Quantitative Modeling of Power Performance Tradeoffs on Extreme Scale Systems

Li Yu, Zhou Zhou, Sean Wallace

Illinois Institute of Technology, Chicago, IL 60616

Michael E. Papka

Argonne National Laboratory, Argonne, IL, 60439

Zhiling Lan*

Illinois Institute of Technology, Chicago, IL 60616

(Phone) 312.567.5710, (Fax) 312.567.5067

Abstract

As high performance computing (HPC) continues to grow in scale and complexity, energy becomes a critical constraint in the race to exascale computing. The days of "performance at all cost" are coming to an end. While performance is still a major objective, future HPC will have to deliver desired performance under the energy constraint. Among various power management methods, power capping is a widely used approach. Unfortunately, the impact of power capping on system performance, user jobs, and power-performance efficiency are not well studied due to many interfering factors imposed by system workload and configurations. To fully understand power management in extreme scale systems with a fixed power budget, we introduce a power-performance modeling tool named *PuPPET (Power Performance PETri net)*. Unlike the traditional performance modeling approaches such as analytical methods or trace-based simulators, we explore a new approach – colored Petri nets – for the design of PuPPET. PuPPET is fast and extensible for navigating through different configurations. More impor-

*Corresponding author

Email addresses: {lyu17,zzhou1,swallac6}@hawk.iit.edu (Li Yu, Zhou Zhou, Sean Wallace), {papka}@anl.gov (Michael E. Papka), {lan}@iit.edu (Zhiling Lan)

tantly, it can scale to hundreds of thousands of processor cores and at the same time provide high levels of modeling accuracy. We validate PuPPET by using system traces (i.e., workload log and power data) collected from the production 48-rack IBM Blue Gene/Q supercomputer at Argonne National Laboratory. Our trace-based validation demonstrates that PuPPET is capable of modeling the dynamic execution of parallel jobs on the machine by providing an accurate approximation of energy consumption. In addition, we present two case studies of using PuPPET to study power-performance tradeoffs on petascale systems.

Keywords: high performance computing, power performance analysis, colored Petri net, extreme scale systems, power capping

---

1. Introduction

Production petascale systems are being designed and deployed to meet the increasing demand for computational cycles made by fields within science and engineering. With their growing performance, energy consumption becomes an important concern. It is estimated that the energy cost of a supercomputer during its lifetime can surpass the equipment itself [5]. This introduces the need for energy-efficient computing. A number of power management technologies have been presented [2, 40], and power capping (i.e., limiting the maximum power a system can consume at any given time) is a well-known approach. For instance, power-aware job allocation and dynamic voltage and frequency scaling (DVFS) are two common power capping mechanisms. To control the peak power within a predefined threshold, the former controls the overall system power by dynamically allocating available resources to the queued jobs according to their expected power requirements [50], while the latter limits the overall system power by adaptively adjusting processor voltage and frequency (DVFS) [26].

While power-aware job allocation and DVFS control the maximum system power through different mechanisms, they both inevitably degrade performance such as system utilization rate, job wait time, etc. Understanding the performance impact of different capping methods is critical for the development of effective power management methods, especially in a system of unprecedented size and complexity. Important questions arising in this context include: *for a given system and workload, which is an appropriate method? And what is the potential impact of power management on performance?* These questions must be answered and their importance grows with the increasing scale of high performance computing. Nevertheless, these are intrinsically difficult questions for many reasons. The optimal power management scheme depends on many factors, such as job arrival rate, workload characteristics, hardware configuration, power budget, and scheduling policies. For DVFS, other relevant factors include the processor power-to-frequency relationship and the workload time-to-frequency relationship. For power-aware job allocation, perhaps the most important factor is the ratio between power at idle processor state versus that at full processor speed. Moreover, these factors are often interrelated, making the analysis even harder. It is simply impossible to examine all these factors via experiments on production systems. To fully understand the impact different power capping mechanisms have on performance, we have developed a new analysis approach, which we

describe in this paper, to predict system performance as a function of these factors.

Models are ideal tools for navigating a complex design space and allow for rapid evaluation and exploration of detailed what-if questions. This motivates us to look for a modeling method that satisfies three basic requirements: (1) *scalability* - it should be capable of modeling extreme scale systems with low overhead; (2) *high-fidelity* - it should be accurate enough to quantify the power-performance tradeoffs introduced by different external factors; (3) *extensibility* - it should be flexible for easy expansion such as adding different system configurations and/or new functionalities.

Existing modeling methods can be broadly classified into three categories: analytical modeling, simulation and/or emulation, and queuing modeling [24]. Unfortunately, none of them meets the above three requirements. Analytical modeling methods are fast, but they only provide rough predictive results, thus do not satisfy the second requirement (i.e., high-fidelity). Further, analytical modeling cannot capture dynamic changes (e.g., dynamic job submission and execution, dynamic frequency tuning). Trace-based simulators can provide highly accurate representations of system behaviors such as dynamic job scheduling [36, 49]. However, no existing simulator has been extended to study power-performance tradeoffs on large-scale systems. Such a functional extension is not trivial, and would require a significant amount of engineering efforts. In other words, this approach does not satisfy the third requirement (i.e., extensibility). The conventional queuing methods (e.g., Markov modeling) have been investigated to deal with complicated programs; however, they suffer from the state explosion problem, thus cannot satisfy the first requirement (i.e., scalability).

In this paper, we explore a new modeling approach by presenting a colored Petri net named *PuPPET (Power Performance PETri net)* for quantitative study and predictive analysis of different power management mechanisms on extreme-scale systems. Colored Petri net (CPN) is a discrete event modeling formalism combining the strengths of Petri nets with the expressive power of programming languages [28, 22]. Petri nets provide the graphical notation for system abstraction, whereas the programming languages offer the primitives for data definition and data value manipulation. Here, we list several reasons why PuPPET is able to satisfy the aforementioned requirements and leave more detailed descriptions of CPN in Section 2.1.

- *Scalability.* The hierarchy and color supports offered by colored Petri

net make it possible to model large systems in a modular way. Unlike conventional queuing methods, PuPPET's model size does not grow dramatically as the number of system components increases. This feature is critical for the modeling of extreme-scale systems targeted in this work. As we will show later in our experiments, PuPPET can model peta-scale systems with low modeling overhead.

- *Accuracy.* By combining the capabilities of Petri nets with high-level programming languages, PuPPET can provide a very precise abstraction of system behaviors, thus enabling an accurate simulation of real systems. Job related factors such as job arrival time, job size and job runtime can be described by language parameters directly, whereas the interactions among these factors can be captured by the graphical notation in an intuitive way.

- *Extensibility.* PuPPET enables us to add, modify or remove the functional modules and their interactions easily. Moreover, the availability of various well-developed Petri net simulation tools allows us to build models at a high level, hence making the model construction and extension much more convenient and faster.

We validate the accuracy of PuPPET by using the system traces (i.e., workload log and power data) collected from the production 10-petaflops IBM Blue Gene/Q system named *Mira* at Argonne National Laboratory. Our experiments show that PuPPET can effectively model batch scheduling and system energy consumption with a high accuracy, e.g., an error of less than 4%. The emulation of executing four-month jobs on Mira took a couple of minutes on a local PC. Given that Mira is a petascale machine with 49,152 nodes, this result demonstrates that PuPPET is highly scalable.

We also present two case studies to explore the use of PuPPET for power performance analysis. We study the performance impact of different power capping strategies under a variety of system configurations and workloads. PuPPET enables us to study the effects of a variety of factors, such as workload characteristics, processor power-to-frequency relationship, workload time-to-frequency relationship, as well as the idle-to-full power relationship, in a unified analysis environment. These case studies provide us useful insights of using power capping on extreme scale systems. For example:

- Given a fixed power cap, both power-aware allocation and DVFS can effectively restrict the overall system power within the limit. While

both mechanisms trade performance for power saving, the degrees of performance degradation are not the same. Although DVFS seems to cause less impact on system performance, it could significantly impact hardware lifetime reliability, as high as up to 3X higher failure rate [38]. Hence, unless in case of a tight power cap and high system utilization, power-aware allocation is a preferred power capping solution due to its comparable performance with DVFS and no impact to system reliability.

- Workload characteristics influence the power-performance efficiency of power capping. In our experiments, greater performance impact is observed in the months where system utilization is relatively high.

- Although batch scheduling policy does not cause substantial power-performance difference when applying power capping, we observe that the scheduling policy adopted by *Mira* is more robust to system performance degradation caused by power management than the well-known first-come, first-served (FCFS) policy, especially with respect to job wait time.

While this paper focuses on power-performance analysis of power capping methods, PuPPET is extensible and can be easily augmented to analyze other architectures, different power management mechanisms, other hardware scaling like memory, or adding new constraints such as reliability. The extension can be achieved by adding new modules. These modules can be integrated or individually disabled for studying different scenarios, thereby providing a very powerful tool for modeling and analyzing extreme scale systems.

The rest of the paper is organized as follows. Section II presents background information. Section III describes our model design. Section IV presents model validation by means of real traces. Section V presents two case studies of using the model. Section VI discusses related work. Finally, we conclude the work in Section VII.

2. Background

Before presenting our model, we provide the necessary background information in this section.

2.1. Colored Petri Net

Petri nets are a modeling formalism for describing systems that are concurrent, asynchronous, distributed, parallel, or stochastic [6]. It combines an intuitive graphical notation with a number of analysis techniques based on a solid mathematical foundation. Petri nets are depicted by weighted and directed graphs, consisting of *places* and *transitions* connected by *arcs*, with *tokens* in places. Places are used to represent system states, whereas transitions are used to represent system events. To indicate the change of system states, tokens (represented as black dots) move from one place to another via the firing of a transition. The firing of a transition transfers tokens from its pre-places to its post-places according to the weights of the outgoing arcs. A transition is ready to fire only when each of its pre-places contains at least the number of tokens specified by the weight of the corresponding arc. Figure 1 illustrates the basic elements used in Petri nets.

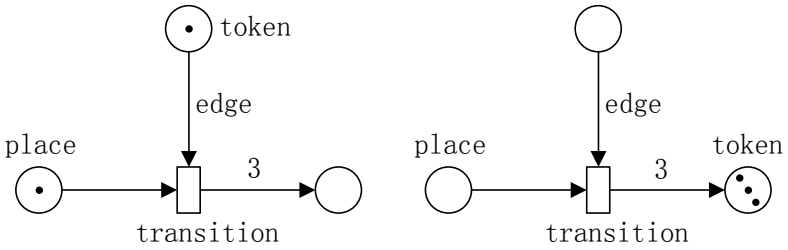
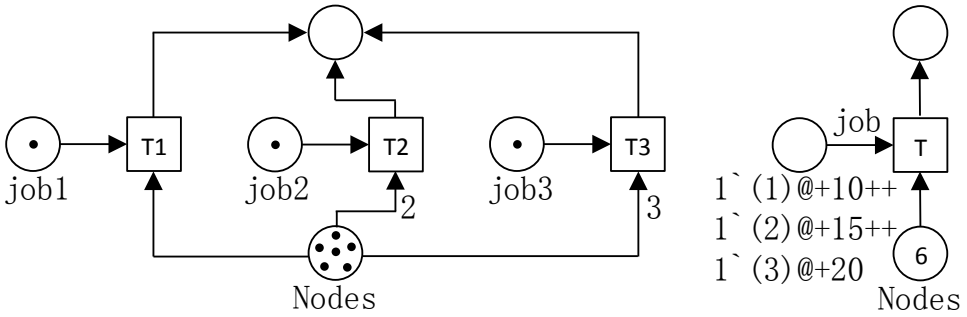


Figure 1: Basic elements used in Petri nets. The left figure indicates the system state before the firing of the transition, and the right figure indicates the state after the firing. Note that the arc weight 1 is not labeled explicitly.

Colored Petri net (CPN) is a recent advancement in the field of Petri nets [23]. By taking advantage of high-level languages, CPN features several extensions over the traditional Petri nets. First, it allows tokens to have a data value attached to them. This attached data value is called token color. As such, tokens become distinguishable according to their colors. Moreover, the transitions in CPN can add, remove or change the color of tokens. These two features improve the expressibility of Petri nets greatly, thus enabling CPN to model complicated system behaviors (e.g., the change of job attributes such as job size, job runtime, job power, etc.) in HPC. Second, tokens or transitions in CPN can be associated with *time*. Timed transitions can fire according to a deterministic delay or a stochastically distributed random variable. This extension provides an accurate control of timing for system

modeling. Third, CPN allows a *hierarchical design*, in which a module at a lower level can be represented by a transition at a higher level. This feature provides a compact model design, and also enables a higher level of scalability. Additional details about CPN can be found in [23]. Figure 2 presents an example to illustrate the major differences between the traditional Petri nets and colored Petri nets.



(a) Petri nets (b) Colored Petri net

Figure 2: An example of comparing Petri nets with colored Petri nets. They are used to model three jobs running on a 6-node system. In the left model, three jobs are represented by three tokens in different places, and their requested numbers of nodes are represented by the weights of arcs from *Nodes* to the transitions. In the right model, the jobs in CPN are distinguished by their colors, i.e., denoted by a value (1), (2), and (3). Also, CPN allows the control of timing (expressed by "@+").

Although CPNs have been widely used for modeling large scale systems like biological networks [31], *to our knowledge this is the first attempt of applying CPN for quantitative power-performance modeling in high performance computing.*

2.2. Power Consumption

For a computing node in a system, the power is mainly consumed by its CMOS circuits, which is captured by

$$P = V^2 \times f \times C_E, \tag{1}$$

where V is the supply voltage, f is the clock frequency, and C_E is the effective switched capacitance of the circuits. According to different environments, the power consumption can be approximated by

$$P \propto f^{\alpha}, \tag{2}$$

where *the frequency-to-power relationship* α typically falls into the scope from one to three [30]. This implies that lowering the CPU speed may significantly reduce the power consumption of the system. However, lowering the CPU speed also decreases the maximum achievable clock speed, which leads to a longer time to complete an operation. Thus, the time to finish an application is inversely proportional to the clock frequency and can be represented as

$$t \propto \frac{1}{f^{\beta}}. \tag{3}$$

where β is *the frequency-to-time relationship*.

Note that the change of CPU speed only affects the time consumed by the frequency-dependent part (e.g., computation) of the application, and the time consumed by the frequency-independent part (e.g., communication) remains unchanged [29]. In the case that an application is 100% frequency-dependent, β can be approximated by 1.0. Studies have shown that the frequency-independent portion could be as high as 40% [10]. According to this ratio, in this study, we set the default value of β to 0.5, and a sensitivity study of β will be provided in Section 5.1

It is worth noting that Equation 2 is only a core-level power model. For an application running on a machine, besides the power consumed by cores, other components such as memory, network, I/O and so on also consume power. For typical applications running on Mira at Argonne, our recent study has found that chip cores contribute to more than 60% of the power consumption [45]. In this study, the change of CPU frequency only influences the power consumed by chip cores, and the default ratio of chip core power to the total power is set to 65%.

2.3. Batch Scheduling

Batch scheduling is typically used for resource management and job scheduling in HPC, where parallel jobs are assigned to disjoint processor resources (i.e., space-sharing) according to resource availability and certain job ordering. Figure 3 illustrates typical batch scheduling on supercomputers. The resource manager is responsible for obtaining information about resource availability, waiting queue, and the running status of compute nodes. It runs several daemon processes on the master nodes and compute nodes to collect

such information. The job scheduler communicates with the resource manager to make scheduling decisions based on the current system status. The job waiting queue receives and stores jobs submitted by users. Users can query the resource manager to get the status of their jobs.

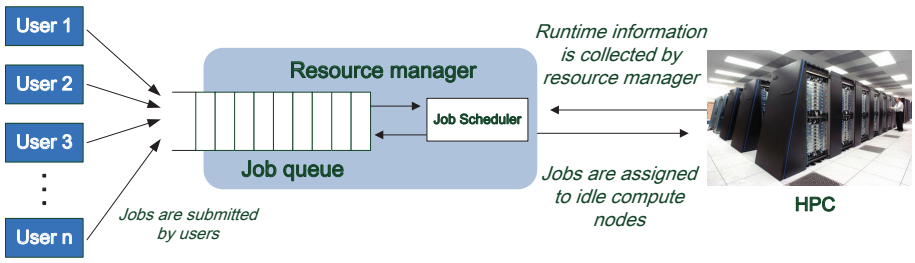


Figure 3: Batch scheduling in HPC.

The job scheduler periodically examines the jobs in the waiting queue and the available resources via the resource manager, and determines the order in which jobs will be executed. The order is decided based on a set of attributes associated with jobs such as job arrival time, job size (i.e., the number of nodes needed), the estimated runtime, etc. First-come, first-served (FCFS) with EASY backfilling is a commonly used scheduling policy in HPC [14]. Under FCFS-EASY, jobs are served in first-come, first-served order, and subsequent jobs continuously jump over the first queued job as long as they do not violate the reservation of the first queued job.

In this paper we also study another scheduling policy named WFP (named for the United Nations World Food Programme), which is designed to avoid large job starvation on IBM Blue Gene systems including the current Blue Gene/Q at Argonne[42, 41]. Unlike FCFS that determines job ordering based on their arrival times, WFP uses a utility function as defined in Equation 4 to determine job ordering. It favors large and old jobs, adjusting their priorities based on the ratio of wait time to their requested wall clock times. WFP with EASY back-filing works as follows: when a job arrives at or leaves from the system, all the queued jobs are sorted according to Equation 4; subsequent jobs may be scheduled over the first queued job as long as they do not violate the reservation of the first queued job.

$$job\_size \times (\frac{job\_queued\_time}{job\_runtime})^3. \tag{4}$$

3. PuPPET Design

PuPPET consists of three interacting modules, namely, *batch scheduling*, *power-aware allocation* and *CPU scaling*, to model user jobs from their submission to their completion. Figure 4 shows the top level design, in which these modules are connected through five states (i.e., places). *Batch scheduling* orders the jobs in the queuing state according to a scheduling policy, e.g., job arrival time (FCFS) or utility score (WFP), and allows small jobs to skip ahead provided they do not delay the job at the head of the queue (i.e., backfilling). *Power-aware allocation* provides coarse-grained power capping by allocating queued jobs onto computer nodes based on the overall system power status and job power requirement. In this work, we develop a net to describe two power-aware allocation strategies for power capping.[1] *CPU scaling* dynamically adjusts the processors' clock frequency for fine-grained power capping. It interacts with the jobs in the running state. Dynamic system states are modeled by job submission, job queuing, job allocation, node allocation, and power state changing.

The five states possess different system information. *Queuing* keeps a list of queued jobs, whose order can be changed dynamically by the *batch scheduling* module. *Running* holds a set of jobs in the running state, where job execution is intercepted by the *CPU scaling* module. *Power* indicates the current power level of the system, based on which *power-aware allocation* module or *CPU scaling* module is able to conduct power capping. *Runtime Info* provides system resource information to the *batch scheduling* module. *Nodes* represents the number of available computer nodes. We give the details of these modules and states in the following subsections, using the inscriptions summarized in Table 1.

3.1. Batch Scheduling

This module is used to model batch scheduling on HPC systems. Figure 5 presents the net design for FCFS and WFP, along with EASY backfilling. As mentioned in Section 2, FCFS-EASY is a widely used batch scheduling policy, and it is estimated that more than 90% of batch schedulers use this as default scheduling configuration [32]. WFP-EASY is the production

[1] In this paper we distinguish job scheduling from job allocation: job scheduling focuses on ordering user jobs, whereas job allocation is dedicated to assigning the queued jobs onto available resources.

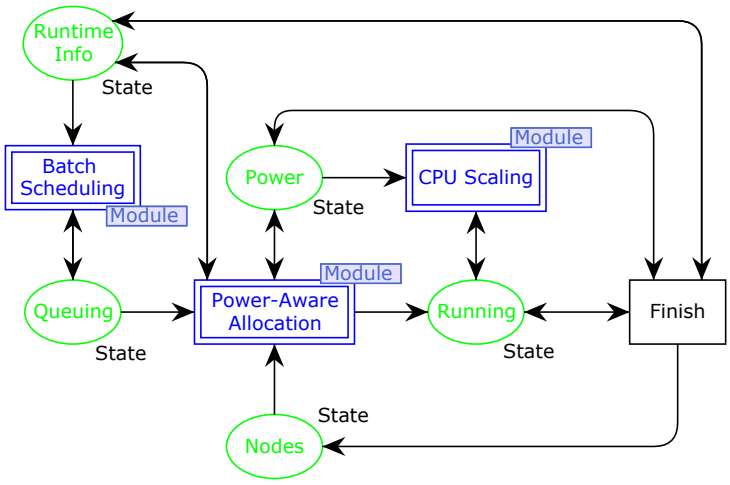


Figure 4: Top level design of PuPPET. There are three hierarchical modules (denoted by double border boxes in blue) and five states (denoted by ellipses in green).

batch scheduling policy used on a number of production supercomputers at Argonne, including the current 48-rack Blue Gene/Q machine.

As shown in Figure 5, jobs leave from the place *User* in the net according to their arrival times. Every job is described in the form of $(js, rt, jp, et, pf)@+st$, where js is job size, rt is job runtime, jp is job power profile, st is job arrival time, et records the time when a job enters a new place, and pf indicates the frequency rate of the processor that a job runs on. Here, js and rt are supplied by users, jp is an estimate that can be obtained from historical data [48], and all the others are maintained by PuPPET.

The job list in the place *Queuing* accepts the jobs submitted from users. Once a job enters or leaves the system (i.e., firing of *Submit* or *Finish*), the place *Trigger* receives a signal and launches a job scheduling process. For FCFS-EASY, the transition *EASY* fires and uses the function *Backfill* to "backfill" jobs according to runtime resource information from the place *Runtime Info*. For WFP-EASY, jobs are sorted by their utility scores (i.e., firing of *WFP*) before backfilling starts.

3.2. Power-Aware Allocation

This module intends to model coarse-grained power capping by allocating queued jobs onto available nodes based on the system power status and job power requirement. Specifically, for each job at the head of the queue,

Inscriptions	Type	Description
job	variable	A job has six attributes: js, rt, jp, et, pf and st.
jobs,ws	variable	A job list.
pow,old_pow	variable	System power level at the current and the previous time steps.
P1,...,Pm	variable	Processor power rates.
f,pf,new_f,f1,...,fm	variable	Processor frequency rates.
[expr]	symbol	A guard on a transition that is able to fire if expr evaluates to true.
@+ expr	symbol	A time delay specified by expr.
T()	function	The current model time.
WFP()	function	Job scheduling using WFP.
Backfill()	function	Job scheduling using backfilling.
PowAllocate()	function	Search for jobs satisfying power cap from the wait queue.
Update()	function	Updating job execution time when CPU speed changes.

Table 1: Major inscriptions used in the PuPPET modules.

if its estimated power requirement makes the overall system power exceed the power cap, the allocator either blocks job allocation or opportunistically allocates subsequent less power-hungry jobs:

- BLOCK: the first queued job and its successors are blocked until there is power available to execute it (e.g., when a job finishes and leaves the system).
- WAIT: the first queued job is held in a wait queue and yields to other less power-hungry jobs, until its wait time exceeds a predefined value or the wait queue is full.

Figure 6 presents the net design. We use *Pcap* to represent the power cap imposed on the system, w to indicate the maximum wait time of a job in the wait queue, and l to restrict the length of the wait queue. The module is centered around three transitions: *Allocate*, *Wait* and *Allocate W*. The head job in the place *Queuing* is either being allocated onto nodes or being held in

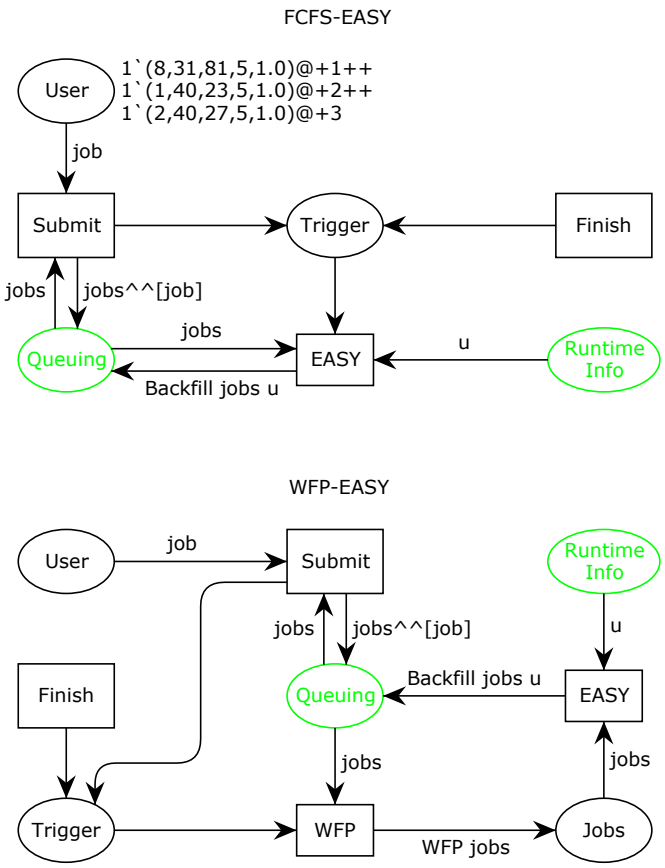


Figure 5: Batch scheduling module

the wait queue. The transition *Allocate* fires if the job's power requirement does not make the overall system power exceed the power cap; otherwise, the transition *Wait* fires as long as the wait queue is not full. Jobs in the wait queue can be allocated onto computer nodes by the firing of the transition *Allocate W* which is set to a higher firing priority than the transition *Allocate*.

To model the BLOCK strategy, we set the wait queue length l to zero. If the head job in the place *Queuing* makes the total system power exceed the power cap, it and its successors are blocked until the power cap is satisfied. Similarly, the WAIT strategy is modeled by setting l to a positive integer. If the head job breaks the power cap, it is moved to the place *Wait Queue*. Once system power changes, the transition *Power Change* fires and starts a selection process for the jobs in the wait queue. The function *PowAllocate*

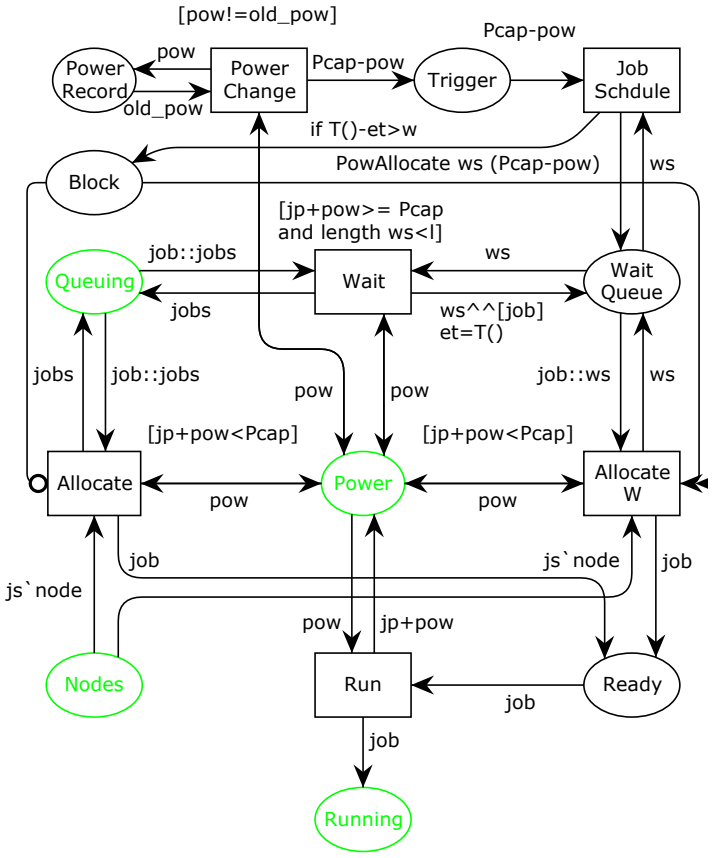


Figure 6: Power-aware allocation module

selects some jobs that can be allocated under the current system power and puts them at the head of the wait queue. However, if any job in the wait queue exceeds the maximum wait time (denoted by w), the job is kept at the head and a signal is sent to the place *Block*. The successors are also blocked until the job at the head of the wait queue is allocated.

Note that the original batch scheduling is modeled by simply setting $Pcap$ to ∞, indicating that the transition *Wait* will never fire and the jobs in the place *Queuing* are allocated as long as there are sufficient computer nodes.

3.3. CPU Scaling

This module models fine-grained power capping through dynamic voltage and frequency scaling (DVFS). Specifically, in the case that a job arrival

makes the total system power exceed the power cap, DVFS is explored to make the processors run at a lower power state, thus limiting the power within the cap.

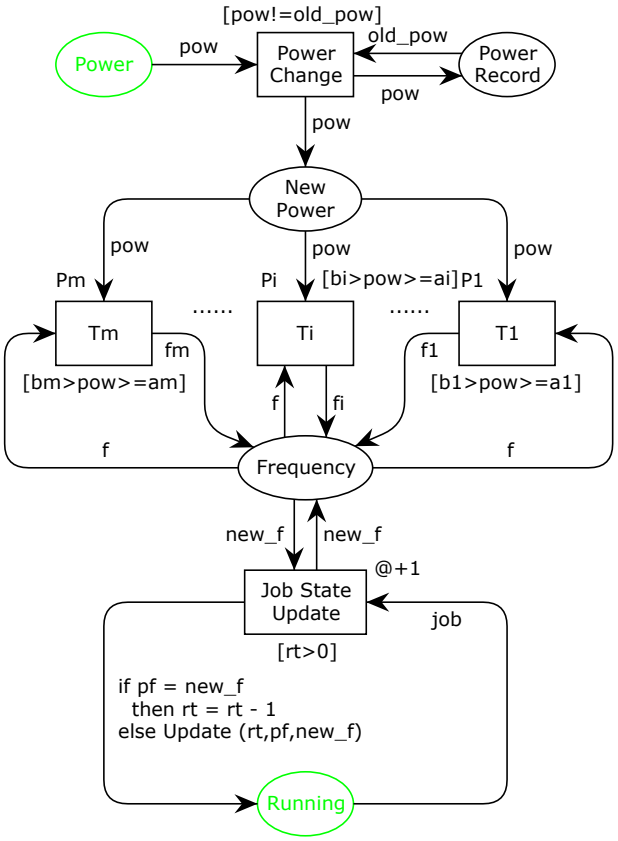


Figure 7: CPU scaling module.

Figure 7 presents the detailed net design. When system power changes due to job arrival or leaving, the power indicated by the place *New Power* will be updated. According to this new system power and the power cap, one of the transitions in $\{T1, T2, ..., Tm\}$ fires, meaning that the processors are going to run at power rate P_i. At the same time, the processor frequency rate and the remaining job execution time is updated according to the power models listed in Section 2.2. Note that we assume there is no latency involved in CPU scaling.

3.4. PuPPET Implementation

We use CPN Tools to construct PuPPET. CPN Tools is a wide-spread tool for editing, simulating, and analyzing colored Petri nets [3]. It is free of charge, and has been in active development since 1999. More importantly, it provides much faster simulation capabilities as compared to other tools we have investigated. PuPPET can be downloaded from our server *http://bluesky.cs.iit.edu/puppet/*. It is directly executable by an appropriate tool like CPN Tools. Currently, PuPPET accepts job traces in the standard workload format adopted by the scheduling community [1]. PuPPET can be simulated interactively or automatically. In an interactive simulation the user is in control. It provides a way to walk through the model, and checking whether the model works as expected. Automatic simulations are similar to job execution.

4. Model Validation

We believe the best way to validate PuPPET is by means of real system traces including both workload log and power data collected from production systems. Unfortunately, these data are not generally accessible. By working with the operational team at Argonne Leadership Computing Facility, we were able to gather a workload log and a corresponding power data from its production IBM Blue Gene/Q system named *Mira*. *Mira* is a US Energy Department petascale resource in support of scientific research. It consists of 48 racks, each containing two mid-planes, eight link cards, and two service cards. A mid-plane contains 16 node boards and each node board holds 32 compute cards. The entire system contains $49,152$ nodes and $786,432$ cores, offering a peak performance of 10 petaflops. The machine uses *Cobalt* for its resource management and job scheduling [42]. *Cobalt* is an open-source, component-based job management suite developed by Argonne and has been used on a number of production Blue Gene systems at Argonne. It adopts the WFP policy described in Section 2 for batch scheduling.

We collected a job log from *Mira* between January and April 2013. In these four months, $16,044$ jobs were executed on the machine. A summary of these jobs is presented in Figure 8. During these months, the machine was going through acceptance testing and a large number of early science applications were submitted from users, hence there are a large number of small sized jobs. As shown in the figure, the least number of jobs were submitted in February. While there are similar number of jobs in the other

three months, most of the jobs in January are small-sized jobs (i.e., less than 8k). In other words, a distinguishing workload difference for these months is that system utilization is low in January and February and high in March and April. More specifically, the system utilization is about 35% higher in March and April, compared to January and February. As we will show later in this paper, this workload difference impacts the effect of power management. Furthermore, *Mira* was deployed with power monitoring sensors that collect and store power-related data in its environmental database [45].

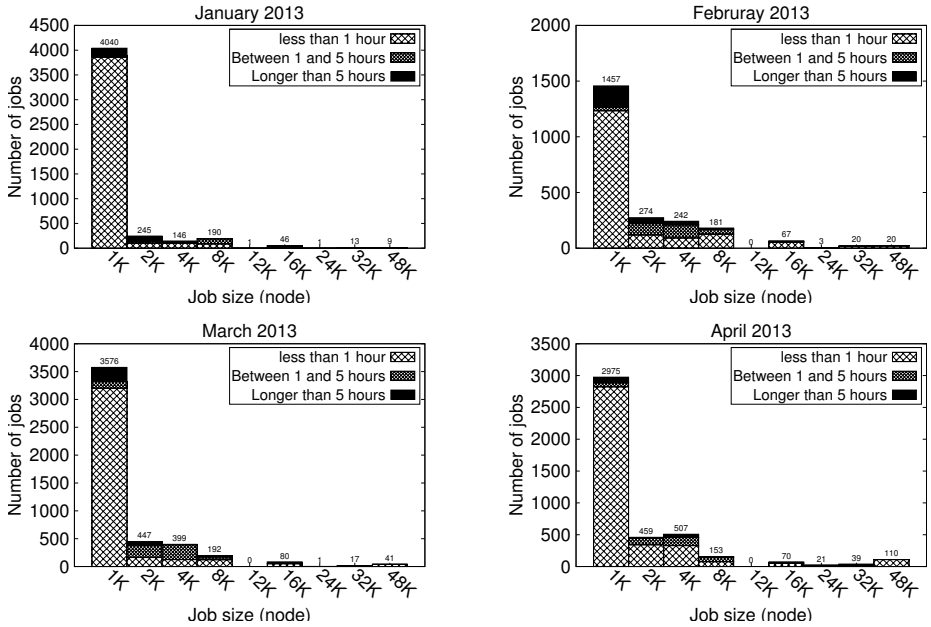


Figure 8: Summary of job traces

From the job log, we extracted job attributes including job size *js*, job runtime *rt*, and job arrival time *st* of all the 16,044 jobs. For each job, its power profile *jp* was obtained by correlating the job log with the environmental log. We measured job size in number of racks, job time in minutes, and job power profile in watts. We also rounded these measured numbers to the nearest integers for the purpose of modeling. The default model simulation time step is set to one minute. The energy consumption within the time step $[t, t+1]$ is approximated as the number of tokens in the place *Power* (see Figure 4) at time t. For example, if at the 5th minute the number of tokens in *Power* is 600, then the energy usage during the following time step

is estimated as $600/1000 * 1/60 = 0.01kWh$.

For model validation, we compared the modeled energy consumption to the actual energy consumption extracted from the environmental data. Figure 9 plots the daily energy consumptions obtained from PuPPET and the actual power data from the environmental database. Note that each data point in the figure indicates the sum of the energy consumed in that day. The plot clearly shows that PuPPET is highly accurate, with an average error of less than 4%. We also compared scheduling metrics (e.g., system utilization rate, average job wait time, etc.) obtained from PuPPET and the job log, which leads to similar results as energy consumption. We did not show them here due to the limit of space. Together these results indicate that PuPPET can effectively model dynamic job scheduling and allocation with high fidelity.

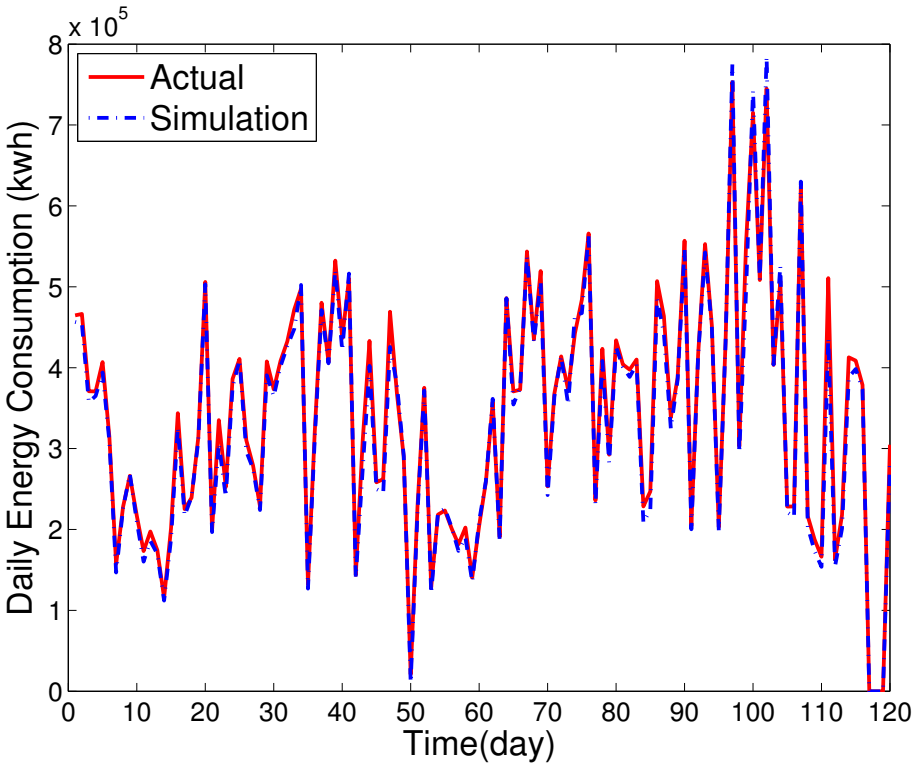


Figure 9: Model validation: daily energy consumption.

Model accuracy and simulation cost are influenced by simulation time

step. We have conducted a set of experiments to evaluate the impact of different simulation time steps, and the results are shown in Table 2. As shown in the table, we vary the simulation time step from 1 minute to 60 minutes, and assess the maximum energy error brought by PuPPET, along with the simulation cost. Obviously, a smaller time step means higher accuracy; nevertheless, a smaller time step also introduces higher simulation overhead. All the experiments were conducted on a local PC that is equipped with an Intel Quad 2.83GHz processor and 8GB memory, running Windows 7 Professional 64-bit operating system. The table demonstrates that PuPPET is fast: for the emulation of the 4-month workload on the $49,152$-node machine, the simulation takes a couple of minutes to complete. For the case of modeling *Mira*, we believe the selection of a time step between $1-15$ minutes provides a good balance between model accuracy and simulation overhead.

Simulation Time Step (min)	Maximum Energy Error	Simulation Overhead
1	3.84%	5.40 min
5	5.03%	5.25 min
15	8.85%	4.95 min
30	15.32%	4.60 min
60	18.09%	4.40 min

Table 2: Impact of simulation time step on model accuracy and simulation overhead

5. Case Studies

We have validated the proposed PuPPET model, and now we proceed to use PuPPET for predictive analysis. In particular, we present two case studies where PuPPET is used to study power-performance tradeoffs under different power capping mechanisms. We analyze both fine-grained (e.g., DVFS) and coarse-grained (e.g., power-aware allocation) power capping strategies. Our study evaluates these strategies by navigating through different configuration spaces. Specifically, three power-performance metrics are used for analysis:

- *System Utilization Rate.* This metric represents the ratio of node hours used by user jobs to the total elapsed system node hours. It is a commonly used metric for evaluating system performance.

- *Average Job Wait Time.* This metric denotes the average time elapsed between the moment a job is submitted to the moment it is allocated to run. This metric is used to measure scheduling performance from the user's perspective.

- *Energy Delay Product (EDP).* This metric is defined as the product of the total energy consumption and the makespan of all the jobs (i.e., the total length of the schedule when all the jobs have finished processing). It is commonly used to measure power-performance efficiency. Obviously, a lower value means better power-performance efficiency [25].

In our case studies, the default setting is as follows: the power cap is set to 45,720kW (i.e., 70% of the maximum power 65,314kW in January), α is set to 2, β is set to 0.5, and the idle processor power is set to be 30% of the power at the full speed. Note that the selection of power cap is based on the previous studies [50, 7, 9] which generally set a power cap from 40% to 80% of the maximal power.

5.1. Case Study 1: Power Capping via DVFS

In this study, we present a case study of using PuPPET to analyze fine-grained power capping via DVFS. In our experiments, we assume there is no latency involved in DVFS tuning. The power-performance efficiency of DVFS is influenced by many factors such as workload characteristics, hardware configuration, and scheduling policies. In order to investigate the impact of different factors on DVFS, we conduct three sets of experiments.

In the first set, we consider various scenarios in which the processors have different *frequency-to-power relationships* (i.e., α in Equation 2). According to the literature and industry reports [16, 33], we found that for most modern processors, the relationship typically falls between linear and cubic. Hence, we examine three different cases: linear ($\alpha = 1$), square ($\alpha = 2$) and cubic ($\alpha = 3$). Figure 10 presents the results, from which we make several interesting observations. First, as expected, in the case of cubic relationship, voltage and frequency tuning has the smallest impact on job execution time, thereby leading to less impact on system performance.

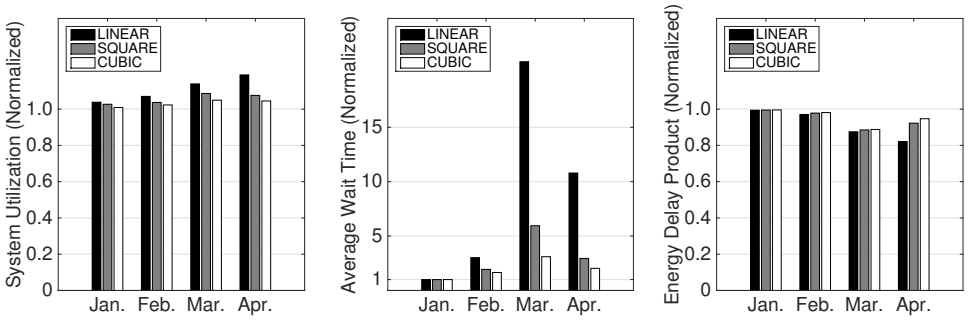
Second, with both batch scheduling policies (FCFS and WFP), DVFS improves system utilization and energy delay product, while increasing average job wait time. In order to limit the overall system power within the cap, DVFS extends job execution times (as the processors do not run at full

speed), thereby prolonging job wait time and improving system utilization. Although extending job execution times increases the scheduling makespan, DVFS reduces the amount of energy required for jobs at the same time (note that for linear relationship, energy consumed by jobs does not change). Further, as DVFS improves system utilization rate, the energy consumed by idle nodes is reduced. This offsets the cost introduced by the makespan extension, thus resulting in better power-performance efficiency. Third, the amount of system performance impact caused by DVFS is greatly influenced by workload characteristics. Specifically, more performance change is observed in March and April than in January and February. As stated earlier, the system utilization rate is higher in March and April.

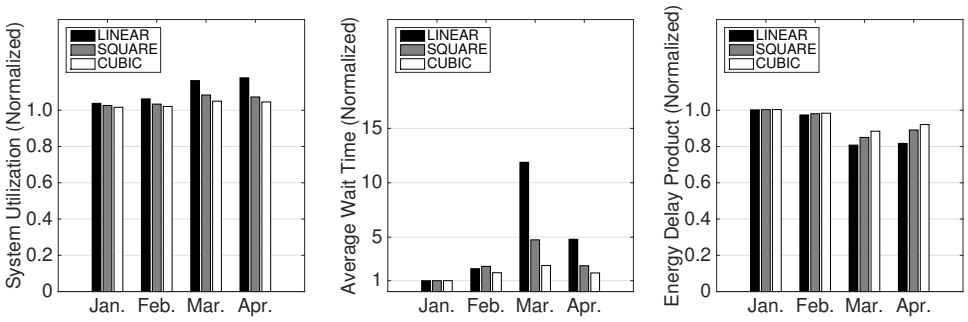
By comparing the top plots with the bottom plots, we can see that the trend of performance change is similar by using FCFS and WFP. An interesting observation is that the increase of average job wait time using WFP is not as significant as that using FCFS, or in other words, WFP is more robust to system performance impact caused by DVFS than FCFS. One reason is that as shown in Equation 4, WFP considers job queued time when ordering jobs for execution.

In the second set, we study the impact of different *power caps*, i.e., from 80% to 40% of 65,314kW (the maximum power in January). Figure 11 presents the normalized results to the cases without applying DVFS. Here, we use the default square frequency-to-power relationship. DVFS tends to increase system utilization rate when a tighter power cap is imposed. Similar to the first set of experiments, workload characteristics influence power-performance efficiency caused by DVFS (e.g., higher performance impact in March and April than that in January and February). An important observation we make from this set of experiments is that tighter power budget (e.g., 40%) could lead to worse EDP, especially in the case of high system utilization.

In the third set, we consider the scenarios where the workload has different *frequency-to-time relationships* (i.e., β in Equation 3). Figure 12 presents the results. As we can see, a higher β value does not increase EDP as excepted. As shown in Equation 3, a higher β value means a longer execution delay when CPU frequency is lowered, thus increasing the scheduling makespan. On the other hand, a longer execution delay increases system utilization and hence saves the energy consumed by idle nodes. This observation implies that the idle power (which is set to 30% of the node power in active state) is a dominant factor for system power-performance efficiency (e.g., EDP here).



(a) FCFS-EASY w/ DVFS



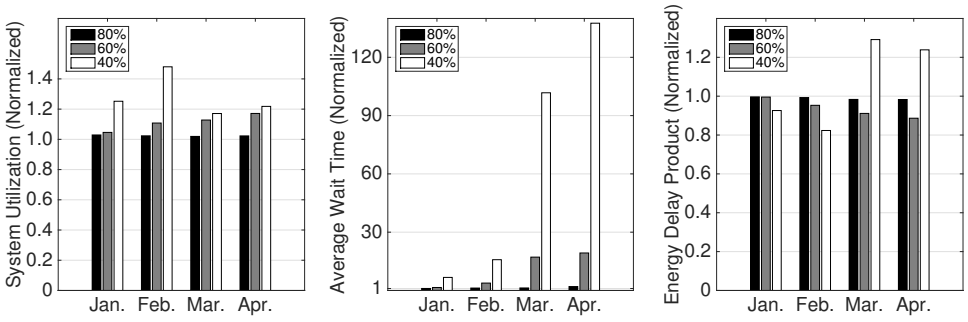
(b) WFP-EASY w/ DVFS

Figure 10: Power-performance impact under varied frequency-to-power relationships (i.e., varying α of Equation 2 from 1.0 to 3.0) by using DVFS for power capping. (1) Top: FCFS-EASY, all the results are normalized to FCFS-EASY w/o DVFS. (2) Bottom: WFP-EASY, all the results are normalized to WFP-EASY w/o DVFS. Note that in these plots, when normalized system utilization is higher than 1.0, it means that the system utilization is increased after applying power capping.

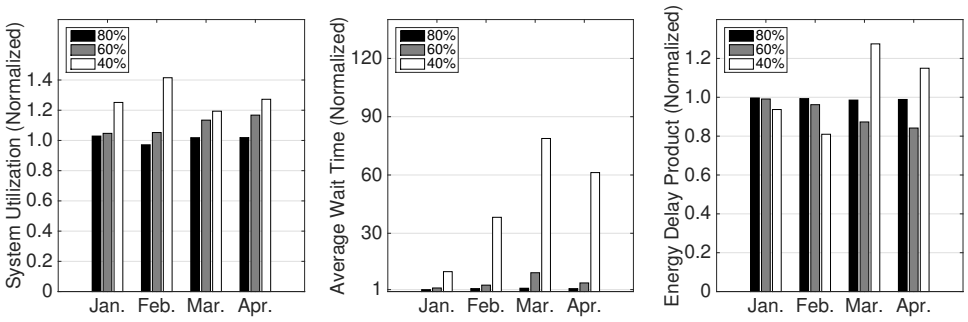
5.2. Case Study 2: Power Capping via Power-Aware Allocation

In this part, we present a case study of using PuPPET to analyze coarse-grained power-aware allocation. Similar to DVFS, the power-performance efficiency of power-aware job allocation is influenced by many factors including workload characteristics, hardware configuration, and scheduling policies. We conduct three sets of experiments to examine performance changes under a variety of factors relevant to power-aware job allocation.

As described earlier, in the current PuPPET, we have developed two power-aware allocation strategies: *BLOCK* and *WAIT*. In the first set of experiments, we compare BLOCK and WAIT using the same power cap. The parameters for WAIT are set to $w = 500$ and $l = 10$. Figure 13 presents



(a) FCFS-EASY w/ DVFS

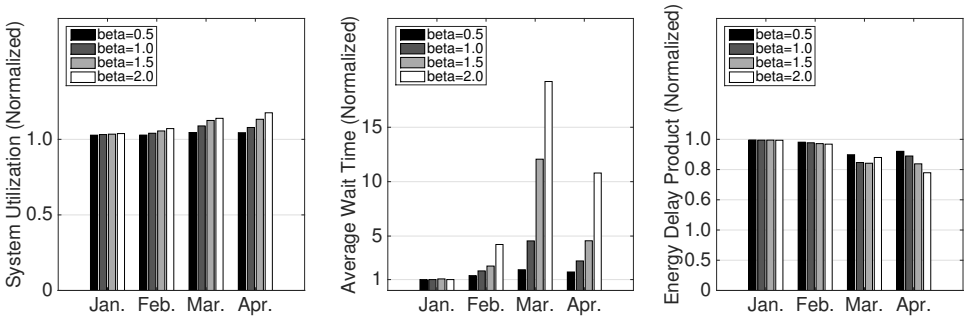


(b) WFP-EASY w/ DVFS

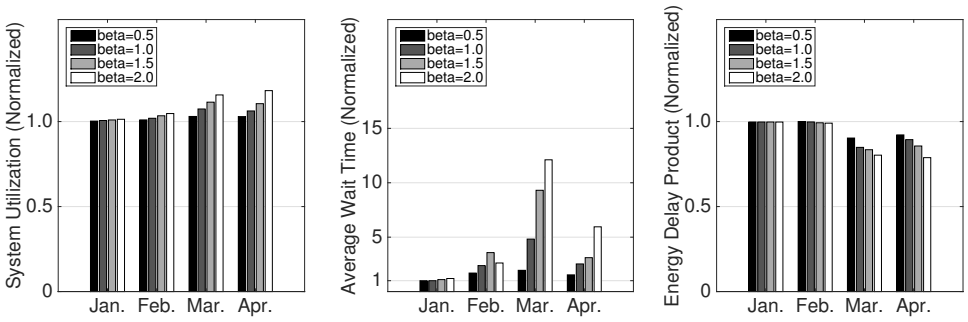
Figure 11: Power-performance impact under varied power caps (i.e., varying the power budget from 80% to 40% of the maximum power) by using DVFS for power capping. (1) Top: FCFS-EASY, all the results are normalized to FCFS-EASY w/o DVFS. (2) Bottom: WFP-EASY, all the results are normalized to WFP-EASY w/o DVFS.

our results. First of all, it is clear that WAIT always outperforms BLOCK across all the metrics. This is quite obvious because the WAIT strategy seeks to improve scheduling performance by opportunistically allocating less power-hungry jobs without violating the power cap.

Unlike DVFS, we find that power-aware allocation results in lower system utilization and higher EDP, especially in the case of March and April. Power-aware allocation intentionally delays the allocation of some power-hungry jobs, thus leading to lower system utilization and longer scheduling makespan. A lowered system utilization rate indicates more energy consumed by idle nodes. Because power-aware allocation does not change the total energy required by the jobs, the extended makespan and the larger amount of energy needed for idle nodes result in an increased EDP. When



(a) FCFS-EASY w/ DVFS

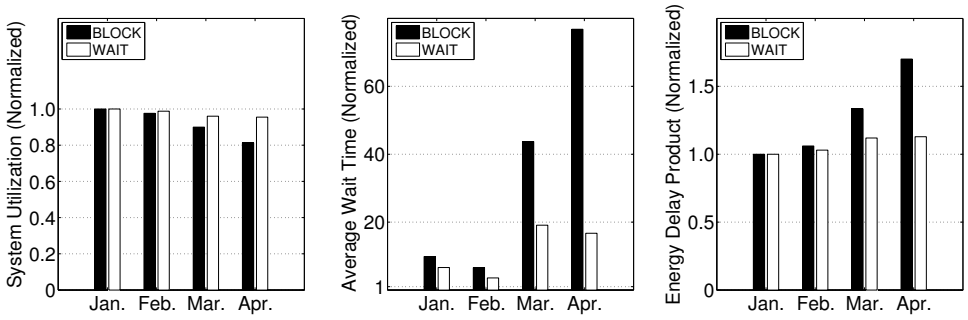


(b) WFP-EASY w/ DVFS

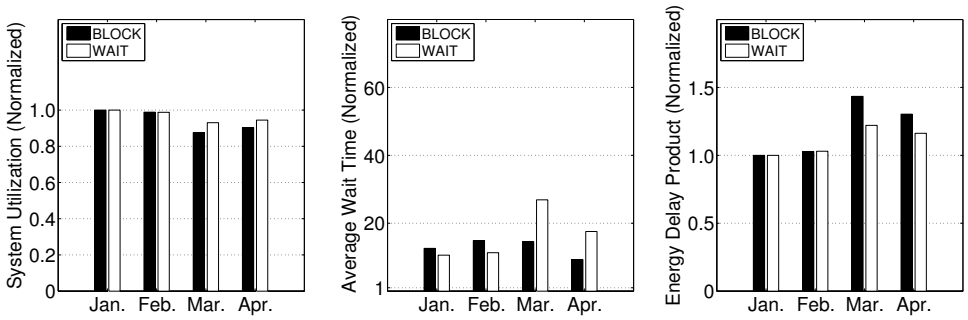
Figure 12: Power-performance impact under varied frequency-to-time relationships (i.e., varying β of Equation 3 from 0.5 to 2) by using DVFS for power capping. (1) Top: FCFS-EASY, all the results are normalized to FCFS-EASY w/o DVFS. (2) Bottom: WFP-EASY, all the results are normalized to WFP-EASY w/o DVFS.

system utilization rate is low (e.g., in Jan. and Feb.), the probability that the power budget will be violated is low, which implies it is less likely for the job allocator to delay job execution. Further, a lower system utilization rate often implies a longer interval between job arrivals, thus delaying a job allocation hardly affects its successors. Similar to Case Study 1, we observe that WFP is more robust to system performance degradation caused by power management than FCFS.

In the second set, we study the impact of different *power caps*, i.e., from 80% to 40% of 65,314kW (the maximum power in January). Figure 14 presents the normalized results to the cases without applying power-aware allocation. By comparing this figure with the DVFS results shown in Figure 11, we find that a tighter power budget has a more severe impact on



(a) FCFS-EASY w/ Power-aware Allocation

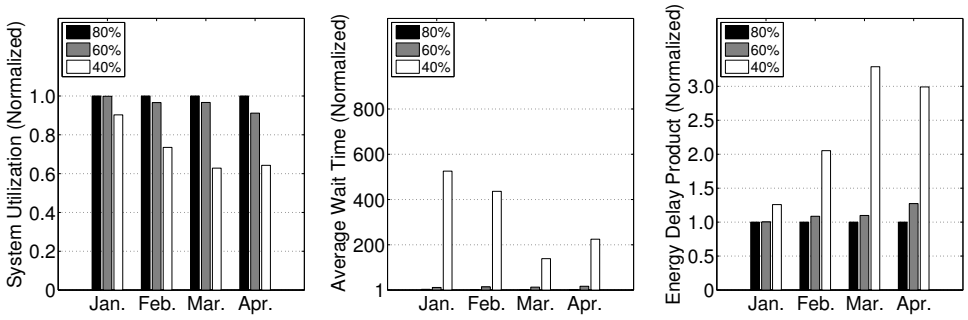


(b) WFP-EASY w/ Power-aware Allocation

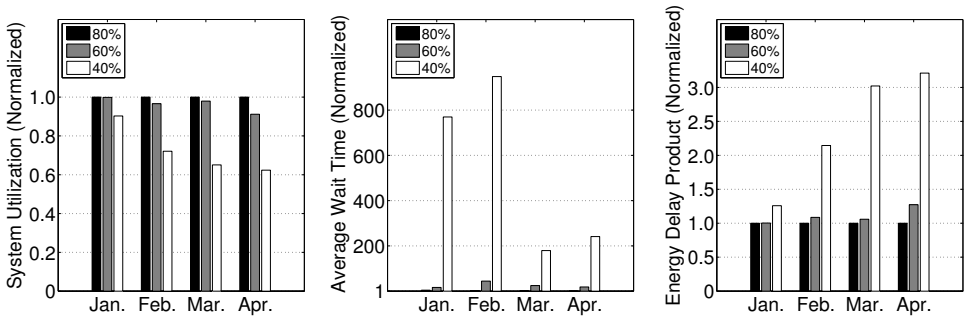
Figure 13: Power-performance impact under different power-aware allocation strategies. (1) Top: FCFS-EASY, all the results are normalized to FCFS-EASY w/o power-aware allocation. (2) Bottom: WFP-EASY, and all the results are normalized to WFP-EASY w/o power-aware allocation.

system performance by using power-aware allocation. When the power cap is set to 80% - 60%, although system utilization, average job wait time, and EDP are all affected by power-aware allocation, the impact is minor. When the cap is lowered to 40%, the impact becomes very obvious. The higher EDP value resulted by using power-aware allocation indicates that it is not a good choice in the case of a tight power budget.

In the third set, we study the impact of *idle power* on power management. We vary the idle power from 0% to 50% of node power in active state. Given that this factor only affects EDP, Figure 15 presents the results regarding energy delay product. First, the use of power-aware allocation does not decrease EDP, which is also depicted in Figure 13. The reason is while power-aware allocation does not change the total energy required by the jobs, it may cause an extended makespan and larger amount of energy required for



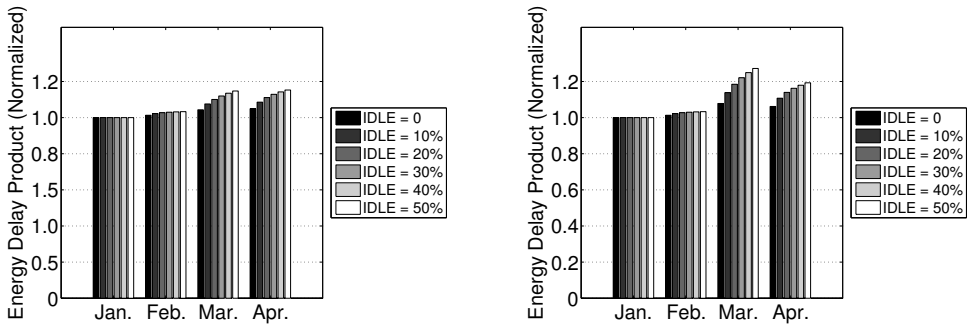
(a) FCFS-EASY w/ Power-aware Allocation



(b) WFP-EASY w/ Power-aware Allocation

Figure 14: Power-performance impact under varied power caps (i.e., varying the power budget from 80% to 40% of the maximum power) by using power-aware job allocation for power capping. (1) Top: FCFS-EASY, all the results are normalized to FCFS-EASY w/o power-aware allocation. (2) Bottom: WFP-EASY, and all the results are normalized to WFP-EASY w/o power-aware allocation.

idle nodes. Furthermore, we observe that this idle power factor has a more significant impact on EDP when the system utilization rate is high (e.g., in March and April). As shown in Figure 13, when the system utilization rate is high, power-aware allocation may decrease system utilization for the purpose of limiting the peak power. A lower system utilization means more energy is required for idle nodes and longer makespan, thus leading to a higher EDP. In addition, the idle power has more significant impacts on WFP than on FCFS in terms of EDP. This is because WFP affects system utilization rate more obviously than FCFS (see Figure 13 WAIT). This results in more remarkable changes of energy consumed by the idle nodes.



(a) FCFS-EASY w/ Power-aware Allocation (b) WFP-EASY w/ Power-aware Allocation

Figure 15: Energy delay product under varied idle power rates by using power-aware job allocation for power capping. (1) Top: FCFS-EASY, all the results are normalized to FCFS-EASY w/o power-aware allocation. (2) Bottom: WFP-EASY, and all the results are normalized to WFP-EASY w/o power-aware allocation.

5.3. DVFS versus Power-Aware Allocation

An interesting question is whether these power capping mechanisms are capable of controlling the overall system power within the limit and how they differ in terms of power change. The answer is presented in Figure 16, in which we plot the average power within a day by applying different policies. Here power at each time point is calculated as the average over the four months. In this experiment, we use WFP-EASY for batch scheduling and $45,720$kW as the power cap. From the figure, it is clear that both methods are able to limit the system power within the threshold. We can also see that these power capping methods result in different power curves. These curves do not exactly fit the straight power cap line. Power-aware allocation controls the overall system power by delaying job execution; however it cannot guarantee the total power of all the running jobs will exactly match the power cap. For DVFS, as the processors can only run at one of the predefined power rates, there may exist a gap between the power cap and the total power of all the resources. In general, the figure clearly demonstrates that both DVFS and power-aware allocation are effective power capping mechanisms.

6. Related Work

Modern systems are designed with various hardware sensors that collect power-related data and store these data for system analysis. System

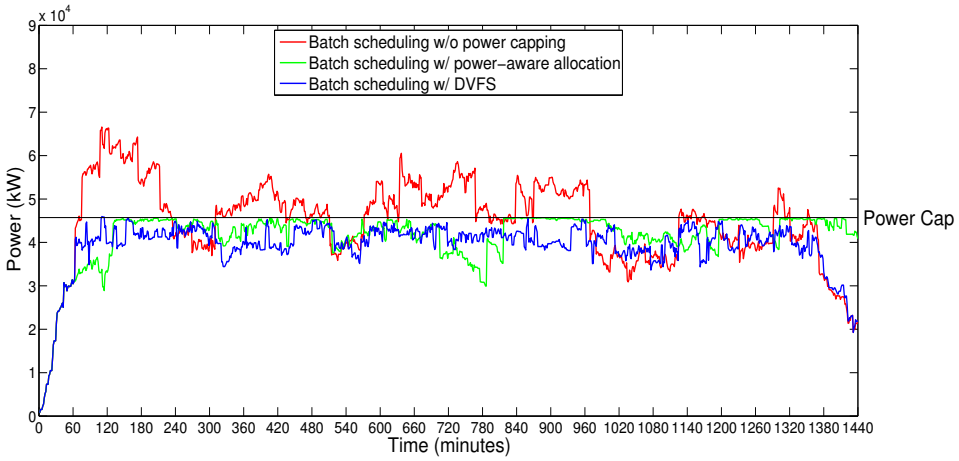


Figure 16: The average daily power trends by using different policies.

level tools like LLView [4] and PowerPack [18] have been developed to integrate the power monitoring capabilities to facilitate systematic measurement, model and prediction of performance. Goiri et al. used external meters to measure the energy consumption of a node during its running time [20]. Feng et al. presented a general framework for direct, automatic profiling of energy consumption on high-performance distributed systems [15]. The performance API (PAPI) has been extended to access internal power sensor readings on several architectures including Intel SandyBridge chips and Nvidia GPUs [47]. In our recent work, we have developed a power profiling library called *MonEQ* for accessing internal power sensor readings on IBM Blue Gene/Q systems [46]. This work focuses on quantitative analysis of power performance tradeoffs. The power data collected by the above power monitoring tools can be used as an input to PuPPET. In other words, this work is complementary to the above power monitoring studies.

Power management policies are widely studied to achieve better energy efficiency in a variety of systems. Ge et al. studied the impacts of DVFS on application performance and energy efficiency for GPU computing in [19]. Patki et al. demonstrated how an intelligent, hardware-enforced power bounds consistently leads to greater performance across a range of standard benchmarks [34]. Fan et al. presented the aggregate power usage characteristics of large collections of servers based on power capping [13]. Lefurgy et al. presented a technique for high density servers that controls the peak power consumption

by implementing a feedback controller [27]. In this study, we investigate two power capping strategies, namely power-aware allocation and DVFS, and applied them to our modeling tool.

Modeling power, performance and their tradeoffs has been done on various systems. Analytical modeling is a commonly used method, which mainly focuses on building mathematical correlations between power and performance metrics of the system. Chen et al. proposed a system level power model for online estimation of energy consumption using linear regression [8]. Curtis-Maury et al. presented an online performance prediction framework to address the problem of simultaneous runtime optimization of DVFS and dynamic concurrency throttling (DCT) on multi-core systems [11]. Dwyer et al. designed and evaluated a machine learning model that estimates performance degradation of multicore processors in HPC centers [12]. Subramaniam et al. built a regression model for the power and performance of scientific applications and used this model to optimize energy efficiency [39]. Tiwari et al. developed CPU and DIMM power and energy models of three widely used HPC kernels by training artificial neural networks [44]. While these models provide good estimation of power and/or performance metrics, they cannot capture the dynamic, complicated power-performance interactions exhibiting in large-scale systems.

There are several studies of applying stochastic models for performance or power analysis. B.Guenter et al. adopted a Markov model for idleness prediction and also proposed power state transitions to remove idle servers [21]. Qiu et al. introduced a continuous-time and controllable Markov process model of a power-managed system [35]. Rong et al. presented a stochastic model for a power-managed, battery-powered electronic system, and formulated a policy optimization problem to maximize the capacity utilization of the battery powered systems [37]. Unlike these studies relying on a Markov process, our model is built on colored Petri nets, thus being more robust to the potential space explosion problem that is commonly observed in Markov models.

The studies closely related to ours are [17] and [43]. In [17], Gandhi et al. used queueing theory to obtain the optimal energy-performance tradeoff in server farms. In [43] Tian et al. proposed a model using stochastic reward nets (SRN) to analyze the performance and energy consumption under different power states. Distinguishing from these studies, our work targets supercomputers with unique batch scheduling and parallel workload. Moreover, our model adopts the advanced feature of colored Petri nets which provides

a more accurate representation of the dynamic system being analyzed. To the best of our knowledge, this is the first colored Petri net modeling work to study the power-performance tradeoffs for high performance computing.

7. Conclusion

In this paper, we have presented PuPPET, a colored Petri net for quantitatively modeling and analyzing power management on extreme scale systems. By using the advanced features of colored Petri nets, PuPPET is capable of capturing the complicated interactions among different factors that can affect power-performance efficiency on HPC systems. We have validated the model accuracy by using system traces collected from the 10-petaflops IBM Blue Gene/Q machine at Argonne Leadership Computing Facility. Our trace-based validation demonstrates that PuPPET can effectively model the dynamic execution of parallel jobs on the system by providing an accurate approximation of energy consumption. A salient feature of the proposed PuPPET is that it can scale to hundreds of thousands of processor cores and at the same time provide high levels of modeling accuracy. Such a feature is crucial for power-performance tradeoff analysis of extreme scale systems. Moreover, we have explored the model to analyze the power-performance tradeoffs under two well-known power capping methods, i.e., power-aware job allocation and DVFS. Our case studies provide us useful insights about using power capping on extreme scale systems. PuPPET is implemented by using the CPN tools, and is freely available for the community research.

PuPPET provides a convenient modeling tool for users to set different parameters and study power-performance tradeoffs on extreme scale systems. We believe it has many other potential usages, in addition to the case studies presented in this work. For instance, it can be used to find the optimal power policy that minimizes energy consumption under a time constraint. It can also be extended to incorporate resiliency analysis by adding a module for describing failure behaviors. The resulting model can be used to study the key tradeoffs among performance, power, and reliability for supercomputing. All these are part of our ongoing work.

Acknowledgments

This work is supported in part by US National Science Foundation grants CNS-1320125 and CCF-1422009. The research used resources of the Argonne

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

[1] The standard workload format. `http://www.cs.huji.ac.il/labs/parallel/workload/swf.html`.

[2] Mira: Next-generation supercomputer. `https://www.alcf.anl.gov/mira`, 2012.

[3] Cpn tools. `http://cpntools.org/`, 2013.

[4] LLview: graphical monitoring of loadleveler controlled cluster. `http://www.fz-juelich.de/jsc/llview/`, 2013.

[5] S. Ashby. The opportunities and challenges of exascale computing. Technical report, DOE Office of Science, 2010.

[6] G. Balbo. Introduction to generalized stochastic petri nets. In *Proceedings of SFM*, 2007.

[7] D. Bodas, J. Song, M. Rajappa, and A. Hoffman. Simple Power-aware Scheduler to Limit Power Consumption by HPC System Within a Budget. In *Proc. of E2SC*, 2014.

[8] X. Chen, C. Xu, R.P. Dick, and Z.M. Mao. Performance and power modeling in a multi-programmed multi-core environment. In *Proceedings of DAC*, 2010.

[9] M. Chiesi, L. Vanzolini, C. Mucci, E. Scarselli, and R. Guerrieri. Power-Aware Job Scheduling on Heterogeneous Multicore Architectures. *IEEE Trans. Parallel Distrib. Syst.*, 26:868–877, 2015.

[10] M. Crovella, R. Bianchini, T. Leblanc, E. Markatos, and R. Wisniewski. Using Communication-to-Computation Ratio in Parallel Program Design and Performance Prediction. In *Proc. of IPDPS*, 1992.

[11] M. Curtis-Maury, A. Shah, F. Blagojevic, D.S. Nikolopoulos, B.R. de Supinski, and M. Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of PACT*, 2008.

[12] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *Proceedings of SC*, 2012.

[13] X. Fan, W.-D. Weber, and L.A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of ISCA*, 2007.

[14] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and

P. Wong. Theory and practice in parallel job scheduling. In *Proceedings of JSSPP*, 1997.

[15] X. Feng, R. Ge, and K.W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Proceedings of IPDPS*, 2005.

[16] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, and et al. Introducing the adaptive energy management features of the power7 chip. *IEEE Micro*, 31:60–75, 2011.

[17] A. Gandhi, M. Harchol-Balter, and I. Adan. Server farms with setup costs. *Perform. Eval.*, 67:1123–1138, 2010.

[18] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21:658–671, 2010.

[19] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *Proceedings of ICPP*, 2013.

[20] I. Goiri, L. Kien, M.E. Haque, R. Beauchea, T.D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenslot: Scheduling energy consumption in green datacenters. In *Proceedings of SC*, 2011.

[21] B. Guenter, N. Jain, and C. Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *Proceedings of INFOCOM*, 2011.

[22] R. Harper. Programming in Standard ML. Available at: `http://www.cs.cmu.edu/~rwh/smlbook/book.pdf`, 2011.

[23] K. Jensen and L.M. Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

[24] S. Kale, K. Bergman, and et al. Ascr workshop on modeling and simulation of exascale systems and applications. Technical report, DOE Office of Science, 2013.

[25] J.H. Laros III, K.T. Pedretti, S.M. Kelly, W. Shu, K.B. Ferreira, J.V. Dyke, and C.T. Vaughan. *Energy-Efficient High Performance Computing - Measurement and Tuning*. Springer, 2012.

[26] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proceedings of ICAC*, 2007.

[27] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11:183–195, 2008.

[28] F Liu, M Heiner, and C Rohr. Manual for Colored Petri Nets in Snoopy. Technical report, Brandenburg University of Technology Cottbus, 2012.

[29] M. Marinoni and G.C. Buttazzo. Elastic dvs management in processors with discrete voltage/frequency modes. *IEEE Trans. Industrial Informatics*, 3:51–62, 2007.

[30] T. Martin and D. Siewiorek. Non-ideal battery and main memory effects on cpu speed-setting for low power. *IEEE Trans. VLSI System*, 9:29–34, 2001.

[31] W. Marwan, C. Rohr, and M. Heiner. *Petri nets in Snoopy: A unifying framework for the graphical display, computational modelling, and simulation of bacterial regulatory networks*. Humana Press, 2012.

[32] A. Mualem and D. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling,. *IEEE Transactions on Parallel and Distributed System*, 12(6):529–543, 2001.

[33] White Paper. Enhanced intel speedstep technology for the intel pentium m processor. Technical report, Intel Corporation, 2004.

[34] T. Patki, D.K. Lowenthal, B. Rountree, M. Schulz, and B.R. de Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of ICS*, 2013.

[35] Q. Qiu and M. Pedram. Dynamic power management based on continuous-time markov decision processes. In *Proceedings of DAC*, 1999.

[36] A.F. Rodrigues, K.S. Hemmert, B.W. Barrett, C. Kersey, and et al. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38:37–42, 2011.

[37] P. Rong and M. Pedram. Battery-aware power management based on markovian decision processes. In *IEEE TCAD*, 2006.

[38] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of DSN'04*, 2004.

[39] B. Subramaniam and W.-C. Feng. Statistical power and performance modeling for optimizing the energy efficiency of scientific computing. In *Proceedings of GREENCOM-CPSCOM*, 2010.

[40] K. Sugavanam, C.-Y. Cher, J. A. Gunnels, R.A. Haring, P. Heidelberger, and et al. Design for low power and power management in IBM Blue Gene/Q. *IBM Journal of Research and Development*, 57:1–11, 2013.

[41] W. Tang, N. Desai, D. Buettner, and Z. Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on Blue Gene/P. In *Proceedings of IPDPS*, 2010.

[42] W. Tang, Z. Lan, N. Desai, and D. Buettner. Fault-aware utility-based job scheduling on Blue Gene/P systems. In *Proceedings of Cluster*, 2009.

[43] Y. Tian, C. Lin, and M. Yao. Modeling and analyzing power management policies in server farms using stochastic petri nets. In *Proceedings of e-Energy*, 2012.

[44] A. Tiwari, M.A. Laurenzano, L. Carrington, and A. Snavely. Modeling power and energy usage of hpc kernels. In *Proceedings of IPDPSW*, 2012.

[45] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M. Papka. Application Profilling Benchmarks on IBM Blue Gene/Q. In *Proc. of Cluster*, 2013.

[46] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M.E. Papka. Measuring power consumption on IBM Blue Gene/Q. In *Proceedings of HPPAC*, 2013.

[47] V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszcek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *International Workshop on Power-Aware Systems and Architectures*, 2012.

[48] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M.E. Papka. Dynamic pricing of electricity into energy aware scheduling for hpc systems. In *Proceedings of SC*, 2013.

[49] G. Zheng, G. Kakulapati, and L.V. Kale. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *Proceedings of IPDPS*, 2004.

[50] Z. Zhou, Z. Lan, W. Tang, and N. Desai. Reducing Energy Costs for IBM Blue Gene/P via Power-Aware Job Scheduling. In *Proc. of JSSPP*, 2013.