LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-JRNL-775437

# A Value-Oriented Job Scheduling Approach for Power-Constrained and Oversubscribed HPC System

N. Kumbhare, A. Marathe, A. Akoglu, H. J. Siegel, G. Abdulla, S. Hariri

May 24, 2019

IEEE Transactions of Parallel and Distributed Systems

## Disclaimer

# A Value-Oriented Job Scheduling Approach for Power-Constrained and Oversubscribed HPC Systems

Nirmal Kumbhare, Aniruddha Marathe, Ali Akoglu, Howard Jay Siegel, *Fellow, IEEE*, Ghaleb Abdulla, and Salim Hariri

**Abstract**—In this study, we investigate limitations in the traditional value-based algorithms for a power-constrained HPC system and evaluate their impact on HPC productivity. We expose the trade-off between allocating system-wide power budget uniformly and greedily under different system-wide power constraints in an oversubscribed system. We experimentally demonstrate that, under the tightest power constraint, the mean productivity of the greedy allocation is 38% higher than the uniform allocation whereas, under the intermediate power constraint, the uniform allocation has a mean productivity of 6% higher than the greedy allocation. We then propose a new algorithm that adapts its behavior to deliver the combined benefits of the two allocation strategies. We design a methodology with online retraining capability to create application-specific power-execution time models for a class of HPC applications. These models are used in predicting the execution time of an application on the available resources at the time of making scheduling decisions in the power-aware algorithms. We evaluate the proposed algorithm on a real HPC system composed of 64 computing nodes and show that our adaptive strategy results in improving resource utilization while delivering a mean productivity that is almost the same as the best performing algorithm across various system-wide power constraints.

**Index Terms**—High performance computing, power-constrained computing, power-aware scheduling, value heuristics, HPC productivity.

## 1 Introduction

Exascale computing is expected to deliver significant computation power to analyze vast amounts of data and to execute ever-growing scientific applications in more realistic time frames. The development efforts of exascale computing systems have given rise to various engineering and research challenges. Out of those challenges, improving scientific productivity and power efficiency are among the top ten [1] and are the focus of this work. In the literature, scientific productivity is defined as the value of the output generated by a mission-critical application on a high performance computing (HPC) system [2]–[5]. The value of the mission-critical application is determined by the system owner [2], [6]. Traditionally, the performance of an HPC system is defined based on the floating point operations it executes per second (flops); hence, flops-per-watt has been the primary metric of interest to improve power efficiency. Even though flops is a useful metric in defining system performance, it is not sufficient to quantify the productivity of an HPC system [2]–[5]. This creates a necessity for a power-aware scheduler that takes into account the system-wide constraints at the time of resource allocation to maximize HPC productivity.

HPC productivity has been an area of concern since the inception of peta-scale computing and it is expected to become increasingly crucial in future exascale systems [1]. To measure the productivity of an HPC system, researchers have proposed the use of time-dependent value (utility) functions [2]–[5], [7]–[11]. These studies are inspired by classical utility theory. The time-dependent value for a job is a monotonically decreasing function of time, and it is used to define the importance of completing the job within a defined time constraint. In HPC domains, the time metric is composed of two components: "time-to-solution" and "time-to-develop" [1]. In our work, we only consider the time-to-solution as our time metric because, for most of the mission-critical HPC applications, their development time can be amortized over their multiple executions during the production phase [12]. Domains for different mission-critical HPC applications can be put into categories including climate prediction, physics simulation, environmental management, nuclear energy, price fluctuation, and weapon trajectory system [12]. These mission-critical applications are sensitive to their completion time. Therefore, in our study, time-to-solution corresponds to the time taken to complete a job after it is submitted into the queue on an HPC system.

Power consumption has emerged as one of the important operating constraints (if not *the* critical constraint) to improve the energy efficiency of future exascale systems [1]. Traditional HPC systems are provisioned with sufficient power to operate all the CPUs at their thermal design

- *N. Kumbhare, A. Akoglu, and S. Hariri are with The Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85719.*
  *E-mail: {nirmalk,akoglu,hariri}@email.arizona.edu*
- *A. Marathe, and G. M. Abdulla are with Lawrence Livermore National Laboratory, 7000 East Ave., Livermore, CA 94550-9234.*
  *E-mail: {marathe1,abdulla1}@llnl.gov*
- *H.J. Siegel is with The Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523.*
  *E-mail: HJ@colostate.edu*

*Manuscript received July 3, 2019; revised August xx, xxxx.*

power (TDP). With the present CPU technologies, the future generation of HPC systems is estimated to cost $2.5 billion annually in power consumption to deliver exascale performance [13]. The Department of Energy (DoE) has set a power consumption target of 30 megawatts to support efficient electricity generation and distribution, and also to keep the operational cost of an exascale computing system manageable. Further, due to the variability in electricity pricing, HPC centers may have to register (with their electricity service providers) for programs like peak shedding, peak shifting, or dynamic pricing to reduce the operational cost for day to day operations [14].

In the past decade, researchers have proposed various power allocation policies to improve the power efficiency of HPC systems (measured in flops-per-watt) by increasing the job completion rate subject to a system-wide power constraint [15]–[17]. However, improving flops-per-watt may not be sufficient in improving HPC productivity. To improve productivity, researchers have proposed value-based heuristics for an oversubscribed system in which each job is associated with a job-value function but they do not take the system-wide power constraint into account while making scheduling decisions [7]–[9], [18]–[22]. To the best of our knowledge, our earlier conference paper is the only study that addressed both power and productivity challenges of an oversubscribed HPC system through power-aware value-based resource management algorithms [23]. In that work, we adapted two, commonly known, static power allocation strategies into a value-based heuristic concept introduced by *Khemka et al.* [8], and designed two power-aware value-based algorithms called *Value Per Time With Common Power Capping (VPT-CPC)* and *Value Per Time With Job Specific Power Capping (VPT-JSPC)*. We compared their performances in terms of total system-value earning and job completion rate under various power constraints on a real but small-scale hardware platform. Our algorithms were designed to use application-specific power-performance models to select and schedule the jobs. VPT-CPC uses a traditional power allocation strategy by distributing the system-wide power budget uniformly across all the nodes. It is designed to minimize the waiting time for the jobs by increasing the node usage within the system. In contrast, VPT-JSPC uses a power-allocation strategy that minimizes the completion time of fewer but higher-value jobs by allocating a system-wide power budget to a selected number of nodes. As a result, it maximizes the power utilization in the system. Through empirical evidence, we showed that under different system-wide constraints on power, the resource scheduler should use different power allocation strategies.

In this paper, we extend our prior work with the following contributions:

- **Hybrid algorithm**: We introduce a new algorithm (hybrid-VPT) that combines the benefits of two power-aware algorithms presented in our prior work in a complementary manner. Our hybrid algorithm adapts to different system-wide power constraints. It is designed to maximize system-value earnings by maximizing resource (both power and node) utilization.
- **Application specific modeling**: We propose a methodology with the online retraining capability to create application-specific power-execution time models for a class of applications that is used to solve Navier-Stokes equations. These models are used to predict the execution time of an application for a given combination of power constraint, node count, core count, and input problem size.
- **Evaluation**: We emulate an HPC prototype of size 64 nodes on a real system and demonstrate the effectiveness of our new hybrid algorithm in combining the benefits of VPT-CPC and VPT-JSPC under different power constraints.

The rest of this paper is organized as follows. We introduce the mathematical model of the *job-value* functions in Section 2, followed by a discussion on the metric used in our work for comparing the performance of different algorithms. In Section 3, we present the details of the HPC environment for which our work is proposed, followed by a description of our emulation testbed in Section 4. Next, we describe our approach to create power-execution time models in Section 5. These models are employed by our power-aware algorithms to estimate the execution time of the jobs when making scheduling decisions. We discuss the details of our design approach in creating power-aware value-based algorithms in Section 6. We describe our workload generation approach and scheduler architecture in Section 7. Later, we show the results of our emulation study on 64 nodes HPC prototype in Section 8. We review the existing literature in Section 9, followed by conclusions and future work in Section 10.

## 2 Timing Constraints and Value Functions

### 2.1 Importance of Timing Constraints in HPC

In the scientific computing domain, many applications are time critical, e.g., weather prediction, missile path tracking, aircraft control, stock price prediction, and surgical path planning. For such time-sensitive applications, a missed completion deadline can lead to either monetary loss or human fatality. These time-constrained applications need to complete on time. Otherwise, they completely lose their value, and a scheduler needs to take that into account to improve productivity. In traditional HPC systems, various studies have shown the importance of using time-dependent job-value functions (discussed next) for quantifying HPC or user productivity [2]–[5], [8], [9], [24], [25]. In an HPC environment, a user submits a job with a specific priority and then waits for an undefined amount of time before resources are allocated for that job. It is not unusual for a job to wait in the resource allocation queue for a duration that is longer than its actual execution time. In the study conducted by *Wolter et al.*, on the HPC system at the San-Diego Supercomputer Center (SDSC) [26], they found that the maximum wait time for a 128 node job was 17 days (with an average wait time of 24 hours and an average run-time of four hours) while for a single node job it was 71 days (with an average wait time of nine hours and an average run-time of 2.4 hours). In their discussion with the HPC users, they observed that a majority of the users do not worry about application execution time as long as it meets their productivity requirements. This observation encourages us to use a constraint on the completion time as a parameter to define user productivity. Assigning a

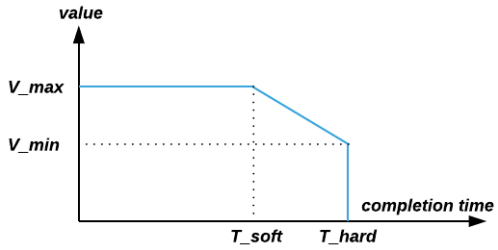



**Fig. 1: Job value function.**

constraint on the completion time will help a user to make a suitable decision to migrate the application to another system or relaunch with an alternate configuration to improve their productivity instead of waiting in the queue for a long duration.

### 2.2 Job-Value Function

In a job-value function, the value of a time-constrained job represents the worth of completing a job from either a user's or a service provider's perspective [7], [8], [27]. In an HPC domain, *Khemka et al.* have proposed the use of monotonically decreasing job-value functions that are created using two important parameters: priority, and urgency [8]. Priority is used to define the level of importance of the job to the enterprise, and urgency indicates how quickly the job loses its importance. In this work, we use a similar monotonically-decreasing job-value function to specify the value earned for completing a job at a specific time instance, as shown in Figure 1 [9], [24], [25]. In the figure, the "*completion time*" axis represents the completion time after the job submission and the "*value*" axis represents the value a job can earn after its successful completion. A job earns "*V_max*" value as long as it is completed before "*T_soft*" (Equation 1). A job earns zero value if it is completed after "*T_hard*" (Equation 3). The value earned from completion of a job decreases linearly between "*T_soft*" and "*T_hard*" from "*V_max*" to "*V_min*" (Equation 2). The slope of this linear decay indicates the urgency of the job.

$i$ = job id indicating $i^{th}$ job;
$T_i$ = time of completion for $i^{th}$ job;
$value_i(T_i)$ = value earned by $i^{th}$ job on its completion;
$V\_max_i$ = maximum value the $i^{th}$ job can earn;
$V\_min_i$ = minimum positive value the $i^{th}$ job can earn;
$T\_soft_i$ = soft completion deadline defined for $i^{th}$ job;
$T\_hard_i$ = hard completion deadline defined for $i^{th}$ job.

$$value_i(T_i) = V\_max_i,\ if\ (T_i \leq T\_soft_i), \tag{1}$$

$$value_i(T_i) = (T_i - T\_hard_i) \times \frac{(V\_max_i - V\_min_i)}{(T\_soft_i - T\_hard_i)} + V\_min_i,\ if\ (T\_hard_i > T_i > T\_soft_i), \tag{2}$$

$$value_i(T_i) = 0,\ if\ (T_i \geq T\_hard_i). \tag{3}$$

The system-value is calculated by accumulating job-values for all the completed jobs over a period of time, which we refer to as HPC productivity. Our aim is to maximize the HPC productivity by maximizing system-value earnings.

## 3 HPC Environment Model

In this work, we propose scheduling algorithms and a resource manager for an HPC environment where:

- Each cluster consists of homogeneous nodes and each node is composed of one or more computing units (multi-core CPU), memory, secondary storage, and network interface card.
- Depending on the operational cost, a resource manager decides the constraint on the system-wide power consumption. The system contains instrumentation to monitor and control the power consumption of the computing units. The power consumption of the remaining components is excluded from the system-wide power budget.
- The scheduling algorithm is responsible for making decisions on power and node allocation to each job. Power allocated to a job is equally distributed among its allocated nodes. The power constraint on a compute node is distributed equally among its computing units.
- Along with a job, each user submits a value function and the job's input parameters (problem size and iteration count for this study).
- For each job, the user also submits a set of valid configurations for the job in terms of the number of MPI ranks and OpenMP threads.
- The node is not oversubscribed, i.e., MPI ranks $\times$ OpenMP threads per rank should be less than or equal to the number of cores on each node.
- Each empirical application power and performance profile is collected offline for the generation of power-execution time model. This power-execution time model is made available to the resource manager at runtime in the production system.
- Jobs are not preemptible. Once a job is scheduled, the scheduler waits for its completion before placing other jobs on the same nodes.
- A node in an HPC system is not shared among multiple jobs. This is a common trend in real systems because resource sharing among multiple jobs can generate unpredictable resource contention and can cause an increase in the execution time. This may lead to the loss of value for a time-constrained job.

## 4 Emulation Testbed

### 4.1 Hardware Platform

For our emulations, we prototype an HPC system (of node count 64) by using the nodes from a real HPC system available at Lawrence Livermore National Labs. Each compute node is a dual socket with 125 GB of memory and InfiniBand QDR for the network interface. Each socket on the node has an CPU (Ivy Bridge-EP) with twelve cores and a maximum frequency of 2.40 GHz. The TDP of each CPU is 115 watts in our system. We access the power-specific registers on each CPU to monitor and control the CPU power consumption. We use this testbed to collect empirical power-performance profiles for our test applications and to run our emulations.

### 4.2 Benchmarks

To create our HPC workload traces, we select the scientific kernels from NAS parallel benchmarks suite [28] that are

**TABLE 1: Jobs in workload.**

| *benchmark* | *description* |
|---|---|
| LU | Lower-Upper Gauss-Seidel solver |
| SP | Scalar Penta-diagonal solver |
| BT | Block Tri-diagonal solver |

**TABLE 2: Memory requirements and iteration count of NAS-NPB classes [28].**

| *class* | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| *memory requirement* | 50 MB | 200 MB | 0.8 GB | 12.8 GB | 250 GB | 5 TB |
| *LU iterations* | 250 | 250 | 250 | 300 | 300 | 300 |
| *BT iterations* | 200 | 200 | 200 | 250 | 250 | 250 |
| *SP iterations* | 400 | 400 | 400 | 500 | 500 | 500 |

used to solve the discretized versions of the compressible Navier-Stokes equations in three spatial dimensions. These benchmarks are listed in Table 1. These benchmarks are enabled with OpenMP and MPI to exploit the multi-level parallelism in an HPC system. Furthermore, these benchmarks are moldable, i.e., the number of MPI ranks and OpenMP thread count can be selected during its launch without changing the high-level problem. Default benchmarks in the suite can be compiled for different classes (A to F) based on problem size (defined by memory footprint) and computation amount (defined by iteration count), as shown in Table 2.

### 4.3 Power Capping

The power capping techniques are used for applying a limit on the maximum power consumption of a CPU. Recent studies on power-constrained computing have primarily used running average power limit (RAPL) for power capping an individual CPU. The RAPL driver provides a user space interface to limit the power consumption of a CPU and DRAM by controlling the combination of P-states and C-states using a built-in software power model [29]. We use the application profiling tool Libpowermon [30] to monitor and control the power consumption of an application on a compute node. Libpowermon is built on top of libmsr, which enables the APIs to interact with CPU's RAPL registers [31].

## 5 Application-Specific Models

### 5.1 Overview

In this work, we create application-specific power-execution time models to predict the execution time of the applications under different power constraints. In reality, this approach may not be feasible if the types of applications executed on an HPC system are unknown in advance. However, the study conducted by *Antypas et al.* is our motivation for creating and using the application-specific models [12]. They analyzed the workload on the Hopper HPC at the National Energy Research Scientific Computing Center (NERSC) and found that 35 applications made up 75% of the HPC workload, whereas the remaining 25% of the workload comprises 600 applications. This finding on the repeatability of applications encourages us to create application-specific models for HPC applications to achieve higher HPC productivity.

Traditionally, researchers have used either historical data or job-specific models to predict the execution time of an application on a given system under different resource constraints [32]–[37]. In our current work, we propose an approach to create models for a class of applications (Table 1). For each benchmark, we create an application-specific model to predict the execution time for a given input problem size, thread count, power constraint, and node count. We use a three-step approach for creating and updating models. In step one, we collect exhaustive data on a small number of resources using representative input problem sizes for each application. We refer to this data as offline data. This is a one-time process, which is performed before application transitions into the production stage in its life cycle. In step two, we use the offline data for training models to predict execution time on the production system.We use these models for making scheduling decisions. The third step is online retraining and involves updating the power-execution time model if the prediction error is out of acceptable limits during run-time. In the next subsection, we present our configuration space parameters, followed by a brief discussion on the range of the configuration parameters we select for offline data collection. Later, we elaborate on our modeling approach.

### 5.2 Modeling Parameters

We use empirical execution time measurements over different combinations of application-specific parameters subject to several hardware-enforced constraints. We represent hardware-enforced constraints as a combination of CPU power cap (using RAPL), core counts per CPU, and the number of nodes. Each unique combination is referred to as a hardware configuration in this paper. We apply the node and core constraints from a hardware configuration by controlling the MPI ranks and the OpenMP counts of the job. The application-specific parameters are the input problem size and iteration count. We use this data to train a power-execution time model that we use to predict application execution time.

### 5.3 Offline Data Collection

Offline data collection is implemented by exhaustively scanning over the multiple combinations of hardware configurations (node count, threads per rank, and power limit) and application-specific input parameters (problem size and iteration count) for each application on a smaller number of resources to reduce training overhead. For application-specific modeling, we use a proportionally smaller input problem size and lower iteration count while ensuring that the derived power-execution time models apply to the production problem sizes on a large-scale system.

For data collection, we use 30 different problem sizes that are evenly spaced in between problem sizes belonging to class C and D. These problem sizes are less resource intensive (Table 2) and are useful in training the application-specific models to predict the execution time for our test problem sizes that are randomly sampled in between class

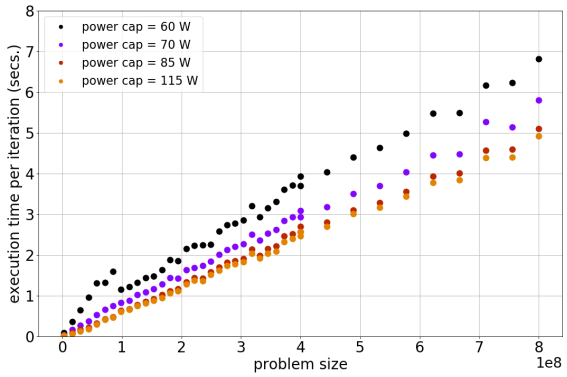



**Fig. 2: LU's execution time per iteration for different power caps and problem size on 16 nodes (24 OpenMP thread count per node). Each color represents a power cap value and each circle represents the execution time per iteration (Y-axis) for a given input problem size (X-axis).**

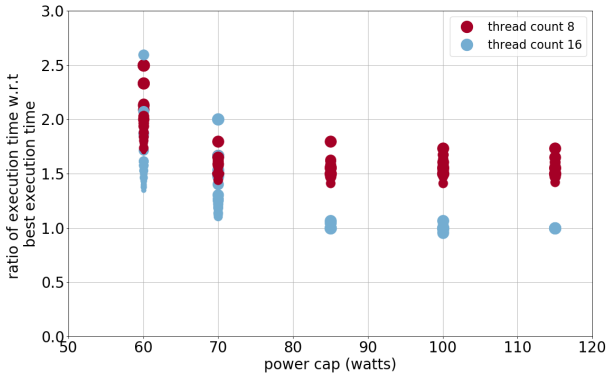



**Fig. 3: LU's clustering with problem size, power cap, and thread count. For all problem sizes, execution time (on Y-axis) is normalized using the execution time on the hardware configuration of 16 node, 24 threads, and 115 watts power constraint per CPU. X-axis presents the power constraint per CPU. Input problem size is represented using circles.**

C and E. For each application, we collect offline data by iterating over each combination of input problem size, node count (4, 8, and 16), thread count (8, 12, 16, 20, and 24), and power limit (55, 60, 70, 85, 100, and 115).

### 5.4 Modeling and Prediction

#### *Modeling*

We use the following parameters to mathematically represent our models.

$N_0$ = fixed node count used to create models;
$P_0$ = fixed power constraint per CPU;
$Th_0$ = fixed thread count per node;
$ps$ = input problem size;
$tc$ = input thread count per node;
$pc$ = input power constraint per CPU;
$it$ = input application's iteration count;
$n$ = input node count;
$LR_{N_0, P_0, Th_0}(ps)$ = linear regression model to estimate the execution time per iteration on the hardware configuration ($N_0$, $P_0$, $Th_0$) for the input problem size $ps$;
$RF_{N_0, P_0, Th_0}(ps, pc, tc)$ = random forest regression model to estimate the ratio of increase in the execution time on the hardware configuration ($N_0$, $pc$, $tc$) compared to the execution time on the hardware configuration ($N_0$, $P_0$, $Th_0$) for the input problem size $ps$;
$Et_{N_0}(ps, pc, tc)$ = estimated execution time per iteration for the input problem size ($ps$) on the hardware configuration ($N_0$, $pc$, $tc$);
$Et(ps, pc, tc, n, it)$ = estimated execution time on the hardware configuration ($n$, $pc$, $tc$) for the input problem size ($ps$) and the iteration count ($it$);

$$Et_{N_0}(ps, pc, tc) = LR_{N_0, P_0, Th_0}(ps) \times RF_{N_0, P_0, Th_0}(ps, pc, tc) \quad (4)$$

$$Et(ps, pc, tc, n, it) = Et_{N_0}(ps, pc, tc) \times (N_0/n) \times it \quad (5)$$

In our experiments, for a given input problem size, we observe a linear correlation between the iteration count ($it$) and the execution time. Similarly, we find a linear correlation between the node count and the execution time. Because these two observations are intuitive, in the following paragraphs we elaborate on the modeling techniques used to capture the effects of other configuration parameters (problem size, power constraint, and thread count) on the execution time, while fixing the node count ($N_0$) and the iteration count to their default values of 16 and 100, respectively.

We observe a linear correlation between the input problem size ($ps$) and the execution time per iteration for the selected benchmarks for all the combinations of hardware configuration parameters (thread count and power constraint). Figure 2 presents the execution time per iteration for a subset of hardware configurations based on the LU benchmark. In this figure, we fix the thread count per node to 24 and only alter the power cap on each CPU to highlight the linear correlation between the problem size and the execution time per iteration.

Only using the linear regression-based models to capture the correlation between the input problem size and the execution time [38] for all the combinations of thread counts and power caps will lead to the creation of a large number of models for each application. To avoid this, we study the effect of the thread count and the power constraint on the normalized execution time for different problem sizes. For normalization, we use the hardware configuration of 24 threads per node ($Th_0$) and CPU power cap of 115 watts ($P_0$) as this configuration was the best-performing configuration for all the problem sizes. Figure 3 presents the normalized execution time for the different combinations of problem size, thread count, and CPU power cap. The size of each circle in the plot is inversely proportional to the size of the input problem size, i.e., the smaller circle represents larger input problem size. This representation allows us to avoid overlapping during clustering and visualize the formation of clusters clearly for larger problems sizes. We observe that as the problem size increases, the normalized execution time starts forming clusters for a given combination of power cap and thread count. We capture this clustering behavior using Random Forest regression [39]. Therefore, instead of creating multiple linear regression models, we create two models for each application. The first model captures the linear trend between the input problem size (from training data) and the execution time on a fixed hardware configuration, ($N_0$, $P_0$, $Th_0$), using a linear regression model ($LR_{N_0, P_0, Th_0}(ps)$). The second model captures the clustering behavior using the Random Forest regression ($RF_{N_0, P_0, Th_0}(tc, pc, ps)$).

#### *Prediction*

To estimate the execution time for an unknown combination of problem size ($ps$), power constraint ($pc$), OpenMP

thread count *(tc)*, node count *(n)*, and iterations *(it)*, we first independently apply the linear regression model and the random forest model on these inputs. Then we multiply the outputs of these models, in Equation 4, to predict the execution time per iteration on a 16 node configuration. We then scale this execution time per iteration to the desired input iteration count and the node count by using Equation 5 to compute the final estimated execution time for the given inputs *(ps, tc, pc, n, it)*.

#### *Online Retraining*

Our evaluation shows, as expected, that the accuracy of our linear regression-based model reduces as we extrapolate beyond the trained problem space. Therefore, our resource manager monitors the execution of each job during its runtime, and if the difference between the predicted execution time and the actual execution time is greater than 15%, the resource manager retrains the model. Based on the actual execution time, iteration count, and nodes used, we calculate the execution time per iteration and scale it for the $N_0$ node configuration using Equation 5. We refer to this actual execution time as the new execution time. We recompute the expected minimum execution time on a fixed hardware configuration of $(N_0, P_0, Th_0)$ by dividing the new execution time with the output of the random forest model. This minimum execution time is the desired output of the linear regression model; hence, we include this data point in the training set to retrain the linear regression model. This retraining of the linear regression model helps the resource manager to increase the accuracy in future predictions.

## 6 Power-Aware Value-Based Scheduling Heuristics

### 6.1 Overview

In our earlier work, we used a value-based heuristic [8] as a basis for constructing two power-aware algorithms (VPT-CPC and VPT-JSPC) [23]. In both algorithms, the VPT metric for a job is computed by dividing the value that the job can earn on its completion with the estimated execution time of the job. The VPT heuristic uses two stages for system-value maximization. The first stage selects the application and its configuration with the maximum VPT among all possible node configurations. The second stage selects the job with the maximum VPT among the outputs from the first stage. We refer to the combination of these two stages as max-max strategy for the remainder of this paper. Before we present our scheduling algorithms, we describe the common steps taken to select the jobs for resource allocation by all the algorithms. We then present an overview of the baseline-VPT and its power-aware variations, followed by a discussion on their limitations. We finally introduce a new power-aware algorithm that combines the strengths of our earlier two power-aware algorithms [23] in a complementary manner, and discuss our design decisions.

### 6.2 Scheduler Execution Flow

Algorithm 1 presents the basic execution flow used by the scheduler to run the different algorithms that are presented in this work. We execute Algorithm 1 at each mapping event. A mapping event is an event at which the scheduling decisions are made by the scheduler. In our scheduler, mapping events are triggered periodically. The jobs considered for scheduling are referred to as the mappable jobs.

**Algorithm 1** Power-aware algorithms execution flow

```
at each mapping event:
1: Update the list of mappable jobs.
2: Update cluster resource status.
3: while (list of mappable jobs is not empty) and (idle
   resources present): do
4:    Estimate execution time for all mappable jobs on all
      possible configurations.
5:    Compute VPT for each configuration of all mappable
      jobs.
6:    Select the mappable job and its configuration with
      maximum VPT.
7:    Remove the selected job from the list of mappable
      jobs.
8:    Update the pool of idle resources by subtracting the
      resources to be assigned to the job.
9:    Configure the assigned nodes with the configuration of
      the job that results in the maximum VPT.
10:   Launch the selected job on the cluster.
```

In step 1, we update the list of mappable jobs to include the jobs from the waiting queue and the newly arrived jobs since the previous mapping event. Then we monitor the completion status of the running jobs, update the system-value with the job-values earned by the completed jobs, and update the trackers used for monitoring the resources in terms of available power and nodes in step 2. In the following sub-sections, we discuss how these resources are used by each algorithm. If the mappable job list is empty or the available resources are insufficient to schedule any of the mappable jobs, then we break out of the while loop in step 3. In step 4, we estimate the execution time for all the mappable jobs on the possible node configurations defined by the user. We compute the VPT metric in step 5 for all the job and node configuration pairs using the outputs from the previous step. In step 6, we filter out the pairs that will need more resources than the number of idle resources in the system, and apply the max-max strategy to select the job and node configuration pair with the maximum VPT for resource allocation. We remove the selected job from the list of mappable jobs in step 7 followed by updating the unused resource trackers in step 8. Then we configure the nodes that are selected for running the job and start the job execution in steps 9 and 10, respectively. A node is configured by setting up the power caps and enabling the thread counts on its CPUs. These steps are repeated over the remaining mappable jobs. After exiting from the while loop (in step 3), we drop the jobs from the waiting queue that may not earn any job-value even if they are scheduled on their best requested configuration, and we wait for a predefined interval before triggering the next mapping event.

### 6.3 Baseline-VPT

The study by *Khemka et al.* [8] introduces the *Utility-Per-Time* algorithm. Their utility function is analogous to our value function. Therefore, we refer to their algorithm as *Value-Per-Time (VPT)*. We implement this algorithm as our baseline-VPT. At the start of an experiment, we equally distribute the power constraint on the HPC system among its nodes. Similar to original VPT, in step 4 of Algorithm 1, our baseline-VPT uses the minimum execution time of a job

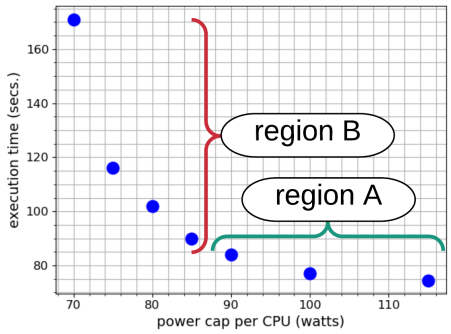



**Fig. 4: Execution time (Y-axis) vs power cap per CPU (X-axis) for the BT benchmark on 16 nodes, and 24 threads configuration. In region A, the impact of the power budget on the execution time is less compared to the impact in region B.**

on a user requested node configuration as the estimated execution time, i.e., it uses the execution time estimates without any power constraints applied to the CPUs. In the baseline-VPT, the *resources* in Algorithm 1 refer to the nodes in the HPC system. Therefore, a mappable job is scheduled only if the system has enough available nodes to meet the timing requirements of the job. This algorithm forms the basis for demonstrating the benefits of using a power-execution time model for making scheduling decisions in a power-constrained HPC system. By not accounting for the effect of the power constraint on the estimated execution time for different jobs, this algorithm assumes that the execution times of all the jobs suffer equally under a given system-wide power constraint. This assumption may not hold as different jobs may have different characteristics resulting in different execution times under the same power constraint [40].

### 6.4 VPT with Common Power Capping (VPT-CPC)

In VPT-CPC, we extend the baseline-VPT to include the effect of power constraints on the job's execution time in step 4 of Algorithm 1. We use the job-specific power-execution time models to estimate execution time for each job. This enables us to calculate the VPT metric with higher accuracy than the baseline-VPT in step 5 and make informed decisions during the job selection stage in step 6. The remaining steps are similar to the steps of the baseline-VPT.

### 6.5 VPT with Job Specific Power Capping (VPT-JSPC)

In the VPT-CPC algorithm, when the power constraint on the system gets tighter, the mappable jobs are either restricted to use the hardware configurations that need a higher node count, or run with an extended execution time. For a given job, when the allocated power reduces below a certain threshold, the percentage amount of decrease in power budget results in higher percentage amount of decrease in application performance [23]. To highlight this point, we present an example in Figure 4 that shows the impact of different power caps on the execution time of the BT benchmark. Uniform power capping forces applications to operate in region B when the system-wide power constraint gets tighter. This decrease in performance reduces the completion rate of the jobs in the system for VPT-CPC. For such cases, allocating the system power and nodes to a subset of mappable jobs will allow completing these jobs and releasing resources earlier for the subsequent mappable jobs in the queue. This greedy allocation of resources pushes application closer to region A in Figure 4. *Patki et al.* have proposed such a resource allocation strategy called power-aware backfilling for a power-constrained HPC environment [15]. In their work, the HPC jobs are moldable and have user defined execution deadlines. Instead of making a job wait for its best node configuration (naive policy) under a system power bound, their approach executes the job on an alternate node configuration that can be created by greedily using the unused nodes and system power at the time of its submission or as early as possible. They have shown that their policy significantly helps in improving the turnaround time and the resource utilization.

VPT-JSPC uses a similar greedy strategy for resource allocation but our work differs from theirs as our objective is to maximize HPC productivity by maximizing system-value earnings as opposed to improving average turn around time for all the jobs. We also consider that each job submitted on our HPC system has a constraint on its completion time instead of its execution time. This increases the limitations on the configuration search space for the job as the current run-time advances towards job's completion time. In our environment, each job has a time-dependent value function whereas in their work all the jobs are assigned a constant priority over time.

In VPT-JSPC, the *resources* in Algorithm 1 refer to the power and nodes in the HPC system. In step 4, while estimating the execution time, we consider all the unassigned power and idle nodes are available to each mappable job. To select a job in step 6, we apply the max-max strategy on the job-configuration pairs whose resource requirements meet the limitations on the available resources. The remaining steps are similar to VPT-CPC.

### 6.6 Hybrid-VPT

Our earlier work [23] shows that VPT-CPC suffers from lower HPC productivity due to decrease in its power efficiency when the constraint on the system-wide power gets tight. Although VPT-JSPC improves HPC productivity under tighter power constraints, it suffers from lower node utilization. The disadvantage of lower node utilization becomes evident when the system-wide power constraint is relaxed and VPT-JSPC under-performs VPT-CPC.

To overcome the aforementioned drawbacks of both VPT-CPC and VPT-JSPC, we propose the *hybrid-VPT*, with the aim of combining the benefits of power allocation policies used by the other two algorithms into a single algorithm. The hybrid-VPT is designed to be similar to VPT-JSPC, but it differs in step 4 in the following way: if the estimated execution time (predicted using job specific models) of a job on its hardware configuration does not exceed its soft threshold, then we replace it with the time remaining to meet its soft threshold. If the predicted completion time of the job falls between its soft and hard thresholds, then its execution time estimation is not altered. The idea is to slow down the jobs that can tolerate spending additional time in the wait queue or can have extended execution

**TABLE 3: Mean % for jobs with maximum job-values.**

| *job-value bin* | [2,4] | [4,6] | [6,8] | [8,10] |
|---|---|---|---|---|
| *mean jobs (%)* | 27 | 22 | 24 | 27 |

on fewer resources before meeting their soft thresholds. This enables the scheduler to save more resources per job. Because the system is oversubscribed, the scheduler uses remaining HPC resources to schedule more jobs to earn more system-value. This is expected to improve the resource utilization and completion rate at a more stringent power constraint compared to VPT-CPC and VPT-JSPC. The potential drawback of this approach is under-utilization of power when the system power constraint is more relaxed because this approach tends to use all the nodes in the system while a considerable amount of power in the system is left unassigned. To bring the benefits of the power allocation policies of VPT-CPC, in step 6, we only consider the node configurations for a given job in which the power constraint on each node is greater than or equal to the power constraint applied by the VPT-CPC for the same system-wide power constraint. With this design choice, our aim is to improve the power utilization of the hybrid-VPT at the relaxed power constraint. The hybrid-VPT is designed to be a single algorithm to improve resource utilization (both, power and nodes) under stringent and relaxed power constraints.

# 7 Emulation Environment

## 7.1 Workload Generation

An HPC system is oversubscribed if, for a given workload, it is not able to earn the maximum job-values for all the jobs in the workload. The subscription level in an HPC system depends on the system-wide power budget and workload characteristics. In this section, we provide an overview of our workload generation approach to ensure the oversubscription in the system under no power constraints. Later, we give a brief overview of our scheduler architecture.

In our study, a synthetic workload trace is a list of jobs in the order of arrival time. We randomly select a scientific routine from Table 1 to form a job in the workload trace. Each job entry in the workload trace consists of job arrival time, job name, maximum job-value, job's input problem size, iteration count, node configuration range, OpenMP thread configuration range, soft threshold, and hard threshold. A combination of all these parameters can affect the oversubscription level in an HPC system. We experimentally select the sampling range for these parameters to ensure our unconstrained HPC system is oversubscribed.

We create 15 unique workload traces. Each trace is composed of 60 jobs in the order of their arrival time. In all the traces, we use an inter-arrival duration of 200 seconds between consecutive jobs and randomly sample the maximum job-value from the range *[2-10]* using a uniform distribution. The mean percentage of jobs in all the workload traces is evenly distributed among four different job-value bins, as shown in Table 3. The sampling range for the problem size is in between class C and class E with a memory footprint between 0.8 and 250 GB, respectively. We randomly sample the iteration count for each job from [50,150]. We randomly select the range for possible node configurations for each job while ensuring that its maximum node count does not exceed the system size. Similarly, we randomly choose the range of OpenMP thread count in between 8 and 24. In a real HPC environment, the selection of soft and hard thresholds is user-dependent. Because our workload trace is artificial, we first estimate the job execution time on an intermediate node count (geometric mean of the maximum and minimum nodes requested by the job). We then use 1.2 and 4.0 times of this estimated execution time as job's soft and hard thresholds, respectively.

## 7.2 Scheduler Design

We create a 64-node HPC prototype on the testbed presented in Section 4. We have a single master node and multiple slave nodes. The master node is only used for running the central scheduler, and it is excluded from the HPC system size. We use the central scheduler to (1) monitor the status of the running jobs, (2) track resource usage, (3) track system-value earnings, (4) retrain application-specific models, (5) execute scheduling algorithms, and (6) launch new jobs on available resources. The scheduler executes these actions at the occurrence of a mapping event in the aforementioned order. For emulation, we set mapping events to occur periodically at an interval of 60 seconds. Slave nodes are used for running real HPC jobs. At the start of our experiments, we launch a light-weight daemon process on each of the slave nodes. This daemon process is used to monitor and report the node and job status to the central scheduler. At the occurrence of each mapping event, for each job, the central scheduler communicates with the slave nodes to collect the running job and node status.

# 8 Emulation Results

## 8.1 Overview

In Section 8.2, we begin by analyzing and comparing the system-value earnings of four resource allocation strategies (baseline-VPT, VPT-CPC, VPT-JSPC, and hybrid-VPT) on a 64-node HPC system under different system-wide power constraints. We then investigate the effect of these algorithms on the utilization of the system resources (power and compute nodes) in Section 8.3. Finally, we demonstrate the strength of the hybrid-VPT algorithm over the other two power-aware algorithms by performing a comparative analysis in Section 8.4.

## 8.2 Performance Analysis

For each algorithm, we run 15 workload traces on our 64-node HPC prototype under three different system-wide power constraints. We apply these power constraints by limiting the system power consumption to a fraction of its maximum power consumption. We choose the three fraction values as 55%, 70%, and 85% to represent tightest, intermediate, and relaxed availability of the power as a resource. This analysis is based on a total number of 180 emulations (four algorithms $\times$ 15 traces $\times$ three power constraints), where each emulation takes anywhere between four to six hours to complete.

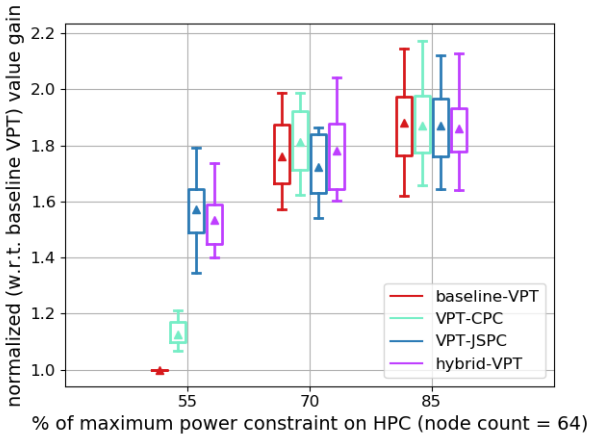



**Fig. 5: The system-value versus system-wide power constraint on the 64 node HPC. The X-axis presents the % constraint on power consumption of the 64 node HPC system, which is a measure of the percentage of the maximum power available for a system. The Y-axis represents the normalized system-value earned by different algorithms. Each box presents the distribution of the normalized system-value for 15 workload traces for a given combination of algorithm and power constraint. The triangle in each box indicates the mean for the normalized system-value.**

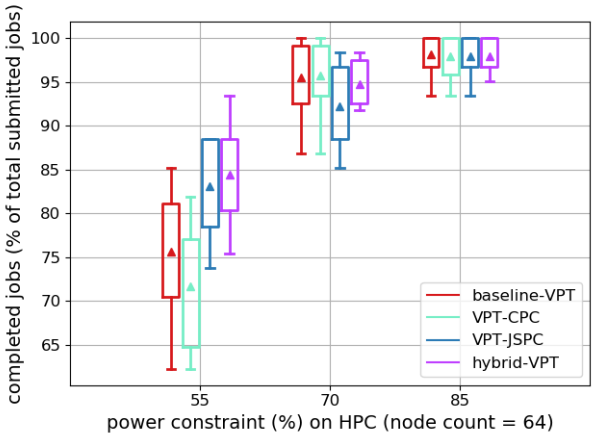



**Fig. 6: The percentage of completed jobs versus system-wide power constraint on the 64 node HPC. The X-axis presents the % constraint on power consumption for the 64 node HPC system. The Y-axis presents the % of submitted jobs that complete their execution. Each box presents the distribution for the completed jobs in 15 workload traces for a given combination of algorithm and power constraint. The triangle in each box indicates the mean % for completed jobs among 15 workload traces.**

In Figure 5, we compare the system-value earnings for the baseline-VPT and our power-aware algorithms under three unique power constraints. The lower magnitude on the X-axis indicates a tighter power constraint. For a given workload trace, the system-value is normalized with respect to the baseline-VPT performance at the 55% power constraint. In Figure 5, at the system-wide power constraint of 55%, the baseline-VPT performs worst compared to other algorithms. At this power constraint, VPT-JSPC earns higher mean system-value (≈38%) compared to VPT-CPC. This behavior reverses at the intermediate power constraint (= 70%) when VPT-CPC earns higher mean system-value (≈6%) compared to VPT-JSPC. These observations highlight the point that the power allocation strategy needs to be more adaptive to the system-wide power constraint. While the mean system-value for the hybrid-VPT is higher (≈35%) than VPT-CPC at the tightest power constraint, it performs similar to VPT-CPC at the intermediate power constraint. Similarly, the hybrid-VPT earns higher mean system-value (≈5%) than VPT-JSPC at the intermediate power constraint, and it performs comparably well at the tightest power constraint. In Figure 5, as the system-wide power constraint is relaxed to 85%, the mean system-value becomes equal for all the algorithms.

To explain the trend in Figure 5, it is important to understand Figures 6, 7, and 8. In Figure 6, we compare the effects of system-wide power constraints on the job completion rate of the presented algorithms. We use Figure 7 to highlight the impact of system-wide power constraints on the job-values earned by the completed jobs. We then derive Figure 8 based on Figure 7 by computing mean system-value (per 100 jobs) of the algorithms under different system-wide power constraints.

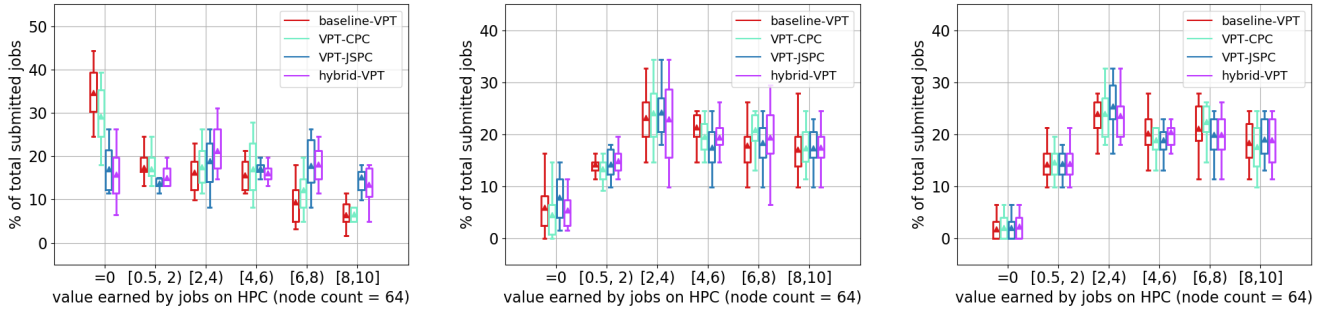
### *Baseline-VPT*

As shown in Figure 7(a), the percentage of submitted jobs earning zero value is higher for the baseline-VPT compared to VPT-JSPC at the tightest power constraint (= 55%). Furthermore, the lower completion rate of the baseline-VPT worsens its system-value compared to VPT-JSPC, as shown in Figure 6. In contrast, the percentage of completed jobs is higher for the baseline-VPT as compared to VPT-CPC but the mean system-value is lower because the number of jobs earning zero value is higher for the baseline-VPT in Figure 7(a). Furthermore, the percentage of jobs earning any positive value by the baseline-VPT is lower than VPT-CPC for the job-value bins in between two and eight. We attribute this poor performance of the baseline-VPT to the fact that it does not account for the job's power budget while estimating the job's execution time. By not accounting for the effect of power limits on the estimated execution time, the baseline-VPT makes an inaccurate estimation of the VPT at the time of making scheduling decisions.

In Figure 8, at the 70% power constraint, the system-value for the baseline-VPT is still lower than VPT-CPC even though the mean percentage of completed jobs is comparable in Figure 6. This is because VPT-CPC has a higher percentage of jobs earning job-value from the bin *[6,8)* compared to the baseline-VPT. In Figure 8, we observe that the mean system-value earned by the baseline-VPT is slightly better than VPT-JSPC because the baseline-VPT completes a higher percentage of jobs as shown in Figure 6. The performance of the baseline-VPT tends to improve and becomes comparable to VPT-CPC as the constraint on the system-wide power is relaxed to 85%. This is primarily because the effect of limiting the power consumption on the execution time diminishes with relaxing power constraint [23].

### *VPT-CPC and VPT-JSPC*

At the tightest power constraint (= 55%), the mean percentage of completed jobs is lower for VPT-CPC as compared to VPT-JSPC (Figure 6). Although the percentage of jobs earning job-value in the interval *[0.5,2)* is higher for VPT-CPC (Figure 7(a)), this is over-shadowed by the higher percentage of jobs earning job-values from bins *[6,8)* and *[8,10]* for VPT-JSPC. As a result, VPT-JSPC has a higher




(a) *power constraint* = 55% (b) *power constraint* = 70% (c) *power constraint* = 85%

**Fig. 7: The percentage of jobs in the workload earning job-values from the different bins for the system power constraint set to (a) *55%*, (b) *70%*, and (c) *85%*. The X-axis shows the binning range for job-value. The Y-axis presents the % of submitted jobs. Each box presents the distribution for the data collected by emulating 15 workload traces.**

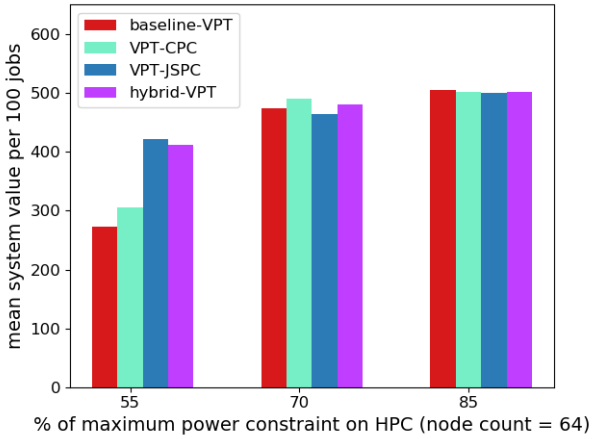
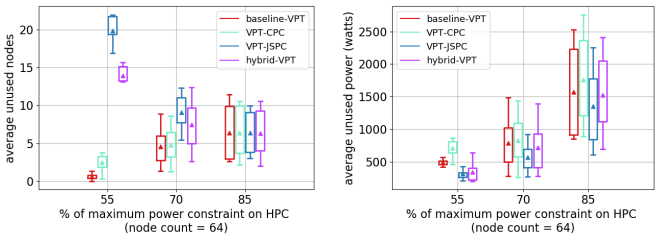



**Fig. 8: Mean system-value per 100 jobs versus power constraint on 64 node HPC system. The X-axis presents the % constraint on power consumption of the 64 node HPC system. The Y-axis presents the mean system-value earned by different algorithms for a workload trace containing 100 jobs.**




(a) *unused nodes* (b) *unused power*

**Fig. 9: Unused resources versus power constraints on the 64 node system. The X-axis presents the constraint on the power consumption of the 64 node HPC system. The Y-axis presents the average (a) *number of unused nodes*, and (b) *amount of unused power* during each emulation. For a given combination of the algorithm and power constraint, each box presents the distribution of unused resources for 15 workload traces and the triangle represents the mean for the unused resources.**

mean system-value compared to VPT-CPC as shown in Figure 8.

At the intermediate power constraint (= 70%), in Figure 7(b), more jobs are earning higher job-values (from bins *[4,6)* and *[6,8)*) for VPT-CPC as compared to VPT-JSPC. VPT-CPC also has a higher percentage of submitted jobs getting completed as shown in Figure 6. Hence, it outperforms VPT-JSPC at the intermediate power constraint. We have discussed the reasons for this behavior while introducing the hybrid-VPT in Section 6.

As we move from the intermediate power constraint to the most relaxed power constraint (= 85%), the mean system-value becomes comparable for all the algorithms as shown in Figure 8. We observe this behavior at the 85% power constraint because allocating a power budget beyond 80% to 90% of the CPU's TDP has a negligible effect on the execution time of our selected benchmarks. A Similar trait is also observed by other researchers in real life HPC applications, e.g., [41], [42]. This loose correlation reduces the variability in the completion time of a job and its VPT estimation in the presented algorithms (Figure 7(c)).

### *Hybrid-VPT*

In Figure 6, the job completion rate for the hybrid-VPT is higher than the baseline-VPT and VPT-JSPC due to more efficient resource utilization at the tightest power constraint (= 55%) (Section 8.3). Even with a higher completion rate, we observe that the system-value for the hybrid-VPT is slightly smaller than VPT-JSPC at the 55% power constraint (Figure 8). Although the hybrid-VPT performs slightly worse than the VPT-JSPC in terms of mean system-value, we will show the superiority of the hybrid-VPT in the next sub-section in terms of resource utilization. As shown in Figure 7(a), a slightly higher number of jobs earns value from higher job-value bins ([4,6) and *[8,10]*) for VPT-JSPC as compared to the hybrid-VPT, hence this leads to slightly higher system-value for VPT-JSPC in Figure 8. The mean system-value for the hybrid-VPT tends to be same or closer to VPT-CPC than the VPT-JSPC at the 70% power constraint as shown in Figure 8.

## 8.3 Resource Utilization

For this analysis, we only include the emulation data until the submission of the last job because the HPC system will eventually transition into under-subscription after the last job is submitted. In Figure 9, we compare the effects of our proposed power allocation strategies under different system-wide power constraints on the utilization of the system resources (nodes and power). The average number

**TABLE 4: Number of times an algorithm performs best in terms of system-value gain under a given system-wide power constraint.**

| *power constraint* | *VPT-CPC* | *VPT-JSPC* | *hybrid-VPT* |
|---|---|---|---|
| 55% | 0 | 10 | 5 |
| 70% | 9 | 0 | 6 |
| 85% | 6 | 5 | 4 |

of unused nodes and the amount of unused power in Figure 9(a) and 9(b), respectively, indicate the unavailability of a compatible job-configuration pair among the waiting jobs to run on idle resources.

At the tightest power constraint (55%), the average number of unused nodes is higher for VPT-JSPC compared to other algorithms. VPT-JSPC accelerates the execution of few high value jobs by allocating more power to them on fewer nodes, and therefore leads to having insufficient amount of power (in Figure 9(b)) to schedule any new jobs from the waiting queue on the idle nodes. In the case of the baseline-VPT and VPT-CPC, the average number of unused nodes is less than the other two algorithms. This is due to the fact the scheduler selects a job-configuration pair that needs a higher number of nodes but suffers from a significant degradation in execution time and job-value because of the equal allocation of power across the HPC nodes. The average amount of unused power is higher for VPT-CPC and the baseline-VPT compared to others because even the idle nodes are allocated a power budget that remains unused whereas other two algorithms allocate system-wide power to the running nodes only. We observe an improvement in the node utilization with the hybrid-VPT compared to VPT-JSPC as shown in Figure 9(a). The hybrid-VPT allows slowing down of a job with higher VPT to meet its soft completion threshold instead of meeting its best execution time (like VPT-JSPC). In return, it saves some amount of unassigned power to schedule more jobs on idle nodes. Hence, we observe more jobs getting completed with the hybrid-VPT as shown in Figure 6.

As the constraint on the system-wide power is relaxed to 70%, the node utilization improves for the hybrid-VPT and VPT-JSPC due to the availability of more unassigned power to schedule new jobs. Whereas, the node utilization for the baseline-VPT and VPT-CPC slightly decreases due to the unavailability of any compatible job-configuration pair. Even though the unused power for the baseline-VPT and VPT-CPC is higher than the hybrid-VPT and VPT-JSPC, their higher node utilization results in a higher percentage of jobs getting completed as shown in Figure 6. The higher system-value earnings of VPT-CPC in Figure 5, compared to the others, indicates its efficient use of system resources.

For the system power constraint of 85%, the unused nodes are comparable for all the algorithms. Even though the power usage of the hybrid-VPT and VPT-JSPC is higher than the others, all the algorithms perform similarly in terms of job completion and system-value gain as shown in Figures 5 and 6, respectively. This is because the selected jobs operate in region A of Figure 4, where the power and performance have a weak correlation.

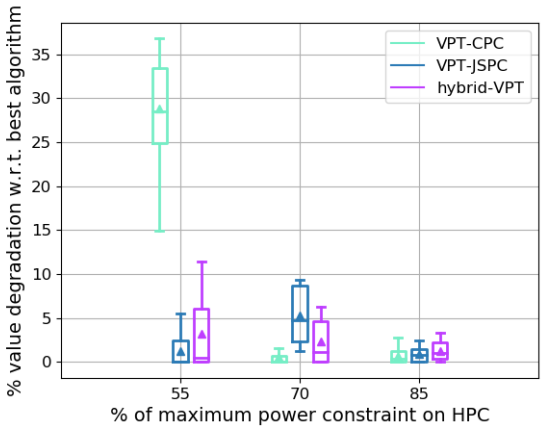



**Fig. 10: Comparative performance of algorithms in terms of % loss of system-value (Y-axis) as compared to the best performing algorithm under different power constraints (X-axis).**

## 8.4 Comparative Analysis

In this section, we present a comparative analysis among our proposed power-aware algorithms using Figure 10 and Table 4. For the comparative analysis of an algorithm with others, we compute the percentage difference (or degradation) in its system-value with respect to the best performing algorithm (Equation 6) for a given combination of power constraint and workload trace by using Equation 7. Each box in Figure 10 represents the distribution for the observed performance degradation on the workload traces that are used in this study. A lower degradation indicates that an algorithm's performance is close to the best performing algorithm. We show the number of times an algorithm outperforms others in terms of system-value for each power constraint in Table 4.

$P$ = set of system-wide power constraints;
$spc$ = system-wide power constraint in %;
$W$ = set of workload traces;
$i$ = workload trace index;
$algo$ = algorithm under consideration (VPT-CPC, VPT-JSPC, or hybid-VPT);
$sv_{algo}(i, spc)$ = system-value earnings of *algo* for $i^{th}$ workload trace at system-wide power constraint of *spc*;
$svd_{algo}(i, spc)$ = % degradation in system-value earnings for *algo* compared to best performing algorithm for $i^{th}$ workload trace at system-wide power constraint of *spc*;

$$sv_{max}(i, spc) = max(sv_{VPT\text{-}CPC}(i, spc), sv_{VPT\text{-}JSPC}(i, spc), sv_{hybrid\text{-}VPT}(i, spc)); \forall i \in W, \forall spc \in P \quad (6)$$

$$svd_{algo}(i, spc) = (sv_{max}(i, spc) - sv_{algo}(i, spc)) \times 100 / sv_{max}(i, spc); \forall i \in W, \forall spc \in P \quad (7)$$

In Figure 10, at the system-wide power constraint of 55%, the mean system-value for VPT-CPC is 28% lower than the best performing algorithm. VPT-CPC under-performs compared to the other two algorithms for all the traces. For ten traces (in Table 4), the hybrid-VPT under-performs compared to VPT-JSPC with the worst observed degradation of 12% relative to the best case of VPT-JSPC. For the remaining five traces, VPT-JSPC under-performs compared to the hybrid-VPT with the worst observed degradation of 6% relative to the best case of hybrid-VPT. The median for our hybrid-VPT is close to zero. This demonstrates the

effectiveness of the hybrid-VPT in delivering a system-value close to VPT-JSPC under the tightest power constraint.

At the system-wide power constraint of 70%, the VPT-CPC algorithm outperforms the hybrid-VPT for nine traces. For the remaining six traces, the performance of VPT-CPC is lower than the hybrid-VPT by less than 5%. VPT-JSPC never outperforms the other algorithms. Its worst case degradation is 10% with the median at 5%. Even though the hybrid-VPT outperforms VPT-CPC for six traces, the VPT-CPC's median and the mean are close to zero. The worst case degradation of 7% is seen for the hybrid-VPT with its median and mean less than 2%. For the system power constraint of 85%, all the proposed algorithms perform equally well.

Based on the observations presented above, we make following conclusions:

- Under the tightest power constraint of 55%, VPT-JSPC and hybrid-VPT are more favorable in improving the HPC productivity as compared to VPT-CPC.
- When the power constraint is 70%, VPT-CPC and hybrid-VPT are more favorable than VPT-JSPC.
- Unlike VPT-JSPC and VPT-CPC, the hybrid-VPT offers an adaptive resource allocation policy under different power constraints.

## 9 Related Work

Time-dependent value functions similar to the ones studied in this paper have been proposed in the literature to improve system productivity [7]–[9], [18], [43]–[47]. Researchers in the cloud computing domain represent the monetary or service level agreement (SLA) value of a job using job-value functions [43]–[47] while the researchers in the HPC domain use job-value functions to represent the importance of completing a job to meet the HPC mission [6]–[9], [18], [20], [21]. In these earlier studies, it is assumed that the expected execution time of the job is known at the time of job submission. This execution time is then used by the scheduler to compute the VPT metric. Most of these earlier studies on value-based heuristics use a greedy approach, similar to the baseline-VPT, to maximize the productivity of the system. However, none of these studies consider the system-wide power as a constrained resource. In our work, we assume that the jobs submitted by the users are malleable and we estimate the execution time at runtime depending on the available resources in the system and input problem size. The performance evaluation approach for most of the earlier work is limited to simulation based experiments whereas we evaluate our algorithms on a real HPC system using real applications. Furthermore, our hybrid-VPT significantly differs from the scheduling algorithms in the literature as we incorporate the time-to-soft-threshold in the VPT metric estimation to improve resource utilization under different system-wide power constraints.

*Yeo et al.* [43] propose a heuristic metric that combines a job's VPT with its completion deadline. In their scheduling algorithm, for each job, they compute the scheduling metric (*return*) by dividing the job's expected VPT with its completion deadline and select the job with the maximum *return*. Their motivation behind this metric is to prioritize a job with the shorter deadline as it needs a shorter commitment, which enables resource manager to accept a later arriving job with a larger *return* value. For a given set of malleable jobs, their approach will always prioritize a job-configuration pair with the shorter execution time and earlier deadline. This will lead to a greedy allocation of the resources (power and node) for the scheduled jobs (similar to VPT-JSPC). As we have shown in our evaluation, under varying system-wide power constraints a system administrator must choose a more flexible power allocation strategy (hybrid-VPT) to achieve higher system-value and job completion count.

In HPC domain, *Khemka et al.* [8], [9] propose multiple greedy heuristics to maximize the total system-value that can be earned by completing jobs. In their work, they use utility functions that are analogous to the job-value functions presented in our study. They do not consider the system-wide power as a constrained resource. *Machovec et al.* [22] propose utility based algorithms for an energy-constraint HPC system. They introduce energy filters for each job to control the rate of energy consumption in the HPC system. In their algorithm, they equally distribute the system-wide energy constraint over the time duration for which the constraint is defined and the scheduler controls the rate of energy consumption by passing each job in the mappable queue through an energy filter to ensure that a selected job-configuration does not exceed the system-wide rate of energy consumption during its execution. If the multiple job-configuration pairs successfully pass through the filter then the one with the maximum Utility-Per-Energy (UPE) metric is used for scheduling. The system administrator selects the time period over which the rate of energy consumption is controlled. However, an energy-constraint system is different than a power-constrained system. The power consumption of an energy constraint HPC system may fluctuate significantly during that controlled time period [34]. In our work, we apply an explicit power cap on each node while ensuring that the total system power consumption stays below the system-wide power constraint. Furthermore, our application modeling and heuristics evaluations are conducted on a real HPC system.

For power-constrained systems, researchers have explored job-specific power allocation strategies (similar to VPT-JSPC) to improve the turn-around time and eventually the completion rate (alternatively FLOPS) of the HPC system [15], [16], [48], [49]. These studies assume that the jobs have equal priority and their completion times are not constrained. As we discussed earlier, FLOPS is a useful metric in defining system performance but it is not sufficient to quantify the productivity of an HPC system [2]–[5]. To quantify productivity of a power-constrained system, we assign job-values and completion deadlines to submitted jobs. Furthermore, the important takeaway from our study is the necessity of a flexible power allocation strategies (hybrid-VPT) to improve system productivity under varying system-wide power constraint.

## 10 Conclusion and Future Work

In this study, we introduced a new power allocation strategy (hybrid-VPT) to improve the productivity and resource utilizaiton of the value-based heuristics in a power-constrained

HPC environment. To realize our proposed allocation strategy, we developed a methodology for creating power-execution time models for a class of HPC applications. These models were integrated with our emulation platform to experimentally compare different power allocations strategies for our VPT algorithms using a real HPC system. We demonstrated the need of a more adaptive power allocation strategy under different system wide power constraints. Later, we successfully addressed this need with our hybrid-VPT.

The power allocation strategies presented in this work are static in nature, i.e., the power budget for a task is allocated at its launch time and it remains constant during its execution. Due to this static allocation, the number of unused nodes increases and the unused power decreases as the system-wide power constraint becomes tighter. This limited amount of power in the system causes newly arrived high value jobs to starve for resources to complete their execution. To overcome this limitation and extract more system-value, we will extend our current research by exploring the impact of dynamic power allocation strategies on value-based heuristics. Our current validation approach is based on a emulation framework executed on a real HPC system. To test our hypothesis on HPC configurations at larger scales, we plan to develop a simulation environment for scheduling algorithms. Such an environment will allow us to reduce the timescale for evaluating new algorithms.

## 11 Acknowledgments

This work is partly supported by National Science Foundation (NSF) research projects NSF CNS-1624668 and CCF-1302693. Part of this work is performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-775437).

## References

[1] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, and J. Dongarra, "Top ten exascale research challenges," accessed date: Mar. 17, 2019, pp. 1-86. [Online]. Available: https://science.energy.gov/ /media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf

[2] M. Snir and D. A. Bader, "A framework for measuring supercomputer productivity," in *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, Nov. 2004, pp. 417–432.

[3] S. Faulk, J. Gustafson, P. Johnson, A. Porter, W. Tichy, and L. Votta, "Measuring high performance computing productivity," in *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, Nov. 2004, pp. 459–473.

[4] J. Kepner, "High performance computing productivity model synthesis," in *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, Nov. 2004, pp. 505–516.

[5] T. Sterling, "Productivity metrics and models for high performance computing," in *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, Nov. 2004, pp. 433–440.

[6] P. C. Broekema, V. L. Allan, and H. E. Bal, "On optimising cost and value in eScience: Case studies in radio astronomy," in *arXiv preprint arXiv:1806.06606*, June 2018, 14 pp.

[7] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility function real-time scheduling and resource management," in *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, May 2005, pp. 55–60.

[8] B. Khemka, R. Friese, L. D. Briceno, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," in *IEEE Transactions on Computers*, vol. 64, no. 8, Sep. 2015, pp. 2394–2407.

[9] B. Khemka, D. Machovec, C. Blandin, H. J. Siegel, S. Hariri, A. Louri, C. Tunc, F. Fargo, and A. A. Maciejewski, "Resource management in heterogeneous parallel computing environments with soft and hard deadlines," in *11th Metaheuristics International Conference (MIC)*, June 2015, 10 pp.

[10] T. Sterling and C. Dekate, "Productivity in high-performance computing," in *Advances in Computers*, vol. 72, Jan. 2008, pp. 101–134.

[11] Z. Marvin, B. Victor, A. Sima, H. Lorin, H. Jeff, and N. Taiga, "Measuring productivity on high performance computers," in *11th IEEE International Software Metrics Symposium (METRICS)*, Sep. 2005, pp. 6–15.

[12] K. Antypas, B. Austin, T. Butler, R. Gerber, C. Whitney, N. Wright, W.-S. Yang, and Z. Zhao, "NERSC workload analysis on Hopper," in *Lawrence Berkeley National Laboratory Technical Report*, vol. 6804, Mar. 2013, 15 pp.

[13] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, and P. Messina, "The opportunities and challenges of exascale computing," in *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, Nov. 2010, 72 pp.

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

[15] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. De Supinski, "Practical resource management in power-constrained, high performance computing," in *24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, June 2015, pp. 121–132.

[16] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," in *21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2016, pp. 545–559.

[17] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "A run-time system for power-constrained HPC applications," in *22nd International Conference on High Performance Computing (HiPC)*, July 2015, pp. 394–408.

[18] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems," in *6th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1985, pp. 112–122.

[19] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," in *Real-Time Systems*, vol. 10, no. 3, May 1996, pp. 293–312.

[20] M. Kargahi and A. Movaghar, "Performance optimization based on analytical modeling in a real-time system with constrained time/utility functions," in *IEEE Transactions on Computers*, vol. 60, no. 8, Aug. 2011, pp. 1169–1181.

[21] C. B. Lee and A. E. Snavely, "Precise and realistic utility functions for user-centric performance analysis of schedulers," in *16th International Symposium on High Performance Distributed computing (HPDC)*, June 2007, pp. 107–116.

[22] D. Machovec, B. Khemka, N. Kumbhare, S. Pasricha, A. A. Maciejewski, H. J. Siegel, A. Akoglu, G. A. Koenig, S. Hariri, C. Tunc, M. Wright, M. Hilton, R. Rambharos, C. Blandin, F. Fargo, A. Louri, and N. Imam, "Utility-based resource management in an oversubscribed energy-constrained heterogeneous environment executing parallel applications," in *Parallel Computing*, vol. 83, Apr. 2019, pp. 48–72.

[23] N. Kumbhare, C. Tunc, D. Machovec, A. Akoglu, S. Hariri, and H. J. Siegel, "Value based scheduling for oversubscribed power-constrained homogeneous HPC systems," in *International Conference on Cloud and Autonomic Computing (ICCAC)*, Sep. 2017, pp. 120–130.

[24] D. Machovec, C. Tunc, N. Kumbhare, B. Khemka, A. Akoglu, S. Hariri, and H. J. Siegel, "Value-based resource management in high-performance computing systems," in *7th Workshop on Scientific Cloud Computing*, June 2016, pp. 19–26.

[25] C. Tunc, D. Machovec, N. Kumbhare, A. Akoglu, S. Hariri, B. Khemka, and H. J. Siegel, "Value of service based resource management for large-scale computing systems," in *Cluster Computing*, vol. 20, no. 3, Sep. 2017, pp. 2013–2030.

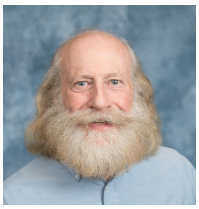
[26] N. Wolter, M. O. McCracken, A. Snavely, L. Hochstein, T. Nakamura, and V. Basili, "What's working in HPC: Investigating HPC

user behavior and productivity," in *CTWatch Quarterly 2*, vol. 2, no. 4A, Nov. 2006, pp. 9–17.
[27] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, "Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system," in *Sustainable Computing: Informatics and Systems*, vol. 5, Mar. 2015, pp. 14–30.
[28] "NAS-NPB benchmark," accessed date: Mar. 17, 2019. [Online]. Available: https://www.nas.nasa.gov/publications/npb_problem_sizes.html
[29] "RAPL," accessed date: Mar. 17, 2019. [Online]. Available: https://lwn.net/Articles/ 545745/
[30] A. Marathe, H. Gahvari, J.-S. Yeom, and A. Bhatele, "Libpowermon: A lightweight profiling framework to profile program context and system-level metrics," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1132–1141.
[31] "Libmsr RAPL," accessed date: Mar. 17, 2019. [Online]. Available: https://github.com/LLNL/libmsr
[32] A. Marathe, R. Anirudh, N. Jain, A. Bhatele, J. Thiagarajan, B. Kailkhura, J.-S. Yeom, B. Rountree, and T. Gamblin, "Performance modeling under resource constraints using deep transfer learning," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2017, 12 pp.
[33] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, June 2007, pp. 835–848.
[34] Z. Liu, J. Lofstead, T. Wang, and W. Yu, "A case of system-wide power management for scientific applications," in *International Conference on Cluster Computing (CLUSTER)*, Sep. 2013, 8 pp.
[35] B. Subramaniam and C. Feng, Wu, "Statistical power and performance modeling for optimizing the energy efficiency of scientific computing," in *International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, Dec. 2010, pp. 139–146.
[36] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers," in *4th ACM European conference on Computer Systems (EuroSys)*, Apr. 2009, pp. 317–330.
[37] Y. Jiao, H. Lin, P. Balaji, and C. Feng, Wu, "Power and performance characterization of computational kernels on the GPU," in *International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, Dec. 2010, pp. 221–228.
[38] D. A. Freedman, *Statistical Models: Theory and Practice*. Cambridge University Press, 2009, vol. 2.
[39] A. Liaw and M. Wiener, "Classification and regression by random forest," in *R news*, vol. 2, no. 3, Dec. 2002, pp. 18–22.
[40] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz, "Dynamic power sharing for higher job throughput," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2015, 11 pp.
[41] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *27th International Conference on Supercomputing (ICS)*, June 2013, pp. 173–182.
[42] A. Marathe, Y. Zhang, G. Blanks, N. Kumbhare, G. Abdulla, and B. Rountree, "An empirical survey of performance and energy efficiency variation on Intel processors," in *5th International Workshop on Energy Efficient Supercomputing (E2SC)*, Nov. 2017, 9 pp.
[43] C. S. Yeo and R. Buyya, "Service level agreement based allocation of cluster resources: Handling penalty to enhance utility," in *International Conference on Cluster Computing*, Sep. 2005, 10 pp.
[44] B. N. Chun and D. E. Culler, "User-centric performance analysis of market-based cluster batch schedulers," in *2nd International Symposium on Cluster Computing and the Grid (CCGRID)*, May 2002, 9 pp.
[45] B. N. Chun and D. Culler, "Market-based proportional resource sharing for clusters," in *Berkeley Computer Science Division Technical Report, University of California*, Jan. 2000, 19 pp.
[46] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing risk and reward in a market-based task service," in *13th International Symposium on High performance Distributed Computing (HPDC)*, June 2004, pp. 160–169.
[47] D. Dib, N. Parlavantzas, and C. Morin, "Meryn: Open, SLA-driven, cloud bursting PaaS," in *1st Workshop on Optimization Techniques for Resources Management in Clouds*, June 2013, pp. 1–8.
[48] T. Cao, Y. He, and M. Kondo, "Demand-aware power management for power-constrained HPC systems," in *16th IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 21–31.
[49] N. Gholkar, F. Mueller, and B. Rountree, "Power tuning HPC jobs on power-constrained systems," in *18th International Conference on Parallel Architectures and Compilation (PACT)*, Sep. 2016, pp. 179–191.

**Nirmal Kumbhare** is a Ph.D. candidate at the University of Arizona under the supervision of Dr. Ali Akoglu. His research interests involve reconfigurable and heterogeneous systems, high performance computing, and power-aware resource management. Prior to joining the Ph.D. program, he worked with Intel for four years as an emulation and validation engineer. He earned his M.Tech in Electronic systems from the Indian Institute of Technology, Mumbai in 2010 and Bachelor Degree in Electronics and Instrumentation Engineering from the Institute of Engineering and Technology in 2008.

**Aniruddha Marathe** is a Computer Scientist at the Center for Applied Scientific Computing (CASC) at the Lawrence Livermore National Laboratory (LLNL). His research focuses on developing performance-optimizing run-time systems for HPC applications on resource-constrained clusters, power-aware computing, trade-offs in HPC cooling systems, and performance variability. Before joining LLNL, he was a Postdoctoral Research associate at the Department of Computer Science at The University of Arizona. He received his Doctorate in Computer Science from the Department of Computer Science at The University of Arizona in August 2014 under the supervision of Dr. David Lowenthal.

**Ali Akoglu** received his Ph.D. degree in Computer Science from the Arizona State University in 2005. He is an Associate Professor in the Department of Electrical and Computer Engineering and the BIO5 Institute at the University of Arizona. He is the site-director of the National Science Foundation (NSF) Industry-University Cooperative Research Center on Cloud and Autonomic Computing. His research interests lie in the fields of high performance computing, reconfigurable computing, and cloud computing with the goal of solving the challenges of bridging the gap between the domain scientist, programming environment, and highly-parallel hardware architectures. He is a member of the IEEE.



**Howard Jay ("H.J.") Siegel** is a Professor Emeritus and Senior Research Scientist/Scholar at Colorado State University (CSU). From 2001 to 2017, he was the George T. Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at CSU, where he was also a Professor of Computer Science. He was a professor at Purdue University from 1976 to 2001. He received two B.S. degrees from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from Princeton University. He is a Fellow of the IEEE and a Fellow of the ACM. Prof. Siegel has co-authored over 460 published technical papers in the areas of parallel and distributed computing and communications, which have been cited over 18,000 times. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was on the Editorial Boards of the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. For more information, please see www.engr.colostate.edu/ hj.

**Dr. Ghaleb Abdulla** earned his Ph.D. in computer science from Virginia Tech in 1998 and M.S. in computer science from the same institution in 1993. He earned his Bachelor Degree in Electrical Engineering from Yarmouk University in Jordan. Before joining LLNL, Dr. Abdulla worked for the Dow Chemical Company as an Information Technology Specialist. Since joining LLNL in 2000, Ghaleb Abdulla has embraced projects that depend on teamwork and data sharing. His tenure includes establishing partnerships with universities seeking LLNL's expertise in HPC and large-scale data analysis. He supported approximate queries over large-scale simulation datasets for the AQSim project and helped design a multi-petabyte database for the Large Synoptic Survey Telescope. Abdulla used machine learning (ML) to inspect and predict optics damage at the National Ignition Facility, and leveraged data management and analytics to enhance HPC energy efficiency. Recently, he led a Cancer Registry of Norway project developing personalized prevention and treatment strategies through pattern recognition, ML, and time-series statistical analysis of cervical cancer screening data. Today, Abdulla is co-PI of the Earth System Grid Federationan international collaboration that manages a global climate database for 25,000 users on 6 continents.

**Salim Hariri** received the MSc degree from Ohio State University in 1982 and the Ph.D. degree in computer engineering from the University of Southern California in 1986. He is a professor in the Department of Electrical and Computer Engineering, University of Arizona and the director of the NSF center for Cloud and Autonomic Computing. His current research focuses on autonomic computing, Cybersecurity, Cyber Resilience, Secure Critical Infrastructures, and Cloud Security.