Market Mechanism-Based User-in-the-Loop Scalable Power Oversubscription for HPC Systems

Md Rajib Hossen
The University of Texas at Arlington
mdrajib.hossen@mavs.uta.edu

Kishwar Ahmed
The University of Toledo
kishwar.ahmed@utoledo.edu

Mohammad A. Islam
The University of Texas at Arlington
mislam@uta.edu

***Abstract*—Significant power consumption is one of the major challenges for current and future high-performance computing (HPC) systems. All the while, HPC systems generally remain power underutilized, making them a great candidate for applying power oversubscription to reclaim unused capacity. However, an oversubscribed HPC system may occasionally get overloaded. In this paper, we propose MPR (Market-based Power Reduction), a scalable market-based approach where users actively participate in reducing the HPC system's power consumption to mitigate overloads. In MPR, HPC users bid to supply, in exchange for incentives, the resource reduction required for handling the overloads. Using several real-world trace-based simulations, we extensively evaluate MPR and show that, by participating in MPR, users always receive more rewards than the cost of performance loss. At the same time, the HPC manager enjoys orders of magnitude more resource gain than her incentive payoff to the users. We also demonstrate the real-world effectiveness of MPR on a prototype system.**

I. Introduction

Motivation. Advances in high-performance computing (HPC) systems have enabled scientists to perform large-scale computations quickly and efficiently. However, with the increasing computational requirement, HPC power consumption has also increased tremendously. The top supercomputers currently consume power in the megawatts range [39], [53]. Massive power consumption remains a central challenge as we move towards exascale and zettascale computing [37], [33].

Addressing the significant power consumption of HPC systems requires the adoption of energy-efficient techniques. *Power oversubscription* is a useful scheme to increase utilization by fitting the HPC system with more computing resources than its capacity. Power oversubscription has been widely adopted in hyperscale data centers of the likes of Google, Facebook, and Microsoft [36], [49], [27], [20], [32], [55], which oversubscribes by as much as 20% [36]. Meanwhile, HPC systems are rife with oversubscription opportunities as these typically suffer from even greater underutilization. Approximately 30% of the power in mid-scale HPC systems remain underutilized [42], while in large-scale HPC systems, about $15 - 40\%$ of the power is never utilized [45]. This underutilization, however, is not due to a lack of demand but due to the HPC's highly specialized usage. To that end, *power oversubscription can reclaim unutilized power capacity and allow HPC expansion without additional infrastructure investment.*

Limitations of existing approaches. Power oversubscription comes with an unwanted pitfall of introducing the possibility of system overload (i.e., power consumption exceeding capacity). Several recent studies propose "power-aware job scheduling" where the HPC job scheduler allocates resources to keep the power consumption within the power budget while also targeting various efficiency improvements, such as increasing the overall system utilization, increasing throughput, and reducing job runtime [45], [57], [41], [50], [51], [52], [29]. However, optimizing such job scheduling with a peak power budget is a combinatorial bin-packing problem that is very hard to solve efficiently [7]. Moreover, the resulting power consumption from resource allocation varies depending on the job's characteristics. Not to mention, HPC jobs also go through different phases that consume different amounts of power [41], [51], [50]. Hence, power-aware scheduling faces the daunting task of estimating the power consumption over the period of each job's execution while also tracking phases of these jobs' progressions. Furthermore, HPC managers trying to maximize the system's performance (e.g., throughput) also need to consider different jobs' varying resource efficiency (e.g., work done per unit resource) during job scheduling. Hence, while existing approaches can proactively avoid overloading an oversubscribed HPC system, they also add a significant burden on job scheduling. More importantly, prior works on HPC oversubscription do not incorporate the HPC users whose job performance is adversely affected (because of power constraints) by oversubscription.

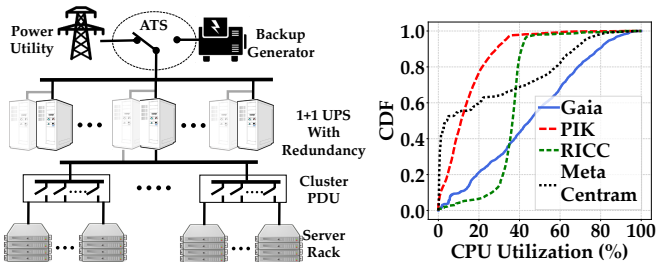
Our contribution. In stark contrast to proactive approaches, we propose a "reactive" approach for managing oversubscribed HPC systems. In our approach, the HPC manager allows the system's power to go beyond the power capacity, causing infrastructure overload. And when such an overload occurs, the HPC manager reduces the system's power consumption to mitigate it. The rationale for this reactive approach is that *first,* the HPC manager can quickly and reliably cutback the power utilizing existing power management techniques such as dynamic voltage frequency scaling (DVFS) and hardware power capping (e.g., Intel's Running Average Power Limit (RAPL) [16]). Such power capping techniques are also available for modern heterogeneous computing architectures with accelerators, such as the `nvidia-smi` tool for Nvidia GPUs [43]. *Second,* we allow overloads by a relatively small margin as the maximum overload depends on the level of oversubscription (e.g., 20%). With such level of overloads, protective circuit breakers operate in the "long-delay" zone and take several tens of minutes before tripping [14], [47], [19].

Meanwhile, HPC data center cooling can also withstand these short-lived (HPC manager reacts to mitigate the overload) overloads due to thermal inertia [40]. In fact, reactively handling power overload is the norm in the cloud data centers [20], [56]. Therefore, we consider that *reactively handling power overloads is a safe approach for managing oversubscribed HPC systems.*

While reactively mitigating power overloads in HPC systems is a viable approach, the HPC manager still needs to decide how to best exercise the power reduction. The power reduction essentially translates into resource reduction (e.g., reduced CPU speed) for the active jobs executing in the system. Hence, overload handling adversely affects the active jobs' performance, and the HPC manager needs to judiciously apply power capping for a graceful power reduction with the minimum performance impact. This, however, brings us back to the challenges of job characteristics profiling of power-aware scheduling and requires the HPC manager to know the impact of resource reduction for every active job.

To avoid this burden on the HPC manager, we propose a market-based approach where the HPC users themselves determine the performance impact and actively participate in making the resource reduction decision during an overload. More specifically, we propose MPR (Market-based Power Reduction), where the users supply, in exchange for incentives/rewards (e.g., free HPC core-hours) from the HPC manager, the necessary resource reduction for handling the overloads. The users use a parameterized supply function to express how much resource they are willing to reduce at what price (i.e., incentive per unit reduction). The HPC manager acts as the market facilitator, and the market outcome determines which job will reduce how much resources and what would be the incentive for the resource reduction. We develop a static market and an interactive market for MPR. The static market offers rapid market decision, while the interactive market guarantees socially optimum power reduction with minimum performance degradation. We also develop strategies that the user can adopt to participate in these markets.

Merits of our approach. (❶) Our reactive approach frees the HPC scheduler from requiring job-wise power estimation and execution tracking. Instead, the HPC manager tracks the system's instantaneous total power consumption to detect overload and uses MPR to reduce power consumption. (❷) MPR brings the user in the loop in managing an oversubscribed HPC system. In MPR, users can integrate their own "perceived value" of performance (i.e., the same amount of performance loss can be valued differently by different users) in the resource reduction process - a user who values their performance more can ask for a greater incentive for resource reduction and vice versa. Such integration of user preference is not available in any existing work on managing an oversubscribed HPC system. (❸) MPR also offers a highly scalable HPC management solution as the HPC manager no longer needs to solve complicated scheduling problems with many variables (e.g., each job's resource allocation). Instead, she only decides the market-clearing price that ensures the active jobs supply the target amount of resource/power reduction. (❹) Finally, by empowering users to influence the HPC system's power consumption through the market mechanism, we believe MPR's user-in-the-loop approach can go beyond handling power oversubscription. For instance, users can also assist in socially responsible HPC management, such as cutting carbon emissions by doing less work with "dirty" power with low/no renewable [61] and participating in demand response to improve the grid's stability [2].



(a) HPC power architecture (b) CDF of HPC utilization

Fig. 1. (a) HPC power architecture. Here, ATS = Automatic Transfer Switch, UPS = Uninterrupted Power Supply, PDU = Power Distribution Unit. (b) CDF of four real-world HPC cluster workloads [13].

Evaluation of MPR. We extensively evaluate MPR using several real-world trace-based simulations using the performance profiles of fourteen different HPC applications. We demonstrate that MPR can effectively handle power overloads while capturing the users' willingness for resource reduction. We show that, by participating in MPR, a user always gets more incentive than its cost of performance loss, while the HPC manager enjoys orders of magnitude more resource gain than her incentive payoff to the users. Finally, to demonstrate MPR's effectiveness in real life, we run experiments on a prototype HPC system and show that MPR can effectively mitigate overloads due to oversubscription.

II. Background

HPC power system. As illustrated in Fig. 1(a), HPC data centers typically use a hierarchical power infrastructure [20], [32]. The utility power is delivered to the data center through an automatic transfer switch (ATS) that switches its power source to the backup generator if the utility power fails. The ATS feeds the UPS (uninterrupted power supply) which is responsible for supplying power while the generator warms up to takeover followed by a utility failure. The UPS typically needs to supply power for two to three minutes. There could be multiple UPSs working in parallel or active/redundant modes. For large HPC systems (e.g., 10-MW systems), the power infrastructure can be divided into multiple pieces, each with dedicated UPSs. The UPS powers the cluster PDUs (power distribution units) which supply power to the server racks.

Power oversubscription. In our context, oversubscription is permanently adding more servers than the HPC power infrastructure's capacity allows. Each layer of the HPC power infrastructure, from the server rack to the ATS/UPS, is subject to capacity limits and can be oversubscribed. However, we

TABLE I
CAPACITY OVERSUBSCRIPTION IN GAIA [11].

Oversubscription	10%	15%	20%	25%
Extra Capacity (core-hours/month)	144K	216K	288K	360K
Probability of Overload	2.5%	5%	9%	14%
Overload Time (hours/month)	17.8	35.5	68.62	101.3
Overloaded Capacity (core-hour/month)	1.25K	3.9K	8.9K	17.5K
Estimated Maximum Overload Payoff	115×	55×	32×	20×

focus on UPS-level oversubscription while considering the cluster PDUs and racks have adequate capacity. We choose this as UPS is typically the dominant contributor in a data center's per kilowatt capital cost for its power system [1], [15]. Oversubscribing an existing HPC data center would mean that we add additional server racks connected to an existing cluster PDU with increased capacity or a new cluster PDU connected to the existing UPS. Cluster PDUs typically have a modular design, and we can increase their capacity by adding more circuit breakers [4].

Opportunities and challenges of power oversubscription. Oversubscription in the HPC data center is enabled by its low average utilization [45]. Fig. 1(b) shows the CDFs of the utilization of four real-world HPC clusters where we see that ∼5% capacity of `Gaia` [11], ∼20% capacity of `Metacentrum` [25], ∼55% of `RICC` [28], and ∼65% of `PIK` [34] are rarely used. Even for the `Gaia` cluster with relatively high utilization, oversubscription can be beneficial.

Table I shows a quantitative analysis of the benefits of different levels of oversubscription on `Gaia` cluster considering the workload is scaled-up proportional to the extra capacity. Here, the unit of one core-hour indicates the availability of one HPC core for one hour. The extra capacity refers to the additional core-hours we can add to the 2004-core `Gaia` system. We have 144K extra core-hours every month at 10% oversubscription, going up to 360K core-hours at 25% oversubscription. The probability of overload tells us how often the total power consumption goes beyond the infrastructure capacity, and overload time gives us the total time the HPC system stays in an overloaded state each month. To understand the impact of these overload periods, we then calculate the total overloaded capacity, which indicates how many core-hours are spent over the HPC capacity. In other words, overloaded capacity tells us how many total core-hours we need to cut back every month to avoid these overloads. It also reveals the most intriguing observation from this analysis that *we add far more core-hours capacity each month (e.g., 144K added vs. 1.25K cut at 10% oversubscription) than we have to cut back to handle the overloads.* Finally, we show the maximum payoffs we can afford if we pay all the added core-hours as payment to users for their core-hour cutbacks during overloads. For instance, at 10% oversubscription, we can pay up to 115× of a user's core-hour reduction.

There are compelling benefits of oversubscription in HPC data centers as the HPC manager can add a significant additional capacity to the system. However, as shown in Table I, an oversubscribed HPC data center may occasionally get overloaded. While UPS circuit breakers can handle power overloads for tens of minutes, sustained overloaded operation will affect UPS's longevity [46], [65]. More importantly, however, the HPC data center's cooling system cannot withstand overloads as long as UPSs [66]. Consequently, even when adopting a reactive approach, an HPC manager's goal is to mitigate the overloads as soon as possible. In the next section, we formalize the problem of handling these power overloads into an optimization problem, identify the HPC manager's challenges, and propose our market-based solution.

III. HANDLING POWER OVERLOADS IN HPC

A. Problem Formulation

Let us consider that at any given time t, there are $M(t)$ jobs running in the HPC system resulting in a total power consumption of $P(t) = \sum_{m=1}^{M(t)} p_m(t, r_m)$, where $p_m(t, r_m)$ is the power consumption attributed to job m running with resource r_m. Note that, instead of traditional approaches of server-wise power modeling (e.g., [50]), here we do job-wise power modeling. The job-wise power model facilitates our market-based design where HPC users (who submit the jobs) play an integral role and are oblivious to how many servers are executing their jobs. Also, such a job-wise power model can be easily extracted by the HPC manager by attributing server power to jobs according to the job's resource share (i.e., number of cores) on that server. In addition, we are considering a unified aggregate power and capacity model for the HPC data center. However, as described in Section II, large HPC data centers can have multiple parallel power infrastructures, each connected to a dedicated UPS. Nonetheless, our model can be seamlessly extended to these data centers by considering individual infrastructure capacity C_i and aggregate power consumption $P_i(t)$ for the data center's i-th parallel power infrastructure.

With HPC data center power capacity of C, a power overload occurs when $P(t) > C$, and the HPC manager needs to intervene to handle this overload by reducing the power consumption by $P(t) - C$. The power reduction target, $P(t) - C$, needs to be met by reducing the power consumption of the running jobs. Reducing power consumption is accomplished through the reduction of resource allocation. For example, a CPU core slowed down to 90% of its regular frequency can be interpreted as allocation of "0.9 cores". Resource reduction to handle power overload leads to performance degradation of the affected jobs. Hence, an HPC manager's goal is to minimize the overall performance degradation while still achieving the target power reduction. We formalize this as the following optimization problem OPT (OPTimum power overload handling)

$$\mathsf{OPT}: \underset{\delta_m}{\text{minimize}} \sum_{m=1}^{M(t)} \mathcal{L}_m(\delta_m) \tag{1}$$

$$\text{subject to} \sum_{m=1}^{M(t)} \mathcal{P}(\delta_m) \geq P(t) - C, \tag{2}$$

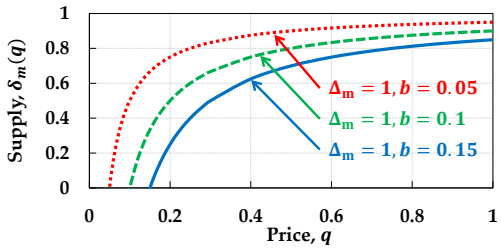


Fig. 2. MPR's supply function, $\delta_m(q) = \Delta_m - \frac{b_m}{q}$. For a job m, $\delta_m(q)$ is the supply of resource reduction at price q, Δ_m is the maximum resource reduction, and b_m is the bid.

where δ_m is the resource reduction for job m and $\mathcal{L}_m(\delta_m)$ and $\mathcal{P}(\delta_m)$ are the performance degradation and power reduction, respectively, due to resource reduction δ_m. Here, the optimization objective (1) is a scalar measure of overall performance impact due to the overload, and the constraint (2) specifies the power reduction requirement to mitigate the power overload.

Challenges. A major challenge for an HPC manager in solving OPT is to accurately determine the performance impact $\mathcal{L}_m(\delta_m)$ for each running job when an overload occurs. However, the user who submits the job can best estimate the potential impact of the resource reduction, both in terms of performance degradation and its perceived impact. On the other hand, determining power reduction for resource reduction, $\mathcal{P}(\delta_m)$, is straightforward with established models for any adopted power capping technique [41].

In this work, *we decouple determining the jobs' performance impact due to resource reduction from the HPC manager and engage the HPC users in the power reduction decision.* We enable users to express their affinity for contributing towards meeting the goal of power reduction during overloads.

B. MPR: Market-Based Power Reduction

We propose a supply function bidding-based market mechanism, MPR, where the HPC users participate in the power overload handling by agreeing to "supply", in exchange for incentives, the required power reduction through resource reduction of their jobs. At their own discretion, the users determine the level of participation in MPR.

Supply function. In our market design, HPC users use a predetermined form of supply function to indicate how much resource they can reduce at what level of incentive. For a job m, its user provides the parameters Δ_m and b_m to form the following parameterized supply function

$$\delta_m(q) = \left[\Delta_m - \frac{b_m}{q}\right]^+, \tag{3}$$

where Δ_m indicates the maximum resource reduction from job m, b_m is the bidding parameter that determines job m's affinity of resource reduction, and q is the incentive/reward per unit resource reduction (e.g., one core of CPU resource reduction for one hour). q can be interpreted as the "unit price" of the market's product which in our case is the resource reduction. $[\cdot]^+$ indicates that $\delta_m(b_m, r)$ is non-negative, meaning that in our market, no job is asked to increase its resource. A similar form of supply function has also been utilized in prior work on electricity markets [21], [58]. The supply function in Eqn. (3) indicates how much resource can be reduced for job m if the HPC manager offers an incentive of q for each unit of resource reduction. Fig. 2 illustrates MPR's supply function for different bids that results in different amounts of resource reduction for the same q.

Rationale for the choice of our supply function. While the form of our supply function in Eqn. (3) is widely used, there are other supply functions, for instance, a linear supply function [31], that can be used for the supply function bidding mechanism. However, our choice is motivated by the fact that Eqn. (3) captures the diminishing return on resource reduction, i.e., as we ask for more supply of resource reduction (δ_m), we need to pay more incentive per unit reduction (q). We see similar behavior in HPC applications (Fig. 7(b)), where as we increase the resource reduction, the performance degradation increases super-linearly. In addition, our supply function in Eqn. (3) is also backed by theoretical performance guarantees under reasonable assumptions [21], [6], [22].

Power reduction during overload. When an overload occurs, the HPC manager invokes the market to reduce the power consumption of the running jobs. Acting as the facilitator of the market, the HPC manager needs to set the market "clearing price" $q'(t)$, which is used as the basis to determine how much resource reduction each running job needs to supply (i.e., $\delta_m(q'(t))$ for job m) towards meeting the total power reduction goal. Setting the market clearing price $q'(t)$ can be formalized as the following optimization problem MClr (Market Clearing)

$$\text{MClr}: \underset{q}{\text{minimize}} \sum_{m=1}^{M} q \cdot \delta_m(q) \tag{4}$$

$$\text{subject to} \sum_{m=1}^{M} \mathcal{P}(\delta_m(q)) \geq P(t) - C. \tag{5}$$

MClr's objective in (4) is to minimize the cost of handling the overload by minimizing the total incentive payoff to the running jobs. The key distinction between OPT and MClr is that the *HPC manager in MClr no longer needs to determine the performance impact $\mathcal{L}_m(\delta_m)$ to set the resource reductions of the active jobs.*

Soliciting bids and exercising the market. As part of the implementation of MPR, we introduce two approaches towards how the bids (i.e., Δ_m and b_m) are collected from the users and how the market clearing price $q'(t)$ is set.

A static market: In the first approach, the bidding parameters Δ_m and b_m are supplied to the HPC manager during job submission. The HPC manager invokes a market instance when there is an overload and uses the already-received bids of all active jobs. The HPC manager sets the market clearing price by plugging the bids into MClr. Since the bids remain unchanged during the market execution, we call this approach MPR-STAT (MPR with static bidding).

An interactive market: In the second approach, the bidding parameters Δ_m and b_m for job m are iteratively updated by the users after the HPC manager invokes the market following an overload. First, the HPC manager declares an initial clearing price $q'_0(t)$. Upon receiving the clearing price, users with jobs running in the system send their bids. The HPC manager plugs the bids into MClr and determines the new clearing price. The HPC manager sends the updated clearing price to the users, who in turn send back their updated bids. This back-and-forth communication continues until the clearing price converges to a stable value. The convergence of clearing price (i.e., Nash equilibrium) is guaranteed if the users take the price set by the HPC manager in each iteration and behave rationally by maximizing their net market gain (Eqn. (7)) [6], [22], [21]. Since the clearing price is determined based on interactions between the HPC manager and the users, we call this approach MPR-INT (interactive MPR). We defer the qualitative comparison of these two approaches to Section III-D.

C. User Bidding in MPR

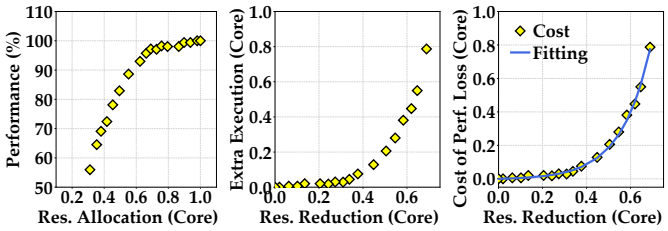
A key step for enabling MPR's performance-oblivious power reduction by the HPC manager is collecting bids from the users/jobs. Our market mechanism is designed to proxy the performance impact $\mathcal{L}_m(\delta_m)$ using the supply function in Eqn. (3). Hence, the bidding parameters need to be decided based on the performance impact from the job's resource (and hence power) reduction. Here, we describe how an HPC user devices its bids based on its performance impact.

Cost of performance loss. We define $\mathcal{C}_m(\delta_m)$ as the user-perceived cost of performance degradation from δ_m resource reduction. The notion of cost enables HPC users to integrate their own relative importance of different jobs in their bidding. While the HPC users can decide how they want to quantify the cost at their own discretion, in this work, we consider the additional work (i.e., increase in execution time or "extra execution") needed to finish the job as the cost of performance loss. Considering $\mathcal{L}(x)$ to be the job runtime with a core reduction of x, we can generalize the cost impact of resource reduction as

$$\mathcal{C}_m(\delta_m) = \alpha_m(\mathcal{L}_m(\delta_m) - \mathcal{L}_m(0)), \tag{6}$$

where $\alpha \geq 1$ is a coefficient that a user can tune to reflect its perceived cost of the additional execution. $\alpha = 1$ indicates that the HPC user does not add any surcharge on the actual performance impact. Alternative to this linear cost and popularly used in system research for performance cost is a "quadratic cost" function, i.e., $\mathcal{C}_m(\delta_m) = \alpha_m \cdot (\mathcal{L}_m(\delta_m) - \mathcal{L}_m(0))^2$, where the cost of performance grows quadratically with increasing performance loss.

As a concrete example, Fig. 3(a) shows the performance of `XSBench` application [54] with different levels of resource allocation. Here, a core allocation of "1" indicates the core is running at 100% speed. Next, in Fig. 3(b), we show the extra execution needed when the core allocation is reduced. In Fig. 3(c), we show the cost associated with different levels of resource reduction using Eqn. (6) and $\alpha = 1$.



(a) Performance (b) Extra execution (c) Performance cost

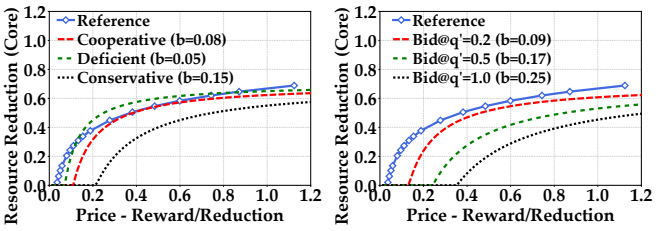
Fig. 3. (a) Performance at different levels of resource allocation. (b) Impact of resource reduction. $ExtraExecution = \frac{100 - Performance}{Performance}$. (c) Cost impact of resource reduction. $Cost = \alpha \cdot ExtraExecution$ with $\alpha = 1$.

Devising the reference cost for bidding. Our supply function is based on the unit price of supply, i.e., payment for per unit resource reduction, and hence to utilize the performance data for bidding we convert the *cost of resource reduction* into *cost per unit resource reduction* as $\mathcal{C}'_m(\delta_m) = \mathcal{C}_m(\delta_m)/\delta_m$. Using this equation, the reference lines in Figs. 4(a) and 4(b) are derived from the cost of performance shown in Fig. 3(c). For any amount of resource reduction (in the y-axis), from the reference lines in Fig. 4, we can find a user's actual cost of per unit resource reduction (in the x-axis). In our context of bidding for supply, we can interpret this cost reference curve as the upper limit on resource reduction without a loss (i.e., the cost is greater than the incentive).

Bidding strategy. An HPC user's net gain from market participation is the payment it gets for resource reduction minus the corresponding performance degradation cost it incurs. Hence, with the market clearing price q', we can write the m-th user's net gain as

$$\mathcal{G}_m = \overbrace{q' \cdot \delta_m(q')}^{\text{Market payoff}} - \overbrace{\mathcal{C}_m(\delta_m(q'))}^{\text{Cost of resource reduction}} \tag{7}$$

The bidding strategy for a user depends on what kind of market is implemented. For MPR-STAT market, the users need to decide their bidding parameters, b_m, without any knowledge of the market clearing price, q'. Note that the bidding parameter Δ_m depends on the HPC application's behavior, and the user does not tune this parameter during bidding. For instance, in `XSBench` we have $\Delta_m = 0.7$. In MPR-STAT, a user cannot maximize its net gain $\mathcal{G}_m$. Nonetheless, we propose a cooperative bidding strategy where the bids are devised to achieve a non-negative net gain over the entire price range. More specifically, in this bidding strategy, a user sets its bidding parameters to keep its bidding curve $\delta_m(q)$ always below its reference cost with the highest supply of resource reduction. We deem this a "cooperative" bidding strategy as the HPC users offer their best resource reduction for handling the power emergency. Fig. 4(a) illustrates the cooperative bid for `XSBench` application. To better understand this bidding strategy for MPR-STAT, we also show a "conservative" bid, where the HPC user is less willing to reduce power and bids for lower resource reduction than its reference. We also show a "deficient" bid, where the HPC user's bid may result in a



(a) Bidding in MPR-STAT (b) Bidding in MPR-INT

Fig. 4. User bidding strategy for market participation.

negative gain for certain clearing prices (for $0.2 \leq q' \leq 0.8$ in Fig. 4(a)).

On the other hand, for MPR-INT market, the clearing price q' is iteratively updated. During each iteration, a user can plug in the clearing price into Eqn. 7, and can find the value of b_m that maximizes its net gain $\mathcal{G}_m$. In this bidding strategy, the user can maximize its market incentive. We illustrate this bidding strategy in Fig. 4(b) for three different clearing prices.

The bidding strategy presented here relies on the estimation of performance impact due to resource reduction. This performance estimation for HPC jobs, however, is non-trivial and introduces additional hurdles on the user's part. We offer a qualitative discussion of such challenges of MPR in Section III-F.

D. Properties of Our Market Design

MPR-STAT vs MPR-INT. The fundamental difference between MPR-STAT and MPR-INT is the market agility, i.e., how quickly the HPC manager can determine the clearing price. While exercising the market in MPR-STAT, the HPC manager has all the information it needs (i.e., the bids and the reduction goal) to handle the overload and, therefore, can very quickly determine how much resource to cut back from each job. Meanwhile, in MPR-INT, multiple rounds of back-and-forth communication between the HPC manager and the users are needed to reach a consensus on the clearing price.

MPR-INT, however, offers theoretical guarantees on the optimality of the overall performance cost [21], [6], and performs as well as OPT. In MPR-STAT, on the other hand, the users devise their bids without any knowledge of the clearing price. Hence, unlike MPR-INT, they cannot guarantee that the power reduction is achieved with the minimum performance impact on the running jobs. MPR-STAT can still capture the relative performance impact of different users' jobs and consistently achieve better cost performance than performance-oblivious power overload handling strategies.

MPR-STAT, due to its agility, is suitable where fast reaction time to power overload is warranted. Meanwhile, MPR-INT can offer the best cost performance, where the HPC system can sustain the power overload long enough for the market to clear. Here, to ensure the safe handling of power emergencies, the HPC manager can set a fixed timeout (e.g., 30 seconds) for MPR-INT's iterations and take the last price as the clearing price. MPR-INT also requires autonomous software agents who send bids without manual user/human involvement. Such bidding agent implementation is relatively straightforward as they require lightweight computation to find the optimum bid for Eqn. (7). Also, to better accommodate MPR-INT, the HPC manager can invoke the market early by predicting power overloads and estimating the power/resource reduction goals.

Scalability. The HPC manager uses MPR's market when a power emergency is detected that needs to be mitigated by power reduction. To set the market clearing price and determine job-wise resource reduction in MPR-STAT, the HPC manager needs to solve MCIr only once using the bids of the active jobs. Moreover, since MCIr has only one optimization variable q and the objective function is monotonically increasing in q, it can be solved by finding the minimum, $q' = \min_q\{q|\sum_{m=1}^{M(t)}\mathcal{P}(\delta_m(q)) = P(t) - C\}$ using a bi-section search. MPR-STAT can scale very well with a growing number of active jobs. In our evaluation, we find that MPR-STAT can find the clearing price in less than a second for even 30,000 active jobs (Fig. 10(a)). In contrast, OPT has M optimization variables (i.e., number of jobs running in the HPC) with exponentially growing problem size (e.g., 40+ minutes to solve a problem of 30,000 jobs).

MPR-INT, on the other hand, is by nature slower as it needs iterative communication between the HPC manager and the users. However, the time required for MPR-INT mainly comes from the communication overhead as MPR-INT also solves the lightweight MCIr only once every communication round. Meanwhile, to determine their bids, the users need to solve even a simpler problem of maximizing $\mathcal{G}_m$ in Eqn. 7 without any constraints. More importantly, the users devise their bids by themselves in a distributed fashion. Hence, the growing number of users only adds more parallel work while the HPC manager's task (to solve MCIr) in every communication round grows similar to MPR-STAT. The main scalability concern for MPR-INT is how many communication rounds are needed for the clearing price to converge. Because of our predefined form of the supply function, the convergence is guaranteed with the user's cost monotonically increasing with resource reduction [6], [22], [21]. In our evaluation, we find that the number of iterations needed for clearing the market for MPR-INT remains almost unchanged even when we increase the jobs from 10 to 30,000 (Fig. 10(b)).

E. Implementation of MPR

Detecting power emergency. MPR uses the HPC cluster's real-time power monitoring to identify when the power consumption exceeds the capacity and there is an overload. MPR then determines the amount of power that needs to be reduced to return the power at or below the capacity. To avoid declaring a power emergency for transient power spikes, the HPC manager may set a minimum duration of overload (e.g., 10 seconds).

Setting the market clearing price. The solicitation of bids and market clearing is done following the adopted version of MPR. The HPC manager plugs in the market clearing price q' and every user's bids (in case of MPR-INT, the bids in the

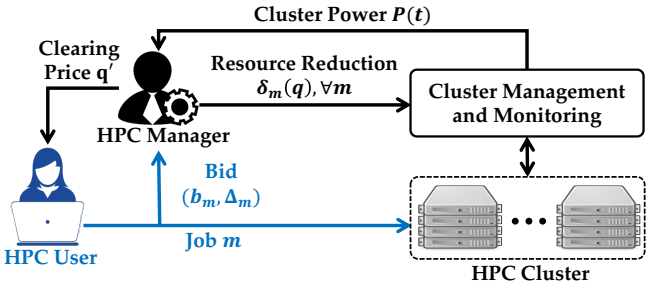


Fig. 5. Interaction between HPC manager and users in MPR.

final iteration) in the supply function (Eqn. 3) to determine the user's corresponding resource reduction.

Executing resource/power reduction. The HPC manager reduces each job's resource allocation utilizing existing techniques such as slowing down the processor using DVFS [24], [38]. During a power emergency, MPR also temporarily halts starting any new HPC job execution.

Resuming normal operation. MPR resumes normal HPC operation when it determines that lifting the power reduction will no longer violate the capacity. Hence, MPR lifts the power emergency when the power consumption falls below the capacity by at least the amount of power reduction. Here, the HPC manager can add a cool-down timer (e.g., 60 seconds) to avoid lifting a power emergency followed by a momentary power dip only to declare an emergency again. The cool-down timer also ensures a minimum time frame for payout to the HPC users participating in the market.

Resource control mechanism. While MPR can be implemented with various resource reduction techniques, such as power capping and node/core scaling, we advocate using DVFS on CPU/GPU cores as it is ubiquitously available with rapid and scalable execution. Moreover, DVFS has a more predictable impact on job execution time as it only slows down the execution (Fig. 16(b)). Hence, confining MPR's resource control knobs to DVFS (or a similar technique) also alleviates the hurdles of performance modeling.

F. Challenges in MPR

Performance prediction for bidding. In MPR, the HPC users devise their bids based on the estimation of the performance impact of resource reduction. Hence, an integral part of MPR is performance prediction which remains challenging [41]. This paper mainly focuses on the user-in-the-loop handling of HPC oversubscription and treats HPC performance modeling as an orthogonal task. Nevertheless, we would like to emphasize that *MPR is not dependent upon rigorous and extensive performance modeling.* In MPR, HPC users express their performance impact through a predefined form of supply function (Eqn. (3)), which is already an approximation of actual performance impact (Fig. 4), allowing margin-of-error in performance prediction. Moreover, HPC users, at their discretion, can intentionally raise their bids b_m (e.g., conservative bid in Fig. 4(a)) to add even more room for estimation error to account for uncertainties of the operating environment, such as interference between different jobs. We evaluate the impact of model error on MPR in Section V-D.

Market participation. Naturally, MPR's user-in-the-loop design requires HPC users' willingness to actively participate in the market to supply the resource reductions necessary to handle the overloads. While the market participation requires additional user efforts (i.e., bidding), as opposed to prior studies on managing oversubscribed HPC systems, MPR compensates users for the inevitable performance impact of oversubscription. Hence, we believe there is a strong incentive for user participation in MPR. We study the impact of user participation on MPR in Section V-D. Meanwhile, to encourage user participation in MPR and ease up their bidding process, the HPC manager can take a more active role by accommodating discounted job execution to assist performance modeling and hosting users' bidding agents.

Market collusion. In theory, MPR's design is susceptible to market collusion where multiple HPC users coordinate and artificially inflate the reward/price for their resource reduction. However, market collusion requires coordination among many users to have enough market power to influence the clearing price. Hence, we believe the efforts outweigh the incentives for market collusion in the HPC system.

Malicious users. Unlike self-serving market colluding users, malicious users want to steal private/secret data and harmfully affect the HPC system. MPR does not add any new attack surface regarding data security. However, allowing users to take an active role in HPC overload handling, MPR creates a new vulnerability. By tracking when the market is invoked, a malicious user would know if the HPC system is experiencing a power overload. The attacker can utilize this information and launch "power attacks" [30], [19], where the attacker triggers power-intensive stage(s) of their active jobs to intensify the overload by creating power spikes. However, such power attacks cannot easily reach dangerous levels (e.g., HPC system/data center shutdown [19]) as the HPC manager actively manages jobs' resources (and hence power capping) and can quickly thwart unwanted power spikes by "directly" reducing the power of all users/jobs bypassing MPR.

Impact on the total cost of ownership (TCO). MPR affects the HPC's TCO in two ways - increase in HPC utilization and reward payoff to HPC users. The infrastructure utilization will increase due to oversubscription affecting the cost of electricity in TCO. Meanwhile, MPR rewards the HPC user for their market participation. The reward payoff will be a MPR specific addition to existing TCO calculations.

IV. Evaluation Methodology

A. Simulation Settings for HPC

Workload traces. We use real-world workload traces for our evaluation. For our core results, we use the workload traces from the `Gaia` cluster at the University of Luxemburg [11]. The `Gaia` trace contains 51,987 jobs spanning a three-month period from May 2014 to August 2014. This trace has been widely used in the literature and referenced in a number of studies throughout the years to generate useful workloads

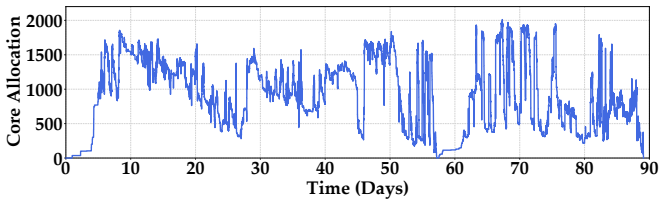


Fig. 6. Core allocation of the Gaia cluster [11].

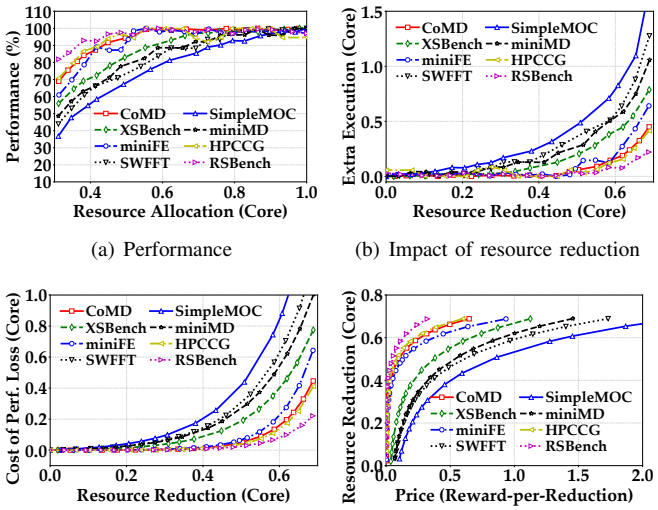
(e.g., [12], [3], [9]). Fig. 6 shows the core allocation of the Gaia cluster with a peak core allocation of 2012.

Power consumption. We convert the core allocations to power consumption using the widely used power model, $\text{Power} = \text{Power}_{static} + \text{Utilization} \cdot \text{Power}_{dynamic}$ [18], considering each core has a dynamic power of 125W and static power of 25W, resulting in peak power of 301.8KW for Gaia. Utilization is calculated as core allocation divided by total available cores. Here, we consider that the power consumption of different components, such as the uncore, DRAM, and storage power, are incorporated in Power_{static} and $\text{Power}_{dynamic}$. The per-core dynamic and static powers are only estimations. Our simulation and analysis hold for other power models as well.

Job simulation. We use Matlab to simulate the HPC job execution by dividing the entire simulation period into one-minute time slots. We get the start time, core allocation, and runtime for each job from the workload traces. We keep a list of active jobs with remaining runtimes. The list is updated at each time slot by adding new jobs (if the system is not overloaded) and discarding completed jobs. For every active job, we also track their core speeds. At the end of a time slot, the remaining runtimes of all active jobs are updated based on their corresponding core speeds. To determine how much work has been done for a given core speed, we use the performance models described in Section IV-B. We use the power model (Section IV-A) to convert the core allocation to HPC power consumption and determine if there is an overload.

Oversubscription levels and power emergencies. For our evaluation, we consider four levels of oversubscription thresholds at 5%, 10%, 15%, and 20%. With x% oversubscription, overloading occurs if the power demand exceeds $\frac{100}{100+x}$ of its peak power consumption (e.g., 301.8 kW for Gaia). To avoid immediate relapse to another overload, we set the power reduction target using an additional 1% buffer as $\Delta P = P(t) - 0.99 \cdot C$. We use a 10-minute cool-down period before we consider resuming normal operation by giving back the capped resources to the active jobs. After the 10-minute cool-down, we resume normal operation if $0.99 \cdot C - P(t) \geq \Delta P$.

Benchmark algorithms. We evaluate MPR against two benchmark algorithms - OPT and EQL. OPT finds the optimum resource allocation by solving the non-linear optimization problem that minimizes the total cost of performance loss of all active jobs. OPT acts as the performance upper limit for handling the overloads. EQL, on the other hand, is oblivious to the performance impact of resource change and equally slows down all cores in the system to reduce power.



(a) Performance (b) Impact of resource reduction

(c) Cost of resource reduction (d) Cost references for bidding

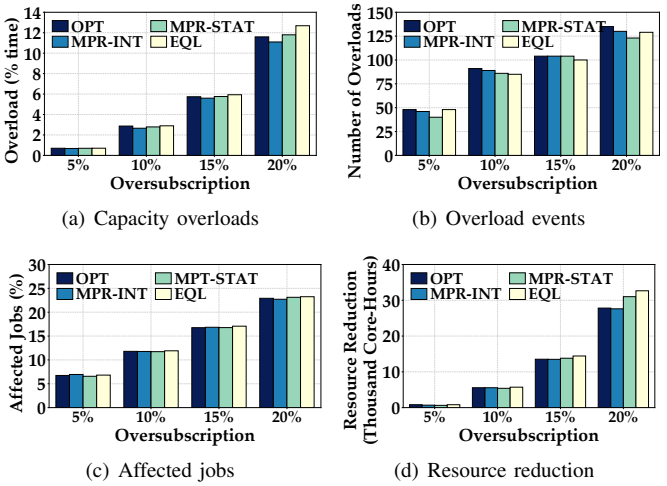
Fig. 7. Performance models, cost models, and bidding references for benchmark applications.

B. Simulation Settings for Users

Performance models. We utilize existing literature to model the performance impact of power capping [41]. We collect the power vs. performance measurements for eight applications that include CoMD- a molecular dynamics simulation application that studies dynamic properties of various materials, XSBench- an application that stresses system through memory capacity, miniFE- a proxy application for unstructured finite element solver, SWFFT- an application for cosmology and astrophysics, SimpleMOC- a three-dimensional reactor simulation application), miniMD- a parallel molecular dynamics code from Mantevo mini-application suite, HPCCG- a conjugate gradient proxy application, and RSBench- a transport application for Monte Carlo neutron transport.

We convert the power capping values from [41] to core allocations by normalizing no power capping (290W) to the core allocation of "1". Fig. 7(a) shows the performance changes for resource allocation changes of our benchmark applications. We see that different applications have different impacts on their performances when their resource allocation is altered. We see that some applications, such as SimpleMOC, SWFFT, miniMD, and XSBench, are more sensitive to changes in resource allocation than others. Here, we do not consider the impact of inter-core/node communication, which is typically much less compared to the impact of power capping [50], [51].

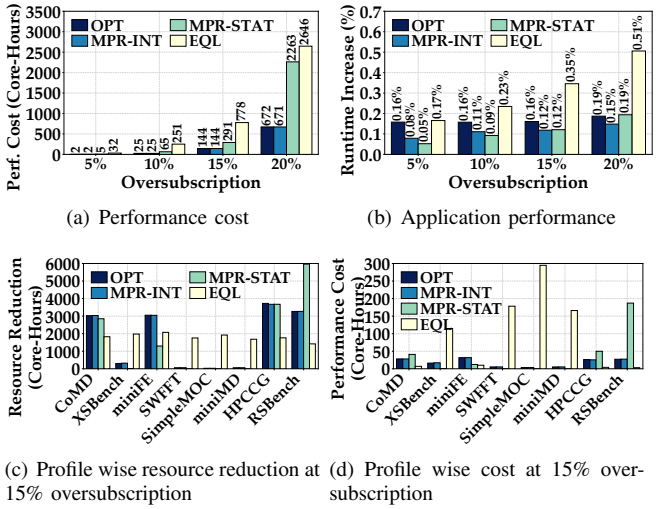
Next, in Fig. 7(b), we show the impact of the performance change in terms of "extra execution" that these applications need to finish the same job due to a change in their performance. In this figure, both the resource reduction and extra execution have the same unit of time - if we consider resource reduction for one hour, then we need the corresponding amount of extra execution cores for one hour as well.



(a) Capacity overloads (b) Overload events

(c) Affected jobs (d) Resource reduction

Fig. 8. Impact of oversubscription on the `Gaia` system and the HPC jobs. `Gaia` has a capacity of ∼4.3million core hours over our simulation period.



(a) Performance cost (b) Application performance

(c) Profile wise resource reduction at 15% oversubscription (d) Profile wise cost at 15% oversubscription

Fig. 9. Comparison of benchmarks over 90-days simulation using `Gaia` trace.

Cost models. We consider the extra execution as the added "cost" for the application when their resource is reduced. Then we use the cost model (Section III-C) to derive the cost. To model the cost of performance loss due to resource reduction, we use a logarithmic curve fitting on our costs calculated based on results from Fig. 7(b). Our logarithmic fitting is $cost = a\log(b \cdot x) - a$, where x is the resource reduction, and a and b are model parameters. Fig. 7(c) shows the cost of resource reduction based on our logarithmic model.

Bidding references. Using the cost calculated in Fig. 7(c), we derive the bidding references for our applications and show them in Fig. 7(d). Here, the price of the bidding references is the cost of unit resource reduction. Since we use cores as both the unit of cost and the unit of resource reduction, our price becomes unitless.

Application profiles. We devise eight application profiles using our performance models, cost models, and bidding references. We uniformly randomly assign an application profile to each HPC job we simulate. The application profile of a job determines its performance impact due to core reduction and its bids in market participation. We also scale up our per-core model with the core allocations of the respective HPC job.

V. Evaluation Results

A. Impact of Oversubscription

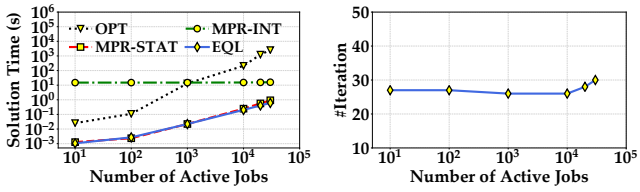
Capacity overloads. Figs. 8(a) and 8(b) show how often the system stays in the overloaded state as we increase the oversubscription level. We see that at 5% oversubscription, the system stays in the overloaded state less than 1% of the time. However, as we increase the oversubscription level, the overload percentage grows super linearly, indicating a diminishing return of oversubscription. All the algorithms perform comparably to each other in terms of causing overloads.

Impact on jobs. In Fig. 8(c), we show the percentage of jobs affected by the overloads. We consider a job has been affected by overload if an overload event occurs when the job was in the active state, regardless of whether the job's resource was cut back or not to handle the overload. We observe that increasing oversubscription affects more jobs. Note that, despite a greater percentage of jobs being affected during the oversubscription, the performance impact on the jobs is not significant (Fig. 9(b)).

Resource reduction. Fig. 8(d) shows the total resource reduction for different algorithms. Since the required resource reduction is dictated by the overloads, all algorithms result in similar amounts of resource reduction.

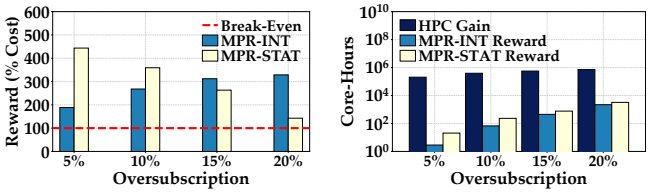
B. Benchmark Comparison

Performance cost. Fig. 9(a) shows the total cost of performance loss due to resource reduction to handle overloads at different oversubscription levels. The unit of cost is "core-hours", indicating how much extra computing is needed to handle the slowdown caused by the overloads. Naturally, the cost increases as we increase oversubscription. Here, we see significant differences in algorithm performances, where EQL suffers from significantly higher performance cost, while MPR-INT achieves cost performance at nearly the same level as OPT. MPR-STAT, however, incurs notably more cost than OPT. EQL's higher cost is due to its performance-oblivious nature. As shown in Fig. 9(c), EQL reduces as much resource from sensitive applications (e.g., `SimpleMOC`) as other less-sensitive applications (e.g., `RSBench`) and incurs high performance cost for the sensitive applications (Fig. 9(d)). Both OPT and MPR-INT achieve a good balance in spreading the resource reduction among the applications, reducing more resources from less-sensitive applications, and vice versa. MPR-STAT, on the other hand, reduces much more resources from the less-sensitive applications (e.g., `RSBench`) and does not reduce any resources from sensitive applications (e.g., `SWFFT`). This is because MPR-STAT uses static bidding from users where users have to bid considering a wide range of



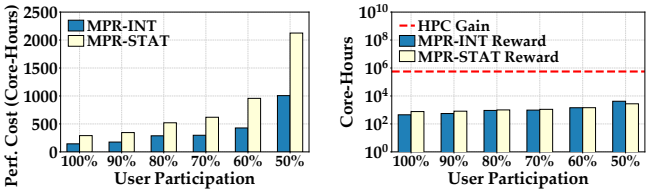
(a) Solution time (b) Iteration number for MPR-INT

Fig. 10. Observed solution time of benchmarks.



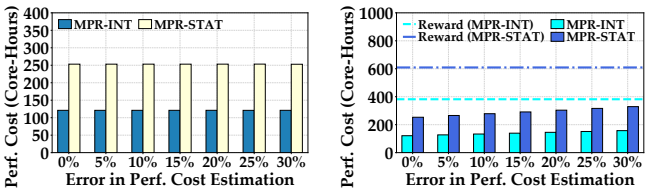
(a) Reward and cost (b) Reward

Fig. 11. User rewards and HPC system's gain from MPR.



(a) Performance cost (b) Reward

Fig. 12. Impact of user participation on MPR. (a) Impact on performance cost. (b) Impact on reward payoff.



(a) Random estimation error (b) Performance cost underestimation

Fig. 13. (a) Random estimation errors do not affect MPR. (b) Even when users underestimate cost, they retain a net gain - more reward than cost.

prices and end up soliciting unnecessary conservative bids for lower price ranges.

Application performance. Fig. 9(b) shows the average increase in runtime (compared to the no oversubscription case) of only the jobs affected by the overload. We see that there is less than 1% increase in average runtime for any algorithm. The performance impact is very small as the overload periods are a small fraction of typical job's total execution time. Moreover, most overloads require less than 20% resource reduction, for which, only sensitive jobs suffer a discernible performance degradation (Fig. 7(b)).

We see that EQL performs worse than other algorithms, and causes a greater increase in the execution time. We also see that MPR-STAT performs better than OPT and MPR-INT on many occasions. This is because MPR-STAT heavily relies on the less-sensitive jobs for reaching the target resource reduction. Nonetheless, the takeaway is that overload handling does not significantly affect performance.

Scalability. Fig. 10 presents the solution times (i.e., time to determine the resource reductions) of OPT, MPR-INT, MPR-STAT, and EQL for varying numbers of active jobs on an iMac computer with Intel Core i9 processor and 128GB of memory. The solution time increases for all the benchmarks as the number of active jobs increases (Fig. 10(a)). Note that, MPR-STAT demonstrates very good performance compared to OPT for varying numbers of active jobs. The EQL benchmark achieves the same level of performance as that of MPR-STAT, however, incurs a significantly higher cost of performance loss compared to MPR-STAT (Fig. 9(a)). Note here that, EQL's increasing solution time is due to the "bookkeeping" (e.g., log each job's new CPU allocation) associated with an increasing number of active jobs. MPR-INT, on the other hand, needs additional communication time for each round of iterative bidding. We present MPR-INT's solution time considering that each communication round adds 500 milliseconds. Nevertheless, we see that MPR-INT can find the resource allocation in less than 30 seconds even with 30,000 active jobs. Fig. 10(b) shows the number of iterations needed for clearing the market for MPR-INT algorithm. The iteration number remains almost unchanged with the number of active jobs.

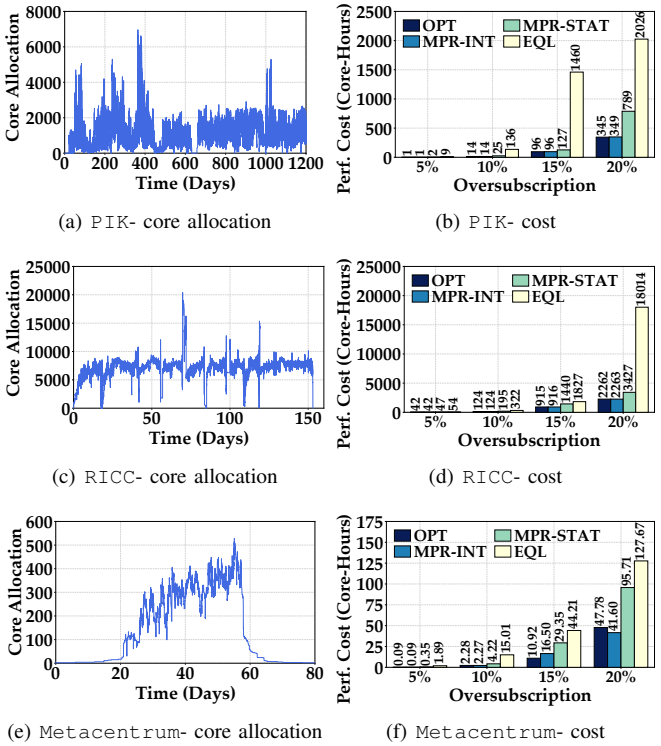
C. Market Performance

User's reward. Fig. 11(a) shows the reward users receive for their participation. The reward is calculated as a percentage of the cost incurred due to performance degradation from resource reduction. As evident from the figure, users always receive more rewards (>100%) than their cost of performance loss. Therefore, users will always enjoy a net benefit for participating in MPR's market for overload handling.

HPC system's benefit. Fig. 11(b) shows the gain of the HPC manager due to oversubscription and the reward earned by the HPC users. We see that the HPC manager gains orders of magnitude more core-hours than she had to pay to the users as the incentive/reward. Also, note that while the HPC gain increases with oversubscription, the user incentive grows at a higher rate. It indicates that it is not beneficial for the HPC manager to oversubscribe the system beyond a certain level. Nevertheless, *these results highlight the economic motivation for both the HPC manager and HPC users to adopt MPR.*

D. Sensitivity Study

User participation. If fewer users participate in MPR, for a given power reduction target, each job needs to supply more resource reduction incurring more performance costs, while the HPC manager will need to pay more rewards. Fig. 12 shows the impact of user participation on the overall performance cost and reward for 15% oversubscription. We see increasing performance cost with decreasing user participation in Fig. 12(a). As shown in Fig. 12(b), the increase in users'



(a) PIK- core allocation (b) PIK- cost

(c) RICC- core allocation (d) RICC- cost

(e) Metacentrum- core allocation (f) Metacentrum- cost

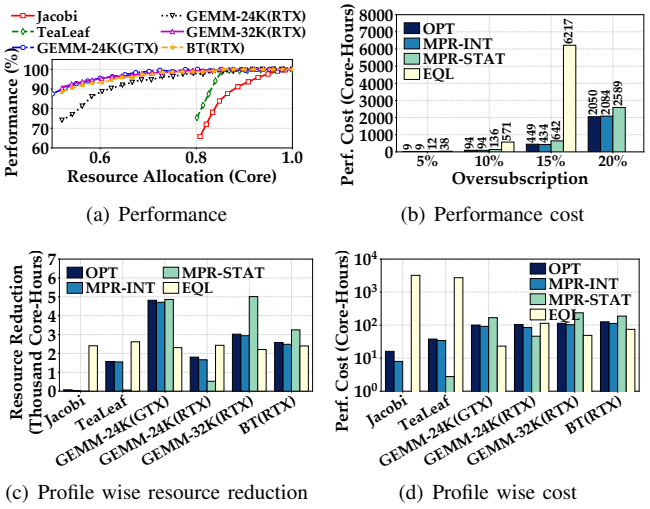
Fig. 14. Performance comparison of MPR under different workload traces demonstrating its effectiveness in various scenarios.

performance cost is offset by an increase in reward payment from the HPC manager. However, note that, even at 50% user participation, HPC gain remains two orders of magnitude higher than the reward payment.

Error in the performance cost model. To understand the impact of errors in the performance cost model, we study the actual performance cost when there are random estimation errors of up to 30% in Fig. 13(a). We see that random estimation errors do not affect the overall performance cost. We then study with a pessimistic setting where users underestimate their true performance cost and show the results in Fig. 13(b). We see that, while the cost increases with underestimation, even at 30% underestimation, the reward is two times the cost for MPR-INT and MPR-STAT.

E. Evaluation Under Different Settings

Different workload traces. We collect three other workloads (PIK, RICC, and Metacentrum) for this study from [13]. The traces are representative of different workload characteristics. The PIK trace is from a medium-scale HPC cluster over a longer time duration (Fig. 14(a), RICC trace is from a large-scale HPC cluster (Fig. 14(c)), and Metacentrum trace is from a small-scale HPC cluster (Fig. 14(e)). The PIK trace contains 742,964 jobs spanning a three-year period from April 2009 to July 2012. The RICC trace contains 447,794 jobs over a 5-month period from May 2010 to September 2010. The Metacentrum trace contains 103,656 jobs, which are collected from January 2009 to May



(a) Performance (b) Performance cost

(c) Profile wise resource reduction (d) Profile wise cost

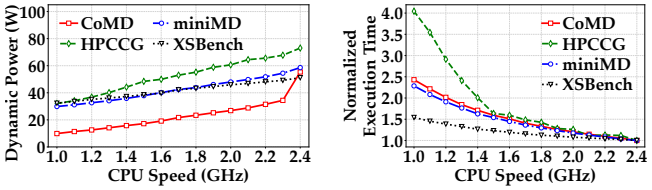
Fig. 15. MPR under a heterogeneous system with GPUs.

2009. PIK trace, RICC trace, and Metacentrum traces have peak CPU allocation of 6,963 cores, 20,4156 cores, and 528 cores, respectively.

Figs. 14(b), 14(d), and 14(f) show the cost of performance loss for the workload PIK, RICC, and Metacentrum, respectively. The performance cost increases with the increase in oversubscription. MPR-INT achieves cost performance almost the same as OPT for all traces. EQL suffers from much higher performance cost compared to MPR-INT and OPT while MPR-STAT also incurs higher costs than OPT.

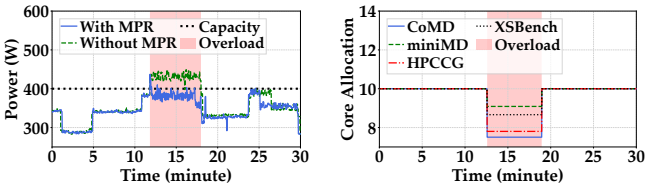
Heterogeneous system with GPU. To evaluate MPR using a heterogeneous HPC system, we collect six different HPC applications' power and performance data on GPU nodes from [5], [26]. The resource-performance relation of the six applications is shown in Fig. 15(a). Jacobi and TeaLeaf are from [5] and runs on NVIDIA P40 GPUs. Meanwhile, the GEMM and BT are from [26] running on NVIDIA GTX 1070 and RTX 2080 GPUs. We use Gaia trace for this evaluation. We normalize each application's maximum power to "one core" allocation to maintain generality. For instance, for Jacobi and GEMM, "one core" is when the power consumption is 225W and 200W, respectively.

Fig. 15(b) shows the overall performance cost under different levels of oversubscription. The results are similar to our prior evaluation using a homogeneous CPU-based system. MPR-INT performs at the same level as OPT, while MPR-STAT incurs additional costs. EQL, in this case, performs much worse and cannot even provide a feasible resource allocation at 20% oversubscription. As shown in Figs. 15(c) and 15(d), this is because Jacobi and TeaLeaf suffer significantly more performance loss under EQL due to resource reduction, while the other algorithms do not ask for much resource reduction from these two performance-sensitive applications. These results highlight that *performance oblivious approaches will suffer significantly when managing HPC applications with a diverse resource-performance relation.*



(a) Power (b) Performance

Fig. 16. Impact of CPU speed change on dynamic power and execution time.



(a) Power (b) Resource allocation

Fig. 17. Demonstration of MPR on our prototype HPC cluster.

F. Prototype Experiment

We run MPR on a prototype HPC cluster consisting of two Dell PowerEdge servers with a total of 40 Intel Xeon CPU cores and 256GB of memory. We implement four applications from our simulation study - `CoMD`, `HPCCG`, `miniMD`, and `XSBench`. We run these applications each with 10 CPU cores. We use `acpi-cpufreq` driver for Linux to control the CPU frequency for resource/power reduction [62]. Fig. 16(a) shows the dynamic power of the applications as we change the CPU speed from 1GHz to 2.4GHz. Fig. 16(b) shows the corresponding execution times normalized to each application's execution time at 2.4GHz. In both figures, we see that the impact of CPU speed change is different for different applications. This supports the need for MPR's application/user-level control approach.

Next, we run two 30-minute experiments - one without MPR and one with MPR, where we create overload conditions by setting the power capacity at 400W. As shown in Fig. 17(a), MPR handles the overload by reducing the power by nearly 50W by slowing down the CPU speeds (i.e., reducing resource allocation) of the applications. In Fig. 17(b), we see that different applications reduce different amounts of resources based on their performance impact and bids. We devise the bids for these applications based on their performance impact (Fig. 16(b)) and follow the steps outlined in Section III-C. Our prototype experiment, albeit on a small scale, demonstrates the effectiveness of using MPR in handling power overloads due to HPC oversubscription.

VI. Related Work

Power overprovisioning in the cloud and HPC systems. Power oversubscription in hyper-scale/cloud data centers has been actively studied to overcome infrastructure underutilization and save capital investment (e.g., [49], [48], [64]). However, power overprovisioning in HPC systems has been relatively less explored but gaining traction in recent years. Works on HPC overprovisioning focus on tackling the job scheduling to satisfy the ensuing operational constraints [45], [52], [44], [10]. Khemka et al. [23], [35] develops dynamic resource management techniques to safely oversubscribe heterogeneous distributed systems. Xiong et al. [57] discuss the interference problem that can be introduced when colocating applications on oversubscribed nodes. The authors then propose an application framework to colocate HPC applications by combining offline profiling, machine learning, and scheduling. Sakamoto et al. [50] explores power-aware resource management techniques at scale in overprovisioned HPC systems. In [51], authors develop a hardware overprovisioning system that allocates extra nodes to the system. The proposed system includes various strategies for dynamic allocation of power capping, various node on-off techniques, and job scheduling techniques. Patel et al. [41] presents a power management framework to improve the system throughput of a hardware-overprovisioned HPC system while ensuring fairness among concurrently running jobs.

Mechanism design applications. Mechanism design is widely-used in many real-life applications. Vickery Clarke Grove (VCG) auction is a sealed-bid auction mechanism, where bidders submit their bids of items with unknown information about other bidders. The mechanism rewards users for their true valuations of the items. VCG auction has been widely used in different fields, including network communication [59], [17], crowdsourcing [8], [60], smart grid [63], among others. Although VCG auction mechanism is efficient and incentive compatible, the mechanism requires the users to reveal their cost functions, which are private function. Supply function bidding is a cost-efficient mechanism that ensures optimality at a Nash equilibrium. Compared to other mechanism models (e.g., VCG auction) supply function is simpler and does not reveal the private cost function of the users. Supply function has been applied in various applications, such as demand response [58], [31] and power emergencies [20].

VII. Conclusion

In this paper, we presented MPR, a market-based approach to managing oversubscribed HPC systems. MPR enables HPC users' participation in resource reduction in exchange for rewards. Using extensive real-world trace-based simulation, we showed that both HPC users and HPC managers are highly incentivized for their market participation. To the best of our knowledge, the solution outlined in this paper is the first market-based approach to handle power oversubscription in an HPC system via active user participation.

VIII. Acknowledgments

This work is supported in parts by the US National Science Foundation under grants ECCS-2152357 and CNS-2300124.

REFERENCES

[1] "Data center cost: Total cost of ownership (tco) pricing breakdown," https://ongoingoperations.com/data-center-pricing-credit-unions/#closemodal, 2022, accessed 8th Oct 2022.

[2] K. Ahmed, J. Liu, and X. Wu, "An energy efficient demand-response model for high performance computing systems," in *MASCOTS*, 2017.

[3] W. Ahmed, S. Fomenkov, and S. Gaevoy, "Reducing approximation time of cluster workload by using simplified hypergamma distribution," in *ICIEAM*, 2018.

[4] APC.com, "Apc 100kva modular power distribution unit," https://www.apc.com/shop/us/en/products/APC-100kVA-Modular-Power-Distribution-Unit-208V-72-Poles-MBP-1-Subfeed/P-PDPM100F-M, accessed 4th Feb 2022.

[5] R. Azimi, C. Jing, and S. Reda, "Powercoord: A coordinated power capping controller for multi-cpu/gpu servers," in *IGSC*, 2018.

[6] N. Chen, X. Ren, S. Ren, and A. Wierman, "Greening multi-tenant data center demand response," *Performance Evaluation*, vol. 91, pp. 229–254, 2015.

[7] E. G. Coffman, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Combinatorial analysis," in *Handbook of combinatorial optimization*. Springer, 1999, pp. 151–207.

[8] W. Dong, S. Rallapalli, R. Jana, L. Qiu, K. Ramakrishnan, L. Razoumov, Y. Zhang, and T. W. Cho, "ideal: Incentivized dynamic cellular offloading via auctions," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1271–1284, 2013.

[9] A. A. El-Moursy, A. Abdelsamea, R. Kamran, and M. Saad, "Multi-dimensional regression host utilization algorithm (mdrhu) for host overload detection in cloud computing," *Journal of Cloud Computing*, vol. 8, no. 1, pp. 1–17, 2019.

[10] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz, "Dynamic power sharing for higher job throughput," in *SC*, 2015.

[11] J. Emeras, "Parallel workloads archive: The university of luxemburg gaia cluster log," https://www.cs.huji.ac.il/labs/parallel/workload/l_unilu_gaia/index.html, 2007.

[12] J. Emeras, S. Varrette, M. Guzek, and P. Bouvry, "Evalix: classification and prediction of job resource consumption on hpc platforms," in *JSSPP*, 2015.

[13] D. Feitelson *et al.*, "Parallel workloads archive," http://www.cs.huji.ac.il/labs/parallel/workload/, 2007.

[14] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers," in *ICAC*, 2011.

[15] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," pp. 68–73, 2008.

[16] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual," *Volume 3B: System programming Guide, Part*, vol. 2, no. 11, 2011.

[17] S. Hua, X. Zhuo, and S. S. Panwar, "A truthful auction based incentive framework for femtocell access," in *WCNC*, 2013.

[18] M. A. Islam, S. Ren, A. H. Mahmud, and G. Quan, "Online energy budgeting for cost minimization in virtualized data center," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 421–432, 2015.

[19] M. A. Islam, S. Ren, and A. Wierman, "Exploiting a thermal side channel for power attacks in multi-tenant data centers," in *ACM CCS*, 2017.

[20] M. A. Islam, X. Ren, S. Ren, A. Wierman, and X. Wang, "A market approach for handling power emergencies in multi-tenant data center," in *HPCA*, 2016.

[21] R. Johari and J. N. Tsitsiklis, "Parameterized supply function bidding: Equilibrium and efficiency," *Operations research*, vol. 59, no. 5, pp. 1079–1089, 2011.

[22] F. Kamyab, M. Amini, S. Sheykhha, M. Hasanpour, and M. M. Jalali, "Demand response program in smart grid using supply function bidding mechanism," *IEEE Transactions on Smart Grid*, vol. 7, no. 3, pp. 1277–1284, 2015.

[23] B. Khemka, R. Friese, L. D. Briceno, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos *et al.*, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2394–2407, 2014.

[24] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *HPCA*, 2008.

[25] D. Klusáček and V. Chlumskỳ, "Evaluating the impact of soft walltimes on job scheduling performance," in *JSSPP*, 2018.

[26] A. Krzywaniak and P. Czarnul, "Performance/energy aware optimization of parallel applications on gpus under power capping," in *PPAM*, 2019.

[27] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura *et al.*, "{Prediction-Based} power oversubscription in cloud platforms," in *USENIX ATC*, 2021.

[28] M. Kurokawa, "Parallel workloads archive: The ricc log," https://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html, 2010.

[29] A. Langer, H. Dokania, L. V. Kalé, and U. S. Palekar, "Analyzing energy-time tradeoff in power overprovisioned hpc data centers," in *IPDPSW*, 2015.

[30] C. Li, Z. Wang, X. Hou, H. Chen, X. Liang, and M. Guo, "Power attack defense: Securing battery-backed data centers," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 493–505, 2016.

[31] N. Li, L. Chen, and M. A. Dahleh, "Demand response using linear supply function bidding," *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1827–1838, 2015.

[32] Y. Li, C. R. Lefurgy, K. Rajamani, M. S. Allen-Ware, G. J. Silva, D. D. Heimsoth, S. Ghose, and O. Mutlu, "A scalable priority-aware approach to managing data center server power," in *HPCA*, 2019.

[33] X.-k. Liao, K. Lu, C.-q. Yang, J.-w. Li, Y. Yuan, M.-c. Lai, L.-b. Huang, P.-j. Lu, J.-b. Fang, J. Ren *et al.*, "Moving from exascale to zettascale computing: challenges and techniques," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 10, pp. 1236–1244, 2018.

[34] C. Linstead, "Parallel workloads archive: The pik ibm idataplex cluster log," https://www.cs.huji.ac.il/labs/parallel/workload/l_pik_iplex/index.html, 2012.

[35] D. Machovec, B. Khemka, N. Kumbhare, S. Pasricha, A. A. Maciejewski, H. J. Siegel, A. Akoglu, G. A. Koenig, S. Hariri, C. Tunc *et al.*, "Utility-based resource management in an oversubscribed energy-constrained heterogeneous environment executing parallel applications," *Parallel Computing*, vol. 83, pp. 48–72, 2019.

[36] S. Malla and K. Christensen, "The effect of server energy proportionality on data center power oversubscription," *Future Generation Computer Systems*, vol. 104, pp. 119–130, 2020.

[37] A. Mann, "Core concept: nascent exascale supercomputers offer promise, present challenges," *PNAS*, 2020.

[38] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: understanding the runtime effects of frequency scaling," in *ICS*, 2002.

[39] D. Monroe, "Fugaku takes the lead," *Communications of the ACM*, vol. 64, no. 1, pp. 16–18, 2020.

[40] C. Nadjahi, H. Louahlia, and S. Lemasson, "A review of thermal management and innovative cooling strategies for data center," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 14–28, 2018.

[41] T. Patel and D. Tiwari, "Perq: Fair and efficient power management of power-constrained large-scale computing systems," in *HPDC*, 2019.

[42] T. Patel, A. Wagenhäuser, C. Eibel, T. Hönig, T. Zeiser, and D. Tiwari, "What does power consumption behavior of hpc jobs reveal?: Demystifying, quantifying, and predicting power consumption characteristics," in *IPDPS*, 2020.

[43] T. Patki, Z. Frye, H. Bhatia, F. Di Natale, J. Glosli, H. Ingolfsson, and B. Rountree, "Comparing gpu power and frequency capping: A case study with the mummi workflow," in *WORKS*, 2019.

[44] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *SC*, 2013.

[45] 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 *HPDC*, 2015.

[46] Predictive Technology Inc, "Four factors that affect data center battery life," https://www.ptisolutions.com/four-factors-data-center-battery-life/, 2020, accessed 18th May 2022.

[47] Raritan, "Data center power overload protection," *White Paper*, 2016.

[48] S. Reda, R. Cochran, and A. K. Coskun, "Adaptive power capping for servers with multithreaded workloads," *IEEE Micro*, vol. 32, no. 5, pp. 64–75, 2012.

[49] V. Sakalkar, V. Kontorinis, D. Landhuis, S. Li, D. De Ronde, T. Blooming, A. Ramesh, J. Kennedy, C. Malone, J. Clidaras *et al.*, "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *ASPLOS*, 2020.

[50] R. Sakamoto, T. Cao, M. Kondo, K. Inoue, M. Ueda, T. Patki, D. Ellsworth, B. Rountree, and M. Schulz, "Production hardware over-

provisioning: Real-world performance optimization using an extensible power-aware resource management framework," in *IPDPS*, 2017.
[51] R. Sakamoto, T. Patki, T. Cao, M. Kondo, K. Inoue, M. Ueda, D. Ellsworth, B. Rountree, and M. Schulz, "Analyzing resource trade-offs in hardware overprovisioned supercomputers," in *IPDPS*, 2018.
[52] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *SC*, 2014.
[53] TOP500.org, "Green 500 list - november 2021," https://www.top500.org/lists/green500/list/2021/11/, 2022, accessed 12th March 2022.
[54] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis," *PHYSOR*, 2014.
[55] G. Wang, S. Wang, B. Luo, W. Shi, Y. Zhu, W. Yang, D. Hu, L. Huang, X. Jin, and W. Xu, "Increasing large-scale data center capacity by statistical power control," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, pp. 1–15.
[56] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, "Dynamo: Facebook's data center-wide power management system," in *ISCA*, 2016, pp. 469–480.
[57] Q. Xiong, E. Ates, M. C. Herbordt, and A. K. Coskun, "Tangram: Colocating hpc applications with oversubscription," in *HPEC*, 2018.
[58] Y. Xu, N. Li, and S. H. Low, "Demand response with capacity constrained supply function bidding," *IEEE Transactions on Power Systems*, vol. 31, no. 2, pp. 1377–1394, 2015.
[59] D. Yang, X. Fang, and G. Xue, "Truthful auction for cooperative communications with revenue maximization," in *ICC*, 2012.
[60] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *MobiCom*, 2012.
[61] F. Yang and A. A. Chien, "Zccloud: Exploring wasted green power for high-performance computing," in *IPDPS*, 2016.
[62] H. Ye, "Using cpufreq on linux servers to manage power consumption," *Lenovo Press*, 2018.
[63] M. Zeng, S. Leng, S. Maharjan, Y. Zhang, and S. Gjessing, "Group bidding for guaranteed quality of energy in v2g smart grid networks," in *ICC*, 2015.
[64] C. Zhang, A. G. Kumbhare, I. Manousakis, D. Zhang, P. A. Misra, R. Assis, K. Woolcock, N. Mahalingam, B. Warrier, D. Gauthier *et al.*, "Flex: High-availability datacenters with zero reserved power," in *ISCA*, 2021.
[65] W. Zheng, K. Ma, and X. Wang, "Hybrid energy storage with supercapacitor for cost-efficient data center power shaving and capping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1105–1118, 2016.
[66] W. Zheng and X. Wang, "Data center sprinting: Enabling computational sprinting at the data center level," in *ICDCS*, 2015.