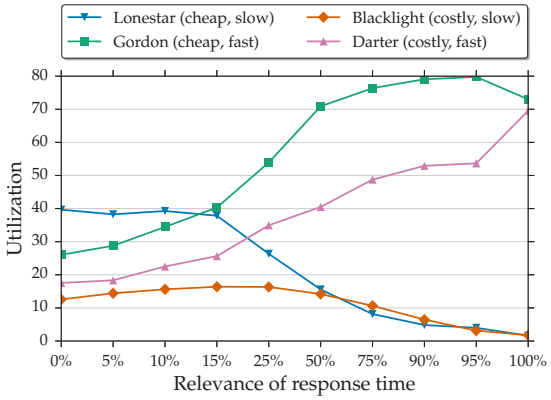

Fig. 4: Variation of system utilization in XSEDE

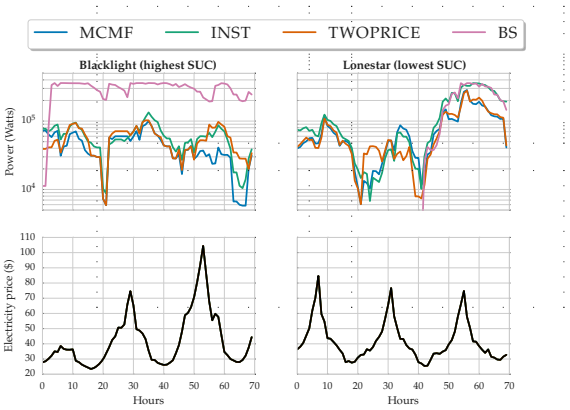

Fig. 5: Power Consumption Variation in XSEDE

and the remaining as *costly*. Similarly we used the HPL peak performance of the systems to label them as *slow* and *fast*. In our experiments, we investigated the effect of service unit cost and machine performance on system utilization. Figure 4 shows that as the importance of response time is increased, jobs are migrated from slow systems (Lonestar, Blacklight) to fast systems (Gordon, Darter). We also see that Gordon has higher utilization than Darter because it has lower SUC. We also note that since Blacklight is relatively costly and slower than the other three systems, it's utilization is low in all the configurations.

We also observed the hourly power consumption due to the scheduling policies. We used $w_t = 25\%$ for these experiments. In Figure 5, we compare the hourly power consumption of Blacklight and Lonestar, which are respectively, the costliest and cheapest systems in the grid. We see that all the electricity price-aware strategies, namely MCMF, INST, and TWOPRICE consume much less power than the baseline in the Blacklight system. During hours 50-60 when Blacklight experiences peak electricity price, the power consumption of our MCMF algorithm is better than both INST and TWOPRICE. However, in Lonestar, the fluctuations in electricity price do not influence the load or power consumption significantly even during peak hours of electricity pricing because it is both the largest and the cheapest system in the grid.

C. Fairness towards individual grid systems

The annual reports published by various supercomputing service providers which are part of XSEDE, show that, the response times of jobs processed at the system, and the number of core hours delivered to specific project allocations and users, are considered important metrics for quantifying the quality of service of each provider. Hence, it is important for service providers to ensure that their participation in the grid does not adversely affect the users of the local batch system. In this section, we compute the job service fairness score of each system when user submissions can be either through the metascheduling portal or the local batch system. In one of our experiments for XSEDE, *Expt1*, we studied the

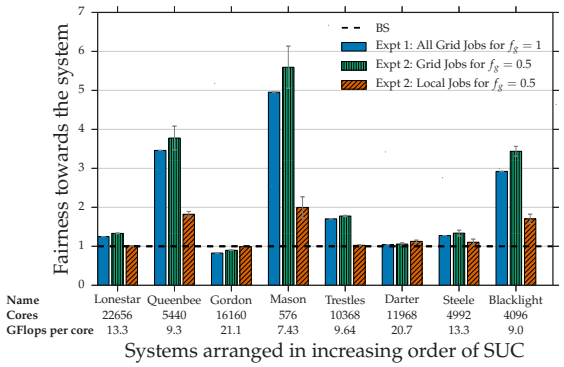

Fig. 6: Job service fairness for systems in XSEDE

job service scores when all users submit their jobs through the metascheduler, i.e., every job is a grid submission. The fairness scores for this experiment are indicated in Figure 6 by the blue bars. In another experiment, *Expt2*, we studied the case where only a subset of the jobs are grid submissions. To perform these experiments, we choose a fixed fraction of grid submissions, f_g (e.g., $f_g = 0.5$ denotes that 50% of the jobs are submitted to the metascheduler), and for each job submission, we conduct a single Bernoulli trial with probability of success equal to f_g. Jobs with successful trials are routed through the metascheduler and the remaining jobs are considered as submissions to local batch system. We performed the experiments for $f_g = 0.5$, repeated each run 5 times and averaged the scores. In Figure 6, the green bars indicate the fairness scores for the 50% grid submissions and the red bars indicate the fairness scores for the 50% local batch queue submissions. We indicate the service fairness of the baseline strategy with a line which is labelled as BS. Service fairness scores more than 1 indicate improved response times compared to the baseline.

When all jobs are grid submissions, we can see that all systems have values more than 1 except Gordon. This indicates that jobs which originated at these systems obtained benefits

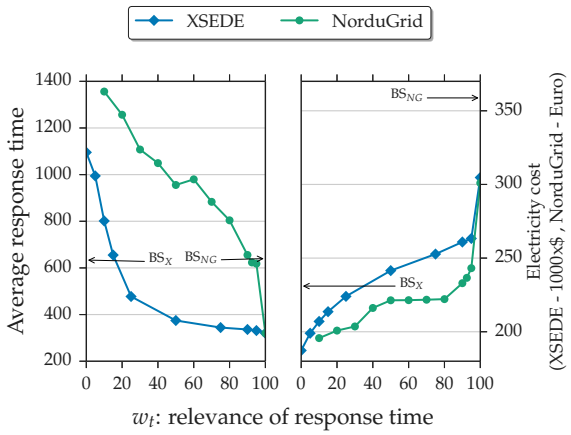

Fig. 7: Effect of varying w_t on response time and cost

in response time due to metascheduling. Gordon, which has service fairness slightly less than 1, is the fastest and 3rd cheapest system in the grid. Hence, our MCMF algorithm migrates many jobs to this system. But, the jobs processed at Gordon incur an average waiting time of only half an hour, which indicates that the users of this system did not suffer much due to grid participation. Jobs which originated at slow smaller sized systems like Queenbee, Mason and Blacklight, obtained large benefits from metascheduling.

When only a subset of the jobs are grid submissions, we see that both users of the grid and the local batch system obtained benefits in response time. Grid users obtained improved performance because of job migration. Local users obtained improved performance at systems like Queenbee, Mason and Blacklight because grid submissions were migrated away from these systems, leaving more resources free to process local submissions. Thus, we see that a system's participation in a grid which uses our metascheduling algorithm, provides benefits even for users who do not submit through the grid portal.

D. Sensitivity to w_t

Recall that w_t denotes the weight of the response time term in the cost function minimized by MCMF. Varying w_t allows the grid administrators to control the relative importance of minimizing response time vs minimizing total electricity cost. These objectives can be conflicting in the presence of daily fluctuations of electricity price. Figure 7 shows the effect of varying w_t in XSEDE and NorduGrid using 10000 jobs. For response time and electricity cost, BS_X and BS_{NG} represent the baseline value in XSEDE and NorduGrid, respectively. We see that increasing the relevance of response time (electricity cost) leads to a decrease in response time (electricity cost). We observed that when only response time is minimized ($w_t = 100\%$) we are able to obtain $48 - 49\%$ reduction in response time over the baseline in both XSEDE and NorduGrid.

Similarly, when only electricity cost is considered ($w_t = 0\%$), our scheduling strategy obtains 18% and 46% reduction in total electricity cost in XSEDE and NorduGrid, respectively. It is interesting to note that, in NorduGrid, for all values of w_t, our MCMF algorithm outperforms the baseline in terms of electricity cost. In XSEDE, we can see that for w_t values between 20-40%, both response time and electricity cost are better than the baseline. So, we selected $w_t = 25\%$ as the optimal value for XSEDE. In NorduGrid we selected $w_t = 92.5\%$. Compared to XSEDE, in NorduGrid, we require a high value of w_t to get improvements in response time. This is because the cost function minimized by MCMF is skewed depending on the magnitude of the response time and electricity cost. In NorduGrid, we observed that average runtime is 3.4x greater than XSEDE and the average electricity cost of a job is 177x lesser than XSEDE. Grid administrators can use a test workload to obtain these trends using our framework and decide an appropriate value of w_t depending on the budget and user service agreements.

VI. Related Work

Approaches which reduce power consumption by lowering CPU frequency or voltage [3] may not be widely and uniformly applicable across the entire grid due to the autonomous systems that are involved. Hence we do not describe related works which primarily employ such techniques to achieve power savings.

Single HPC system scheduling: The works of Yang et. al. [1] and Zhou et. al. [29] formulate the electricity price aware job scheduling problem for a single computing system as a 0-1 knapsack model. These works do not use hourly electricity prices. Instead, they consider two electricity price values corresponding to on and off-peak hours. Their algorithm is applied during peak hours to maximize utilization while maintaining the power consumption within a power budget that is specified a-priori. Our work considers hourly electricity pricing and have shown improvements over a strategy which uses only on-peak and off-peak prices.

Datacenter scheduling: The concept of *geographic load balancing* [6] has been used for distributing Internet traffic across distributed data centers. Qureshi et.al. [7] proposed electricity price aware request routing for Akamai's web traffic workload. Liu et. al. [6] proposed geographic load balancing of Hotmail traffic requests to achieve energy savings. Rao et. al. [5] use minimum cost flow for scheduling service requests in geographically distributed Internet data centers. Ren and He developed COCA [8], a scheduling framework which uses Lyapunov optimization to minimize operational cost of the data center while satisfying carbon neutrality constraints. This work uses one hour ahead electricity price prediction.

These approaches are applicable only for Internet data center workloads and not batch system workloads. They assume that requests are uniform with similar service times and employ techniques which use overall request arrival and service rate statistics. In a typical HPC or grid workload, requests/jobs are highly non-uniform in terms of running time, requested number of processors and queueing delay. These works consider that the request is serviced in the submission

hour and do not consider requests which require many hours or days of computation. Thus, the combination of workload and service policy used in HPC centers cannot be accurately modeled by these previous works. Our work predicts the execution period of a job using a history based queue waiting time predictor and considers actual/predicted electricity prices during this future period.

Grid scheduling: Mutz and Wolski [30], developed auction based algorithms for implementing job reservations in grid systems. Sabin et. al. [31] proposed a metascheduling algorithm based on the multiple simultaneous reservations at different systems in a heterogeneous multi-site environment. None of these previous works are cognizant of electricity price or job power characteristics. To our knowledge, ours is the first work on metascheduling HPC workloads across grid systems which optimizes both response time and electricity cost.

VII. Conclusions

In this paper, we presented a Minimum Cost Maximum Flow based formulation of the grid scheduling problem to optimize the total electricity price and average response time of HPC jobs in large scale grids operating in day-ahead electricity markets. Using two currently operational computational grids, we demonstrated that our algorithm can effectively use predictions of queue waiting time and electricity prices to optimize job placement across the grid. We compared our algorithm to aggressive baselines and showed that using hourly electricity pricing is more beneficial than approaches which use only fixed on and off-peak prices. We also showed most systems which participate in the grids which use our metascheduling algorithm, are able to offer improvements in response time for both grid and local users. In future, we plan to show the benefits of our metascheduling considering different prediction models for response times, and performing sensitivity studies with different prediction errors.

References

[1] 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*, ser. SC '13, 2013, pp. 60:1–60:11.

[2] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas, "Save Watts in Your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems," in *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, Dec 2008, pp. 171–178.

[3] N. Rizvandi, J. Taheri, A. Zomaya, and Y. C. Lee, "Linear Combinations of DVFS-Enabled Processor Frequencies to Modify the Energy-Aware Scheduling Algorithms," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010.

[4] I. Goiri, K. Le, M. Haque, R. Beauchea, T. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "GreenSlot: Scheduling energy consumption in green datacenters," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–11.

[5] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing Electricity cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[6] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew, "Greening Geographical Load Balancing," in *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '11, 2011, pp. 233–244.

[7] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the Electric Bill for Internet-Scale Systems," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09, 2009, pp. 123–134.

[8] S. Ren and Y. He, "COCA: Online Resource Management for Cost Minimization and Carbon Neutrality in Data Centers," in *Super Computing*, 2013.

[9] Extreme Science and Engineering Discovery Environment (XSEDE). [Online]. Available: https://www.xsede.org/

[10] NorduGrid. [Online]. Available: http://www.nordugrid.org/

[11] XSEDE: Metascheduling with Condor-G. [Online]. Available: https://www.xsede.org/web/guest/metascheduling-condor-g

[12] P. Murali and S. Vadhiyar, "An Adaptive Framework for Prediction of Queue Waiting Times in Supercomputer Systems," Middleware And Runtime Systems Lab, Technical Report: TR-MARS-SERC-IISC-3-2014, Oct. 2014. [Online]. Available: http://mars.serc.iisc.ernet.in/downloads/batchsystems/techreports/prakash-qwait-trmarsserc32014.pdf

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

[14] D. C. Nurmi, J. Brevik, and R. Wolski, "QBETS: Queue Bounds Estimation from Time Series," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '07, 2007, pp. 379–380.

[15] H. Li, J. Chen, Y. Tao, D. Gro, and L. Wolters, "Improving a Local Learning Technique for Queue Wait Time Predictions," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 335–342, 2006.

[16] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather Than User Runtime Estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[17] U. Lublin and D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *J. Parallel Distrib. Comput.*, vol. 63, no. 11, pp. 1105–1122, Nov. 2003.

[18] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Modeling User Runtime Estimates," in *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP'05, 2005, pp. 1–35.

[19] I. Kim and O. de Weck, *Structural and Multidisciplinary Optimization*, vol. 31, pp. 105–116, 2006.

[20] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[21] The Grid Workloads Archive. [Online]. Available: gwa.ewi.tudelft.nl/

[22] O. Peleg, "Python Scheduler Simulator," Feb. 2010. [Online]. Available: http://code.google.com/p/pyss/

[23] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel Job Scheduling: a Status Report," in *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP'04, 2005, pp. 1–16.

[24] D. L. Hart, "Measuring Teragrid: Workload Characterization for a High-performance Computing Federation," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 4, pp. 451–465, Nov. 2011.

[25] Federal Energy Regulatory Commission. [Online]. Available: http://www.ferc.gov/industries/electric/indus-act/rto.asp

[26] Nord Pool Spot. [Online]. Available: www.nordpoolspot.com/

[27] BSP SouthPool Regional Energy Exchange. [Online]. Available: www.bsp-southpool.com/

[28] S. Kamil, J. Shalf, and E. Strohmaier, "Power Efficiency in High Performance Computing," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–8.

[29] Z. Zhou, Z. Lan, W. Tang, and N. Desai, "Reducing Energy Costs for IBM Blue Gene/P via Power-Aware Job Scheduling," in *Job Scheduling Strategies for Parallel Processing*, ser. JSSPP, 2013, pp. 96–115.

[30] A. Mutz and R. Wolski, "Efficient Auction-based Grid Reservations using Dynamic Programming," in *SC*. IEEE/ACM, 2008, p. 16.

[31] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, "Scheduling of Parallel Jobs in a Heterogeneous Multi-site Environment," in *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP'03, 2003, pp. 87–104.