Production Hardware Overprovisioning: Real-world Performance Optimization using an Extensible Power-aware Resource Management Framework

Ryuichi Sakamoto*, Thang Cao*, Masaaki Kondo*, Koji Inoue†, Masatsugu Ueda†,
Tapasya Patki ‡, Daniel Ellsworth‡, Barry Rountree‡, Martin Schulz‡
*The University of Tokyo
†Kyushu University
‡Lawrence Livermore National Laboratory
*r-sakamoto@hal.ipc.i.u-tokyo.ac.jp

***Abstract*—Limited power budgets will be one of the biggest challenges for deploying future exascale supercomputers. One of the promising ways to deal with this challenge is *hardware overprovisioning*, that is, installing more hardware resources than can be fully powered under a given power limit coupled with software mechanisms to steer the limited power to where it is needed most. Prior research has demonstrated the viability of this approach, but could only rely on small-scale simulations of the software stack. While such research is useful to understand the boundaries of performance benefits that can be achieved, it does not cover any deployment or operational concerns of using overprovisioning on production systems.**

This paper is the first to present an extensible power-aware resource management framework for production-sized overprovisioned systems based on the widely established SLURM resource manager. Our framework provides flexible plugin interfaces and APIs for power management that can be easily extended to implement site-specific strategies and for comparison of different power management techniques. We demonstrate our framework on a 965-node HA8000 production system at Kyushu University. Our results indicate that it is indeed possible to safely overprovision hardware in production. We also find that the power consumption of idle nodes, which depends on the *degree of overprovisioning*, can become a bottleneck. Using real-world data, we then draw conclusions about the impact of the total number of nodes provided in an overprovisioned environment.

***Index Terms*—Power-constrained HPC system; Overprovisioned; Design of resource manager; Plugin interface; Power characteristics of HPC system;**

I. Introduction

High performance computing (HPC) systems are critical for advancing modern science and are a major asset for research organizations. These systems are large scale resources; and most HPC systems are built as clusters of hundreds to thousands of compute nodes. The amount of power required for HPC system operation results in high costs for owning organizations. With many of the current top HPC systems already capable of consuming near 10 megawatts, we are heading towards a regime where future HPC systems are no longer limited by the amount of hardware resources they can deploy, but by the power they will consume.

Traditionally, organizations use *worst case power provisioning*, where enough power is procured to run all components at their peak power consumption at all times. Under this constraint, energy efficiency becomes a major focus for system design. At a coarse granularity, nodes might be powered off to reduce energy consumption (reducing the affected node's power consumption to zero watts), but the time taken to power the nodes back on is significant. Dynamic component level techniques that reduce energy consumption of individual nodes with much finer logical and temporal granularity exist (for example, power gating within processors, dynamic voltage or frequency scaling) and, in aggregate, provide substantial energy savings. However, they also result in time varying behavior without reducing the peak power consumption. As a consequence, worst case power provisioning when used with energy efficiency optimizations—in hardware or software—results in poor power utilization at scale.

As the alternative, hardware overprovisioning [27], [30] has been shown to be an approach to improve power utilization at scale. A hardware overprovisioned system contains more compute nodes than the organizational power budget allows to operate simultaneously at peak consumption. For example, in a system composed of 1,000 nodes, each of which could potentially consume 1kW of power, 1MW of power must be purchased for worst case provisioning. Knowing that the individual nodes are unlikely to consume 1kW at all times during operation, the owning organization might purchase some number of *extra* nodes under the same power budget of 1MW. The *extra* nodes are able to take advantage of the otherwise unutilized power for productive computation.

Study of hardware overprovisioned HPC systems started fairly recently and there are many open questions regarding the design and operation of such systems. Most existing research has focused on small-scale systems or simulations and preliminary performance results; the impact of hardware overprovisioning at large scale and in production environments has not, yet, been examined. Furthermore, the software stack for making such systems accessible to users and administrators

is missing. For overprovisioning to be successful, several practical deployment and operations management questions need to be answered, the three most important ones being:

1) How can power be managed in an overprovisioned system safely in order to maintain the systemwide power budget at scale?
2) How can we fairly measure and evaluate the performance of various power management strategies across HPC centers?
3) How many *extra* nodes should be added to a system to gain the most benefits?

In this paper, we make four research contributions that help to answer these questions: a) we present an extensible power-aware framework based on the widely used SLURM [34] resource manager. This framework provides flexible plugin interfaces that can be used by different HPC centers to enforce site-specific power management policies; b) as a proof by construction, we show results of deploying our power-aware SLURM extensions on a 965-node production HA8000 system that demonstrates that is is possible to deploy a hardware overprovisioned system at scale safely; c) using our platform capable of implementing several power management strategies, we discuss how our SLURM extensions can be used to implement and fairly compare competing power management solutions found in the literature; and finally, d) we use experimental results to investigate the impact of varying the number *extra* nodes, or the *degree of overprovisioning*, given job mixes with differing peak power consumption.

II. Background and Related Work

In this section, we describe general related work about power-aware resource management. The main research topics are job scheduling, dynamic power management, processor variation, and performance estimation. We also discuss SLURM extensions and resource management challenges.

A. Job Scheduling

Previous non-overprovisioned HPC systems optimize node utilization with job scheduling techniques such as backfilling [26]. Energy-aware and power-aware techniques for non-overprovisioned systems have been studied [17], [16], [18], [19], [5]. In an overprovisioned system, the scheduler optimizes both node and power utilization. The scheduler thus needs to adjust the power cap and the job size. Gholkar et al. [21] and Patki et al. [28] optimize system throughput by considering moldable jobs on overprovisioned HPC systems. Patki et al. use a power-aware backfilling algorithm. Sarood et al. [29] design a scheduling algorithm for optimizing both moldable and malleable jobs using integer linear programming. All the aforementioned strategies search for an optimal job allocation and power allocation, while considering performance optimization.

B. Dynamic Power Management

The job scheduling techniques described above determine the distribution of power allocation statically before running a job. This can result in poor power utilization and can also impact system throughput as the unutilized power could have been used to schedule more jobs. To tackle this problem, dynamic power management schemes have been proposed [13], [12], [8], [14]. Cao et al. [8] reallocate power resources dynamically between high priority jobs and low priority jobs. Ellsworth et al. [12] suggest balancing unused power between jobs by using continuous monitoring and capping.

C. Processor Variation

Modern CPUs and DRAMs exhibit heterogenous power and performance characteristics caused by manufacturing variations despite having the same microarchitecture [6], [22]. To make matters worse, this effect is amplified under power caps. For example, if we set the same power cap to processors in an HPC cluster, each processor may end up having a different operating frequency based on its individual characteristics. This introduces load imbalance in perfectly load balanced applications, limiting the performance of to that of the slowest processor. Processor variation adds complexity to finding the best power capping method.

Significant research exists to address this problem [23], [8], [32], and it focuses on two areas of resource management. The first area explores multicore SMP CPUs. Ehsan et al. [32] consider the number of cores that should be used under a power constraint and what the physical layout of these cores should be. On the other hand, Inadomi et al. [23] and Cao et al. [8] focus on multi-node HPC systems. They decide the best node allocation and the power caps in order to maximize application performance in clusters with processor variation. Gholkar et al. [21] propose a 2-level variation-aware approach: at the macro-level, they handle job scheduling, while at the micro-level, they refine the job scheduler's resource selection by using processor variation-awareness.

D. Performance Prediction

Estimating application performance under a power constraint is critical for designing overprovisioned HPC systems and for implementing an efficient software stack with power-aware job schedulers and runtime systems. Prediction accuracy is important, as underestimating power consumption of applications can lead to the system-wide power budget being exceeded. Several researchers are trying to find good methods for estimation. Patki et al. [28] estimate job execution times under the various power caps by using regression models based on profiled data. Inadomi et al. [23] estimate the performance of a job by using a two point estimation method with process variation. First, they generate a process variation table across all nodes at machine setup time and then, once a job gets scheduled, run the job twice on a single node, once without any power caps and once with a tight power cap. Finally, they calculate a best capping value by using the profiling data and the variation table. Sarood et al. [29] propose a power-aware strong scaling performance model based on Downey's [11] model.

E. SLURM Extensions

Most of the aforementioned research evaluates the efficiency of proposed methods using simulators or small-scale schedulers that cannot be used in production systems. Production level HPC systems need to support many features such as accounting, priorities and advanced plugins. It is thus difficult to adapt these algorithms proposed in research literature to production-level HPC systems.

Only a small amount of research work has focused on these practical aspects by extending the existing SLURM resource manager to be power-aware and evaluating their strategies on real HPC systems. For example, Georgiou et al. [20] monitor their HPC system and present results using SLURM. Bodas et al. [3] propose three types of frequency capping methods and also evaluate the efficiency of these with SLURM. Dynamic power allocation results are shown on Cray systems using another SLURM extension [25]. Other research on power-aware resource management in smaller-scale production systems has also been presented [5], [4]. However, all these implementations are specialized for a single site and depend on specific underlying mechanisms, so their extendability is poor. Ellsworth et al. [15] have recently worked towards developing a preliminary infrastructure for unified power management using SLURM, but this is mostly a research effort albeit in the correct direction.

We propose a power-aware resource manager with practical features and with extendability based on the existing SLURM resource manager. We provide plugin interfaces and provide job scheduling and node scheduling plugins and we show a deployment of our approach in a large scale production system. Our proposed SLURM can be easily used to implement various performance estimation, processor variation, and dynamic power management strategies.

III. Design of a Power-Aware Resource Manager

This section describes the design of our proposed power-aware resource manager. It enables the implementation of a wide spectrum of power management algorithms within its framework. Each of these algorithms is expected to have two main elements:

- Estimating a job's power consumption and execution time
- Determining resource allocations (nodes and power) for static and dynamic power management

Figure 1 shows the design of our proposed power-aware resource manager, which is implemented as a SLURM extension. SLURM supports the implementation of new job scheduling and node scheduling algorithms using plugin interfaces. These plugins are compiled as shared objects, enabling different scheduling algorithms (such as backfilling) to be loaded dynamically. For our approach, we reused these existing job and node scheduling plugin interfaces and added plugin interfaces to support static and dynamic power management. We also added power monitoring and power capping control functions available to plugin developers to ease the development of power management functionality.

Our extensions are highlighted in Figure 1. SLURM consists of three primary parts—a user side job submission application (`sbatch`), a centralized scheduler that manages all resources (`slurmctld`), and per-node daemons for local control (`slurmd`). Users request nodes and time on a shared cluster by submitting their job to SLURM. The centralized scheduler, `slurmctld`, then deals with job submission, job scheduling, and node management as shown in the figure. The *job scheduler* optimizes the execution order of jobs from the job queue based on constraints such as priorities, accounts, and job resource requirements. The *node scheduler* selects the best physical node allocation for a job by taking into consideration factors such as number of cores, memory usage, or demands of a specific accelerator.

We add a *power scheduler*, a *node power manager* and a *low-level power plugin interface* to the existing SLURM code. The power scheduler is responsible for scheduling all of the system power, and it does so by monitoring the compute nodes and distributing power. It has three components: (a) Power monitor, (b) Power analyzer, and (c) Power allocator. The power allocator and power analyzer provide extensible plugin interfaces. This allows other plugin developers to implement and test their own power management algorithms. We discuss our extensions to SLURM in detail in the following subsections.

A. Functionality of Power Management

a) Power Monitor:: The power monitor is responsible for periodically monitoring node and system power and for capping the node-level power when required. In the case of hardware overprovisioned systems, it is possible that the total system power may exceed the designated budget unexpectedly due to software bugs or security attacks. Should such a scenario occur, the power monitor will issue an *alert* and lower the power cap value of some compute nodes or kill some of jobs, if necessary, to guarantee safe operation.

b) Power Analyzer:: The power analyzer predicts the execution time of a job based on the number of nodes allocated for the job. The job scheduling plugin uses this information to determine when and where each job in the job queue should be executed for maximizing system throughput. The job scheduling plugin along with the power analyzer also estimates the availability of power and hardware resources for future time windows by sending queries to the node scheduler and the power allocator. The execution time prediction algorithms are provided as plugins.

c) Power Allocator:: The power allocator cooperates with the power analyzer and is responsible for both static and dynamic power control. It determines the optimal power allocation of each job before it is launched. It also periodically updates the power allocation of all the executing jobs dynamically to manage the total system power consumption. Based on requests from the job scheduler, it also estimates future power resource utilization. This functionality is provided for further scheduling optimization, for example, with techniques such as backfilling.

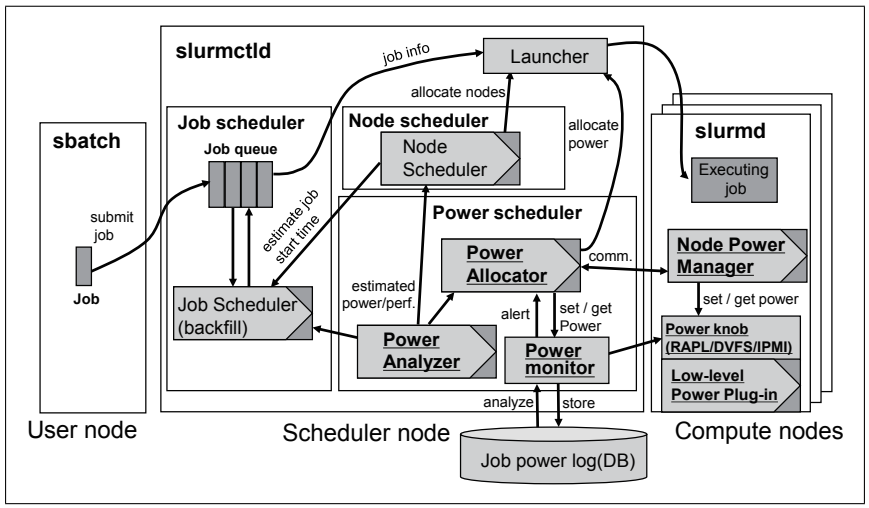

Fig. 1. Design of our power-aware framework in SLURM, including the power scheduler, node power manager and low-level power plugin.

d) Node Power Manager:: A two-level *hierarchical* power management is supported in our design with the help of a node-level power manager. The node power manager has the role of making local power control decisions while cooperating with the job scheduler.

e) Power Knobs and Low-level Power Plugins:: Processor vendors provide different power knobs for measuring and limiting the power consumption of components. Examples include techniques such as DVFS, RAPL, IPMI [1] or PowerInsight [10], [7]. An additional low-level power plugin interface allows us to abstract the differences between the underlying vendor-specific power control mechanisms.

B. APIs

Researchers or system developers can create their own modules for power-aware scheduling and power control using the APIs provided by the plugin interface and the power control daemon. We list these APIs in Table I.

C. Implementing Site-Specific Power Management Strategies

In this subsection, we show how plugin developers can implement different power management strategies by exploring the techniques that have been discussed in the literature so far. Table II shows a summary of existing power-aware research. Ehsan et al. [32] optimize for variation-awareness on a SMP CPU. In their case, the node power manager extensions can be used for implementation. Gholkar et al. [21] use a two-level hierarchical variation-aware approach, and this is supported with the job scheduler, power allocator and the node power manager. Sarood et al. [29] and Patki et al. [28] optimize for job throughput using power-aware job scheduling techniques that depend on integer linear programming and power-aware backfilling. In these cases, the job scheduler and power allocator API is helpful for implementation. Power allocator with `power_schedule` is suitable for dynamic power management work such as the research proposed by Ellsworth et al. [12]. Processor variation is a node characteristic, and the node scheduler APIs can be used to add variation-awareness.

IV. Implementation and Deployment

We demonstrate the operation and effectiveness of our power-aware SLURM variant by deploying it on a large-scale production system. This will help us understand the feasibility and impact of overprovisioning HPC systems at scale. In this section, we describe the system specification and an example power control policy that was implemented using the plugin interface discussed in Section III.

A. Experimental Environment

We use the HA8000 supercomputer system at Kyushu University. The system contains 965 compute nodes and each node has two Xeon processors with 128-GB DDR3 memory. A detailed specification of the system is presented in Table III. The processor supports the RAPL interface [10], [24], which provides users with the ability to monitor and limit power consumption of the CPU and DRAM. In this evaluation, we measure only CPU package power though, due to BIOS limitations.

B. Example Power Management Policy

One way to optimize system performance in overprovisioned HPC systems is to let the job scheduler determine the best power cap for each CPU socket based on the characteristics of submitted jobs. For example, Schöne et al. [31] and Weissel et al. [33] use the information of frequency and LLC-miss rate as hints for determining the power cap value for each CPU. In Cao et al. [8], the users provide the information about acceptable performance degradation of their jobs to the scheduler, which then automatically decides the power cap for each job.

Although there are several ways to optimize the power cap value, our goal in this paper is to analyze the system power consumption and performance in a large-scale HPC

TABLE I
DETAILS OF THE PLUGIN APIS. ALL FUNCTIONS RETURN THE STATUS VALUE.

API Provider	API Name	Description	Main inputs
Job scheduler	init	This function is called when starting SLURM. If the site-specific job scheduler plugin wants to update the job queue periodically, a separate thread should be forked. SLURM provides a slurm_sched_p_schedule function, however, this is not suitable for plugin use.	N/A
Node scheduler	select_p_job_test	This function allocates nodes to the job. It can also reserve nodes for future and calculate job start time. The MODE argument (now or future) determines whether nodes are allocated immediately or in the future.	MODE (Operational mode can be now or future)
		Now is used just before job launch. The best nodes are selected with processor variation-awareness and the node list is populated.	JOB The job descriptor has information about a single queued job. It contains the node list, power cap, start time, etc.
		Future is used for backfilling. This function calculates future node allocation based on *all* job descriptors in the job queue. The future start time of the job is updated based on backfilling requirements.	JOB (Same as above)
Power allocator *(new)*	power_p_job_test	This function reserves power for the job. It checks the current power consumption and returns SUCCESS if enough power is available.	JOB
	power_schedule	Used for dynamic power management. It periodically checks for available power and redistributes power intelligently.	N/A
	power_alert	Used for *emergency* power management. This creates an *alert* when the system power consumption goes over a certain threshold. This function is called by power monitor.	N/A
Power analyzer *(new)*	estimate_job_time	Estimates a job's execution time given power constraints.	JOB
Power monitor *(new)*	get_remote_power	Returns the power consumption of the specified nodes.	NODE_LIST NODE_LIST contains current node-level information such as power consumption, CPU frequency, power cap, etc.
	get_remote_frequency	Returns the CPU frequencies of the specified nodes.	NODE_LIST
	set_remote_power	Sets the power caps for the specified nodes.	NODE_LIST with specified power budgets
	set_remote_frequency	Sets the CPU frequency for the specified nodes.	NODE_LIST with specified maximum

TABLE II
CORRESPONDENCE BETWEEN EXISTING POWER MANAGEMENT STRATEGIES AND THE PROPOSED FRAMEWORK

Power Management Strategy	Job Scheduler	Node Scheduler	Power Allocator	Power Analyzer	Node Power Manager
Ehsan [32]					✓
Gholkar [21]	✓		✓		✓
Sarood [29]	✓		✓		
Patki [28]	✓		✓	✓	
Ellsworth [12]			✓		
Inadomi [23]	✓	✓	✓	✓	
Cao [8]	✓	✓	✓	✓	

TABLE III
SPECIFICATION OF THE EXPERIMENTAL SYSTEM

Total nodes	965
Sockets per node	2
Cores per socket	12
Total cores	23,160
Memory per node	128GB
CPU	Xeon E5-2670 v2 (2.5GHz)
Interconnect	Infiniband FDR
Operating System	Linux with kernel 2.6.32
MPI	Open MPI v1.7.3
Theoretical Peak (Rpeak)	1000 TFlop/s
Linpack Performance (Rmax)	712.5 TFlop/s

production system using our power aware SLURM variant. Therefore, in order to make the analysis independent of the specific optimization methodology, we use a simple, baseline power capping strategy that is driven by user input. We assume that the every user provides a power cap for his/her job at submission time. This power cap can be hidden from the users and determined by system software using a power monitor, profiling tools or dynamic estimation in the future.

For this paper, the SLURM scheduler directly uses the power cap value specified by users. Whenever the scheduler selects a job to be executed, it calculates the expected total system power consumption based on the specified power cap value and tries to keep the system power below the power budget. If the expected power consumption exceeds the budget when trying to execute a job at the head of the job queue, the job has to wait in the queue until enough power is available. The scheduler also uses a backfilling algorithm to improve system performance.

V. ANALYSIS AND DISCUSSION

We perform a variety of experiments to depict the validity and usefulness of the developed power-aware scheduler. First, we evaluate basic power consumption under various conditions on a large-scale system. Second, we analyze power consumption behavior of overprovisioned HPC systems in terms of unutilized compute nodes. We then discuss an important design issue in overprovisioned HPC systems of how many *extra* nodes should be procured.

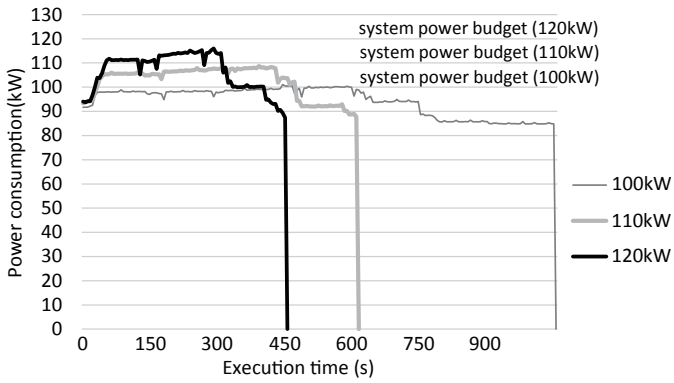

Fig. 2. Validation under different system power budgets

A. Assumptions

In our evaluation, we vary the total power budget of the system and the characteristics of the job mixes executed. We create a synthetic job that computes parallel matrix multiplication on each node. Matrix multiplication is a CPU intensive application and consumes high CPU power, which is about about 85W per CPU socket in our case. In our study, we use three CPU power caps, 85W, 70W, and 55W—the first one being equivalent to not having a power bound (based on the maximum consumption for our synthetic job). Memory intensive applications usually see little performance penalty when low CPU power cap values are used as their performance is not computation bound [9]. We capture the case of memory intensive jobs by using 55W capped jobs, as lowering the CPU frequency affects the rate at which memory operations are performed [23]. Even though our matrix multiplication program is a single node job, we replicate it on several compute nodes to mimic multi-node jobs. This does not generate any communication between compute nodes. In this analysis, we simply omit the effect of communication behavior on power consumption and application performance. While this can be viewed as an unrealistic assumption, it is important to note that the impact of communication performance on several scientific applications is minimal when compared to slowing down the CPU by a factor of 2 (which is what happens between the unbounded 85W and power-capped 55W case). Also note that our goal in this paper is to focus on evaluating the proposed resource manager for deployment. Understanding the impact of communication costs on performance and power in an orthogonal problem, and we plan to include such research in our future work.

The job arrival rate and the number of nodes for each job mimic a real HPC system. We use a job submission trace from the RIKEN RICC supercomputer [2] and follow 600 jobs from the head of the trace for the evaluation.

B. Validation Under Different System Power Budgets

We first evaluate the power consumption of the system with all nodes in the HA8000 system while varying the total power budget. We evaluate three power budgets, 100kW, 110kW, and 120kW. Figure 2 presents the aggregate power consumption across all of the compute nodes.

As shown in Figure 2, our power-aware SLURM variant successfully controls power usage of executing jobs. When the system is fully loaded (middle section of each curve on the graph), the actual power is almost the same as the allocated power budget. This indicates that power-aware SLURM correctly schedules jobs under a restricted power budget in a large scale HPC system and ensures high utilization. Though we focus only on CPU power consumption obtained by RAPL in this experiment, the outcome is applicable to system wide power since the power for the other components is practically constant and can be regarded as an offset.

C. Evaluation of System Power Consumption

In this subsection, we evaluate system power consumption for several configurations of job arrival rates, job mixes and the number of compute nodes with the same total system power budget. We set the system power budget equal to the peak CPU power (TDP) of 500 nodes (125kW).

Since the job arrival rate affects the scheduling efficiency, we first vary the job submission rate. Figure 3 shows the system power usage over time when we vary the job submission rate by 200x, 400x, and 600x of the original rate when 400 nodes are used. The cases of 200x, 400x and 600x indicate low, medium, and high job submission rates. We use the job submission trace shown in V-A and same 600 jobs. The figure shows the power consumption from submission of the first job to the completion of the last job. In the figures, gray and dark gray areas show the power consumption for nodes that are actively assigned to jobs and nodes where no jobs are assigned (idle nodes), respectively. The black lines present the number of jobs waiting in the queue.

The original job submission rate is not enough to fully utilize the available resources as the job execution trace used in this evaluation was collected for much smaller machines than our environment. Even in the case of the job submission rate of 200x, the submission rate is too low to make full use of available compute nodes in the system, resulting in low overall system performance and longer running time. There is almost no job in the waiting queue and the scheduler cannot fully utilize available power and compute node resources. When we increase the submission rate, the power consumption almost reaches the power budget, meaning that system is fully loaded. When comparing the 400x and 600x scenarios, note that almost all the nodes are fully utilized and compute resources are exhausted in both cases. As a result, the execution times do not change, but the number of queued jobs increases faster in 600x. We use the job submission rate of 400x in the rest of the experiments.

Figure 4 shows power consumption for three kinds of job mixes: all low-power (55W) jobs, mix of high (85W) and low (55W) power jobs, and all high-power (85W) jobs. In the case of all low-power jobs, shown in Figure 4-(A), if we take a look at the stable period (portion that the system is fully loaded), almost all of the compute nodes are assigned for executing

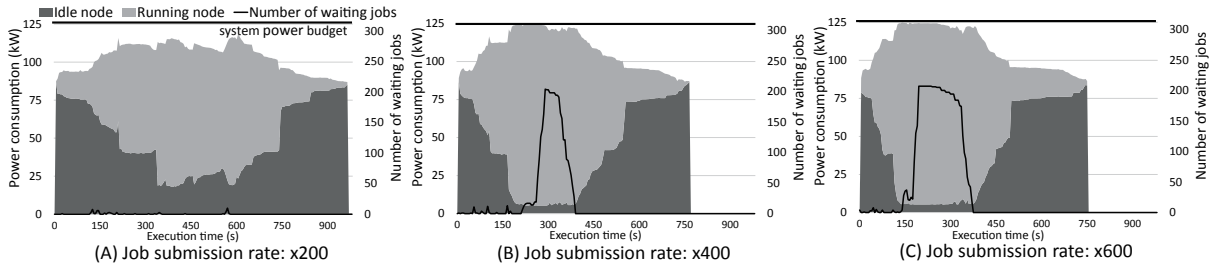

Fig. 3. Power consumption over time for different job arrival rates

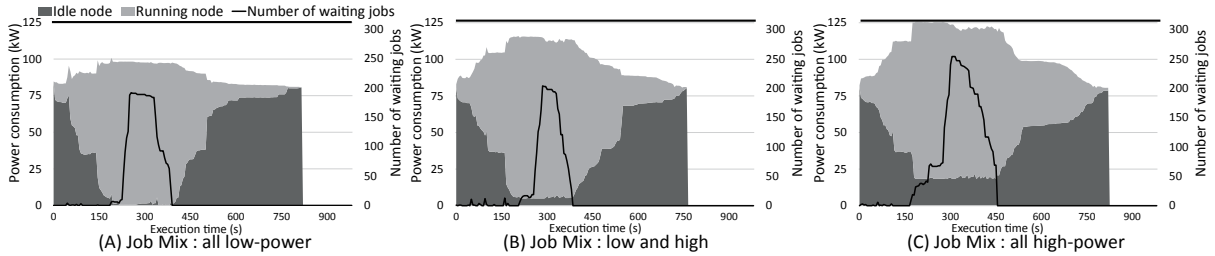

Fig. 4. Power consumption over time for different job mixes

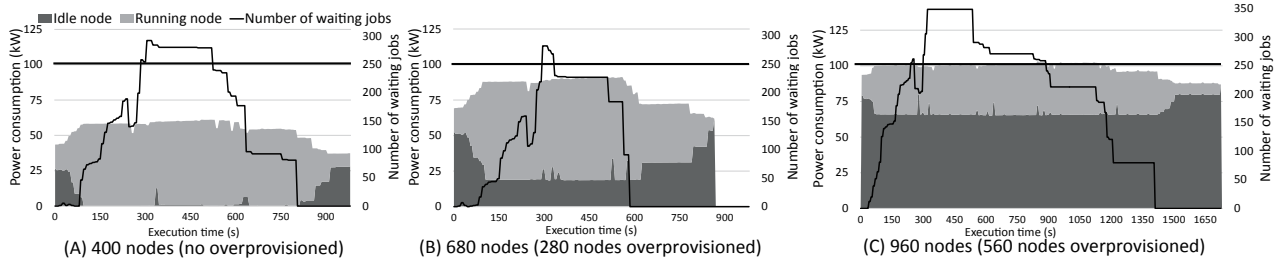

Fig. 5. Power consumption over time for different number of overprovisioned nodes

jobs. However, the power consumption is far below the power budget, which indicates that the power resource is not fully utilized because we run out of compute nodes. On the other hand, in the case of all high-power jobs, shown in Figure 4-(C), the power consumption is close to the system power budget. However, there are many idle nodes as well as many jobs waiting in the queue. This is because we run out of power. The important observation here is that idle nodes consume a non-negligible amount of power and prevent the scheduler from executing more jobs. As stated in Section I, it is not easy to turn off the unutilized compute nodes for a very short period of time in real production systems, and the power consumption of the idle node should thus be taken into account.

Figure 5 shows the results for when we vary the number of overprovisioned nodes (or the *degree of overprovisioning*). under a fixed system power budget. Figure 5 shows the power consumption of 400 nodes, 680 nodes and 960 nodes under the power budget of 100kW (which is the peak for operating 400 nodes). Figure 5-(A) shows that, while most of nodes are actively executing jobs, the power consumption is significantly lower than the system power budget. In this case, the scheduler is blocked because there are not enough nodes to launch more jobs. Figure 5-(B) shows the power consumption for 680 nodes (280 nodes are overprovisioned). In this case, the total execution time is shorter than the 400 node case, leading to better throughput. This is the expected case for overprovisioning. Unutilized power still remains, so more nodes can potentially be added. Figure 5-(C) shows power consumption of 960 nodes (560 nodes are overprovisioned). In this case, the power is almost fully utilized. However, in this case, the scheduler is blocked on power, as most of idle nodes cannot start new jobs due to insufficient power budget. As a result, excessive overprovisioning reduces system performance due to power wasted on idle nodes.

D. Discussion on Designing Overprovisioned Systems

As shown in the previous subsection, power consumed on idle nodes affects system efficiency. The number of idle nodes depends on the characteristics of the job mix and the *degree of overprovisioning*, which reflects the number of *extra* nodes

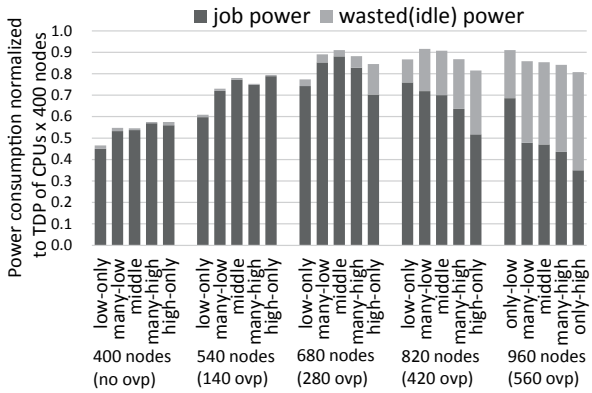

Fig. 6. Average power resource utilization

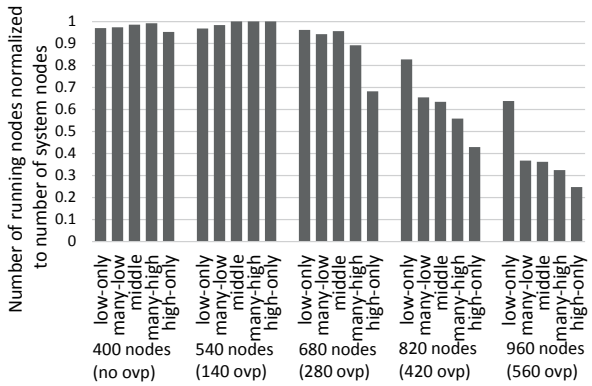

Fig. 7. Average node resource utilization

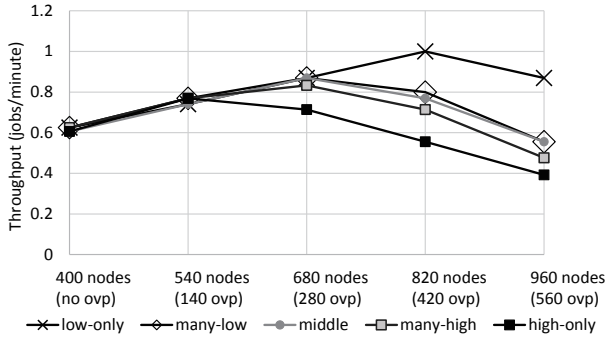

Fig. 8. System throughput (jobs per minute)

that are provisioned. In this subsection, we analyze the power and performance behavior of the system focusing on idle node power. Based on this analysis, we also discuss how to design an overprovisioned system.

We assume the base system design is a non-overprovisioned HPC system with 400 compute nodes whose system power budget is the peak power consumption of 400 nodes of the HA8000 system. Then, we evaluate several cases of overprovisioned configurations: 540, 680, 820, and 960 compute nodes with the same system power budget as that of 400 nodes at peak power consumption.

First, we show average power resource utilization, average node resource utilization, and system throughput under several configurations in Figure 6, Figure 7, and Figure 8. We execute five job mixes and consider different job power ratios of 85W:70W:55W for each job mix. These are 1:0:0 (high-only), 3:2:1 (many-high), 1:1:1 (balance), 1:2:3 (many-low), and 0:0:1 (low-only). The power utilization is normalized to the power budget (400-nodes at peak power). We are interested in comparing the steady state resource utilization in these experiments and produce our statistics from the middle portion of the execution. More specifically, our statistics are based on the system performance while running the middle portion of 200 jobs of the 600 total jobs from the trace.

From Figure 6, it can be seen that when we increase the number of overprovisioned nodes to 680-nodes, power resource utilization increases. However, utilization decreases if the node count increases any further. The node resource is exhausted in cases of small numbers of overprovisioned nodes due to the inability of concurrent work to fully consume the power budget. The power resource is exhausted in cases of large numbers of overprovisioned nodes due to the nodes that the scheduler must leave idle to maintain the power budget. This phenomenon is supported by Figure 7 where node utilization is almost 1.0 in the 400-node and 540-node cases.

Figure 8 shows that system throughput, which is defined as the number of completed jobs per unit of time, tends to improve when we increase the number of nodes. This is expected as overprovisioning provides more computational resources. However, the performance trends in the figure are more complex because utilization decreases for the 960-node case and depends on job mixes. Power-hungry job mixes (with many CPU intensive jobs) prefer a relatively small number of overprovisioned nodes. This is because the average node power consumption is high and power resource is exhausted by small number of nodes. On the other hand, low-power job mixes (with more memory intensive jobs) prefer relatively large numbers of overprovisioned nodes as the power resource cannot be exhausted easily and a larger number of nodes helps increase the number of jobs executing concurrently.

Figure 9 presents energy consumption broken down in terms of energy for actual execution and energy for idle nodes. Here, *job energy* is energy consumed by executing nodes and *wasted energy* is that consumed by the idle nodes. As shown in the figure, wasted energy increases when the number of overprovisioned nodes increases beyond 680. This is expected as with a large number of nodes, we have more idle nodes and insufficient remaining power budget for scheduling jobs.

From the above analyses, we conclude that the degree of overprovisioning and types of job mixes affect both overall system performance and energy efficiency. Therefore, we need to design overprovisioned HPC systems carefully, while taking into account the number of nodes to be overprovisioned as well as typical job characteristics expected on the system.

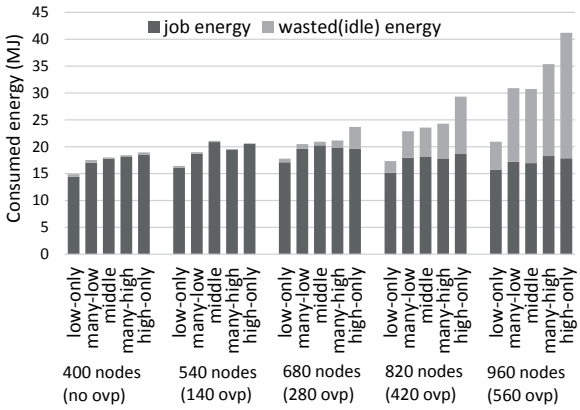

Fig. 9. Breakdown of energy consumption

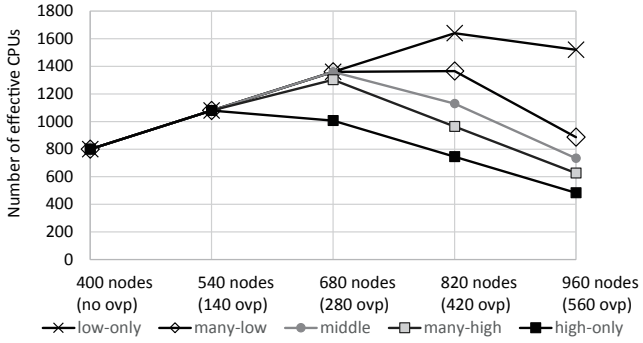

Fig. 10. Model based estimation of aggregate system performance

E. Guideline for Overprovisioned System Design

Based on the analysis presented in the previous subsection, we briefly present a guideline for designing overprovisioned systems. For the sake of simplicity, we approximate system performance by the number of effective CPUs utilized for computation, CPU_{eff}, and use it as our target metric. This guideline determines the number of effective CPUs, CPU_{eff}, based on three things: the system power budget, P_{budget}, the number of installed CPUs, $CPU_{installed}$, and the power characteristics of the job mix. We assume that the job characteristics can be determined roughly at design time based on targeted applications and targeted users. Here, we consider jobs with an average power consumption of P_{avg}.

First, we calculate the number of available CPUs, $CPU_{available}$, based on the system's power budget as shown below in Equation 1.

$$CPU_{available} = \frac{P_{eff}}{P_{avg}} \tag{1}$$

Here, P_{eff} is effective power that can be utilized for computing. P_{eff} is calculated as follows using P_{base}, which is the static power component of the CPUs.

$$P_{eff} = P_{budget} - (P_{base} \times CPU_{installed}) \tag{2}$$

Finally, the number of effective CPUs for computation, CPU_{eff}, is calculated as shown in Equation 3. Note that we cannot use more CPUs than installed.

$$CPU_{eff} = min(CPU_{available}, CPU_{installed}) \tag{3}$$

Figure 10 shows the number of effective CPUs, CPU_{eff}, as our metric for system performance as a function of the installed total nodes and different job mixes. P_{avg} is simply calculated by the combination of power cap assumption in each job mix. The results closely match the throughput measurements shown in Figure 8 from our experiments on the HA8000 system. This validates our guideline and we believe that this metric is a good indicator for system performance. If the expected average CPU power consumption of executing jobs in a particular HPC system is known, this model can tell us how many *extra* nodes we should buy for an overprovisioned system. The limitations of the model are that we do not consider power consumption for other components such as DRAM, fans and network. We also do not consider scheduling slacks. Taking these elements into account is part of our future work.

VI. Conclusions

Hardware overprovisioning is a viable approach for increasing power utilization for HPC systems at scale, but so far existing work was limited to showing its benefits on small scale systems or by solely using simulation. In this work, we have presented the first resource management system targeting large production overprovisioned systems. Based on the widely used SLURM, our framework can maintain a system wide power limit at scale using a set of new plugin interfaces in combination with portable APIs for power measurement and control. We have further shown how several power management strategies in the literature for hardware overprovisioned systems can be mapped to our framework. Finally, through experimentation on a large scale production system at Kyushu University, we have shown that our framework is safe for deployment on production systems and that the job mix executed on a system greatly impacts the degree to which the system should be overprovisioned.

Acknowledgments

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-706278). Additionally, this work was supported by the Japan Science and Technology Agency (JST) CREST program, A Power Management Framework for Post Peta-Scale Supercomputers.

References

[1] http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html.

[2] The RICC log. http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html, 2010.

[3] D. Bodas, J. Song, M. Rajappa, and A. Hoffman. Simple power-aware scheduler to limit power consumption by hpc system within a budget. In *Proceedings of the 2Nd International Workshop on Energy Efficient Supercomputing*, E2SC '14, pages 21–30, Piscataway, NJ, USA, 2014. IEEE Press.

[4] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Predictive Modeling for Job Power Consumption in HPC Systems. In *High Performance Computing: 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016*.

[5] A. Borghesi, C. Conficoni, M. Lombardi, and A. Bartolini. Ms3: A mediterranean-stile job scheduler for supercomputers-do less when it's too hot! In *High Performance Computing & Simulation (HPCS), 2015 International Conference on*, pages 88–95. IEEE, 2015.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th Annual Design Automation Conference*, DAC '03, pages 338–342, New York, NY, USA, 2003. ACM.

[7] M. Broyles, C. Francois, A. Geissler, G. Grout, M. Hollinger, T. Rosedahl, G. J. Silva, M. Vanderwiel, J. Van Heuklon, and B. Veale. Ibm energyscale for power7 processor-based systems, 2011.

[8] T. Cao, Y. He, and M. Kondo. Demand-aware power management for power-constrained hpc systems. In *The 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid'16, 2016.

[9] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[10] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: Memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, pages 189–194, New York, NY, USA, 2010. ACM.

[11] A. B. Downey. A model for speedup of parallel programs. Technical report, Berkeley, CA, USA, 1997.

[12] D. Ellsworth, A. Malony, B. Rountree, and M. Schulz. Dynamic power sharing for higher job throughput. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 80:1–80:11, New York, NY, USA, 2015. ACM.

[13] D. Ellsworth, A. Malony, B. Rountree, and M. Schulz. POW: System-wide Dynamic Reallocation of Limited Power in HPC. In *High Performance Parallel and Distributed Computing (HPDC)*, June 2015.

[14] D. Ellsworth, T. Patki, S. Perarnau, S. Seo, A. Amer, J. Zounmevo, R. Gupta, K. Yoshii, H. Hoffman, A. Malony, M. Schulz, and P. Beckman. Systemwide Power Management with Argo. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1118–1121, May 2016.

[15] D. Ellsworth, T. Patki, M. Schulz, B. Rountree, and A. Malony. A Unified Platform for Exploring Power Management Strategies. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing (to appear)*. IEEE Press, 2016.

[16] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Optimizing Job Performance Under a Given Power Constraint in HPC Centers. In *Green Computing Conference*, pages 257–267, 2010.

[17] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Utilization Driven Power-aware Parallel Job Scheduling. *Computer Science - R&D*, 25(3-4):207–216, 2010.

[18] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Linear Programming Based Parallel Job Scheduling for Power Constrained Systems. In *International Conference on High Performance Computing and Simulation*, pages 72–80, 2011.

[19] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Parallel Job Scheduling for Power Constrained HPC Systems. *Parallel Computing*, 38(12):615–630, Dec. 2012.

[20] Y. Georgiou and M. Hautreux. Evaluating scalability and efficiency of the Resource and Job Management System on large HPC Clusters. In *Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '12, May 2012.

[21] N. Gholkar, F. Mueller, and B. Rountree. Power tuning hpc jobs on power-constrained systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, PACT '16, pages 179–191, New York, NY, USA, 2016. ACM.

[22] S. Hong, S. H. K. Narayanan, M. Kandemir, and O. Özturk. Process variation aware thread mapping for chip multiprocessors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 821–826, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[23] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 78:1–78:12, New York, NY, USA, 2015. ACM.

[24] Intel. *Intel-64 and IA-32 Architectures Software Developer's Manual, Volumes 3A and 3B: System Programming Guide*, 2011.

[25] M. Jette. Slurm Power Management Support. https://slurm.schedmd.com/SLUG15/Power_mgmt.pdf, 2015.

[26] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, June 2001.

[27] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *International Conference on Supercomputing*, June 2013.

[28] 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 *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 121–132, New York, NY, USA, 2015. ACM.

[29] O. Sarood, A. Langer, A. Gupta, and L. Kale. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 807–818, Piscataway, NJ, USA, 2014. IEEE Press.

[30] O. Sarood, A. Langer, L. V. Kale, B. Rountree, and B. R. de Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *International Conference on Cluster Computing*, 2013.

[31] R. Schöne and D. Hackenberg. On-line analysis of hardware performance events for workload characterization and processor frequency scaling decisions. In *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ICPE '11, pages 481–486, New York, NY, USA, 2011. ACM.

[32] E. Totoni, A. Langer, J. Torrellas, and L. Kale. Scheduling for HPC Systems with Process Variation Heterogeneity. January 2015.

[33] A. Weissel and F. Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '02, pages 238–246, New York, NY, USA, 2002. ACM.

[34] A. Yoo, M. Jette, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60, 2003.