2017 IEEE International Symposium on High Performance Computer Architecture

Fast Decentralized Power Capping for Server Clusters

Reza Azimi
School of Engineering
Brown University
Providence, RI
reza_azimi@brown.edu

Masoud Badiei
SEAS
Harvard University
Cambridge, MA
mbadieik@seas.harvard.edu

Xin Zhan
School of Engineering
Brown University
Providence, RI
xin_zhan@brown.edu

Na Li
SEAS
Harvard University
Cambridge, MA
nali@seas.harvard.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI
sherief_reda@brown.edu

***Abstract*—Power capping is a mechanism to ensure that the power consumption of clusters does not exceed the provisioned resources. A fast power capping method allows for a safe over-subscription of the rated power distribution devices, provides equipment protection, and enables large clusters to participate in demand-response programs. However, current methods have a slow response time with a large actuation latency when applied across a large number of servers as they rely on hierarchical management systems. We propose a fast decentralized power capping (DPC) technique that reduces the actuation latency by localizing power management at each server. The DPC method is based on a maximum throughput optimization formulation that takes into account the workloads priorities as well as the capacity of circuit breakers. Therefore, DPC significantly improves the cluster performance compared to alternative heuristics. We implement the proposed decentralized power management scheme on a real computing cluster. Compared to state-of-the-art hierarchical methods, DPC reduces the actuation latency by 72% up to 86% depending on the cluster size. In addition, DPC improves the system throughput performance by 16%, while using only 0.02% of the available network bandwidth. We describe how to minimize the overhead of each local DPC agent to a negligible amount. We also quantify the traffic and fault resilience of our decentralized power capping approach.**

I. Introduction

Over the last decade, the architecture of datacenter has evolved significantly to accommodate emerging online resource-intensive workloads such as software as a service and social networking. Modern architectures facilitate power management and reduce power provisioning costs by capping the peak power capacity of the datacenter. Fast power capping is a desirable feature for three main reasons:

- **Power Over-subscription:** To increase the efficiency of datacenters, many servers are normally hosted by an electric circuit than its rated power permits [1]. This power over-subscription is justified by the fact that the nameplate ratings on servers are higher than the servers' actual power utilization. Moreover, rarely all servers work at their peak powers simultaneously. In the case that the power consumption of subscribed servers peak at the same time and exceed the circuit capacity, power must be capped quickly to avoid tripping the circuit breakers.
- **Equipment Protection:** Power capping can be used as a safety measure to reduce power consumption of servers when supporting equipment fails. For instance, the breakdown of a datacenter's computer room air conditioning (CRAC) system may result in a sudden temperature increase of IT devices [2]. In this scenario, power capping can help maintain the baseline temperature inside the datacenter.
- **Demand-Response Participation:** Dynamic power capping regulates the total power consumption under dynamic time-varying power caps. This is an important feature for short-term trading where energy transactions are cleared simultaneously to match electricity supply with demand in real time [3]. Even in a more static day-ahead energy market, fluctuations in diurnal pattern of submitted queries may necessitate a fast response from power management tools to ensure an optimized performance for the cluster.

To facilitate the enforcement of individual server power caps, current generation of servers are equipped with power management mechanisms that can throttle the power usage via RAPL [4] or dynamic voltage frequency scaling (DVFS) [5]. Based on these two mechanisms, different power capping schemes have been developed to coordinate the power caps dynamically across multiple servers and applications [6], [1], [7]. To coordinate power caps across servers, current techniques require data aggregation from all computing units that include workload, application priority, and thermal footprint information among many others. Facebook's Dynamo is an example of such hierarchical structure [6].

We observe that hierarchical power capping techniques such as Dynamo have a slow response time with dead time caused by its actuation latency, which makes it inadequate for tracing dynamical power caps at a fast time scale [1]. Moreover, due to employing heuristic methods for reducing the power of servers, power capping techniques can result in a significant performance degradation. Our main insight is that decentralized power capping allows for localizing power capping computation at each server which improves the speed and minimizes the performance degradation due to power capping. We propose a fully decentralized power capping (DPC) framework, where each server has a DPC

2378-203X/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCA.2017.49

agent that locally computes its power usage such that (i) the aggregated throughput of the entire cluster is maximized, (ii) workload priorities are taken into account, and (iii) the power usage of the cluster is capped at a certain threshold. Specifically, in our proposed framework, each DPC agent first transmits messages to neighboring DPC agents using the cluster's network. It then updates its power consumption in the form of a state space model, where the states are local power consumption and power cap violation estimation, and the inputs are estimates of neighboring agents' states transmitted over the cluster's network.

The DPC scheme expands and improves upon the distributed power management framework we developed earlier [8]. In the proposed DPC scheme, we exploit the workloads' priorities to mitigate the performance degradation that may result from a server power capping. Moreover, the new DPC framework incorporates the capacity of multiple circuit breakers into the power capping decision making process. We also focus on the implementation and practical aspects of the DPC method whereas in our earlier work, the algorithm performance analysis was limited to simulations. In this work, we also analyze the network topology and propose a new more effective penalty function for our algorithm.

In this paper, we implement and evaluate our proposed DPC framework on an experimental computing cluster of 16 Xeon-based servers. For comparison, we also implement and test three other classes of power management methods, namely a uniform power allocation, Facebook's Dynamo algorithm [6], and a centralized power capping method [9]. To investigate the performance of DPC framework, we evaluate a number of metrics, including (i) attained system performance for workloads with priority, (ii) response to varying workload utilization, (iii) convergence rates and network traffic as a function of DPC communication topology, (iv) dead time to actuation, and (v) fault resilience.

The rest of this paper is organized as follows. In Section II, we present a decentralized power capping algorithm and provide the underlying architecture. In Section III, we provide a comprehensive set of experimental results and compare our distributed framework with existing methods including Facebook's Dynamo algorithm. In Section IV, we review related works on power capping techniques. In Section V we discuss our results and conclude this paper.

II. Proposed DPC Framework

Before we describe a mathematical formulation for the proposed power capping technique, we provide a general overview of the DPC framework.

The decentralized power capping has a structure as shown in Figure 1. In this structure, users submit jobs with different priorities to the job scheduler (SLURM)[10]. The scheduler in turn allocates jobs to the servers based on their priorities. Each server is equipped with a DPC agent with three components; 1) Workload monitor (WM), 2) DPC, and 3) power controller (PC). DPC agent of server i receives the current workloads' characteristics including a throughput

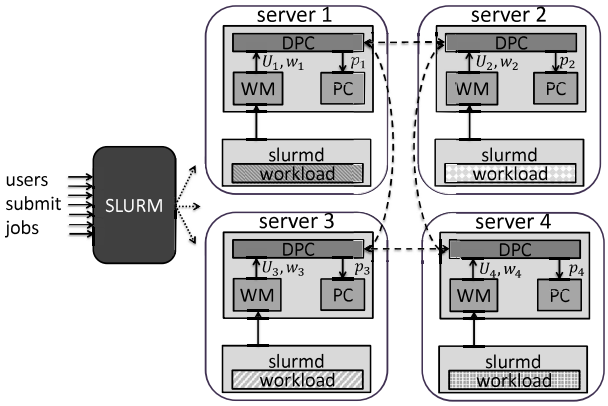

Figure 1: Structure of DPC algorithm. Jobs get submitted to SLURM and workload monitor (WM) get the workload information from SLURM daemon (`slurmd`). DPC gets the workload information from WM and actuate the power cap using the power controller (PC).

function $U_i(\cdot)$ and the workload's priority information w_i from the workload monitor. It then solves the optimization problem formulated in Subsection II-A. The solution of the optimization provides the local power cap of server i, p_i, for its power controller module to apply.

Note that our decentralized power capping framework is iterative. At each iteration, each DPC agent computes local decision variables and communicates their local information with their neighbors. We take into account the physical power limits of the Circuit Breakers (CBs) to avoid circuit tripping. Therefore, the DPC power capping framework allows form power over-subscription. In other words, the planned peak power demand can be higher than what is supplied, which improves the efficiency of cluster.

A. Problem Formulation

The DPC algorithm is based on a weighted sum throughput maximization problem subject to (i) a power cap R_0 on the cluster power consumption, (ii) a power cap $R_k, k = 1, 2, \cdots, r$ for each circuit breaker, and (iii) the power constraint of each server. More specifically, we consider a heterogeneous system where the i-th *active* server in the cluster of n-nodes $N = \{1, 2, \cdots, n\}$ has a utility function $U_i(p_i)$, where $P_i^{\min} \leq p_i \leq P_i^{\max}$, where $P_i^{\min}$ and $P_i^{\max}$ represent the minimum and maximum power consumption of the i-th server. Following the method of [11], we also consider a set of weight factors $w_i \in \mathbb{R}^+, i = 1, 2, \cdots, n$ that determine the workload priority, *i.e.*, a large weight corresponds to a high priority workload. Note that the throughput function $U_i(\cdot)$ and the weight $w_i > 0$ of each server are not fixed. Rather, they change depending on the type of the workload being processed by each server. However, in the design of DPC, we formulate the sum throughput maximization problem with a set of fixed utility functions for servers. Upon change in the workload of each server, DPC re-adjusts the optimal power consumption

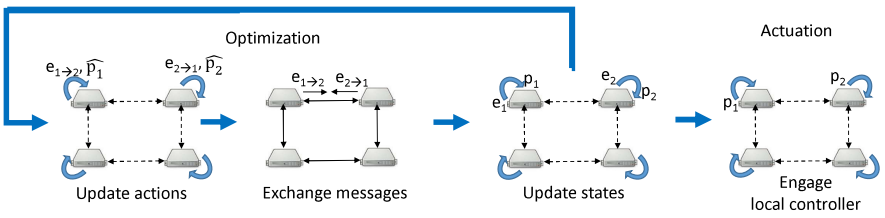

Figure 2: The main steps of DPC algorithm.

for the new workload configuration. Also for all types of practical workloads under consideration, the utility functions of all servers are concave, as verified by our results in Section III.

We also consider a set of r circuit breakers $\{CB_k\}_{k=1}^r$ that form a cover of the cluster of n servers, *i.e.*, each of n servers is mapped to one or more of the circuit breakers. To simplify our formulation, we define $CB_0 := N$ as the set of all servers in the cluster.

To develop the DPC algorithm, we formulate the following sum throughput optimization problem:

$$\max_{p_1,p_2,\cdots,p_n} \sum_{i=1}^{n} w_i U_i(p_i) \tag{1a}$$

$$\text{subject to :} \quad \sum_{i\in \text{CB}_k} p_i \le R_k, \quad k=0,1,\cdots,r, \tag{1b}$$

$$P_i^{\min} \le p_i \le P_i^{\max}, \quad i=1,2,\cdots,n. \tag{1c}$$

B. DPC Algorithmic Construction

Herein, we outline the DPC algorithm. We defer the more technical detail of our derivations to Appendix A.

The main objective of DPC is to provide a decentralized procedure for solving the optimization problem in Eqs. (1a)-(1c). However, the challenge in designing such a procedure is that the power usages of servers are coupled through the power capping constraints in Eq. (1b). We decouple the optimization problem (1a)-(1c) by augmenting the local utility $U_i(p_i)$ of each server $i \in N$ with a penalty function defined on the estimation terms. Maximizing the augmented utility at each server guarantees that the power cap constraints in Eq. (1b) are satisfied.

For each power capping constraint $k = 0,1,\ldots,r$ in Eq. (1b) in DPC algorithm, we define an estimate (belief) variable e_i^k for each server i. Specifically, $e_i^k(t) \le 0$ means that the ith server's estimate from the kth power capping constraint in Eq. (1b) is satisfied, whereas $e_i^k(t) \ge 0$ indicates a constraint violation proportional to the magnitude of $e_i^k(t)$. Servers communicate these estimation terms with their neighboring agents to obtain accurate values of the constraint violation. The main steps of DPC algorithms are depicted in Figure 2. Each server maintains a vector of state variables $\left(p_i(t), \{e_i^k(t)\}_{k=0}^r\right)$ where $p_i(t)$ is the power usage and $e_i^k(t)$ is the estimation term for the kth power capping constraint. Our algorithm is iterative and at each iteration t and for each server $i \in N$, DPC has the following steps (see Fig. 2): REPEAT STEPS I-III

i) *Update actions*: compute a vector of actions, $\left(\hat{p}_i(t), \{e_{i\to j}^k(t)\}_{j\in N(i)}^{k\in\{0,1,\cdots,r\}}\right)$, where $\hat{p}_i(t)$ is the change of power usage and $e_{i\to j}^k(t)$ is the message passed from agent i to its neighbor j; Here $N(i) \subset N$ is the set of all neighbors of the agent i. To compute the actions, we apply a gradient ascent method on the local augmented utility function which is a combination of local utility function $U_i(p_i)$ and a penalty function defined on the estimation errors $e_i^k, k \in \{0,1,\cdots,r\}$.

ii) *Exchange messages*: pass the messages $e_{i\to j}^k(t)$ and $e_{j\to i}^k(t)$ to (resp. from) neighbors $j \in N(i)$.

iii) *Update states*: update the state variables $\left(p_i(t), \{e_i^k(t)\}_{k=0}^r\right)$ based on the action vector computed in the previous step and the received messages.

v) *Engage local controller*: actuate the power cap to the server.

We have formalized the above description in the form of the pseudo-code in Algorithm 1. See Appendix A for details of our derivations. To initialize the algorithm, we set the power usage $p_i(0) = P_i^{\min}$. Moreover, when $i \in CB_k$ for $k = 0,1,\cdots,r$ the initial estimate terms are given by

$$e_i^k(0) = \frac{1}{|\text{CB}_k|}\left(R_k - \sum_{j\in\text{CB}_k} P_j^{\min}\right), \tag{2}$$

and otherwise,

$$e_i^k(0) = 0. \tag{3}$$

The algorithm we proposed requires choosing free parameters including the step size ϵ and μ. These parameters must be selected based on the cluster size and convergence speed. For example, while choosing a small value for the step size ϵ guarantees that the solution of DPC is sufficiently close to the optimal solution of Eqs. (1a)-(1b), it results in a slower convergence rate and thus a longer runtime for DPC. For all the experiments in this paper, we set $\epsilon = 4$ and $\mu = .01$.

C. DPC Implementation Choices

We conclude this section by discussing a few aspects of DPC implementation on the cluster.

Choice of Network: DPC is a decentralized method and thus naturally utilizes datacenter's network infrastructure to establish connection between DPC agents. We analyze the impact of communication topology on DPC resource

Algorithm 1 DPC: DECENTRALIZED POWER CAPPING

Initialize $\mu > 0$ and a constant step size $\epsilon > 0$. Choose $p_i(0) = P_i^{\min}$ and $e_i^k(0)$ as in Eq. (2).
AT TIME ITERATION t SERVER i:
Compute the action $\hat{p}_i(t)$ according to,

$$\hat{p}_i(t) = \epsilon \frac{\partial U_i(p_i(t))}{\partial \hat{p}_i(t)} + \epsilon\mu \sum_{k \in M(i)} \max\{0, e_i^k(t)\};$$

Let $M(i) \subseteq \{0\} \cup \{1 \cdots r\}$ be the subset of CBs that the server i is subscribed to.
if $\hat{p}_i(t) + p_i(t) \leq P_i^{\min}$ **then** $\hat{p}_i(t) = P_i^{\min} - p_i(t)$
if $\hat{p}_i(t) + p_i(t) \geq P_i^{\max}$ **then** $\hat{p}_i(t) = P_i^{\max} - p_i(t)$
Compute the action $e_{i\to j}^k(t)$ according to,

$$e_{i\to j}^k(t) = \mu\epsilon \left(e_i^k(t-1) - e_j^k(t-1)\right)$$

Send $e_{i\to j}^k(t)$ and receive $e_{j\to i}^k(t)$ to (resp. from) all neighbors $j \in N(i)$.
Update the states $p_i^k(t+1)$, $e_i^k(t+1)$ for all $i \in N$, and $k \in \{0, 1, \cdots, r\}$ according to

$$p_i(t+1) = p_i(t) + \hat{p}_i(t),$$
$$e_i^k(t+1) = e_i^k(t) + \hat{p}_i(t)\mathbb{1}_{\{i \in CB_k\}} + \sum_{j \in N(i)} (e_{j\to i}^k(t) - e_{i\to j}^k(t)),$$

where $\mathbb{1}_{\{i \in CB_k\}} = 1$ if $i \in CB_k$ and $\mathbb{1}_{\{i \in CB_k\}} = 0$ otherwise.
Output: the power consumption p_i for all $i \in N$.

utilization and convergence. Specifically, we consider Watts-Strogatz model [12] to generate connectivity networks with small-world properties [12]. Small world networks are characterized by the property that they are clustered locally and has a small separation globally. That is, most servers can be reached with only few hops.

Watts-Strogatz model generates small-world networks with two structural features, namely clustering and average path length. These features are captured by two parameters: the mean degree k and a parameter β that interpolates between a lattice ($\beta = 0$) and a random graph ($\beta = 1$).

In this model, we fix $\beta = 0$ to obtain regular graphs and select various mean degrees k. Figure 3 illustrates the generated graphs from this model with $N = 16$ servers. For $k = 2$, we obtain the ring network, where each DPC agent is connected with two neighbors. For $k = 16$ we obtain a complete graph, where each agent is connected to all other agents in the given cluster.

For each underlying graph, we execute the decentralized DPC algorithm on our cluster where the total power cap is $R_0 = 2.6$kW and we assume to have two CBs, each supplying power to 8 of our 16 server cluster where $R_1 = R_2 = 1.6$kW. We measure various performance metrics such as 1) the network bandwidth (BW) utilization

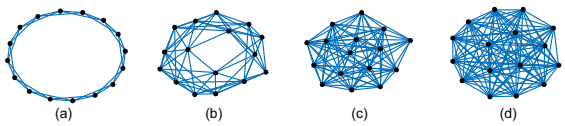

Figure 3: Generated graphs for DPCs agent where each vertex is a DPC agent and each edge indicates two agents that are neighbors. Graphs are generated from Watts-Strogatz model where each with $\beta = 0$ and mean degree (a) $k = 4$ (b) $k = 8$ (c) $k = 12$ and (d) $k = 16$.

Average degree k	# iter	node BW (kB/s)	cluster BW (kB/s)	communication (ms)	computation (ms)
16	547	1880.8	27644	331.0	1.9
14	550	1200.3	18021	314.0	2.2
12	571	1064.8	15990	295.1	2.4
10	599	934.7	14038	274.6	2.4
8	636	790.2	11875	233.0	2.3
6	796	739.5	11109	247.9	3.1
4	1108	685.6	10287	253.6	3.5
2	3234	993.0	14917	634.6	10.6

Table I: The effect of changing topologies on DPC.

per node, 2) the total cluster BW, 3) communication time per node, and 4) the computation time per node. The results of these measurements are summarized in Table I. From the table we observe that the number of iterations required for DPC to converge decreases as k increases. This observation is intuitive as for large k, the total number of edges increases and thus the consensus among servers can be reached faster. This in turn reduces the number of DPC's iterations required to converge to the optimal solution. However, increasing the edges increases the number of messages and network utilization at each iteration of the algorithm. As k increases, the network utilization decreases until the turning point $k = 4$ where the trend reverses. Based on this data, we use a network topology with $k = 4$ in all the following experiments for DPC.

Fault Tolerance: Server failure occurs frequently in large scale clusters. In the case of failure of a DPC agent, the agent's socket[1] will be closed and neighbors can send a query to the failed server and restart its DPC agent. If the DPC agent stops responding due to a more severe issue, two scenarios may occur: 1) the socket gets closed, 2) the communication time between agents exceeds a specified timeout threshold. In either case, the neighbors take this occurrence as a failure. After agents identify a failure in their neighborhood, they take the failed server out of the neighborhood list. Moreover, each agent continues the optimization process on a different connectivity graph that excludes the failed server. In the network generated by Watts-Strogatz model with $\beta = 0$, the connectivity of the network is maintained as long as the number of failed agents in the vicinity of a agent is smaller than the average degree k. After the failed server is fixed, its corresponding DPC agent communicates with its neighbors so that it will be

[1]The communication between agents is established using the standard Linux socket interface.

included again to their neighborhood list.

In Subsection II-B, we explained how DPC algorithm adopts if the total power cap changes. There are two scenarios under which a discrepancy between the total power cap and the algorithm beliefs of total power cap can occur, namely (i) when there is a change in the power cap, or (ii) when there is a server failure. In practice, this discrepancy can be injected as an error term to any randomly chosen server. The error propagates to all servers due to the consensus step in the decentralized algorithm. Note since the server is chosen arbitrarily, if it is not responsive, another random server can be chosen.

Computation/Communication Overhead: Any decentralized power management method consumes both communication and computational resources to operate. Therefore, we propose a 'sleeping' mechanism for each DPC agent to minimize the impact of DPC on the cluster's performance. Specifically, each DPC agent can execute as many as five thousand iterations of DPC per second. After convergence, DPC's speed can be reduced by sleeping in between iterations. This in turn reduces both network utilization and CPU cycles required for optimization. In our DPC implementation, the sleeping cycle is enforced by updating DPC at the rate of ten iterations per second. This rate is sufficient to effectively capture the changes in workload characteristics and the power cap. To minimize cache contentions and the overhead of scheduling, we always fix the core affinity of the DPC agent to one fixed core.

Comparison with other power capping schemes: We compare DPC to other existing power capping techniques. In particular, we implement three other methods:

Dynamo: Dynamo is the power management system used by Facebook datacenters [6]. It uses the same hierarchy as the power distribution network, where the lowest level of controllers, called leaf controllers, are associated with a group of servers. A leaf controller uses a heuristic high-bucket-first method to determine the amount of power that must be reduced from each server within the same priority group. In this framework, priorities of workloads are determined based on the performance degradation that they incur under power capping.

Compared to the DPC algorithm, Facebook's Dynamo has a larger latency for two reasons:

1) DPC agents continually read the power consumption and locally estimate power cap violations. Therefore, DPC agents respond to variations in the workload and power caps almost instantly. Notice that the time required to reach the consensus in our decentralized framework is much smaller than the timescale of variations in workload and power caps. In contrast, Dynamo scheme has a large latency due to its hierarchical design. In particular, the leaf controllers in Dynamo scheme compute the power caps and broadcast them to local agents of servers to actuate. However, to obtain stable power readings from servers and to compute the power caps, the leaf controllers must measure the power consumption only after they reach their steady states. As a result, if there is a sudden change in system variables after power consumption are measured, these changes are *not* taken into account in the power caps until the next power reading cycle.
2) By localizing the power cap computations at each server, the computation and actuation of power cap can be overlapped concurrently. In contrast, Dynamo requires stable power readings from servers, and thus power cap computation and actuation are carried in separate phases.

We also note that DPC improves the system throughput performance compared to heuristic methods such as Dynamo because DPC incorporates workload characteristics and their priorities to maximize the system throughput under specified power caps. Due to the fact that the power cap of each server is determined based on an optimization problem, the DPC framework reduces the negative impact of the power capping on the system throughput performance. In comparison, heuristic power capping methods, such as Dynamo, may adversely affect the system performance.

Uniform Power Allocation: In this scheme, the total power budget is evenly divided among active servers, regardless of their workloads' priorities.

Centralized Method: In this method, the optimization problem in Eqs. (1a)-(1b) is solved in a centralized manner on a single server [9]. In particular, the centralized coordinator aggregates servers' local information and solves the optimization problem in Eqs. (1a)-(1b) using off-the-shelf software packages such as `CVX` solver [13]. The centralized coordinator then broadcasts the local power consumption to servers. While a centralized method ensures the maximum throughput, it scales poorly for large clusters of thousands servers and cannot be employed in practice.

III. Evaluation

A. Experimental Setup

In this section, we report the experimental setup of DPC on our cluster. Our capping software is available online. [2]

Infrastructure: The experimental cluster consists of 16 Dell PowerEdge C1100 servers, where each server has two Xeon quad-core processors, 40 GB of memory, and a 10 Gbe network controller. All servers have Ubuntu 12.4 and are connected with top-of-the-rack Mellanox SX1012 switch. Performance counter values are collected from all servers using the `perfmon2`. We use SLURM as our cluster job scheduler [10]. Servers at full loads can consume about 220 Watts. We instrument each server with a power meter

[2]The DPC implementation codes can be found in our research lab github repository. https://github.com/scale-lab/DPC.

that reads the server's power consumption at 10 Hz.

Workloads: We use two types of workloads. For batch processing applications, we use the HPC Challenge (HPCC) and NAS parallel benchmark (NPB) suites to select our HPC workloads [14], [15]. In particular, in our experiments we focus on a mix of known CPU-bound workloads, e.g., `hpl` from HPCC and `ep` from NPB, and memory-bound workloads, e.g., `mg` from NPB and `RA` from HPCC, as they represent two ends of the workload spectrum. We use class size C for the workloads selected from NPB which approximately takes a minute to complete in the absence of power capping the servers. For workloads from HPCC, we select a matrix size with the same runtime. For latency-sensitive transactional workloads, we use MediaWiki 1.22.6 [16] which is a 14 GB copy of the English version of Wikipedia on a MySQL database. We use two clients to generate loads using Siege 3.1 [17].

Performance metric: We use retired jobs per hour as the throughput metric. To quantify the total throughput of our cluster of 16 servers, we calculate the total number of jobs retired per hour during the experiment. For total power, we sum the measured power of all 16 servers. For the web serving, we evaluate at the tail latency (99th percentile) as our performance metric.

Power controller (PC): To enforce the local power target at each servers, we implement a software feedback controller similar to Pack & Cap [18], where the controller adjusts the number of active cores according to the difference between the power target and the current power consumption. To avoid an oscillatory behavior around the power target, we consider a buffer area of two percent below the power target in which the controller remains idle. When the difference between the power target and current power consumption is positive and more than two percent, the controller increases the number of active cores. Similarly, a negative difference results in a decrease in the number of active cores. In our experiments, we engage the controller every 200 ms.

For a fair comparison between different methods, all the experiments and reported results in the paper, including those of Dynamo, are based on our core-affinity power controller. Nevertheless, both the RAPL controller in the Dynamo scheme [6] and the core-affinity power controller we use in this paper take approximately two seconds to stabilize. Therefore, we followe the three seconds sampling rule that is recommended in [6] for our implementation of Dynamo's leaf controllers.

Workload monitor (WM): The main task of WM is to determine the workload priorities and utility function for DPC algorithm to solve optimization in Eqs. (1a)-(1c). WM also monitors different resource utilization, performance counters, and workloads information. All the information gets logged with the time-stamp for further analysis.

rate (per minute)	1	2	3	6	12	20	30	60
overhead (%)	0.1	0.1	0.4	0.6	0.7	0.9	1.7	2.0

Table II: The effect of update rate of DPC on the overhead.

Overhead: As described earlier, any decentralized power management framework must be computationally inexpensive to minimize the performance degradation. To measure the overhead of DPC algorithm, we execute it without enforcing the power caps. We then calculate the overhead by comparing the results with the case that the cluster does not use any optimization algorithm and thus no resources is allocated to DPC.

The overhead of running the DPC agent on each node is determined by how often DPC needs to be run at full speed. DPC runs at full speed when power caps are needed to be re-calculated. Re-calculating the power caps are only required when the total power cap or the configuration of workloads changes. After re-calculations, DPC agents run in the background to monitor changes but use sleeping cycles to reduce overhead. To quantify the overhead of DPC algorithm, we varied the rate at which DPC needs to calculate power caps from once per minute to every second and recorded the system throughput for three different runs. Table II shows the average overhead of DPC as the function of re-calculation rate. In our experiments, the typical re-calculation rate is 1-2 per minute, which leads to negligible overhead.

We considered the following three main experiments.

1) **Prioritized Workloads**: In the first set of experiments we compare DPC and Dynamo focusing only on the workloads' priorities. We show the advantage of DPC over Facebook's Dynamo.
2) **Utility Maximization**: In the second set of experiments we consider the case when the throughput utility functions are known, and we show the performance improvement over heuristics that can be attained through solving the utility maximization problem.
3) **Scalability and Fault Tolerance**: In the third set of experiments, we evaluate the advantage of DPC compared to the centralized method and Dynamo in terms of scalability and fault-tolerance.

B. Prioritized Workloads

To demonstrate the performance gain achieved by minimizing the latency in DPC, we consider a scenario where two types of workloads are served in the cluster, namely a batch of HPC jobs running on half of the servers, and the other half are web servers. We take the latter as the cluster's primary workload. Because web queries are latency-sensitive, a coarse power capping on this type of services can result in violation of service level agreement (SLA). Therefore,

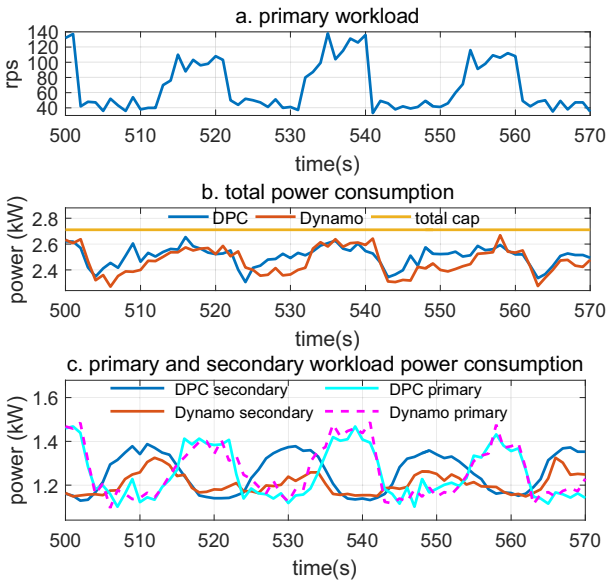

Figure 4: Detailed comparison between DPC and Dynamo for a minute of experiment. Panel (a): load on the web servers, Panel (b): total power consumption of the cluster, and Panel (c): power consumption of each sub-cluster running the primary (web servers) and secondary (batch jobs) workload for each method.

to meet the total power cap and the provisioned resource constraints, the power consumption of secondary workload must be capped. Accordingly, the objective is to satisfy the power constraints and maximize the throughput of the secondary workload.

To attain this objective, we prioritize the web queries in both Dynamo and DPC to avoid power capping. We generate approximately 40 to 120 queries per second to the web servers while a batch of HPC jobs are processed on the rest of the clusters as the secondary workload. In the implementation of Dynamo, we assume that all servers and two CBs are assigned to a single leaf controller.

The experiment is implemented for an hour duration, where we fix the total power cap to $2.7kW$ and assume all 16 servers are protected by two CBs with power capacity of $R_1 = R_2 = 1.6kW$. Figure 4.(a) shows the load on the web servers as the primary workload. From Figure 4.(b), we observe that both methods successfully cap the total power consumption. Moreover, from Figure 4.(c), we observe the variations in the primary workload's power consumption due to changes in the number of submitted queries. Local DPC agents are able to estimate cap violations in a fast decentralized way, and thus they provide a faster reaction time to the changes in the power cap profile of the primary workload.

DPC agents are running on all of the 16 servers and power is divided between the primary and secondary workloads. The primary workload always has the maximum number of active cores due to a higher priority. As explained earlier, power controller (PC) constantly monitors the power target

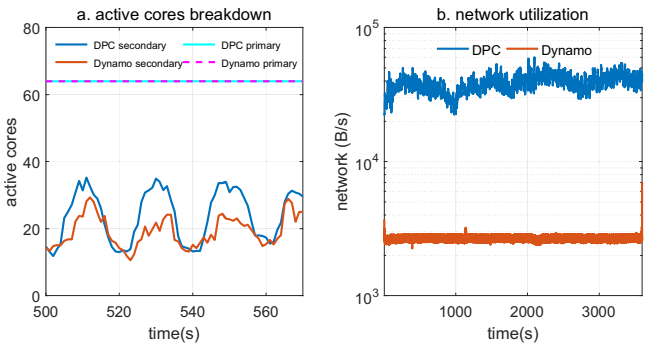

Figure 5: Panel (a): DPC and Dynamo's active number of cores for each type of workload in a minute of experiment. Panel (b): Network utilization of the Dynamo's leaf controller and average server for DPC through the experiment.

and power consumption and sets $P^{\max}$ accordingly. When the primary workload power consumption decreases, PC updates $P^{\max}$ on each node in the DPC algorithm. The affected nodes in DPC algorithm updates their power caps and using the messages communicated between the agents, power passes from the primary to the secondary workload. When the primary workload's power consumption increases, again PC updates $P^{\max}$ and because of higher priorities, power passes from the secondary to the primary workload.

Due to different structural design, DPC and Dynamo have different response times. As mentioned earlier, Dynamo must wait for the local power controllers to stabilize to compute the power caps, and actuation delay of the controllers determine how fast Dynamo can sample. In contrast, DPC agents estimate power cap violations independently in a decentralized way. In addition, DPC overlaps power cap calculation with the actuation because both are done locally. This fast reaction time in turn results in a more efficient power allocation to the secondary workload.

Figure 5.(a) shows the active number of cores for each workload. In both DPC and Dynamo methods, all the available cores are allocated on the eight servers that are processing the primary workload. However, due to a higher power target, DPC allocates more cores to the secondary workload. During an hour of experiment, DPC provides 16% improvement in the secondary job's throughput compared to Dynamo. The response-time tail latency (99th percentile) of the primary workload is unaffected in both methods since both methods allocate maximum number of active cores to avoid any performance degradation to the latency sensitive workload; see Figure 5.(a). From Figure 5.(b) we observe that the network utilization of Dynamo's leaf controller and the average network utilization of all nodes for DPC through the experiment. Although DPC has a higher network utilization compared to Dynamo, at its peak network utilization, DPC consumes only 0.02% of the available bandwidth of a 10Gb Ethernet network controller. Thus the DPC network overhead is negligible.

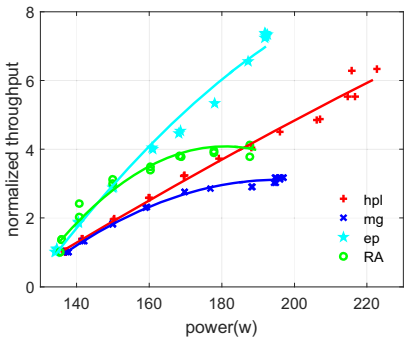

Figure 6: Modeled (lines) and observed (markers) normalized throughput as the function of power consumption.

C. Utility Function Experiments

In this section, we consider the case where the utility functions are known. We compare solutions like DPC and the centralized to other heuristic methods such as uniform and Dynamo. We consider the behavior of these methods in two cases of a dynamic power caps and a dynamic load.

Utility functions determine the relationship between the throughput and power consumption using empirical data. There are many studies on characterizing this relationship [19], [20], [9]. We adapt a quadratic form to characterize the relationship between throughput and power consumption. We assume that all the workloads and their corresponding utility functions are known a prior. The workload monitor (WM) receives the current workload information from SLURM daemon (`slurmd`) running on each node and selects the correct utility function from a bank of known quadratic functions. `Slurmd` is the daemon of SLURM which monitors current jobs on the server and accepts, launches, and terminates running jobs upon request.

Figure 6 shows the normalized throughput function of workloads selected from the HPCC and NPB benchmarks, where we fix the server power cap at various values and measure the average throughput for each workload. We also observe from Figure 6 that for all practical workloads in this paper, the throughput is a concave function of the server power. Throughout this experiment we assume DPC and centralized method only uses the throughput/power relationship as utility function. As Figure 6 shows, the throughput of CPU-bound workloads (`hpl` and `ep`) are affected more by power capping. To use the only knob that Dynamo offers to do workload-aware power capping, we assumed higher priority for CPU-bound workloads. Again for Dynamo we assume that all servers and two CBs are assigned to a single leaf controller.

1. Dynamic Caps: We now consider a scenario where the total power cap must be reduced from 2.8 to 2.5 kW due to a failure in the cooling unit. Throughout the experiment, our 16-node cluster is fully utilized with a mix of different workloads. We again assume all the 16 servers are protected by two CBs where each can handle 1.6 kW. Figure 7.(a) shows how each method reacts to changes in the total power cap and the mix of workload as shown in Figure 7.(b). While all the power capping methods can successfully cap the total power under the given power cap, DPC and the centralized methods consistently provide 8% higher job throughput over uniform. Further, Dynamo can only improve the jobs throughput by 5.2% compared to the uniform power capping. The corresponding network utilization is depicted in Figure 7.(c) with a logarithm scale in bytes per second. The centralized approach, Dynamo, and uniform power capping methods make a negligible use of the network infrastructure. DPC engages in full capacity only when the power cap or the mixture of workloads change which can be observed as the spikes in Figure 7.(c). We also observe that DPC slows down after convergence, which in turn reduces the communication rate of DPC and minimizes the overhead. Although DPC has the highest network utilization among all the considered methods due to its decentralized design, it uses only 0.1% of the available network bandwidth of each server's 10Gbe network at its peak. Hence, from a practical point of view, the network overhead in DPC is negligible.

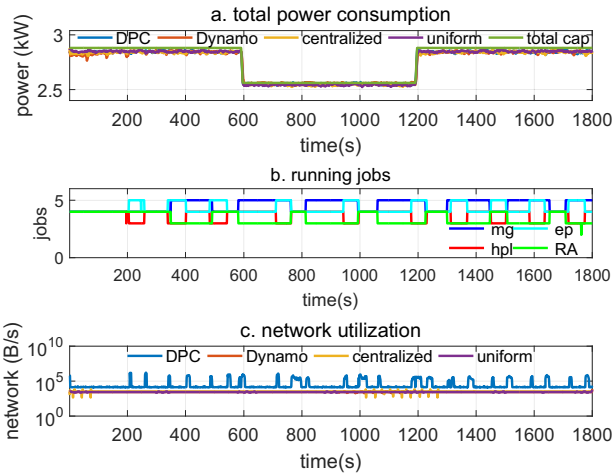

Figure 7: Comparison between DPC, Dynamo, centralized, and a uniform power capping under a dynamic power cap.

2. Dynamic Load : In this experiment, we evaluate each method under dynamic workloads. In this experiment, the total power cap is set to 2.8 kW and all the 16 servers are protected by two CBs each of 1.6 kW capacity. The first batch of workloads is submitted to the cluster, where five jobs are running on the cluster at the beginning of the experiment. We choose the workloads to be a mix of memory and CPU-bounds applications. In the beginning of this experiment, the jobs occupy five servers in the cluster and the remaining servers are idle. Approximately nine minutes in the experiment, the second batch of jobs (a mix of memory and CPU-bounds applications) are submitted to the cluster such that all the servers are fully utilized. We demonstrate the total power consumption and corresponding job status in Figure 8. When only a few workloads are running on the cluster, a large amount of power is available

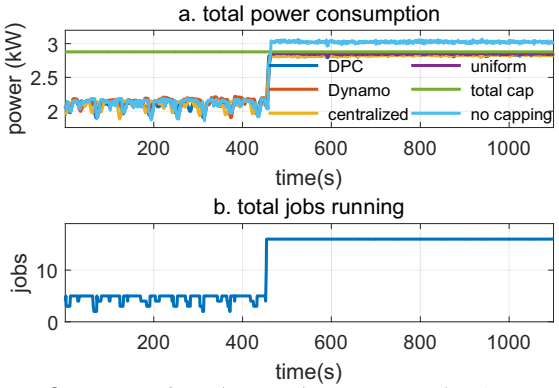

Figure 8: Power and number of jobs running on the cluster in the dynamic-load experiment.

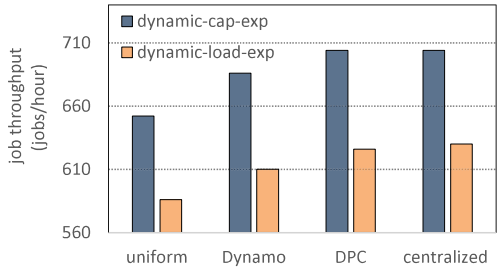

Figure 9: job throughput of the two experiments.

to be allocated to utilized servers. However, when more jobs are submitted under a restrictive power cap, the power cap of each server needs to be computed to maximize the cluster throughput performance. The centralized and decentralized methods find the optimal power caps for each server based on the workload characteristics. Therefore, these two methods provide a better job throughput compared to Dynamo and uniform power capping. More precisely, the centralized and DPC outperform the uniform power capping by 7%. In comparison, Dynamo only provides 4% improvement over uniform power capping. Similar to the previous experiment, DPC has the most network utilization which at its peak is about 0.1% of the available bandwidth for each server.

Figure 9 shows the jobs throughput performance in the two experiments with a dynamic power cap and dynamic workloads. We observe that DPC outperforms the uniform power allocation and Dynamo schemes by 7% and by 3%, respectively. To obtain these results, experiments are repeated three times, which we observed 0.2% standard deviation between three trials.

D. Scalability and Fault Tolerance

1. Scalability: In this section, we compare DPC with the centralized and Dynamo methods in terms of the scalibility. The *actuation latency* consists of three parts, (i) the computation time, (ii) the communication time, and (iii) the controller actuation time. In the following experiments, we are interested in the computation and communication time, *i.e.*, the time it takes for a power capping method to compute the power caps and set it as the power target for the power controller. We exclude the actuation time of the power controller in the reported results as it adds the same amount of delay to all methods we consider. Based on the data from measurements on the real-world 16-server cluster, we compute the latency of a cluster with 4096 servers.

To estimate the computation and communication time of DPC for large number of nodes, we first used Matlab simulations to compute the number of iterations required for DPC convergence in large clusters. The computation time of each iteration is measured from our cluster. We use the utility functions of the workloads from our cluster. To consider the randomness effect, we average the DPC convergence for 10 different trials. Communication time of each iteration is the time needed to send and receive messages from neighbors and we measure it again from our cluster. Then, we estimate the computation and communication time of DPC by multiplying the computation and communication time of each iteration and the number of iterations.

The computation time of the centralized method is the runtime of the `CVX` solver which we measure from our system. To measure the communication time of the centralized method, we measure the time needed for sending and receiving messages from a centralized coordinator, where the local power cap of each server is computed, to the servers. As Dynamo reported [6], Dynamo's leaf controller that can handle up to a thousand nodes has the pulling cycle of 3 seconds. For more than a thousands nodes, upper-level controllers are needed which have pulling cycle of 9 seconds.

The latency of different power capping methods are shown in Figure 10 in logarithm scale. The latency of centralized method grows cubically, as centralized method uses `CVX` quadratic programming solver that suffers from computational complexity of approximately $O(n^3)$ [21]. Dynamo's latency is the function of a hierarchy depth and DPC's latency is the function convergence iteration and as our results shows it grows linearly.

Dynamo has smaller actuation time compared to the centralized method; however, it lacks performance efficiency as we see throughout experiments. DPC is both the fastest solution and delivers the optimal performance. Centralized method solves the optimization like DPC but it cannot be used in practice because of the large *actuation latency*.

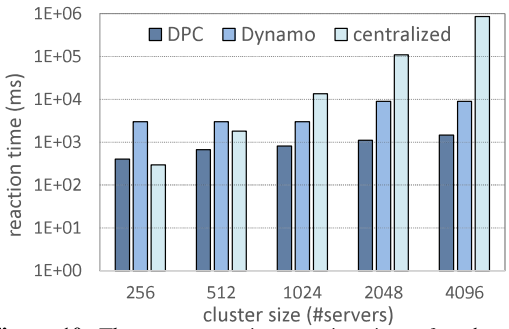

Figure 10: The power capping reaction time of each method.

DPC also has a negligible network overhead for larger clusters. The messages between each pair of servers at each iteration of DPC has the same length. Because the number of neighbors is fixed, network utilization is only a function of number of convergence iterations. Again we used Matlab to compute the number of iterations required for DPC convergence in larger clusters. To calculate the network utilization for larger clusters, we measured network utilization per iteration from our cluster and multiply it by the number of iterations needed for larger cluster to converge. Although DPC utilize the network more than the centralized method and Dynamo, it only occupies upto less than 1% of the available network bandwidth. Thus the network overhead is negligible.

2. Fault Tolerance: Another advantage of the decentralized method is that it does not have a single point of failure by its nature. In this paper, we focus on server failures and do not consider the correlated failures due to network switch failures and power outages. A failed node is assumed to be taken out of the cluster and consumes no power from the power cap throughout the experiment.

We choose $k = 4$ for the topology of DPC, where each node is connected to 4 other nodes. In this case, a node will still remain connected, even if 3 of its neighbors fail. In our DPC agent implementation, when one node fails, its neighbors will notice the failed node due to lack of response and remove the node out of their neighbors list. The optimization will run with the active nodes since they are still connected. The difference between the total power target and actual power that the cluster is consuming is injected as an error to one of the node estimation as explained in Subsection II-C. One of the nice features of DPC is that the error estimation can be injected to any of the active nodes. This avoids a single point of failure and reinforces our fully decentralized claim for DPC.

In this experiment, the total power cap was fixed at 2.8 kW and three neighbor servers fail approximately at 30th, 40th and 50th seconds of the experiment. Our power monitor infrastructure monitors the total power consumption of the cluster and power cap. In case of an error, it pings servers and injects the error to one of the working servers for DPC. Figure 11.(a) shows the power consumption of DPC methods during the nodes failure experiment. Figure 11.(b) shows the servers network utilization and it shows DPC re-calculates the power caps after each server failure to use the power budget of the failed server for other running servers. After each failure, power consumption of the cluster undershoots because the failed servers are taking out of the cluster. DPC compensates by allocating the failed server's budget to other active servers. After the third failure, all remaining thirteen servers are uncapped since the total power cap is above what is consumed by the cluster.

IV. Related Work

The problem of coordinated power capping in large scale computing clusters and its effect on different classes of workloads has been studied extensively [6], [3], [5], [1], [7]. Among different types of workloads, however, power capping on online data intensive services (OLDI) has received a great deal of attention [22], [5]. This is due to the fact that OLDI services exhibit a wide dynamic load range and thus would benefit significantly from 'energy proportionality' wherein servers consume power in proportion to their loads.

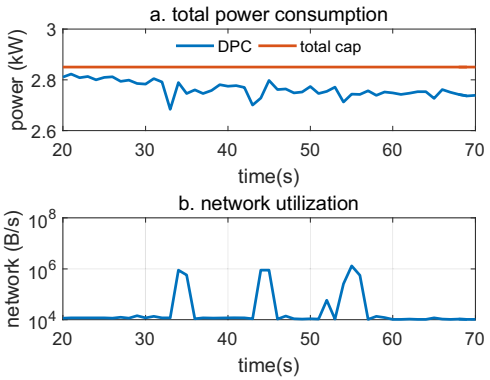

Figure 11: Total power consumption of the cluster and the average network utilization in the case of servers failure.

An important issue in power capping techniques is to select an appropriate power cap in order to maximize the number of hosted servers in a datacenter [23]. A common practice is to ensure that the peak power never exceeds the branch circuits capacity as it causes tripping in circuit breaker (CB). However, this approach is overly conservative. By carefully analyzing the tripping characteristics of a typical CB, the system performance can be optimized aggressively through power over-subscription without the risk of tripping CB. It must be mentioned nevertheless that over-subscribing power at the rack level is unsafe, as noted by Fan. *et al.* [24], given that large Internet services are capable of driving hundreds of servers to high-activity levels simultaneously.

The main challenge in implementing coordinated power capping is to control the latency. A detailed examination of latency in hierarchical models [1] shows that even a small *actuation* latency, i.e., the latency in control knobs, can cause instability in hierarchical, feedback-loop power capping models. The main difficulty is due to the fact that when feedback loops operate on multiple layers in the hierarchy, stability conditions demand that the lower layer control loop must converge before an upper layer loop can begin the next iteration. Despite this observation, recently a hierarchical structure for power capping, dubbed as *Dynamo*, has been proposed [6] and implemented on Facebook's datacenter which we compared our work against thoroughly.

V. Conclusions

Power capping techniques provide a better efficiency in datacenters by limiting the aggregate power of servers to the branch circuit capacity. As a result, power capping is deemed as an important part of modern datacenters. In this

paper, we proposed DPC, a fast power capping method that maximizes the throughput of computer clusters in a fully decentralized manner. In contrast to most existing power capping solutions that rely on simplistic heuristics, we used distributed optimization techniques that allow each server to compute its power cap locally. We showed that DPC provides a faster power capping method compared to Facebook's Dynamo heuristic. It also improves the system throughput by 16% compared to Dynamo while using only 0.02% of the available network bandwidth. The power capping framework we proposed also takes into account workloads priority by augmenting the throughput functions of each workload with a multiplicative factor. Additionally, we showed that when the utility function at each server is known, DPC provides the optimal performance in terms of jobs per hour metric.

ACKNOWLEDGMENT

The research of Reza Azimi, Xin Zhan, and Sherief Reda is partially supported by NSF grants 1305148 and 1438958. The research of Masoud Badiei and Na Li is supported by IBM PhD Fellowship and NSF CAREER grant 1553407.

REFERENCES

[1] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," in *Sustainable Computing: Informatics and Systems*, vol. 3, no. 3. Elsevier, 2013, pp. 183–193.

[2] R. Azimi, X. Zhan, and S. Reda, "Thermal-aware layout planning for heterogeneous datacenters," in *ACM Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, 2014, pp. 245–250.

[3] H. Chen, C. Hankendi, M. C. Caramanis, and A. K. Coskun, "Dynamic server power capping for enabling data center participation in power markets," in *IEEE Proceedings of the International Conference on Computer-Aided Design*, 2013, pp. 122–129.

[4] T. Kidd, "Power management states: P-states, c-states, and package c-states," *Intel Development Resources*, 2014.

[5] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *IEEE SIGARCH Computer Architecture News*, vol. 42, no. 3, 2014, pp. 301–312.

[6] 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 *IEEE International Symposium on Computer Architecture (ISCA)*, 2016, pp. 469–480.

[7] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," in *ACM ASPLOS*, 2016, pp. 545–559.

[8] M. Badiei, X. Zhan, R. Azimi, S. Reda, and N. Li, "Diba: Distributed power budget allocation for large-scale computing clusters," in *IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, pp. 70–79.

[9] X. Zhan and S. Reda, "Power budgeting techniques for data centers," in *IEEE Transactions on Computers*, vol. 64, no. 8, 2015, pp. 2267–2278.

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[11] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *IEEE Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 807–818.

[12] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," in *nature*, vol. 393, no. 6684. Nature Publishing Group, 1998, pp. 440–442.

[13] M. Grant, S. Boyd, and Y. Ye, "CVX: Matlab software for disciplined convex programming," 2008.

[14] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS parallel benchmarks," in *International Journal of High Performance Computing Applications*, vol. 5, no. 3. SAGE Publications, 1991, pp. 63–73.

[15] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC challenge (HPCC) benchmark suite," in *conference on Supercomputing*. Citeseer, 2006, p. 213.

[16] MediaWiki, "Mediawiki — mediawiki, the free wiki engine," 2015. [Online]. Available: https://www.mediawiki.org/w/index.php?title=MediaWiki&oldid=1446376

[17] J. Fulmer, "Siege http regression testing and benchmarking utility," *URL http://www. joedog. org/JoeDog/Siege*.

[18] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: adaptive DVFS and thread packing under power caps," in *ACM International Symposium on Microarchitecture*, 2011, pp. 175–185.

[19] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, 2009, pp. 157–168.

[20] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. De Supinski, "Practical performance prediction under dynamic voltage frequency scaling," in *IEEE International Green Computing Conference and Workshops (IGCC)*, 2011, pp. 1–8.

[21] M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe, "Interior-point methods for large-scale cone programming," in *Optimization for machine learning*. MIT, 2011, pp. 55–83.

[22] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *IEEE International Symposium on Computer Architecture (ISCA),*, 2011, pp. 319–330.

[23] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers," in *ACM International Conference on Autonomic Computing*, 2011, pp. 21–30.

[24] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, 2007, pp. 13–23.

[25] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," in *Journal of Parallel and Distributed Computing*, vol. 67, no. 1. Elsevier, 2007, pp. 33–46.

APPENDIX

In this appendix, we provide more detail for the derivations in Subsection II-B. We consider a network between servers which we abstract by the graph $G = (N, E)$, where $E \subseteq N \times N$ denotes the set of edges, i.e., servers i and j are connected to each other iff $(i, j) \in E$. Let $N(i) \subseteq N$ be the set of all servers that are connected to server i on the network, i.e., $j \in N(i)$ iff $(i, j) \in E$. In a general form, we define the augmented utility function of each server $i = 1, 2, \cdots, N$ as follows:

$$U_i'\left(p_i(t), \{e_j^k(t)\}_{j\in N(i)\cup\{i\}}\right) := U_i(p_i(t)) - \mu V_i\left(\{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}}\right), \quad (4)$$

where $V_i : \mathbb{R}^{(r+1)(|N(i)|+1)} \to \mathbb{R}_{\geq 0}$ is a penalty function that takes the estimates $e_j^k(t)$ of all constraints and all neighbors $N(i)$ as well as server i as the argument, and outputs a non-negative real value. Here, $\mu \geq 0$ is a tunable parameter that determines the significance of the penalty function. The utility function in Eq. (4) consists of two parts. The first part $U_i(p_i(t))$ is associated with the throughput at server i, while the second part $V_i\left(\{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}}\right)$ is included to reduce the error terms $e_i^k(t)$ to zero. By choosing a proper penalty function $V_i(\cdot)$ and the parameter μ, we can define a local procedure for each server $i \in N$ such that $p_i(t)$ converges to the optimal solution of the formulated optimization problem in Eqs. (1a)-(1c).

Implicit in the structure of the penalty function V_i in Eq. (4) is some form of communication among servers. In particular, servers exchange information about their estimates $e_i^k(t)$ to create consensus. Let $\{e_{i\to j}^k, e_{j\to i}^k\}_{j\in N(i)}$ denotes such messages sent and received from the i-th server to its neighbors $j \in N(i)$. Based on the received messages, the belief of the i-th server is updated in the form of a state-space model, wherein $(p_i(t), \{e_i^k(t)\}^{k\in\{0,1,\cdots,r\}})$ are states. Further, for all $k = 0, 1, \cdots, r$ and $i = 1, 2, \cdots, n$ we have

$$p_i(t+1) = p_i(t) + \hat{p}_i(t), \quad (5a)$$

$$e_i^k(t+1) = e_i^k(t) + \hat{p}_i(t)\mathbb{1}_{\{i\in CB_k\}} + \sum_{j\in N(i)} (e_{j\to i}^k(t) - e_{i\to j}^k(t)), \quad (5b)$$

where $\mathbb{1}_{\{\cdot\}}$ is an indicator function that takes the value of one if $i \in CB_k$ and zero otherwise. We note that $\hat{p}_i(t)$ in the first equation and $\hat{p}_i(t)\mathbb{1}_{\{i\in CB_k\}} + \sum_{j\in N(i)}(e_{i\to j}^k(t) - e_{j\to i}^k(t))$ in the second equation are the inputs of the linear state-space system in Eqs. (5a)-(5b). In this formulation, $\hat{p}_i(t-1)$ can also be sought as the amount of change in the power usage limits of each server i at iteration t. We also note that in the machinery characterized in Eqs. (5a)-(5b), the messages $\{e_{i\to j}^k(t), e_{j\to i}^k(t)\}_{j\in N(i)}(t)$ propagate the belief $e_i^k(t)$ of server i to its neighbors $j \in N(i)$.

By putting Eq. (4) and Eqs. (5a)-(5b) together, we arrive at the following local convex optimization problem at the i-th server,

$$\max_{\hat{p}_i(t), \{e_{i\to j}^k(t)\}_{j\in N(i)}^{k\in\{0,1,\cdots,r\}}} U_i'\left(p_i(t), \{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}}\right) \quad (6a)$$

subject to : (5a) − (5b)

$$P_i^{\min} \leq p_i(t) \leq P_i^{\max}, \quad \forall i \in N. \quad (6b)$$

It now remains to determine the form of the penalty function V_i in Eq. (4) as well as the values of power change $\hat{p}_i(t)$ and messages $\{e_{i\to j}(t)\}_{j\in N(i)}^{k\in\{0,1,\cdots,r\}}$ at each algorithm iteration $t \in [T]$. By our construction in Eq. (4), we observe that an admissible penalty function must satisfy

$$V_i\left(\{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}}\right) = 0,$$

if for all $j \in N(i) \cup \{i\}$ and $k \in \{0, 1, \cdots, r\}$ we have $e_j^k(t) \leq 0$. That is, when the beliefs of all servers in $N(i)\cup\{i\}$ indicates that the power capping constraints in Eq. (1b) are satisfied, the penalty function must vanish. Although such a choice of penalty function is not unique, we consider in this paper an admissible function given by

$$V_i\left(\{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}}\right) := \frac{1}{2}\sum_{k=0}^{r}\sum_{j\in N(i)\cup\{i\}}\left[\max\{0, e_j^k(t)\}\right]^2.$$

To determine the values of power change $\hat{p}_i(t)$ and messages $\{e_{i\to j}^k(t)\}_{j\in N(i)}^{k\in\{0,1,\cdots,r\}}$, we use a projected gradient ascent direction for $U_i(p_i(t), \{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}})$. Let $M(i) \subseteq \{0, 1, \cdots, r\}$ be the subset of CBs that the server i is subscribed to. We compute:

$$\tilde{p}_i(t) = \epsilon \cdot \frac{\partial U_i'(p_i(t), \{e_j^k(t)\}_{j\in N(i)\cup\{i\}}^{k\in\{0,1,\cdots,r\}})}{\partial \hat{p}_i}\Big|_{\hat{p}_i=0} = \epsilon\frac{dU_i(p_i(t))}{d\hat{p}_i(t)} - \epsilon\mu\sum_{k\in M(i)}\max\{0, e_i^k(t)\}, \quad (7)$$

where $\epsilon > 0$ is a time-independent step size in the gradient ascent method. The resulting action is obtained by the projection $\hat{p}_i^t = \mathbf{Pr}_{\mathcal{P}(p_i(t))}[\tilde{p}_i(t)]$ onto the set $\mathcal{P}(p_i(t)) := \{\tilde{p} \in \mathbb{R}_{\geq 0} : p_i(t) + \tilde{p} \in [P_i^{\min}, P_i^{\max}]\}$. This projection step guarantees that the new updated power consumption profile given by $p_i(t+1) = p_i(t) + \hat{p}_i(t)$ respects the power restriction of the server, i.e., $P_i^{\min} \leq p_i(t+1) \leq P_i^{\max}$. Simplifying this projection results in the thresholding step (Step 4) of Algorithm 1.

Similarly, for all $j \in N(i)$ we choose

$$e_{i\to j}^k(t) = \mu\epsilon\left(e_i^k(t) - e_j^k(t)\right). \quad (8)$$

Note that based on the structure of the messages in Eq. (8), we obtain the following state-space update for the estimates from Eq. (5b),

$$e_i^k(t) = (1 - 2\mu\epsilon)e_i^k(t) + 2\mu\epsilon\sum_{j\in N(i)} e_j^k(t) + \hat{p}_i(t)\mathbb{1}_{\{i\in CB_k\}},$$

which is a standard step for achieving consensus in distributed averaging algorithms, e.g. see [25].