PERQ: Fair and Efficient Power Management of Power-Constrained Large-Scale Computing Systems

Tirthak Patel
patel.ti@husky.neu.edu
Northeastern University

Devesh Tiwari
tiwari@northeastern.edu
Northeastern University

Abstract

Large-scale computing systems are becoming increasingly more power-constrained, but these systems employ hardware over-provisioning to achieve higher system throughput because applications often do not consume the peak power capacity of nodes. Unfortunately, focusing on system throughput alone can lead to severe unfairness among multiple concurrently-running applications. This paper introduces PERQ, a new feedback-based principled approach to improve system throughput while achieving fairness among concurrent applications.

ACM Reference Format:
Tirthak Patel and Devesh Tiwari. 2019. PERQ: Fair and Efficient Power Management of Power-Constrained Large-Scale Computing Systems. In *The 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19), June 22–29, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307681.3326607

1 Introduction

Continued progress in high-performance computing has enabled computational scientists to expedite the scientific discovery, but the high power consumption of large-scale systems is one of the top ten challenges for future exascale systems. Large-scale enterprise computing systems in modern data centers also remain power-constrained. This limited power availability requires intelligent job scheduling methods to obtain high system throughput. Prior works have shown that *hardware over-provisioning* can be effective in increasing the efficiency of a power-constrained large-scale system. Under this approach, large-scale systems are provisioned with more number of compute nodes than the system power budget can accommodate if each node were to operate at the maximum power level of the node at all times [45, 55].

Why over-provision large-scale computing systems? The idea of over-provisioning relies on a key insight: applications typically consume lower power per-node than the compute node's specified thermal design power (TDP) limit, as also shown in Table 1 (average per-node power consumption of multiple HPC applications is in the range of 25%-70% of the TDP on an Intel Xeon E5-2686 node). Traditionally, a system designer would provision as many nodes in the system as can be powered up at their peak capacity (TDP) under the given system power budget – referred to as *worst-case provisioning*. Over-provisioning enables the system to concurrently execute higher number of jobs compared to a worst-case provisioned system, thereby, increasing the system job throughput (i.e., jobs completed per unit time). Recognizing this opportunity, researchers have made significant strides in making over-provisioned systems more efficient [19, 20, 45, 46, 54, 55, 59]. Prior works have demonstrated the economic viability of over-provisioned large-scale clusters [47] and provided experimental evidence to support that over-provisioning systems at production-scale is practically feasible and beneficial [53]. Enterprise computing data centers also tend to over-provision hardware resources to meet the peak-load demands [8, 15, 44].

Table 1: Applications from Exascale Computing Project Proxy App Suite [1], used for experiments described in Sec. 3.

Application	Domain	Avg. Power (% of TDP)
ASPA	Multi-scale physics	27%
CoHMM	Material shockwave analysis	27%
CoMD	Molecular dynamics	48%
HPCCG	Conjugate gradient proxy	57%
RSBench	Multipole resonance	39%
SimpleMOC	3D neutron transport in reactor	69%
SWFFT	Cosmology	28%
XSBench	Monte Carlo neutronics	43%
miniFE	Unstructured finite element solver	61%
miniMD	Parallel molecular dynamics	65%

Need for balancing system throughput and fairness on power-constrained hardware over-provisioned system: An intuitive approach to increase system throughput on an over-provisioned system is to execute jobs on all the nodes but power-cap all the nodes at the same level. Such an approach is promising since it is able to use more number of nodes than the base system (worst-case provisioned system) and hence, likely to achieve higher job throughput. At the same time, this approach is also "fair" to different jobs since the power is distributed evenly across all the compute nodes. However, prior works have shown that such a "fairness-oriented policy" does not yield sufficient improvement in the system throughput of over-provisioned systems [19, 33, 42, 46]. Essentially, "fairness-oriented policy" may improve the system throughput but not sufficiently enough to overcome the capital and operational cost of over-provisioning – determined by the *over-provisioning factor* (i.e., the ratio of total number of nodes in the over-provisioned system to the number of nodes in the worst-case power provisioned system, referred as f throughout this paper).

Therefore, to outweigh the capital and operational cost of an over-provisioned system, researchers have designed "throughput-oriented policies" where the scheduler may focus on allocating more power to jobs which are likely to improve system throughput. For example, an intuitive approach is to give maximum power to jobs which are the closest to finishing and are running on the least number of nodes. While effective in improving the system throughput, such policies are fundamentally unfair by design (as our results show in the worst case, such approaches can lead to more

HPDC '19, June 22–29, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6670-0/19/06…$15.00
https://doi.org/10.1145/3307681.3326607

than 200% performance degradation for some applications from their fair performance level (Sec. 3). We note that although large-scale HPC systems are primarily designed for high performance, failure to integrate fairness property into resource management policies can have undesirable severe side-effects (e.g., unintended delay in scientific discoveries, inaccurate resource consumption accounting, unfair treatment of users).

In summary, there is a need for a power management framework that can meet the two conflicting objectives simultaneously: (1) achieve higher system throughput to make up for the cost of over-provisioning, and (2) maintain fairness among jobs. Note that a rule-based or ad-hoc power management strategy would not be able to provably fulfill both the objectives simultaneously because such approaches lack dynamic feedback and theoretical underpinning to provide guarantees. Machine learning based techniques can learn the behavior of applications and adjust power allocation reactively, but can not provide theoretical guarantees when optimizing for multiple conflicting objectives.

We propose PERQ [1], a control-theoretic policy to allocate power in a way which achieves high system throughput, while being fair to the jobs. PERQ uses the principles of multi-objective control theory to achieve both of these goals simultaneously. PERQ uses dynamic feedback to adjust to jobs with diverse characteristics, and to assess the performance impact of its power-capping decisions. The characteristics of HPC applications which enable PERQ to meet these goals are discussed in Sec. 2.

Contributions: This paper introduces PERQ, a new, principled power allocation technique, that ensures that the system is able to make up for the cost of over-provisioning by improving system throughput, while maintaining fairness among the concurrently running jobs. To provide fairness and maximize system throughput, PERQ provisions more power to jobs which have a large impact on system throughput, without hurting the performance of other jobs. This opportunity is enabled by leveraging the observation that applications have different levels of sensitivity to power-capping: some jobs perform equally well at lower power-caps as they do at higher power-caps, while others are more sensitive. Using optimal and robust control theory, PERQ carefully reduces power-caps of certain jobs and increases the power allocation of other jobs to maximize the system throughput. But, making these decisions requires accurately estimating power allocation level and performance relationship for different (unseen) jobs – which is the most challenging task toward providing fair and efficient power management. To overcome this challenge, PERQ builds a novel state-space system model, derived using system identification theory [34], that accurately estimates power allocation level and performance relationship without over-fitting the model to training workloads. Our evaluation using a previously unseen application set confirms that PERQ meets desired target levels and leads to improved performance, although PERQ uses a completely different set of benchmarks to build the system model. Complete PERQ experiments, measurement data, and prototype implementation is available open-source for the research community at `https://github.com/GoodwillComputingLab/PERQ`.

Our experimental and simulation-based evaluation shows that PERQ leverages the difference in power-cap sensitivity of different jobs to achieve higher system throughput for over-provisioned systems, while remaining fair to concurrently running jobs. PERQ's evaluation is driven by characteristics of real-world large-scale HPC systems and jobs. PERQ provides adaptive, stable, and fair treatment for jobs of different characteristics. PERQ provides better system throughput for systems with different levels of over-provisioning. Overall, PERQ improves system throughput by up to 50% points, compared to the fairness-oriented allocation policy, while remaining fair. PERQ helps HPC systems achieve a higher profit over the capital and operational expenses of over-provisioning.

PERQ makes the state-of-art for data center power management landscape richer and easier to advance: While over-provisioned large-scale clusters are the primary target systems for PERQ, the core technical innovations in PERQ solution are applicable and useful for managing power in enterprise data centers as well. Data-center power management has been to central to advancing and improving data center operations. Researchers have designed intelligent power allocation techniques that use feedback-based, multi-level, coordinated approaches to meet different targets such minimizing power budget violations, reducing power oscillations, and improving efficiency [4, 15, 28, 32, 40, 48, 51, 58, 60]. However, as noted by Li et al. in a recent study published in HPCA 2019 [31], concurrent to our work, current state-of-art data center power management schemes are oblivious to workload priority levels and hence, can not maintain fairness. Li et al. [31] make significant advances in designing first workload-priority-aware scalable approach to manage data center power, but do not provide theoretical guarantees about meeting multiple conflicting objectives, convergence, and stability like PERQ does. Integration of PERQinto data center power management framework will bridge an important gap: jointly achieving fairness and efficiency objectives with provable guarantees about stability and convergence in dynamically changing environment. Additionally, researchers can leverage PERQ's principled multi-objective control-theoretic approach to design data center power management schemes. Our open-sourced PERQ model predictive controller modules lower the barrier to entry for designing new data center power management techniques for different target objectives and systems.

2 PERQ Framework

PERQ leverages multi-objective control theory to manage two conflicting objectives: (1) achieving high system throughput to compensate for the cost of over-provisioning (system's objective), and (2) remaining fair to all concurrently-running jobs (job's objective). PERQ employs dynamic feedback to learn about the performances of individual jobs and adjusts their power allocation by power-capping individual nodes to ensure that both job- and system-level targets are met. PERQ provides theoretic guarantees about converging to and remaining stable at the job and system targets. Next, we discuss characteristics of HPC applications to demonstrate the need and feasibility of a feedback-based solution and how PERQ exploits these characteristics to meet conflicting objectives efficiently.

2.1 Enabling HPC Applications Characteristics

Observation 1: HPC jobs are typically long-running and provide opportunity for feedback-based dynamic solution. For a feedback-based dynamic solution to be effective, the jobs have to be sufficiently long-running to make changes and observe their effects. For instance, if the job only lasts for a few milliseconds,

[1] PERQ is a conveniently chosen acronym for power provisioning for efficiency and fair equality, and is pronounced as *perk* (an additional benefit or advantage).

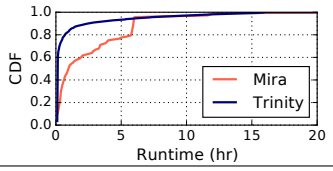

Figure 1: HPC applications are sufficiently long-running to be suitable for a feedback based control.

it would be challenging to apply power-capping and observe its effects in a timely manner to readjust the power-capping level because power-capping itself may take a few milliseconds to take effect. Fortunately, HPC jobs typically execute for several minutes to hours, and some times even days. As shown in Fig. 1, HPC jobs are sufficiently long (in order of hours) on multiple supercomputers, including LANL's Trinity [14] and Argonne's Mira [27] supercomputers. The mean runtime of jobs is 30 minutes on Trinity, and 72 minutes on Mira. Over 62% of the jobs on Mira and 46% of the jobs on Trinity have runtimes longer than half hour. These characteristics enable PERQ to easily apply feedback-based solution where it can apply the control (power-capping) and observe the effect (performance) within the lifetime of a job.

Observation 2: HPC applications may exhibit dynamic power consumption behavior during their execution; making static or ad-hoc power allocation not suitable and requiring adaptive solution like PERQ. Typically, HPC applications do not have monotonous resource utilization during their execution due to their phase-based behavior [18, 26, 30]. Consequently, they exhibit varying power consumption during their execution time. To support this, Fig. 2 shows how the power consumption profiles of three of the applications from Table 1 change during their runtimes (other applications show similar behavior). This result demonstrates that it is thus not possible to employ a static solution which sets one power-cap for the job when it starts, and maintains the same power-cap during its run. This is why, PERQ employs dynamic feedback to adjust to these phases and changes the power-caps accordingly. These phases are often sufficiently long in duration, and do not change very frequently [18, 43]; as HPC applications are often iterative and perform similar computations repeatedly. This enables PERQ's controller to adjust and converge to a stable solution and react appropriately when the phase changes.

Observation 3: HPC applications have varied sensitivity toward power-capping. PERQ exploits an interesting characteristic of HPC applications: power-capping affects the performance of HPC applications to different degrees at different power-capping levels. As shown in Fig. 3, the effect of power-capping on HPC applications can be classified into three categories: (1) low-sensitivity, (2) medium-sensitivity, and (3) high-sensitivity.

Applications with low-sensitivity toward power-capping do not observe dramatic performance degradation as the power-cap level is reduced significantly below the TDP. For example, even at 90W power-cap level (TDP is equal to 290W), applications such as ASPA, CoHMM, and HPCG observe less than 20% performance degradation compared to highest power-cap (290W). This is because such applications are primarily memory intensive or communication intensive. On the other hand applications with high-sensitivity toward power-capping observe dramatic performance degradation

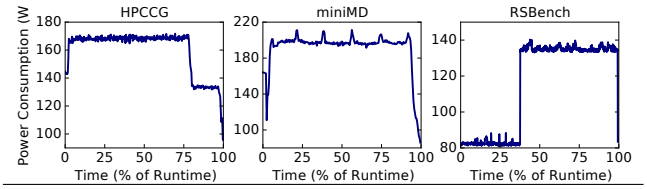

Figure 2: Power consumption profiles of different HPC applications change during their execution time.

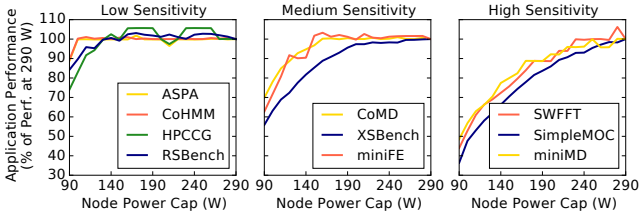

Figure 3: Different HPC applications have different performance sensitivity toward different power-capping levels.

with a steep-curve as the power-cap level drops, more than 60% performance degradation. Applications with medium-sensitivity toward power-capping exhibit in-between behavior, not very steep or flat power-capping-performance curve. We discovered that the number of active cores used by an application is not an absolute indicator of its sensitivity to power-cap. For instance, both, SimpleMOC and miniFE use all of the available CPU cores; however, SimpleMOC is much more sensitive to power-capping, while miniFE is less sensitive than even applications which run on only 32 cores such as XSBench and miniMD. In fact, miniFE's performance curve is similar to that of CoMD, which utilizes only 16 cores. Moreover, an application's sensitivity also changes according to the phase it is in (i.e., compute-intensive phase vs. memory-intensive phase). Due to these complexities in performance impact of power-capping, it is non-trivial to use any ad-hoc policies to meet the two goals.

As described next, PERQ exploits this observation to intelligently re-adjust and shuffle power allocation among jobs such that it remains fair to different jobs while maximizing system throughput.

2.2 PERQ Approach

PERQ uses the principles of multi-objective control theory to achieve both system- and job- level objectives simultaneously. The system-level objective is to achieve proportionally higher throughput than it would have achieved with a worst case provisioned system. The job-level objective is to achieve the performance it would have achieved had the power been distributed equally among the nodes. Note that these targets are not static; they change over time. System-level target varies according to the jobs running on the system, while job-level target varies according to the job's individual performance and phase behavior. PERQ detects the change in the target and re-adjusts itself.

One of the key features of PERQ is that it leverages the insight that different applications exhibit different levels of sensitivity to power-capping. PERQ can cap an application exhibiting low-sensitivity toward power-capping at lower level, while achieving the same performance, and thus, being fair to it. The power saved from such application can be allocated to applications which are likely to improve system throughput. *We note that PERQ does not*

rely on precomputed application models to derive their power-cap and performance relationships. Instead, PERQ uses feedback to dynamically adjust to jobs with diverse characteristics, and to assess the performance impact of its power-capping decisions.

PERQ uses multi-objective control theory to meet the dynamic goals, while satisfying hard constraints of the physical system such as the minimum and maximum power-capping levels of the nodes and the system's total power budget. Multi-objective control theory enables PERQ to individually assign weights to the system-level throughput target and the job-level performance targets. Additionally, PERQ uses dynamic feedback to adapt and adjust to changing situations, and make power provisioning decisions at discreet intervals and for discreet targets. As discussed next, grounded in the theory of multi-objective controls, PERQ enjoys formal guarantees of responsiveness, convergence, and stability.

2.3 Multi-Objective Control

Table 2: Characteristics of control theory.

Feature	Types
Feedback	Continuous, Discrete
Techniques	Classical, Adaptive, Stochastic, Optimal
Size	SISO, SIMO, MISO, MIMO
Model	Static, Dynamic, Uncertain, Constrained

2.3.1 Controller Design A control theoretic solution requires the selection and design of an appropriate controller to meet the desired goal(s). Table 2 outlines the features and system characteristics which categorize different controller types. Traditionally, "classical" controllers belonging to the Proportional-Integrative-Derivative (PID) family are used in many situations, as they provide strong convergence and stability guarantees. However, these controllers are not suitable for PERQ since PID controllers cannot optimally control a system with multiple conflicting goals. On the other hand, controllers that can control multiple objectives and belong to the "optimal" technique family instead of "classical" (e.g., Linear-Quadratic Regulators (LQR) controllers) will not be effective for PERQ's case. This is primarily because such controllers can only provide a one-time optimal strategy for a system (assuming a static system model for infinite time) and hence, are not capable of providing optimal strategies for dynamically changing system and job characteristics, as is the case for PERQ. Therefore, PERQ chooses a Model Predictive Controller (MPC) [29], from the optimal and robust branch of control theory, which can handle multiple conflicting objectives simultaneously (multiple inputs and multiple outputs (MIMO)) and can provide optimal solution in a dynamically changing environment.

Model Predictive Controller (MPC) [29] devises an optimal control strategy for a fixed time window at the current decision instance. The controller analyzes the impact of its previous decision and the new targets to reassess it's control strategy and implements another strategy over the next time window. Use of a receding time window ensures that the control strategy employed is adaptive and adjusts itself to provide optimal solution for the targeted time window. This *time window* is referred to as the *MPC prediction horizon* in MPC literature. PERQ chooses MPC controller as it provides the following desirable properties: (1) PERQ requires the controller to have MIMO controlling ability in order to handle all targets together. MPC allows PERQ to achieve this successfully by allowing to put equal weights on the system-level and job-level objectives (and it can also be configured to assign different weights). (2) Large-scale computing systems are complex and effect of power-cap on performance depends on various factors. An MPC controller can deal with systems with uncertain models and dynamic conditions. (3) An MPC controller can work at discrete intervals, which is required since power-caps take some time to take effect and nodes require a discrete interval to study the impact of the power-cap on performance. (4) Lastly, a "constrained" MPC controller allows PERQ to meet hard system constraints. PERQ needs to ensure that the power-cap set by the controller for each node remains within the thermal design power of the node and the overall power usage of the system remains below the system power budget.

In addition to meeting the above targets, a MPC controller can be designed to ensure that the targets are reached quickly and optimally. The optimality condition refers to (1) the cost of not reaching the target during the prediction horizon, and (2) the cost of making large changes to the "actuator" values from one instance to another (actuator is the component which enables the control; in the case of PERQ, this is the node power-cap). The above two costs reflect the convergence and stability of the controller. A strategy that does not reach the target is non-convergent, and would thus incur a large cost pertaining to (1). On the other hand, it is not desirable to continuously drastically change a compute node's power-cap between two consecutive decision intervals (e.g., from 290W (TDP of an Intel Xeon E5-2686 node) to 100 W to 290W) since it may lead to highly unstable job and system behavior, and may even have adverse effect on the node-health in the long term. Thus, a strategy which is unstable should incur a large cost pertaining to (2). Therefore, the optimal strategy minimizes both of these costs. Next, we describe the mathematical formulation of how a MPC controller minimizes multiple costs.

2.3.2 Mathematical Theory of MPC Controllers The MPC controller minimizes the cost, J, by solving a quadratic programming problem (Eq. 1) every decision instance. It finds the actuator values which minimize the overall cost for the next instance, providing a convergent and stable solution. In Eq. 1, M is the prediction horizon, N is the number of targets, $T_{i,j} - Y_{i,j}$ is the difference between the target value and actual value (W_T is the corresponding weight), and $U_{i,j,k+1} - U_{i,j,k}$ is the change in the actuator values between instance k and $k+1$ (W_U is the corresponding weight). Thus, the cost is proportional to the square of the differences between the targets and the actual values, and the changes in the actuator values.

$$J = \sum_{i=1}^{M} \sum_{j=1}^{N} (W_T (T_{i,j} - Y_{i,j})^2 + W_U (U_{i,j,k+1} - U_{i,j,k})^2) \quad (1)$$

MPC controllers have been shown to provide convergence and stability by using the conditions of "terminal cost" and "terminal region" [6, 9, 39]. Terminal cost is a cost which is added to the total cost if the solution does not converge by the last stage in the prediction horizon. Typically, a large terminal cost is chosen in order to enforce convergence. On the other hand, terminal region is a set of states (a state is one permutation of all actuator values i.e., in the case of PERQ, a state is a permutation of node power-caps of all nodes in the system) which the last state in the prediction horizon must belong to in order to ensure that the solution converges by the last stage in the prediction horizon. A large penalty cost is placed on the controller if it does not converge to one of the states in the terminal region, therefore, enforcing convergence and stability. Typically, the terminal region is dictated by the constraints

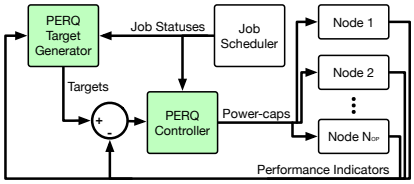

Figure 4: Conceptual overview of PERQ feedback control flow.

of the system and the optimality condition of the controller [39]. In the case of PERQ, the terminal region includes states which meet the power constraints, and minimize the cost sufficiently (as determined by the weight parameters). It might in some cases be infeasible to achieve a cost of zero (i.e., impossible to meet all goals simultaneously); therefore, MPC controller allows convergence to a non-zero cost state as well. However, any state which does not meet the above requirements is penalized, and therefore, not allowed. By meeting these conditions, it can be ensured that the controller converges as close as possible to the targets, and remains stable around the final values.

2.4 PERQ Feedback Control

Fig. 4 provides an overview of PERQ dynamic feedback control loop that helps PERQ adapt to changing job behavior and system environment, while meeting the system- and job-level performance objectives. Compared to a traditional system, PERQ adds two new components: "PERQ Target Generator" and "PERQ Controller". PERQ target generator module determines the target levels: system throughput for the whole system (efficiency) and the job-level performance for each job (fairness). PERQ controller receives two inputs: (1) target for system throughput for the whole system and job-level performance for individual jobs from the target generator, and (2) the current system throughput and current performance level of different jobs from compute nodes. At each decision interval, PERQ controller determines the power allocation for each job (and hence, power-cap level for individual compute nodes) based on the difference between the target level and current performance level. Power-capping bounds the power consumption of compute nodes as determined by the controller for a given decision interval. The feedback process is iterative and discrete, where at the end of each decision interval, compute nodes send the job's performance under the set power-cap for this decision interval to the PERQ target generator. Then, PERQ target generator determines the new targets based on the dynamic job behavior and system state – note that the target levels for both, the system and the individual jobs, may change over decision intervals as different jobs arrive, change phases, or finish. Then, the controller receives the difference between the targets and the observed performances, and determines the power-caps to ensure accurate goal tracking (the difference between the goal and the actual performance should be as small as possible, i.e., the performance should track the goal).

PERQ target generator and controller communicate with the job scheduler and resource manager to learn the status of currently running jobs and their node allocations, whenever a new job arrives or finishes. At every decision interval, compute nodes provide performance data that is used to calculate the current job-level performance and system throughput by the controller and estimate the target job-level performance and system throughput by the target generator.

PERQ uses a simple, effective and easy-to-measure metric, instructions per second (IPS), as the performance/progress indicator. Compute nodes send IPS of the MPI ranks running on the node at the end of each decision interval. Note that other performance indicators can be used, but PERQ prefers IPS for various reasons. IPS is available every decision interval. Traditional methods of judging progress of HPC jobs such as length of a time loop or time between MPI barriers cannot be used as performance indicators because the information is not available every decision interval. It can take a long period to assess the performance impact of a certain power-cap if the time to execute a loop iteration is long. Moreover, the length of a loop iteration may vary from one job to another job, which makes it impossible to make power provisioning decisions for all of them at the same time (i.e., end of a decision interval). We note that the target generator and the controller use the IPS of the slowest job (MPI) process to determine its performance – this avoids the misleading scenario where a process waiting on a synchronization barrier may falsely indicate fast progress for the job.

The controller uses the difference between the target IPS and the current IPS to set the power-cap for each of the nodes. The target generator aggregates the IPS of all the active jobs to estimate the current system-level throughput. We note that target generator needs projected IPS information at different power-caps to generate new system- and job-level performance targets for next decision interval. Sec. 2.4.1 and 2.4.2 discuss how a state-space system model is identified to achieve this purpose.

2.4.1 PERQ System- and Job-Performance Target Generation The target generator determines the system-level and job-level targets at each decision instance.

From the system's perspective, the system throughput target of the over-provisioned system (T_{OP}) should be higher than the throughput of an equivalent worst case provisioned system (T_{WP}). It can be expressed, in a general form, as $T_{OP} = T_{\text{improv-ratio}}\ T_{WP}$, where $T_{\text{improv-ratio}}$ is the factor of improvement which is expected to be proportional to the increase in the size of system. For example, a system with 20% over-provisioned nodes may expect the throughput to go up by 20%. We note that such an expectation is clearly unrealistic at very high over-provisioning factors (f). Our evaluation in Sec. 3 shows that PERQ is not sensitive to the value of system throughput improvement ratio ($T_{\text{improv-ratio}}$).

T_{OP} is determined dynamically based on the set of currently running jobs on the over-provisioned system, A_{OP}, and their respective performances. To determine the system throughput target, PERQ needs to know the performance of a worst case power provisioned system given a set of jobs $A_{WP} \subset A_{OP}$ running at highest power-cap (a worst case power provisioned system has fewer jobs than an over-provisioned system because it has fewer nodes; therefore, A_{WP} contains the subset of jobs in A_{OP} which can be accommodated by a worst case power provisioned system on FCFS basis). The performance of all jobs A_{WP} is summed up to provide the system's performance. However, it is infeasible to actually run the jobs on a worst-case provisioned system to determine T_{WP}. Therefore, PERQ uses a system model, discussed next, which enables PERQ to predict the IPS of jobs in A_{WP} when they are running at maximum power-cap TDP (Sec. 2.4.2).

From the jobs' perspective, the performance target for each job is to achieve the performance that it would under equal power allocation across the nodes (i.e., fairness-oriented policy). This fair power allocation can be estimated $P_{OP} = TDP \times \frac{N_{WP}}{N_{OP}}$, where N_{OP}

and N_{WP} are the number of nodes in the over-provisioned and worst case power provisioned systems, respectively. Thus, PERQ needs a system model to be able to predict the performance of each jobs during its current phase at TDP and P_{OP}. We discuss the details of this system model in the next section (Sec. 2.4.2).

2.4.2 PERQ System Model To estimate the performance of a job at maximum power-cap (i.e., TDP) and fair power allocation (P_{OP}) in an over-provisioned system, PERQ develops a system model that captures the power-cap and performance relationship for different jobs on a given system. PERQ builds a state-space model [49] derived using system identification theory [34]. To ensure that the model is not an over-fit for the running workloads, PERQ uses a completely different set of benchmarks to build this model, and as our evaluation (Sec. 3) shows, PERQ performs effectively for a different set of unseen application set.

To develop a system model that captures power-cap and performance relationship, one could attempt to build an analytical model for the processor but the complexity of a processor would render such an approach ineffective. For example, Intel processors supporting Running Average Power Limit (RAPL) [12] power-cap features have a complicated power-cap vs. performance relationship due to hardware features such as dynamic voltage and frequency scaling (DVFS), dynamic clock modulation, C-state and P-state management, and Turbo-boost. The power-cap vs. performance relationship is further complicated by software and job activity such as OS jitter, number of active threads, CPU and memory intensity. To address this challenge, PERQ employs a data-driven model. The performance of different NAS Parallel Benchmarks (NPBs) with different input sizes was collected for different levels of power-cap on the Intel processor used for evaluation. NPB benchmarks cover a wide range of application behaviors with different input sizes, which is used to obtain power-cap vs. performance curves of HPC applications with different sensitivity toward power-capping. Results were measured by running each benchmark one hundred times and switching the power-cap frequently using a uniform distribution, to emulate a real switching environment that captures impact of power-capping for different phases of the benchmarks. PERQ, then, uses the system identification method [34] to develop a state-space model for the power-cap vs. performance relationship of the node. Note that, such a model is sufficient for all nodes of the same type; but it needs to be rebuilt for a different type of node. However, an HPC system does not change its node type frequently, therefore, development and use of "build-one-time-use-through-out-lifetime" model is reasonable to develop the PERQ controller.

State-space model predicts the performance impact of a particular power-cap based on previous inputs (previous power-cap levels) and outputs (estimated performance vs. observed performance) [49]. PERQ uses a 3^{rd} order state-space model that uses the previous three power-caps ($P(k-3)$, $P(k-2)$ and $P(k-1)$) and outputs IPS at the current instance, denoted as $Y(k)$, based on the current power-cap $P(k)$ at decision interval k. Note that it is not possible to assign a static power-cap vs. performance relationship to the system as the job performance depends not only on the system hardware features, but also its own characteristics. Moreover, jobs are also prone to showing varying performance based on the phase in which they are operating. Therefore, PERQ employs a model which takes into account the dynamic impact of power-cap on the jobs' performance. Moreover, state-space models are linear and time-invariant allowing for low-overhead analysis.

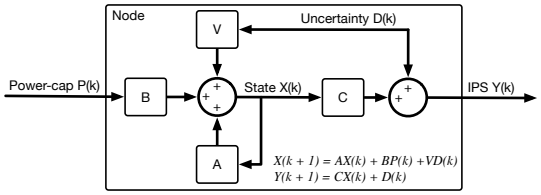

Figure 5: Conceptual view of the PERQ's state-space model of the system that captures the power-cap and performance relationship, accounting for processor complexity and uncertainties.

Table 3: MPC matrices referred to in Eq. 2, Eq. 3 and Eq. 4.

N_J	Number of jobs running on the system
N_{OP}	Number of nodes in the over-provisioned system
M	Prediction horizon of MPC controller
T	Vector of performance targets
Y	Vector of actual performances
P	Vector of power-caps of nodes
ΔP	Vector of changes in power-caps from previous instance $(k-1)$ to current instance (k)
W_T	Matrix of weights on system ($W_{T_{sys}}$) target and job ($W_{T_{job}}$) targets
$W_{\Delta P}$	Matrix of weights on changes in power-caps (ΔP)
X_0	Matrix of current node states at instance k
P_0	Vector of current node power-caps at instance k
Q	Vector of system states for the next M intervals
G, F	Control theoretic matrices which map X_0 and P_0 to Y, respectively
H, D	Control theoretic matrices which map W_T and $W_{\Delta P}$ to Q, respectively
J	Cost function to be minimized

Fig. 5 provides a visual representation of the state-space model of a node. MATLAB's system identification tool is used to develop a controllable state-space model, taking into account the uncertainties of the system. The figure also shows the equations which govern the state space node model, where $P(k)$ is the node power-cap, $X(k)$ is the state of the node, $Y(k)$ is the IPS of the node, and $D(k)$ is a disturbance signal which accounts for system noise and uncertainties. Matrices A, B, C and V map the corresponding signals to the output signal. These matrices define the system behavior and are identified by the system identification tool based on the measured data. The internal state $X(k)$ of the node gets updated every decision instance based on the active input-output relationship of the currently running job. PERQ uses this model to estimate the performance of a job at maximum power-cap (i.e., TDP) and fair power allocation (P_{OP}) in an over-provisioned system.

2.4.3 PERQ MPC Controller Next, we proceed to design a controller for the above system model. As discussed in Sec. 2.3.1, PERQ employs an MPC controller to optimize the decision cost and attain both targets simultaneously. Expanding on Eq. 1, and specializing it for PERQ, the decision costs related to PERQ's MPC controller are mathematically represented in Eq. 2, which shows the overall cost of a control strategy, denoted as J, during a prediction horizon M. The controller predicts the performance of the N_J jobs and the system for the next M control decision intervals on the N_{OP} nodes based on the optimizing strategy. $T_{j,i} - Y_{j,i}$ refers to the difference between the target performance and the actual performance during instance j of job i, and $W_{T_{job}}$ is the corresponding weight. $\Delta P_{j,i}$

refers to the change in power-cap on node i between instances $j-1$ and j, and $W_{\Delta P}$ is the corresponding weight. The parameters and variables used in Eq. 2 are listed in Table 3.

$$J = \sum_{j=1}^{M} \left(W_{T_{job}} \sum_{j=1}^{N_J} (T_{j,i} - Y_{j,i})^2 + W_{\Delta P} \sum_{i=1}^{N_{OP}} (\Delta P_{j,i})^2 + W_{T_{sys}} (T_{j,sys} - Y_{j,sys})^2 \right) \quad (2)$$

Representing J in matrix form gives the following Eq. 3.

$$J = (T - Y)^T W_T (T - Y) + \Delta P^T W_{\Delta P} \Delta P \quad (3)$$

The MPC controller minimizes the cost, J, by solving the quadratic programming equation shown in Eq. 4 (reduced from Eq. 2). It finds the power-caps for all the nodes which minimize the overall cost for instance $k+1$. This controller provides a dynamically convergent and stable solution, as was discussed in Sec. 2.3.2.

$$\text{find } P \text{ to minimize } J = \frac{1}{2} P^T Q P + Y^T P \quad (4)$$
$$\text{where } Q = H^T W_T H + D^T W_{\Delta P} D$$
$$\text{and } Y = H^T W_T (T - G X_0 - F P_0)$$

Overall, PERQ manages multiple targets by monitoring job characteristics and reacting accordingly. PERQ takes the job progress indicators (IPS) as the input, and outputs the power-caps for the nodes, solving the above optimization problem, and sends the new power-cap level to the nodes at every decision instance to reach the optimal and stable solution, as shown in the evaluation (Sec. 3).

2.4.4 PERQ Scope and Limitations PERQ requires node-level power-capping feature to be enabled in the processor (e.g., Intel's Running Average Power Limit (RAPL) interface). It may take a few milliseconds to apply and observe the effect of power-capping. Therefore, the control decision interval has to be order of seconds. Our control decision interval is ten seconds, which provides effective performance (Sec. 3). Consequently, PERQ is most effective when jobs run for at least longer than a few minutes to span a few control intervals – as is often case for a productive and useful HPC system; jobs shorter than a few minutes do not require complicated power allocation decisions. We note that the PERQ's decision making is not on the critical path and it makes adjustments at the end of each decision interval for new job arrivals and terminations. The length of PERQ's decision interval does not need to change dynamically and can be configured as needed. PERQ has no limitations or restrictions in terms of a job's size (i.e., number of nodes it spans). Our evaluation considers real job trace from different supercomputers that comprise of different job sizes and runtime lengths. Finally, PERQ's performance is not dependent on the benchmark suite used for developing the state-space model, but development of this model requires one-time effort for a given node-type. We anticipate this to be a reasonably low investment effort (less than a week of time on only a few nodes – one for each type), for a multi-thousand node cluster expected to run for multiple years. We will make the PERQ controller code publicly available for easier adoption and extension of PERQ inspired approaches.

3 Evaluation

Methodology: PERQ evaluation is driven by system parameters and job characteristics of real-world supercomputers. We perform both real-system experiment based evaluation and simulation based exploration to understand the performance trends of PERQ. Note that performing a real-system experiment on a production supercomputer with power-capping is both time- and cost-prohibitive. To address this, we perform simulation based exploration to gain deeper insights and demonstrate that PERQ performs effectively on a wide variety of real-world system parameters and job characteristics. To drive our simulation study, we use trace of jobs which were executed on Mira and Trinity supercomputers [2, 14, 16, 27]. In particular, job characteristics such as the number of nodes they are running on and their runtime are used for the simulation. System characteristics such as the number of nodes are also emulated from Mira and Trinity. Mira contains a total of 49,152 IBM PowerPC A2 nodes, while Trinity consists of 19,420 Intel Xeon nodes. For the simulation, we use First-Come-First-Serve (FCFS) with back-filling job scheduling, while making sure that there is always a job available to run at the head of the queue. This is done in order to obtain the true improvement in the job throughput of an over-provisioned system. The statistical distribution of job characteristics such as job size and runtimes are kept same the simulated cluster (Mira and Trinity), but to make the evaluation tractable, the simulation is configured to emulate one day (24 hours) of an actual system. This translates to 1052 and 1024 jobs on Mira and Trinity systems, respectively, for the over-provisioned system with $f = 2.0$. Note that power consumption profile of the jobs running on these clusters is not collected and hence, is not directly available for using as an input to our evaluation. Therefore, the power-performance relationship profiles and phase-behavior of jobs are taken from the application mentioned in Table 1 (taken from Exascale Computing Project (ECP) Proxy Application Suite). Each job is assigned the power-performance characteristics of one of the ten applications using a uniform distribution to have diverse and representative range of behavior. We use the CVXOPT [3] package to solve the quadratic programming aspect of PERQ's MPC controller.

The PERQ prototype is deployed on a local HPC cluster: Tardis. A private network is used on Tardis consisting of 16 nodes: one node being the scheduler node (running the python job scheduling, target setting and controlling features), and others being the cluster nodes (running the actual jobs and performing power-caps). The nodes consist of Intel Xeon E5-2686 processors, and have socket-level RAPL power-capping functionality. The cluster uses Infiniband interconnect for communication over the network. All nodes communicate with the scheduler over a TCP socket about power-cap, IPS, and job start and finish information. Like the simulation, the prototype also uses the CVXOPT package to solve the quadratic programming problem every instance, and uses FCFS scheduling. Lastly, a 100 jobs are run for the prototype for each over-provisioning factor and policy, and the benchmarks used for the jobs are taken from Table 1. These runs last for hours on the full cluster and are repeated multiple times to obtain statistically significant results.

We first present simulation based evaluation driven by Mira and Trinity parameters to demonstrate that PERQ works across different job and system characteristics and then, followed by prototype results that support our simulation results and provide research community an implementation prototype for adoption.

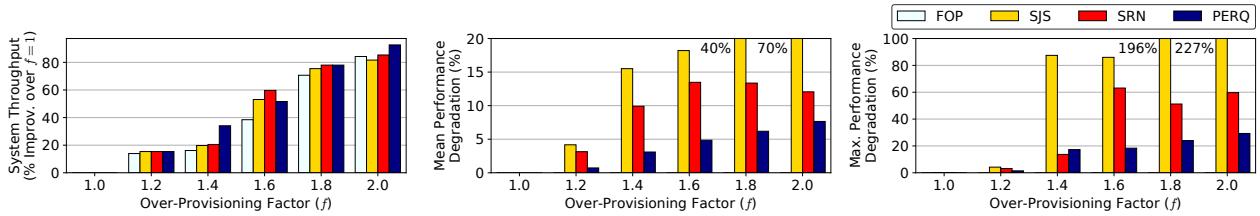

Figure 6: PERQ provides proportionally high system throughput, while being fair for Mira supercomputer parameter-driven evaluation.

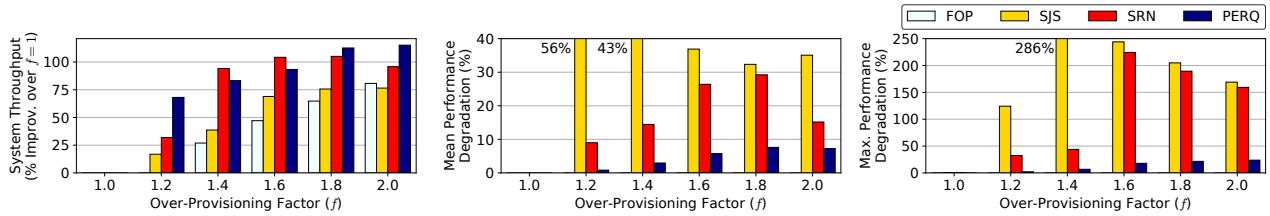

Figure 7: PERQ also provides proportionally high system throughput, while being fair for Trinity parameter-driven evaluation.

Power Provisioning Policies: We compare PERQ to several policies. For fairness, we compare it to the *fairness-oriented policy* (FOP), which allocates equal power to all the nodes on the over-provisioned system. For system throughput, we compare it to several throughput-oriented policies. The first policy prioritizes jobs which are running on the fewest number of nodes: *smallest job size* (SJS). This policy allocates more power to small jobs, anticipating that accelerating them would improve system throughput. Conversely, one might also imagine prioritizing large jobs (LJS) and helping them finish faster, in order to free up more nodes for other jobs, thus, improving system throughput. However, based on our experiments such a policy actually degrades system throughput. This is because it takes a lot of power to accelerate large jobs, and diverting power to such jobs adversely impacts other concurrently running jobs, slowing them down considerably.

The second throughput-oriented policy prioritizes jobs which are closest to finishing and are running on the fewest number of nodes: *smallest remaining node-hours* (SRN). This policy diverts power to shortest and smallest jobs, knowing that finishing them would improve throughput. It uses *future* knowledge of when the job is going to finish, which is not known for HPC jobs apriori – users typically overestimate runtime and runtime prediction is prone to inaccuracy. However, we compare PERQ to this policy in order to demonstrate that PERQ provides comparable throughput improvement to a policy which may have prior knowledge and solely focuses on throughput.

Objective Metrics: To quantify the system's performance, we use the system's *job throughput*, which is the number of jobs which complete execution during the duration of the experiment. To assess how fairly the jobs are treated, we use their *mean performance degradation*: the mean runtime degradation of *only* the jobs which experience degradation in their runtime with PERQ compared to their runtime with FOP. Note that this metric takes into account only the jobs which experience degradation because considering jobs that benefit from unfairness will skew our assessment of fairness. Jobs which experience better or equal performance as compared to FOP are not considered as they are deemed to be treated fairly. We use the *maximum performance degradation* metric to quantify the worst case for a given policy. By definition, FOP enjoys a mean and maximum job performance degradation of 0%.

PERQ improves system throughput significantly, while remaining fair to jobs. Fig. 6 and Fig. 7 show the improvement in system throughput of different power provisioning policies, over worst-case provisioning ($f = 1$) at different over-provisioning factors, for Mira and Trinity supercomputer parameter driven simulations, respectively. First, we observe that FOP and other ad-hoc policies such as SJS are not able to achieve proportional increase in system throughput at any over-provisioning factor, i.e., the improvement in system throughput is always less than the over-provisioning factor. We obtained similar results for LJS (prioritizing power to largest job size) policy. Second, we observe that a throughput oriented policy such as SRN which accounts for both job size and remaining time can improve throughput, but not proportionally. However, *PERQ is able to improve system throughput significantly and proportionally at all over-provisioning factors (f) and for both Mira and Trinity settings which represent different job characteristics. Notably, PERQ is able to beat the competing policy, SRN, which specifically focuses on improving throughput and has "future" knowledge (job completion time); PERQ does not have access to such knowledge and still provides throughput gains.* Note that beyond $f = 2.0$, the improvement in system throughput saturates because the system power budget becomes the bottleneck.

Next, we discuss the fairness performance of PERQ. Fig. 6 and Fig. 7 also show the mean and maximum performance degradation with different power-provisioning policies at different over-provisioning factors. The throughput-oriented policies have significantly high mean and maximum performance degradation, with SJS performing particularly worse. In fact, SRN, which provides high improvement in throughput, is 2-3x worse in terms of mean and maximum performance degradation compared to PERQ. For example, for Trinity, the maximum performance degradation of SRN is over 150% at $f = 2.0$ while PERQ incurs less than 30% maximum performance degradation. Lastly, we note that PERQ achieves up to 50% better system throughput than FOP, while maintaining a mean performance degradation of less than 8% for both Mira and Trinity. This shows that PERQ achieves a large improvement in system

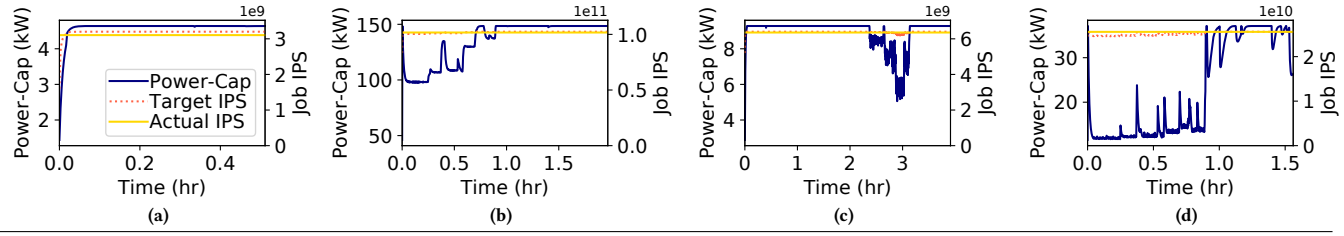

Figure 8: PERQ tracks job-level performance targets effectively for varied jobs, and provides convergent and stable power allocation.

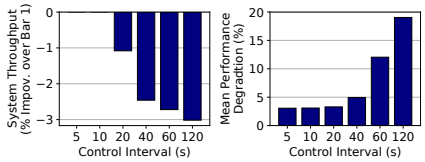

Figure 9: PERQ's throughput is not very sensitive to larger control intervals.

throughput at a lower over-provisioning factor, while remaining fair compared to ad-hoc and future-knowledge policies.

Interestingly, our results reveal that PERQ is able to achieve similar throughput as FOP with much lower over-provisioning cost. For instance, for Trinity, FOP achieves 80% improvement in system throughput over worst case over-provisioning at $f = 2.0$. However, PERQ achieves the same improvement in throughput at a much lower over-provisioning factor of $f = 1.4$ – thereby, requiring 30% less number of nodes. According to recent data presented by Cray Inc., 47% of the total cost-of-ownership (TCO) of an HPC facility is used as capital expense for purchasing the equipment, while 7% is used as operational expense for the equipment [38]. Thus, 30% fewer nodes can help reduce the overall cost of equipment by 30%, and the TCO by 16.2%. If these numbers are converted in monetary savings, it would result in more than a million dollars saving over the typical life span of a supercomputer [35] while improving system throughput and maintaining high level of fairness.

PERQ is convergent, responsive and stable. Fig. 8 shows how a job's power-cap level and performance under a given target (determined based on fairness) evolve over its execution time for four example jobs on Trinity. These examples jobs are selected to cover a wide range in job and PERQ characteristics.

Recall that convergence is the ability of the controller to actually meet the target performance, responsiveness is its ability to reach the target quickly, and stability is the ability to provide constant performance once the target is reached.

Fig. 8(a) shows that PERQ converges to the job-level target quickly (within a few minutes, less than a minute in many cases). In fact, PERQ achieves slightly better performance than the target because it is also trying to meet the system target which requires higher performance from some jobs. We also observe that the job performance is stable once the controller meets the target.

Fig. 8(b) shows a case where the job is not allocated enough power initially to meet its job-level target, but its performance remains close to the target. This is because the controller deems that this power can better serve to meet the system's or other jobs' targets. We found that this job has low sensitivity toward power-capping and this allows PERQ to borrow approx. 50kW of power to distribute it to other jobs, without significantly affecting the performance of this job. We note that eventually, the job is given enough power to meet its target, due to the completion of other jobs and arrival of jobs with different characteristics.

Fig. 8(c) shows a case where a job is given enough power to meet its goal for the most part. However, there are disruptions in between, which can occur when there is a burst of jobs starting or jobs finishing. During such periods, the controller responsively tries to figure out the power-caps for the new combination of jobs. In the case of this job, the power-cap is lowered during disruptive phase, but the performance does not get affected considerably.

Lastly, Fig. 8(d) shows a job which is executed during a time period when there is a lot of job start and finish activity, and the job's power-cap changes frequently during its execution, especially because the controller has observed that this job has low-sensitivity toward power-capping. During such high-activity time periods, the controller readjusts the power-cap gradually and the job's performance converges and stabilizes to a new value.

PERQ performs effectively across different control parameters and is not sensitive to selection of parameters values. Our results show that PERQ continues to provide good performance (higher system throughput while meeting fairness goals) as we vary different control parameters (length of decision interval and prediction horizon, system throughput improvement ratio, weights on system throughput and ΔP). In fact, our results support that PERQ does not need to highly tune or carefully select these parameters to achieve high performance.

Fig. 9 shows the effect of the control interval on the performance of PERQ for Mira supercomputer job trace. Recall that the control interval is the time between two decision instances when the job performances are examined and the nodes are assigned new power-caps. We notice that system throughput degrades minimally even at high control intervals (less than 3%), and mean performance degradation for unfairly treated jobs goes above 5% only at control intervals higher than 40 seconds. We observe similar results for the MPC prediction horizon, which is the number of future control intervals which the controller optimizes its decisions over. So a longer MPC prediction horizon is expected to provide faster convergence to the target, but may incur larger computation overhead. We found that that with our chosen control interval length of 10 seconds, PERQ was not sensitive to the length of MPC prediction horizon (i.e., number of control intervals). Note that we avoid control intervals less than 5 seconds to avoid aggressive power-cap switching, even though they are equally effective.

Recall that the system throughput improvement ratio determines how high the system throughput target should be set compared to the over-provisioning factor. The larger this ratio, the higher is the

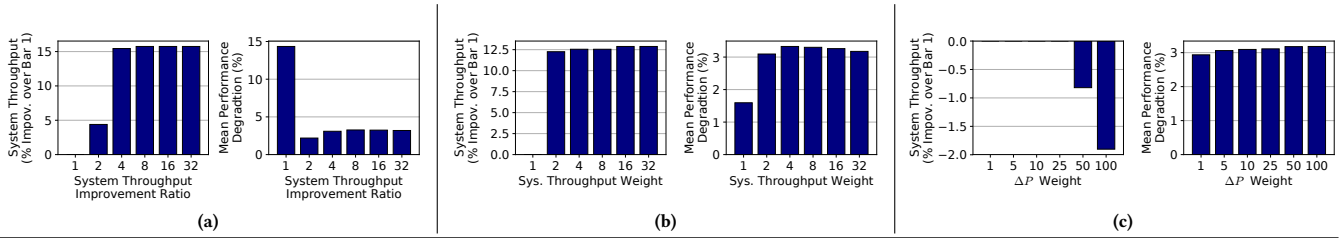

Figure 10: PERQ's effectiveness is not sensitive to control parameters such as (a) system throughput improvement ratio, (b) system throughput weight, and (c) ΔP weight on system throughput and job fairness.

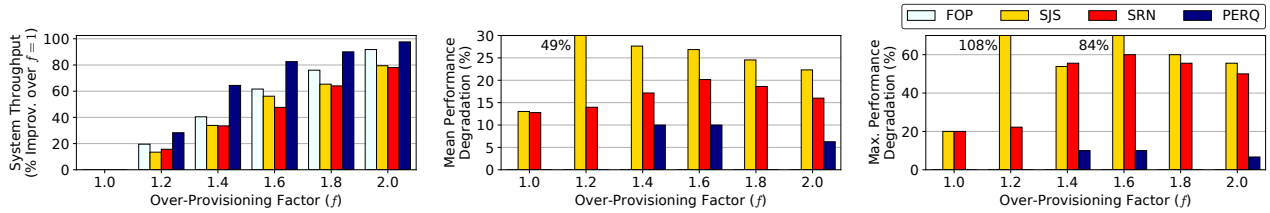

Figure 11: PERQ real-system experimental evaluation also confirm that PERQ is more effective than competing policies both in terms of system throughput and fairness.

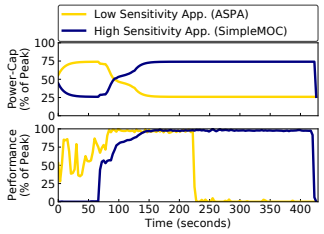

Figure 12: PERQ prototype automatically detects applications with different sensitivity toward power-capping and intelligently changes power allocations to meet job-level targets. The second job starts around 50 seconds mark and the first job ends at around 225 seconds mark. Note that the power-cap setting has a minimum limit too (as an idle node still consumes power) and hence, can not be set to zero even after an application finishes its execution.

target for system throughput. If the value of ratio is equal to one, it indicates that the target for system throughput is proportional to increase in the number of over-provisioned nodes in the system. This is conservative and hence, the system throughput improvement ratio is set much higher and our results show that PERQ is not sensitive to it once the value chosen is 4 or higher (Fig. 10(a)). Finally, our results also demonstrate that PERQ is not sensitive to selecting the weight placed on meeting the system throughput target or ΔP (the amount of change in the power-cap of a node from one control interval to another) (Fig. 10(b) and (c)). Recall that system throughput weight is different than the system throughput improvement ratio. The system throughput improvement ratio determines how high the throughput target is set, while the system throughput weight determines how hard the controller should try to meet this target. On the other hand, ΔP weight determines how much cost is associated with large ΔP, i.e., how must cost is associated with large switching of the power-caps from one control instance to next. Fig. 10(b) and (c) show that system throughput improvement and fairness metric is not affected significantly even across a wide range of parameter values. Note that PERQ can also be used to solely focus on improving the system throughput by placing orders of magnitude higher weight on throughput than fairness. We found that following this strategy results in up to 5% system throughput improvement over PERQ, but increases the maximum performance degradation close to 70%. Finally, we present PERQ prototype results and analysis.

The PERQ prototype implementation results show that PERQ is an effective strategy in meeting both throughput and fairness targets, and the observed improvements align with simulation results. Fig. 11 shows the system throughput and mean performance degradation of PERQ for different over-provisioning factors. Similar to Mira and Trinity simulation results, PERQ prototype results show that it can deliver up to 25% better throughput than FOP, while maintaining a mean performance degradation of less than 10%. Other ad-hoc throughput-oriented policies, SJS and SRN, are neither able to improve throughput as much as PERQ, nor are they able to remain as fair as PERQ because of their lack of feedback-based principled approach. In fact, SRN has a mean performance degradation of up 20% (double of PERQ), and maximum performance degradation of up to 60%, while not providing the necessary system throughput improvement.

Lastly, we use the prototype to provide insights into the dynamic behavior of PERQ. Fig. 12 shows how the provisioned power trades hands between two different types of applications running on the Tardis HPC cluster. Initially, a low sensitivity application starts running and is allocated as much power as possible, while the empty node is allocated minimum power. At around the 50 seconds time mark, a high sensitivity application starts running on the empty node. The controller determines that it is a high sensitivity application and gradually transfers power to that application. This does not degrade the performance of the low sensitivity application, while it improves the performance of the high-sensitivity application. Eventually, the controller completely switches the power allocations of the two applications (at about 150 seconds time mark). The low-sensitivity application continues to perform at close to 100% of its peak performance even at minimum power allocation,

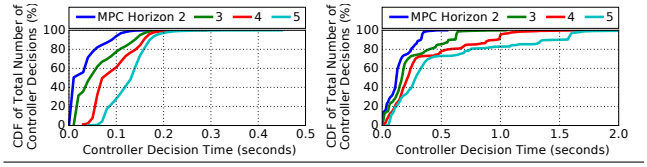

Figure 13: PERQ controller makes its power-capping decision within 0.5 seconds for more than 80% cases: Mira supercomputer (left) and Trinity supercomputer (right).

until it finishes at 55% time mark. This demonstrates how PERQ dynamically detects application characteristics and allocates power in a manner which benefits the system and the jobs.

Overhead Analysis: Finally, we discuss the scalability and overhead aspects of PERQ. PERQ has two sources of overheads: (1) MPC controller decision making overheads, and (2) communication of performance indicators (e.g., IPS) from each node to the controller. As we discuss next, our results provide quantitative evidence to show that these overheads are relatively small compared to the control interval. However, more importantly, we first note that these overheads are not on the critical path of application execution. These overheads only delay the optimal power-capping targets for different jobs and hence, these overheads do not pause or interrupt the application execution or computing cluster operations. In other words, these overheads simply make the effective length of the control decision intervals slightly longer. Our simulation and experimental evaluation includes these overheads and shows that PERQ is still effective at achieving its goals. Fig. 13 shows that MPC controller makes most of its decisions within 0.5 seconds for both the simulated systems Mira and Trinity (with 1052 and 1024 jobs, respectively) for MPC prediction window of length four. As the prediction window size increases, the overhead increases but the overhead still remains fairly low. We also note this overhead is not dependent on the scale of the system, but instead depends on the number of concurrently running jobs and their characteristics. For Trinity, only in less than 2% cases, the MPC controller decision making takes more than one second. Increasing the number of concurrently running jobs in the order of 10,000 can prohibitively increase the MPC controller decision making time in some cases due to computationally intensive nature of the calculations. However, several well-known strategies can be applied to alleviate this bottleneck: offloading the computation to GPUs, hierarchical decision making, eliminating the need to perform calculation for every job at every decision instance, creating groups of jobs with similar characteristics. Future efforts can exploit such opportunities to optimize MPC decision controller performance even further.

The second source of overhead (communication of performance metric from the nodes to the controller) was measured to be even lower. We stress tested our system by spawning 100,000 clients in our Tardis cluster and found that communicating such a information from these nodes to controller incurs only 0.19 seconds of delay. Note that we did not perform any optimizations to reduce this delay (e.g., using a dedicated network, performing reductions by filtering redundant information, etc.) that HPC clusters typically perform to monitor system health at much finer granularity and sending it to management workstations. In summary, MPC decision making is the primary source of delay in optimal power-capping control. But, even MPC decision time is relatively small and helps PERQ achieve effective results in terms of fairness and throughput (these delays and corresponding effects are modeled in our simulation results).

4 Related Work

This section provides an overview of prior solutions proposed for efficiently power provisioning on HPC systems. Prior studies have examined the performance of jobs under different power management techniques [24, 25, 61]. However, they do not provide solutions to provide performance guarantees under target power budgets.

Many studies have looked into maximizing resource utilization (system job throughput) while satisfying power constraints [22, 46, 55, 56]. These works are mostly concerned with dynamic application scheduling to maximize system utilization. Thus, they are compatible with PERQ and can be combined with it. Similarly, job scheduling and placement techniques [36, 57] to maximize performance under fixed energy budgets are also complementary to PERQ, as PERQ begins to function after the jobs have been scheduled, and is not concerned with scheduling.

Many studies have proposed various uses of dynamic voltage and frequency scaling (DVFS) and power capping to meet power constraints [7, 11, 33, 52]. However, these solutions (1) do not have a system-level consideration of power budget and therefore, do not take into the account the trade-offs of allocating different amounts of power to different jobs, and (2) they do not provide any feedback-based dynamic functionalities. Some works have also proposed heuristics-based approaches to meet multiple goals [5, 10], while others have proposed monitoring-based solutions to manage the application performance [23, 37]. However, these works have similar issues as mentioned above related to lack of a global system view to manage multiple jobs and lack of consideration for fairness among the jobs. Several studies have also proposed the use of control theory to manage the power efficiency of applications, while delivering performance guarantees [13, 17, 21, 37, 41, 50, 61, 62]. These studies highlight the effectiveness of control theory in achieving software and hardware targets in HPC applications. However, these studies lack a universal system-level view and continuous estimation of fairness; therefore, are unable to distribute power amongst applications based on what the system can deliver.

5 Conclusion

This paper described design and evaluation of PERQ, which achieves high system throughput while maintaining job-level fairness. PERQ employs multi-input multi-output control theory which provides theoretic guarantees about satisfying conflicting goals. Extensive real-system and simulation evaluation demonstrate that PERQ improves system throughput by up to 50% points, compared to the fairness-oriented allocation policy, while remaining fair to jobs.

Acknowledgment: This research is supported by the Northeastern University, Amazon AWS research program, and Massachusetts Green High Performance Computing Center (MGHPCC). We are thankful to anonymous reviewers for their feedback. We are especially thankful to Liana Fong for her detailed feedback and efforts which helped us improve the quality of the paper significantly.

References

[1] 2018. Exascale Computing Project Proxy Applications: https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/.
[2] George Amvrosiadis, Jun Woo Park, et al. 2018. On the Diversity of Cluster Workloads and its Impact on Research Results. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 533–546.
[3] M Andersen, Joachim Dahl, et al. 2013. CVXOPT: A Python Package for Convex Optimization. *abel. ee. ucla. edu/cvxopt* (2013).

[4] Reza Azimi, Masoud Badiei, Xin Zhan, Na Li, and Sherief Reda. 2017. Fast Decentralized Power Capping for Server Clusters.. In *HPCA*. 181–192.

[5] Prasanna Balaprakash, Ananta Tiwari, et al. 2013. Multi Objective Optimization of HPC Kernels for Performance, Power, and Energy. In *International PMBS Workshop*. Springer, 239–260.

[6] Alberto Bemporad, Francesco Borrelli, et al. 2000. Optimal Controllers for Hybrid Systems: Stability and Piecewise Linear Explicit Form. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, Vol. 2. IEEE, 1810–1815.

[7] Sridutt Bhalachandra, Allan Porterfield, et al. 2017. Improving Energy Efficiency in Memory-constrained Applications Using Core-specific Power Control. In *Proceedings of the Workshop on Energy Efficient Supercomputing*. ACM, 6.

[8] Abhishek Chandra, Pawan Goyal, and Prashant Shenoy. 2003. Quantifying the benefits of resource multiplexing in on-demand data centers. *Computer Science Department Faculty Publication Series* (2003), 20.

[9] Hong Chen et al. 1998. A Quasi-Infinite Horizon Nonlinear Model Predictive Control Scheme with Guaranteed Stability. *Automatica* 34, 10 (1998), 1205–1217.

[10] Jee Choi, Marat Dukhan, et al. 2014. Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 447–457.

[11] Ryan Cochran, Can Hankendi, et al. 2011. Pack & Cap: Adaptive DVFS and Thread Packing under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 175–185.

[12] Howard David, Eugene Gorbatov, et al. 2010. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, 189–194.

[13] Gökalp Demirci, Ivana Marincic, and Henry Hoffmann. 2018. A divide and conquer algorithm for DAG scheduling under power constraints. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 466–477.

[14] Douglas W Doerfler. 2014. *Trinity: Next-Generation Supercomputer for the ASC Program*. Technical Report. SNL, Albuquerque, NM (United States).

[15] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH computer architecture news*, Vol. 35. ACM, 13–23.

[16] Dror G Feitelson et al. 2014. Experience with using the parallel workloads archive. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2967–2982.

[17] Antonio Filieri, Henry Hoffmann, et al. 2015. Automated Multi-Objective Control for Self-Adaptive Software Design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 13–24.

[18] Rong Ge, Xizhou Feng, et al. 2005. Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 34–34.

[19] Neha Gholkar, Frank Mueller, et al. 2016. Power Tuning HPC Jobs on Power-Constrained Systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. ACM, 179–191.

[20] Neha Gholkar, Frank Mueller, et al. 2018. PShifter: Feedback-Based Dynamic Power Shifting within HPC Jobs for Performance. In *2018 HPDC*.

[21] Henry Hoffmann. 2015. JouleGuard: Energy Guarantees for Approximate Applications. In *Symposium on Operating Systems Principles*. ACM, 198–214.

[22] Connor Imes et al. 2015. Minimizing Energy under Performance Constraints on Embedded Platforms: Resource Allocation Heuristics for Homogeneous and Single-ISA Heterogeneous Multi-Cores. *ACM SIGBED Review* 11, 4 (2015), 49–54.

[23] Connor Imes, Lars Bergstrom, et al. 2016. A Portable Interface for Runtime Energy Monitoring. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 968–974.

[24] Canturk Isci et al. 2006. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Symposium on Microarchitecture*. IEEE, 347–358.

[25] Melanie Kambadur and Martha A Kim. 2014. An Experimental Survey of Energy Management Across the Stack. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 329–344.

[26] Jaimie Kelley et al. 2016. Adaptive power profiling for many-core HPC architectures. In *ICAC 2016*. IEEE, 179–188.

[27] Kalyan Kumaran. 2016. Introduction to Mira. In *Code for Q Workshop*.

[28] Kien Le, Ricardo Bianchini, Thu D Nguyen, Ozlem Bilgir, and Margaret Martonosi. 2010. Capping the brown energy consumption of internet services at low cost. In *International Conference on Green Computing*. IEEE, 3–14.

[29] Jay H Lee. 2009. A Lecture on Model Predictive Control. *Pan American Advanced Studies Institute Program on Process Systems Engineering* (2009).

[30] Bo Li, Hung-Ching Chang, et al. 2014. The Power-Performance Tradeoffs of the Intel Xeon Phi on HPC Applications. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, 1448–1456.

[31] Yang Li et al. 2019. A Scalable Priority-Aware Approach to Managing Data Center Server Power. In *HPCA 2019*. IEEE, 701–714.

[32] Harold Lim, Aman Kansal, and Jie Liu. 2011. Power budgeting for virtualized data centers. In *2011 USENIX Annual Technical Conference (ATC)*.

[33] Yanpei Liu, Guilherme Cox, et al. 2016. FastCap: An Efficient and Fair Algorithm for Power Capping in Many-Core Systems. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 57–68.

[34] L Ljung. 1999. System Identification-Theory for the User 2nd edition PTR Prentice-Hall. *Upper Saddle River, NJ* (1999).

[35] Thomas Ludwig and Manuel Dolz. 2014. Total Cost of Ownership in High Performance Computing. In *Talk at the University of Hamburg*.

[36] Olli Mämmelä et al. 2012. Energy-Aware Job Scheduler for High-Performance Computing. *Computer Science-Research and Development* 27, 4 (2012), 265–275.

[37] Ivana Marincic et al., Venkatram Vishwanath, and Henry Hoffmann. 2017. PoLiMEr: An Energy Monitoring and Power Limiting Interface for HPC Applications. In *Proceedings of the 5th E2SC Workshop*. ACM, 7.

[38] Steven Martin. 2017. Total Cost of Ownership and HPC System Procurement. In *Talk at the 2017 International Conference in Supercomputing*.

[39] David Q Mayne, James B Rawlings, et al. 2000. Constrained Model Predictive Control: Stability and Optimality. *Automatica* 36, 6 (2000), 789–814.

[40] David Meisner, Christopher M Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F Wenisch. 2011. Power management of online data-intensive services. In *ACM SIGARCH Computer Architecture News*, Vol. 39. ACM, 319–330.

[41] Nikita Mishra, Connor Imes, et al. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 184–198.

[42] Frank Mueller, Barry Rountree, et al. 2016. *Power Tuning for HPC Jobs under Manufacturing Variations*. Technical Report. North Carolina State University. Dept. of Computer Science.

[43] Bin Nie et al. 2017. Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities. In *MASCOTS 2017*.

[44] Chandrakant D Patel and Amip J Shah. 2005. Cost model for planning, development and operation of a data center.

[45] Tapasya Patki et al. 2013. Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing. In *Proceedings of the International Conference on Supercomputing*. ACM, 173–182.

[46] Tapasya Patki et al. 2015. Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the Symposium on High-Performance Parallel and Distributed Computing*. ACM, 121–132.

[47] Tapasya Patki et al. 2016. Economic Viability of Hardware Overprovisioning in Power-Constrained High Performance Computing. In *Proceedings of the Workshop on Energy Efficient Supercomputing*. IEEE Press, 8–15.

[48] Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F Wenisch, and Jack Underwood. 2010. Power routing: dynamic power provisioning in the data center. In *ACM Sigplan Notices*, Vol. 45. ACM, 231–242.

[49] Ian R Petersen and Roberto Tempo. 2014. Robust Control of Uncertain Systems: Classical Results and Recent Developments. *Automatica* 50, 5 (2014), 1315–1335.

[50] Raghavendra Pradyumna Pothukuchi, Amin Ansari, et al. 2016. Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 658–670.

[51] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No power struggles: Coordinated multi-level power management for the data center. *ACM SIGOPS Operating Systems Review* 42, 2 (2008), 48–59.

[52] Barry Rountree, Dong H Ahn, et al. 2012. Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. IEEE, 947–953.

[53] Ryuichi Sakamoto, Thang Cao, et al. 2017. Production Hardware Overprovisioning: Real-World Performance Optimization using an Extensible Power-Aware Resource Management Framework. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 957–966.

[54] Ryuichi Sakamoto, Tapasya Patki, et al. 2018. Analyzing Resource Trade-offs in Hardware Overprovisioned Supercomputers. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 526–535.

[55] Osman Sarood, Akhil Langer, et al. 2014. Maximizing Throughput of Overprovisioned HPC Data Centers under a Strict Power Budget. In *Supercomputing (SC)*. IEEE Press, 807–818.

[56] Kun Tang et al. 2016. Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy. In *DSN 2016*. IEEE, 311–322.

[57] Akshat Verma, Puneet Ahuja, et al. 2008. Power-Aware Dynamic Placement of HPC Applications. In *Proceedings of the 22nd Annual International Conference on Supercomputing*. ACM, 175–184.

[58] Xiaorui Wang, Ming Chen, Charles Lefurgy, and Tom W Keller. 2012. Ship: A scalable hierarchical power control architecture for large-scale data centers. *IEEE Transactions on Parallel and Distributed Systems* 23, 1 (2012), 168–176.

[59] Will Whiteside, Shelby Funk, Aniruddha Marathe, and Barry Rountree. 2017. PANN: Power Allocation via Neural Networks Dynamic Bounded-Power Allocation in High Performance Computing. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*. ACM, 8.

[60] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. 2016. Dynamo: facebook's data center-wide power management system. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 469–480.

[61] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *ACM SIGARCH Computer Architecture News* 44, 2 (2016), 545–559.

[62] Huazhe Zhang and Henry Hoffmann. 2018. Performance & energy tradeoffs for dependent distributed applications under system-wide power caps. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 67.