RC25680 (AUS1711-001) March 28, 2018
Computer Science

IBM Research Report

CapMaestro: Exploiting Power Redundancy, Data Center-Wide Priorities, and Stranded Power for Boosting Data Center Performance

Yang Li[1], Charles Lefurgy[1], Karthick Rajamani[1], Malcolm Allen-Ware[2], Guillermo J. Silva[1], Daniel D. Heimsoth[3], Saugata Ghose[4], Onur Mutlu[5]

[1]IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758
USA

[2]IBM Tucson
9000 S Rita Road
Tucson, AZ 85744
USA

[3]IBM Raleigh
4205 South Miami Boulevard
Durham, NC 2770
USA

[4]Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue
Pittsburgh, PA 15213
USA

[5]ETH Zurich
Rämistrasse 101
8092 Zürich
Switzerland

CapMaestro: Exploiting Power Redundancy, Data Center-Wide Priorities, and Stranded Power for Boosting Data Center Performance

Yang Li[+×], Charles Lefurgy[+], Karthick Rajamani[+], Malcolm Allen-Ware[+], Guillermo J. Silva[+], Daniel D. Heimsoth[+], Saugata Ghose[×], and Onur Mutlu[×^]

[+]*IBM* [×]*Carnegie Mellon Univeristy* [^]*ETH Zürich*

***ABSTRACT*—**Power infrastructure is a critical component of cloud and HPC data centers, and costs as much as tens of millions of US dollars for a large data center. The infrastructure must be highly reliable and must tolerate load variation, which traditionally requires significant redundancy and overprovisioning. This redundant and overprovisioned capacity is significantly underutilized during normal operation (typical load, non-failure mode). Power capping reduces underutilization by adding more servers to the existing power infrastructure, and throttling power consumption in the infrequent cases where demand exceeds the provisioned capacity. However, state-of-the-art power capping solutions are (1) not practical for the properties of real-world redundant power infrastructure in highly-available data centers, and (2) oblivious to differing priorities of workloads across the entire data center when power consumption needs to be throttled. As a result, these solutions are inefficient and can even be unsafe.

In this work, we present *CapMaestro*, a new power management architecture for cloud and HPC data centers. CapMaestro has three major new contributions. First, CapMaestro is designed to work with multiple power feeds, and exploits server power capping to independently cap the load on each feed of a server. It exploits the underutilized redundant power infrastructure commonly employed in data centers to safely accommodate a much greater number of servers. Second, CapMaestro uses a scalable, distributed, multi-level power capping approach, which accounts for power capacity at each level of power distribution hierarchy. It is *global* priority-aware, ensuring that no high-priority server *anywhere in the data center* is throttled before *all* lower-priority servers in the data center are throttled, as long as this can be achieved safely. Third, CapMaestro exploits stranded power (i.e., power budgets that are not utilized) in redundant power infrastructure to boost the performance of applications running in the data center. We deploy CapMaestro in our cloud data center control plane to demonstrate its effectiveness on real-world machines. We then simulate a data center with thousands of servers using published load distribution data, and demonstrate that CapMaestro safely increases the number of servers under the existing power infrastructure by 50%.

I. INTRODUCTION

Power infrastructure is a critical part of data centers, both in terms of its cost (tens of millions of US dollars) and its impact on availability. For highly-available data centers, the power distribution infrastructure often relies on redundancy at each level of the power distribution hierarchy to ensure reliable power delivery, spanning from multiple power supplies within individual servers up to multiple utility feeds into the data center, as shown in Figure 1. This redundancy within the power distribution infrastructure, referred to as N+N power delivery design, ensures continued availability of the *full* power demand in the event of the failure of half the power devices at each level. If a data center's total power consumption exhibits wide variations, or cannot be well anticipated (such as in a public cloud), each side of the power infrastructure (i.e., A-side/B-side power feed in Figure 1) is conventionally overprovisioned to meet the *maximum possible* power consumption (including uncertainty) and avoid risking failures. During normal operation, the data center's total power consumption may be much lower than its peak power, and maximum power loads rarely occur, resulting in underutilized infrastructure. Additionally, the extra power capacity from redundant feeds is not utilized during normal operation.

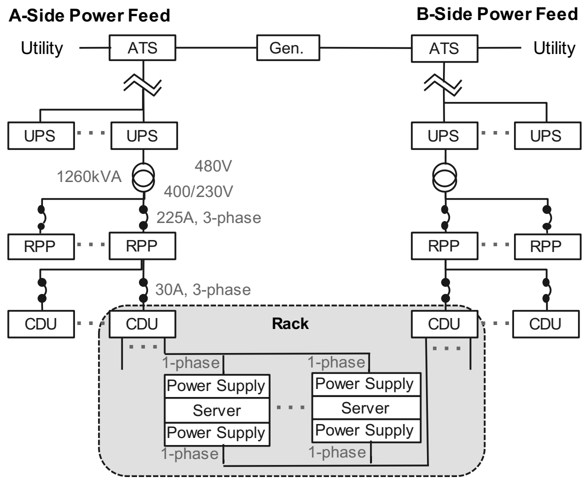


Figure 1. Example power delivery layout in a data center.

Power capping, introduced a decade ago [1], throttles the amount of power consumed by servers. It can be used to reduce the amount of power load applied on the power infrastructure during periods of peak demand. With the gradual, industry-wide adoption of server power capping, today's data centers have the means to shape power consumption in real time, so that potential *excursions* above the power capacity of the infrastructure can be avoided [3-5, 22-24, 28-31, 39, 40]. This lets us more efficiently utilize the existing power infrastructure by increasing server capacity (i.e., housing more servers) in the data center based on their *typical* load and/or quality-of-service (QoS), rather than overprovisioning the power infrastructure to meet infrequently-occurring maximum possible loads.

Unfortunately, existing power capping solutions face three serious challenges, which prevent them from guaranteeing safety and effectively increasing server capacity for highly-available data centers. First, in highly-available data centers, a server draws power from multiple power supplies, each connected to a different power feed. We find that there is typically an imbalance in the power drawn from each supply (see Section III.A). In such cases, power capping must ensure that the power consumed by *each* power supply does not exceed the supply's power budget. State-of-the-art controllers cannot do this today, and enforce only a *single* combined budget across all power supplies of a server. These controllers cannot ensure that the budgets for individual power supplies are respected, which can cause one of the power feeds to become overloaded, leading to tripped circuit breakers and power loss on the overloaded feed.

Second, when existing power capping techniques are invoked during periods of high demand, they are oblivious to the importance of each workload *globally* across the entire data center (at best, they may be aware of the workload importance within a limited *local* group of servers, e.g., [5]). As a result, existing techniques may inadvertently cap a critical (i.e., high-priority) workload in one group of servers unnecessarily, even though lower-priority workloads in another group remain uncapped. Without capturing priority across the entire data center, a power capping solution may not redistribute power budgets to critical workloads, unnecessarily hurting their performance.

Third, existing power capping techniques cannot guarantee the power budgets allocated to different power supplies of a server exactly match the power load sharing between these supplies (see Section III.C and VI.C). As a result, some power supply budget(s) may not be fully utilized, leaving the unutilized budget becoming *stranded power*.

To address these challenges, we propose *CapMaestro*, a new power capping architecture for cloud and HPC data centers that can control an arbitrary, multi-level, redundant power infrastructure. CapMaestro unlocks the unused power capacity of a highly-available data center, which is provisioned for peak power and redundancy, to power more servers under a fixed power budget, while still protecting every level of the power infrastructure from overload. Our architecture performs efficient *global* priority-aware budget allocation, and includes an optional optimization that adjusts the budget to redistribute stranded power throughout the data center. CapMaestro performs budget allocation using a distributed algorithm with multiple coordinated power controllers, which enables fault-tolerant and scalable capping, and reduces communication and dependencies between the controllers.

We implement CapMaestro as a scalable, managed control plane service within our cloud data center and demonstrate its effectiveness for capping multi-feed power infrastructure with *global* priorities. To evaluate its effectiveness for increasing data center server capacity, we study the impact of CapMaestro on the power infrastructure of a data center using a detailed simulation model. Based on the load data for a Google data center [27], we find that for a typical shared data center where we designate 30% of the servers as high-priority, CapMaestro can enable the data center to support 50% more servers than if power capping were not employed, and can support 20% more servers than a state-of-the-art power capping controller [5] that we modify to support multiple power feeds. This increased server capacity negligibly impacts the performance of high-priority workloads even during a worst-case power emergency, and negligibly impacts all workloads during normal conditions.

Importantly, *our solution is designed to be applicable with minimal changes to existing data centers.* For example, our design integrates with conventional baseboard management controller (BMC) based server infrastructure management [33]. Our solution can also handle (1) load imbalances on different power feeds that can otherwise lead to stranded power, and (2) a shared power distribution hierarchy where some of the infrastructure does not have power capping technology.

We make the following key contributions in this work:

- CapMaestro is a new architecture for data center power management with distributed, coordinated power controllers that enforce power limits at all levels of the power infrastructure. We propose and implement a global priority-aware algorithm *for the first time*.
- We design the *first* closed-loop, feedback power controller for servers with multiple power supplies. This allows us to manage the power consumption at each supply in response to the unique power loads and limits seen at each upstream circuit breaker. This enables CapMaestro to manage multi-feed power hierarchies in the data center.
- We provide a mechanism that can reduce *stranded power* within the power infrastructure, by shifting the stranded power to servers that are currently throttled and thus improve performance.
- We implement our solution in a cloud control plane and demonstrate its functionality and effectiveness. Also, we perform simulations with real data center measurements to estimate how much we improve the performance (by increasing the number of servers) of large-scale power-constrained data centers.

II. BACKGROUND

In this section, we review the typical design of the power infrastructure and then discuss power capping techniques at the server and data center levels.

A. Data Center Power Delivery

Figure 1 illustrates the power infrastructure for a typical data center. Power from the utility is delivered to the building at 12.5kV, and is stepped down to 480V for distribution. Onsite generators provide power through the ATS (*Automatic Transfer Switch*) if the utility feed fails. Another layer of transformers steps the voltage down further to 400V.[1] RPPs (*Remote Power Panel*) are 42-pole boxes with CBs (*Circuit Breaker*) that connect to *Cabinet Distribution Units* (CDUs) in the racks. 3-phase power is delivered to the CDUs from the RPPs. The outlets on the CDU receive power at 230V from one of the three phases.

At each branch of a distribution point, there are CBs that limit the amount of current, to protect the power infrastructure and guard against cascading upstream failures from short circuits and overload conditions. In this paper, we use the distribution point *power limit* to refer to the maximum amount of power that the corresponding CB[2] or transformer allows. When CBs trips, downstream power delivery is interrupted, potentially causing server power outage.

Redundant power feeds provide higher availability and resilience against power interruption. Servers rely on two or more power supplies connected to independent power feeds. Even if one of the power feeds or supplies fails, the remaining keep(s) the server operational. Ideally, power supplies equally share the server power load. In practice, load varies from supply to supply (see Section III.A).

Conventional practice in data centers is to not have sustained power load exceed 80% of the maximum rating for CBs and transformers [21] to avoid risk of damaging the power infrastructure. For example, a 30A 3-phase breaker may only be loaded to 24 A on each phase. When a power feed fails, its power load shifts to the remaining power feed, whose CB will see double the load. The server power connections must ensure that this doubled load doesn't exceed 80% of the CB rating. Otherwise, the CB may trip during the failure, and the servers downstream of the CB will lose power [5]. In our example with redundant (dual) 30A feeds, the per-phase load on each feed would need to be limited to 12A (40% of 30A) to ensure that the load during a failure is limited to 24A (80%).

In our work, we load CBs up to 80% under normal conditions because we employ power capping. When a feed fails, the breaker on the redundant feed becomes overloaded to 160%. The time it takes for a CB to trip depends on the amount of overload. For example, CBs covered under the UL 489 standard (a widely-adopted CB industry standard) will operate for a minimum of 30 seconds before tripping when under 160% load [11][17]. Within that 30-second window, our capping solution throttles the associated servers to ensure the load comes down to within 80% of the rating, which thus avoids tripping the CB.

B. *Server Power Capping*

In the past decade, power capping has become a common feature in servers to keep the server within a power limit [6][7][8][25]. Typically, the power controller measures the server power and throttles the CPU (scaling its voltage and frequency) and other components to enforce the power limit. Prior work has shown that such controllers can generally perform decisions within a few seconds [6].

The range of power control can be characterized by running the most power-demanding workload at the highest and lowest performance states of the server at the highest allowed ambient temperature. $ServerPcapmin$ is the power consumed by the server at the lowest performance state. $ServerPcapmax$ is the power at the highest performance state. Any power budgeted to the server above $ServerPcapmax$ is wasted, and capping cannot guarantee adherence to any budget below $ServerPcapmin$.

Capping server power consumption allows us to directly control the power load seen by CBs and transformers. The timescale of power capping is an order of magnitude faster than the trip time of CBs. This allows server power capping to adequately protect against tripping of CBs.

III. DESIGN GOALS

CapMaestro addresses three key challenges in leveraging server power capping for power management: (1) accounting for *multiple* power feeds, (2) applying priorities *globally* in power capping decisions, and (3) capturing *stranded power* in a redundant power infrastructure. These are important to address for highly-available data centers but have not been addressed in prior works.

A. *Power Capping for Multiple Power Feeds*

The power load across multiple power feeds is not perfectly balanced. We observe this at all levels of the infrastructure for three reasons. First, servers with multiple power supplies do *not* split their power load equally between their power supplies. In our servers with two power supplies there can be as much as 15% mismatch across the two supplies, with either A-side or B-side feed dominating across all power load levels. This power mismatch varies from server to server and is an intrinsic property of servers (i.e., is independent of the workloads), and cannot be adjusted during use. Additional data is available in Section 0. Second, power device failures (e.g., failed power supplies or power feeds) may also lead to imbalanced load between different power feeds. Third, some servers now have an energy efficiency mode that puts a redundant supply in standby (drawing no power) when the server load is below a certain level [34].

An imbalanced load between different power feeds forces power managers to assign and regulate *separate* budget for each power supply of a server. For example, a server with two power supplies (Supply A and Supply B) may get a budget of 200W on Supply A (which is connected to the A-side feed) and only 100W on Supply B (which is connected to the B-side feed), because other servers exert a greater power load on the B-side feed, leaving less power available to assign to Supply B. However, existing server power capping solutions, which only limit the combined load across *all* power supplies, do not consider this need for individual per-supply power budgets, and therefore may not adequately protect the upstream circuit breaker for each power supply. To tackle this challenge, we

[1] 400V is the line-to-line voltage of the 3-phase power. The corresponding line or phase voltage is 230V.
[2] CBs are rated in terms of maximum current, but we convert them to their equivalent power values.

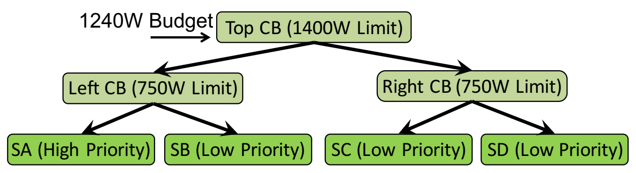


Figure 2. Example I: power feed with various server priorities.

propose a server power controller that enforces an individual power budget per supply in Section IV.B.

B. *Global Priority-Aware Power Capping*

In a data center, some workloads can be more important than others (e.g., due to different SLAs, pricing, workload heterogeneity, or service function). We want to prioritize these workloads *globally* (across the entire data center) and give them sufficient power during a power emergency, by letting them borrow power from lower-priority workloads on a common power feed, regardless of the physical server location.

Existing priority-based power allocation schemes cannot do this, as they consider priorities only between *local* groups of servers under a single power constraint (e.g., a branch circuit), and are unable to incorporate priority across higher levels of power limits in the data center (e.g., the servers attached to a common RPP in our power delivery layout) [4][5]. This limits the ability to truly share power across the entire data center.

Figure 2 shows an example with four servers SA, SB, SC, and SD under a total power budget of 1240W for a single power feed. The servers connect to a power feed of three CBs: a top-level CB rated at 1400W, and two child CBs rated at 750W each, which we call Left CB and Right CB. These four servers have an equal power demand of 430W each, and a *ServerPcapmin* of 270W. Suppose SA has high priority, while the other servers have low priority. Table 1 shows how much power each server is budgeted under a *local* priority-aware power capping policy, and how power would be budgeted if the policy is instead *global* priority-aware. At the top level, the local priority-aware policy splits the total power budget equally across the Left and Right CBs, as only the lowest-level CBs have the knowledge of and enforce server priorities. Therefore, under a total power budget of 1240W, both Left CB and Right CB are assigned a power budget of 620W. As SB can at most be throttled down to 270W (*ServerPcapmin*), SA can only receive a power budget of 350W (i.e., 620W – 270W), even though it demanded 430W. In contrast, a global priority-aware policy knows *at the top level* that one of the servers under Left CB has high priority. As a result, the policy allocates more power to Left CB, allowing SA to be budgeted the full 430W that it demands. A global priority-aware policy ensures that a higher priority server is not throttled when lower priority

Server	SA	SB	SC	SD
Priority (1 = high, 0 = low)	1	0	0	0
Power Demand (W)	430	430	430	430
Budget with Local Priority (W)	**350**	270	**310**	**310**
Budget with Global Priority (W)	**430**	270	**270**	**270**

Table 1. Power budget assignments using local per-CB vs. global priorities.

servers anywhere in the data center can be capped to meet the concerned power constraint. We introduce a global priority-aware power capping algorithm in Section IV.C.

C. *Stranded Power in Redundant Power Feeds*

For servers with redundant power supplies, it is possible that the available power on the power feed to each power supply may not be matched (Section III.A). This could be the result of different set of loads on either feed. If the load sharing of the server across its feeds does not match available power on those feeds, the available power on one of the feeds would be left *stranded* (i.e. unutilized). It is desirable to reallocate the stranded power to other power-constrained servers to improve their performance. To perform power reallocation, we propose a stranded power optimization mechanism in Section IV.D.

IV. DESIGN OVERVIEW

CapMaestro is a new scalable power management solution for data centers that achieves the three design goals described in Section III. At a high level, CapMaestro employs a lightweight *power control framework* to efficiently collect power demand information and enforce power budgets for each node in the power infrastructure hierarchy, including each individual server power supply (Sections IV.A and IV.B). CapMaestro uses the collected power demand information to determine power budgets for each node based on a new *global priority-aware power capping* algorithm (Section IV.C). Once global priority-aware allocation finishes, CapMaestro can *optimize stranded power* by identifying the nodes where assigned power is underutilized, and then reassigning this power elsewhere in the hierarchy (Section IV.D).

A. *System Overview*

CapMaestro uses a *power control tree* (shown in Figure 3) that mirrors the hierarchy of the power infrastructure. At the bottom of the tree, *capping controllers* manage the power of individual server/IT equipment using the built-in server power capping mechanism (see Section IV.B). At each higher level of the tree, we use a power *shifting controller* that distributes the power budget at that level among the nodes fed from that distribution point. Each shifting controller is mapped to a single physical device, and adheres to the device's *power limit*

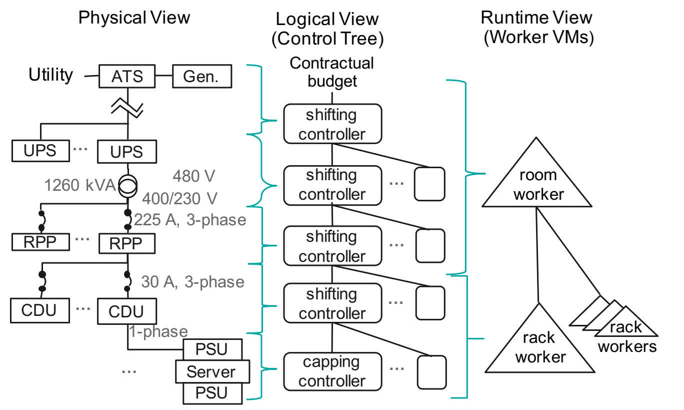


Figure 3. Mapping physical equipment to control tree.

(e.g., maximum power allowed by a transformer, RPP, or CDU) or *contractual budget* (i.e., the maximum power that a data center has negotiated to draw in total across all of its utility feeds).

To account for redundant power infrastructure, we replicate the power control tree for each power feed of the data center. We also replicate the power control tree for each phase of power delivery to protect each phase independently, since loading on each phase is not always uniform. In our example in Figure 3, our *power control framework* (two power feeds with three phases each) has six control trees. The shifting controllers on one power feed operate independently from shifting controllers on the other feed, while each server has a single capping controller that is shared across multiple trees.[3]

Each server can have a specific priority level.[4] Its capping controller generates metrics (e.g., power demand) for the server, which flow upstream in the control trees to the shifting controllers in the next level up. Each shifting controller produces priority-based metrics summarizing the sub-tree that it controls, based on the metrics that the shifting controller receives from its child nodes. *To perform global priority-aware power capping, a key insight is that we need to convey upstream only the metrics summarized by priority level, and not individual server metrics for all servers in a sub-tree*. In practice, we expect a data center to have only a small number of priority levels (on the order of 10); thus, the priority-based summaries provide us with a compact way to represent metrics for thousands of servers. This allows the shifting controller at the root node to efficiently have a global view of the power demand across the entire data center. With this view, the root shifting controller easily routes power (by assigning power budgets to its child nodes) towards the most critical servers by comparing priority-based metrics from each of its child nodes, while respecting the power limits of the intervening CBs and transformers along the control tree. These budgets flow downstream, and are recursively allocated until the budgets reach the capping controllers (see Section IV.C for algorithm details). After a power budget is assigned to a capping controller, the controller (Section IV.B) ensures that for *each* power supply of the server, the per-supply power budget is not exceeded by the power consumption on that supply.

Our control trees mirror the physical electrical connections of the data center, allowing us to model situations unique to each data center or portions of it. For example, CapMaestro can (1) manage both multiple- and single-corded devices; (2) deal with equipment that does not include power capping technology, by setting the metrics to assign a fixed maximum power for that equipment; (3) capture servers plugged into multiple phases of power; and (4) work with shifting controllers that accept power budgets based on restrictions aside from physical equipment limits, e.g., contractual budgets.

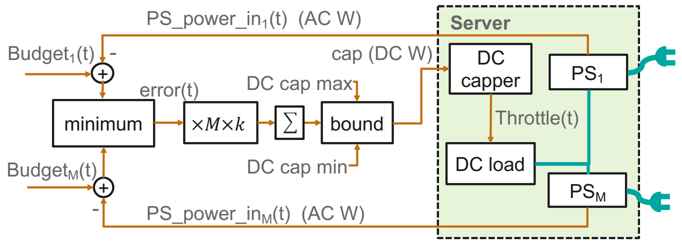


Figure 4. Power capping controller (“PS” means power supply).

B. Power Supply Budget Enforcement

To protect the independent power feeds of the redundant power infrastructure, we design a proportional-integral (PI) [15] feedback controller for CapMaestro that guarantees adherence to AC power budgets on the power consumption of each power supply in a server. Our controller utilizes the server power capping controls of Intel Node Manager [7], which caps only the total DC power of the server. The input to our controller is the external AC power budget for each power supply. These budgets are determined by the power capping algorithm that protects each power feed. The budgets for the power supplies of a server may have unequal values, depending on the load on each power feed. The controller determines the proper DC power cap for Node Manager to adhere to the given AC power budgets for all supplies.

Figure 4 shows our control diagram. First, each control iteration calculates an *error* for each power supply by subtracting its measured power from its budget value. This error quantifies how close the AC power is to the AC budget on each power supply. After that, the minimum error is selected as how much we should adjust for the AC power on every supply, and passed to the next stage. This ensures that the power supply that the minimum error corresponds to will settle to its desired budget and the remaining ones will be at or below their budgets[5]. Second, the error is scaled by the power supply efficiency (k) to transform from AC power domain to DC power domain (k can be determined from the power supply specification manual), and then further scaled by the number of working[6] power supplies (M) to account for how much DC power the full system power needs to be adjusted by. Third, the scaled error is added to the integrator (which stores the previously desired DC cap) to form the currently desired DC power cap. After clipping to the server’s controllable range, the currently desired DC power cap is sent to Node Manager which then manages the frequency and voltage of the processor to meet the DC power cap.

[3] For a server with multiple power supplies, its capping controller adjusts the frequency/voltage of the entire server, impacting the load on *all* of the server’s power supplies. As a result, different control trees need to share a single capping controller per server.

[4] For cloud platforms that run VMs or containers with different priorities, one could set server priority based on the priorities of the set of VMs/containers assigned to a server and further assign VMs/containers to servers based on their priorities.

[5] The stranded power optimization mechanism will later shift the unused power budgets to other servers for better utilization.

[6] Working power supplies refer to ones with non-zero power consumption. If a power supply fails, we ignore the quantities associated with it and decrease M in the capping controller.

C. Global Priority-Aware Power Capping Algorithm

CapMaestro's global priority-aware power capping algorithm allocates power budgets across a tree of shifting and capping controllers, respecting the data center contractual budgets and the power limits of multiple levels of CBs and transformers while safely trying to satisfy as much of the power demands of the servers. Both single and redundant power distribution environments can use this algorithm. In both cases, each control tree runs this algorithm independently.

Our algorithm runs iteratively, with each iteration consisting of two phases. First, in the **metrics gathering phase**, each shifting controller receives power allocation requests (and other metrics) from its child nodes. These metrics are tagged by the priority value *j*. The shifting controller then aggregates these metrics from all its children by priority value, and sends the aggregated metrics upstream to its parent node. Second, in the **budgeting phase**, each shifting controller receives its power budget from its parent node, and then computes and sends power budgets downstream for its child nodes based on the power budget assigned to the controller and the priority-based metrics of its child nodes. At the bottom, each capping controller receives an individual budget for each of its power supplies (from the corresponding *leaf* shifting controller), and uses the method discussed in Section IV.B to set a power cap for the corresponding server.

1) Metrics Gathering Phase

CapMaestro computes the following metrics at each *node* (a node may correspond to a shifting or capping controller):

- $Pcapmin(j)$: the minimum total power budget that must be allocated to servers with priority *j* under the node.
- $Demand(j)$: the total power demand of all servers under the node with priority *j*, without power capping. This metric is workload-dependent. For example, the metric will be larger if these servers run more power-hungry workloads.
- $Request(j)$: the maximum total power budget that can be safely allocated to servers with priority *j* under the node to satisfy their total power demand. If the node corresponds to a capping controller, this will be $Demand(j)$. If the node corresponds to a shifting controller, this may be lower than $Demand(j)$. This metric accounts for the power limit of the node, other requested power at higher priority, and the minimum power budget for lower priorities, all of which reduce the power budget available to this priority class.
- $Constraint$: the upper limit for power budgeted to a node across all priority classes. It is limited by the power limit of the node, power limits for downstream shifting controllers and $ServerPcapmax$ for downstream capping controllers.

The computation of these metrics differs between the capping and shifting controllers. At each capping controller, we calculate the metrics for each power supply of the server governed by the controller as:

$$Pcapmin(j) = r \times ServerPcapmin$$

$$Demand(j) = r \times max\{ServerDemand,\ ServerPcapmin\}$$

$$Request(j) = Demand(j)$$

$$Constraint = r \times ServerPcapmax$$

where *j* is the server priority, *r* is the fraction of the server load borne by that power supply (nominally 1/M, where M is the number of working power supplies; we adjust it in practice based on how the load is actually split between the working power supplies of the server), $ServerPcapmin$ and $ServerPcapmax$ are the minimum and maximum controllable AC power budgets for the server, and $ServerDemand$ is the amount of power that workloads running on the server consume at full performance (we discuss how to estimate it in Section 0). When we calculate $Demand(j)$, we choose the maximum of $ServerDemand$ and $ServerPcapmin$ and then scale it with *r*. This is because if the server is running light workloads, $ServerDemand$ may be below the minimum power budget $ServerPcapmin$. In this scenario, our power capping algorithm needs to ensure that the aggregate power budget allocated to the server across its power supplies stays within the controllable range; otherwise, the power cap on the server may not be enforceable if the server load suddenly increases later. For *j* not equal to the server priority, the corresponding metric values are zero.

At each shifting controller, we calculate the metrics, in descending order of priority (i.e., highest priority first) as:

$$Pcapmin(j) = \Sigma_i Pcapmin_i(j)$$

$$Demand(j) = \Sigma_i Demand_i(j)$$

$$Request(j) = min\left\{\begin{matrix} limit - \Sigma_{h>j} Request(h) - \Sigma_{l<j} Pcapmin(l), \\ \Sigma_i Request_i(j) \end{matrix}\right\}$$

$$Constraint = min\{limit,\ \Sigma_i Constraint_i\}$$

where *i* is a child node index, and *j*, *h*, and *l* are priorities, and $limit$ is the power limit of the shifting controller. For $Pcapmin(j)$ and $Demand(j)$, the corresponding metrics $Pcapmin_i(j)$ and $Demand_i(j)$) are aggregated across all the child nodes. For $Constraint$, either the power limit of this node or the sum of the constraints of the child nodes may limit the maximum budget for this controller. For $Request(j)$, it is first limited by the total power requests from the child nodes ($\Sigma_i Request_i(j)$). There is no reason to request more power at this controller than the controllers downstream can consume for this priority class. Second, $Request(j)$ is also limited by the power limit of the controller. For the sake of priority-aware power capping, we need to deduct the total power allocation requests for higher priorities h ($\Sigma_{h>j} Request(h)$). We also deduct the total minimum power allocation for lower priorities l ($\Sigma_{l<j} Pcapmin(l)$), as capping cannot restrict their consumption below their $Pcapmin(l)$.

2) Budgeting Phase

The budgeting phase at each shifting controller distributes its budget among its child nodes in four steps:

1. Allocate a minimum budget to each child *i* that is the sum of the child's $Pcapmin_i(j)s$ over all its priorities.
2. Iterate over the priority levels (*j*) from highest to lowest, to further allocate the portion of power requested above the minimum budget (i.e., $Request_i(j) - Pcapmin_i(j)$) to each child (*i*) from the controller's budget. If the power remaining in the controller's budget is not enough to meet the

power requested for any priority level during this step go to Step 3, else go to Step 4.

3. For the last priority j whose power demand could not be completely fulfilled in Step 2, proportionally give the remaining budget to each child (i) based on its power demand over minimum power budget ($Demand_i(j) - Pcapmin_i(j)$).
4. If there is still some remaining power budget, assign it to the child nodes up to their constraints ($Constraint_i$).

We have rigorously proved that our global priority-aware power capping algorithm allows servers with high priority to always be throttled after servers with lower priorities, as long as the power limits in the data center allow. The proof is in Section 0.

D. Stranded Power Optimization (SPO) Mechanism

As Section III.A points out, imbalanced loads may exist between different power feeds of a data center. We observe this causes a server to receive mismatched power budgets for its power supplies, such that the power budget for one of them may be underutilized, i.e., *stranded*. It is desirable to shift this stranded budget to other servers on the same power feed whose power supplies are constrained. To achieve this, we propose a stranded power optimization mechanism.

Our SPO mechanism runs after CapMaestro performs the global priority-aware power capping algorithm. With SPO, CapMaestro does not apply those budgets immediately. Instead, based on each power supply's budget, and the power sharing between them, we lower the requested power on the side with stranded power and run the power capping algorithm again. This allows the server to "return" its underutilized budget to the higher-level shifting controllers, and shift this extra power budget to other servers that are being capped.

V. IMPLEMENTATION

We implement a prototype of CapMaestro as an integral service in a cloud data center control plane. We group and run the shifting and capping controllers of CapMaestro in VMs called *workers* (shown in Figure 3). A worker communicates with other upstream or downstream workers to exchange metrics and budgets. This communication is on the order of milliseconds. We read server sensors via IPMI [26], and use these readings to generate server metrics proposed in Section IV.C. We use Intel Node Manager [7] to control server power based on the DC server power caps determined by CapMaestro. Our detailed implementation, including how to read and control power, how to estimate power demand, and how to ensure reliability, is described in Section 0.

Scalability analysis. Our solution has a negligible hardware cost. We deploy a rack-level worker for each rack of servers to protect its CDU, and a room-level worker for the two power feeds to protect their RPPs, transformers, and contractual power budget. We reserve one core per rack (each rack has 1260 cores) to run the rack-level worker, which consists of 6 shifting controllers (2 feeds x 3 phases) and 45 server capping controllers. For the entire data center, we use four additional cores to operate 1 room-level worker and 3 redundant manager

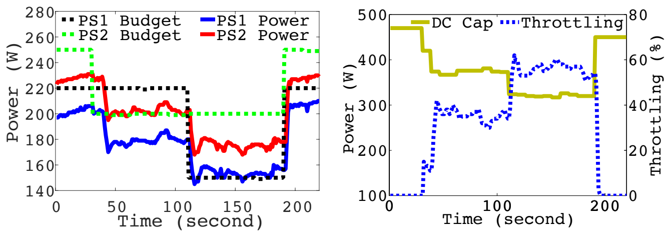


Figure 5. Power capping for redundant power supplies (PS). *Throttling* refers to power cap throttling (see Section 0).

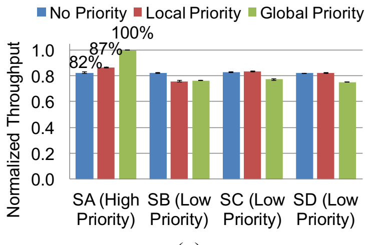
VMs (which are used to ensure reliability; details are provided in Section 0). The computation logic for these controllers in the rack-level worker takes less than 10 milliseconds. We estimate the computation time for a room-level worker shifting across 500 racks to be well under 300 milliseconds (based on the fact that the computation time of a shifting controller grows linearly with the number of its child controllers and the room-level worker has no capping controllers). In total, CapMaestro uses less than 0.1% of the data center's resources, regardless of the number of racks in the data center. Due to these negligible costs, we expect that our design will scale well even for very large data centers.

VI. EXPERIMENTAL RESULTS

In this section, we demonstrate our approach can successfully (a) enforce different budgets for multiple power supplies of a server using server power capping (Section VI.A), (b) implement global priority-aware power capping across hierarchical power constraints across the data center (Section VI.B), and (c) implement stranded power optimization for redundant power feeds (Section VI.C) based on real system experiments. Our servers run Apache HTTP Server [18] as a representative cloud workload (with separate client cluster running the Apache benchmarking tool *ab* [19]). Finally, we perform a data center-scale simulation based on characteristics of our real servers. We report the effectiveness of our solution for server capacity increase (Section VI.D).

A. Results for Server Power Cap Enforcement

Figure 5 shows that our controller in Section IV.B enforces power budgets on the individual power supplies (labeled PS1 and PS2 in the figure) of a server. At the beginning, the budgets for both supplies are higher than the loads, and there is no throttling. At t=30s, we lower the budget for PS2 to 200W. Our controller responds by computing and applying the resulting DC cap for the server that would lower the PS2's power down to the new budget. The Node Manager then applies the DC cap to the server, which lowers the load on *both* PS1 and PS2. At t=110s, an even smaller budget of 150W is placed on PS1, making it the more constrained of the two power supplies. Our controller computes and applies the corresponding DC cap to bring down PS1's power consumption. In both cases, our controller recognizes which of the power supplies has the more constrained budget, and ensures that the server load is lowered enough so that the power supply loads satisfy the more constrained budget. Overall, the power settles to within 5% of the assigned budgets



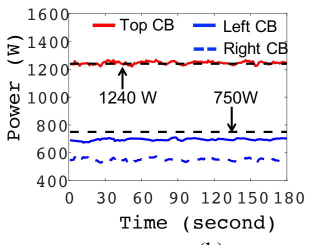


Figure 6. Example I: (a) Throughput normalized to uncapped cases; (b) Power at each CB.

Server	SA	SB	SC	SD
Priority (1 = high, 0 = low)	1	0	0	0
Power Demand (W)	**420**	413	417	423
Budget with No Priority (W)	**314**	306	311	316
Budget with Local Priority (W)	**344**	274	314	317
Budget with Global Priority (W)	**419**	276	275	275

Table 2. Server power for Example I.

within two control periods (16 s; the details of control period is in Section 0).

B. *Example I: Global Priority-Aware Power Capping*

We use four servers, set up as shown in Figure 2, to evaluate the conceptual example we described in Section III.B. Please note these servers are powered by a single power feed – this is an important power failure scenario in redundant power infrastructures where the other power feed fails. Each server runs the same workload and consumes an average of 420W without capping. Each server has a $ServerPcapmin$ of 270W. The total power budget is 1240W. Since this does not cover the full power demanded by all four of the servers, we need to perform power capping. Server SA is assigned high priority, and the other three servers (SB, SC, and SD) are assigned low priority. We evaluate the power allocated to each server under three different power capping polices: 1) *No Priority*: after guaranteeing that each server receives $ServerPcapmin$, distribute the remaining power proportionally to each server based on $(ServerDemand - ServerPcapmin)$; 2) *Local Priority*: enforce the notion of priority only at the lowest controller level, while the higher-level controllers distribute power to each branch using No Priority policy. This policy is an extension of Facebook's Dynamo [5], the most advanced data center power capping solution as far as we know that has been deployed in a production environment. We extend Dynamo to handle redundant power infrastructure; 3) CapMaestro's *Global Priority*: enforce a common priority system at every power controller (Section IV.C).

Table 2 shows the results for these three policies when the workloads are in a steady state. Similar to the conceptual example in Section III.B, our Global Priority policy lets the high-priority server SA consume 419W, which is very close to its full power demand (420W), while neither the Local Priority or No Priority policy can achieve this. This allows the workload running on SA to achieve a higher throughput and lower latency with Global Priority than with the Local Priority or No Priority policies. Figures 6(a) shows the detailed throughput normalized to an ideal baseline where no power capping takes place. For SA, No Priority results in 18% lower throughput (and 21% higher latency) relative to the uncapped performance of SA; while Local Priority results in 13% lower throughput (and 15% higher latency) than the uncapped performance. With Global Priority, SA achieves the same throughput (and latency) as if it were uncapped.

Figure 6(b) shows the total power consumption at the Top, Left, and Right CBs under our Global Priority policy. We observe that the total power consumption is below the respective limits at the Top CB (1240W), and at Left and Right CBs (750W). This demonstrates that our policy can successfully redistribute power for better performance at high-priority servers when power constrained while ensuring the power consumption respects power limits and budgets in data center, guaranteeing power safety.

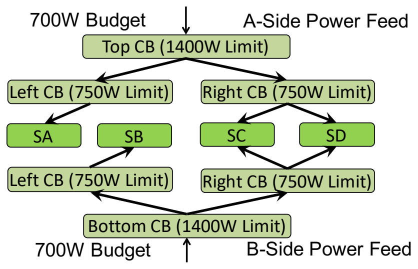
C. *Example II: Stranded Power Optimization*

We now demonstrate our policy's capability of utilizing stranded power in redundant power feeds with four servers (SA, SB, SC, and SD) connected to two power feeds. The configuration of the servers, and their power distribution hierarchy, is shown in Figure 7(a). SA has high priority, while the other servers have low priority. We assume that the B-side power supply of SA and the A-side supply of SB have failed. SC and SD have two healthy power supplies, so they draw power from both power feeds (though not in equal amounts, due to the power split mismatch). Each power feed has a budget of 700W (i.e., the total budget is 1400W), and the Top/Bottom CB is rated at 1400W, the Left/Right CB at 750W.

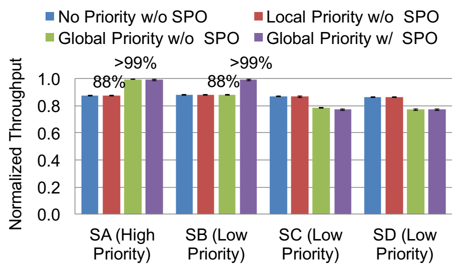
Table 3 shows the allocated power budgets and the power each server consumes on the A-side and B-side power feeds, under different power capping policies. If we use our Global Priority policy without our proposed SPO mechanism (i.e., Global Priority w/o SPO), SC and SD receive a power budget of 164W and 187W on the B-side, respectively. However, the servers can consume only 137W and 158W respectively, due to their more limited A-side budgets. This leaves 27W and 29W stranded on the B-side for SC and SD, respectively. If we apply our SPO mechanism (i.e., Global Priority w/ SPO), SC and SD lower their B-side power budgets, and CapMaestro shifts 67W of underutilized power to SB in Figure 7(c). Without SPO, SB has a 12% lower throughput (and 14% higher latency) relative to its uncapped performance. With SPO, SB now performs similar to its uncapped performance, as shown in Figures 7(b),

Server		SA	SB	SC	SD
Priority		1	0	0	0
Demand		414	415	433	439
No Priority w/o SPO	Budget	342/0	0/344	188/164	167/189
	Consumption	345/0	0/348	189/166	168/190
Local Priority w/o SPO	Budget	342/0	0/345	187/166	169/187
	Consumption	345/0	0/348	189/166	167/190
Global Priority w/o SPO	Budget	415/0	0/346	152/**164**	132/**187**
	Consumption	413/0	0/348	156/**137**	135/**158**
Global Priority w/ SPO	Budget	416/0	0/413	152/**132**	132/**155**
	Consumption	413/0	0/412	153/**134**	133/**156**

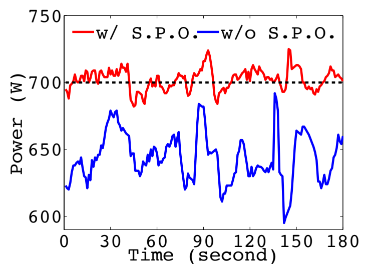
Table 3. Server power budgets and actual power consumption for Example II (a/b is respectively for the A-side/B-side feed).



(a)

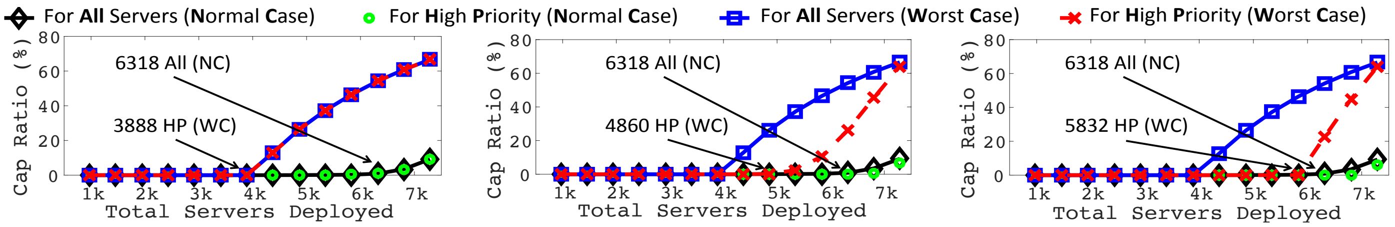


(b)



(c)

Figure 7. Example II: stranded power optimization. (a) Power feed organization; (b) Throughput normalized to uncapped case; (c) Total power of the B-side power feed with/without SPO under the Global Priority policy.



(a) No Priority (b) Local Priority (c) Global Priority

Figure 8. Average cap ratio for all servers and high-priority servers during normal case and worst-case power emergency.

even though SPO did not take away any power actually consumed by SC and SD.

Combining our Global Priority policy with SPO, CapMaestro can successfully avoid unnecessary reductions in performance when power capping is employed.

D. Impacts on Data Center Performance

We perform a simulation to study the data center performance improvement (number of servers the data center can support) when a power management system is employed under different conditions and policies. Our simulations model a production data center infrastructure as shown in Figure 1, with the parameters summarized in Table 4. The data center has 2 power feeds (A and B), 4 transformers, 36 RPPs, 324 CDUs, for a total of 162 racks (2 CDUs from different power feeds power 1 rack). In our simulation, we vary the total number of servers deployed by changing the number of servers in each phase of a CDU (from 2 servers to 15 servers), while keeping the rest of the infrastructure constant. We load the CBs and transformers to 80% of the rated power. The contractual budget for the data center is 700kW per phase, 2.1MW in total. We use 95% loading for this contractual budget, and reserve 5% as margin to tolerate errors (e.g., server parameter error,

contractual budget*	transformer rating*	RPP rating*	CDU rating*
700kW	420kW	52kW	6.9kW
ServerPcapmin	ServerPcapmax	ServerIdlePower	
270W	490W	160W	

transformers per	RPP per transformer	CDU per	servers per CDU
2	9	9	From 6 to 45

Table 4. Data center parameters (* means rating per phase; contractual budget is shared by the two power feeds - if a feed fails, the remaining feed consumes the full contractual budget.

power measurement error). Without employing a power management system, each phase of the CDU can contain at most 8 servers (700kW x 95% / 490W / 162CDU = 8.4) to accommodate peak power demand, resulting in a total of 3888 servers deployed in the data center.

For the simulations, we consider both *normal condition* which is typical load and fully operational dual-feed power infrastructure and *worst-case condition* which has all the servers at maximum power and one entire power feed down. For our baseline results, we assume 30% of servers are high-priority and perform full data center Monte Carlo simulations with random assignments of priorities to servers. For the *typical* load for the normal case, we use a published load profile from Google (Figure 9) [27], giving the distribution of average CPU utilization of a shared data center across time. We perform 20k Monte Carlo simulations for every data point in our results for the normal case. In each simulation, we draw an average CPU utilization for the data center from the distribution, and let the CPU utilization of each server randomly vary around it. Each server has a power demand between its idle power and maximum power, determined by its CPU utilization based on [2]. (We expect our conclusions still hold if we use other methods to generate server power demand from CPU utilization.) For the worst case, since load is fixed (maximum), we can converge with 1k Monte Carlo simulations for every data point (there is still random variation of server priorities). As a measure of capping impact on performance, we define a metric called *cap ratio*, which is the percentage of the server's dynamic power demand that is capped ($CapRatio = (Demand - Cap)/(Demand - ServerIdlePower)$). This metric enables us to have an application-neutral way of characterizing the potential impact on performance imposed by capping.

Figure 8 shows the average cap ratio during the normal case and the worst case for the three policies as number of servers are increased. The x-axis shows the number of servers and y-axis shows the corresponding Cap Ratio. Cap Ratio for high-priority servers and all servers are shown. The Cap Ratios grow with the number of servers deployed in the data center. For the priority-aware policies, high-priority servers are throttled last, therefore they will have a lower Cap Ratio compared with the average for all servers. Comparing Figure 8(b) and (c), we can see that with Global Priority, high-priority servers have a better performance compared with the Local Priority case, as Global Priority lets high-priority servers take power from low-priority servers even under other shifting controllers (while with Low Priority that is restricted to same shifting controller). The differences are stark under worst-case load and small for normal operations *at this typical load*.

In this study, our goal is to increase server count while negligibly impacting the average performance for all the servers during the normal case, and the average performance for high-priority servers during the worst case. Thus, we use 1% average cap ratio as the limit to determine the number of servers from the *all servers* curve for the normal case and from the *high-priority servers* curve for the worst case. Figure 8 (and 10) shows that to guarantee the performance for high-priority servers for worst-case, Global Priority can deploy up to 5832 servers in the data center, while Local Priority and No Priority can only support up to 4860 and 3888 servers, because it better differentiates servers with different priorities while capping. Figure 8 (and 10) also shows the three policies support the same number of servers (up to 6318 servers) during the normal case – this is because the normal condition criteria is impact for all servers which is priority agnostic. If we used a priority-sensitive impact criteria, we might also see higher numbers for priority-aware schemes for the normal case. If the typical load was much higher, the number of servers that can be supported for the normal case (requires negligible impact on all the servers) can be lower than those that can be supported for the worst case (requires negligible impact on just high-priority servers). Thus, to achieve our performance goals for both the normal and worst cases, we need to pick the minimum of the number of servers that can be deployed for the normal case and the worst case (shown as third set of bars in Figure 10). CapMaestro's Global Priority can support up to 5832 servers, which is 20% better than Local Priority (i.e., Facebook's Dynamo [5] with our extension to support multiple power feeds) does (4860 servers). Finally, compared to a data center with no power capping, *Global Priority achieves a 50% improvement* in the number of servers (5832 vs. 3888 servers).

Additional results: We performed a sensitivity study on the impact of key data center parameters including 1) the percentage of high-priority servers, 2) $ServerPcapmin$, and 3) the data center contractual budget.

First, we varied the percentage of high-priority servers to see its impact, as shown in Figure 11. With zero high-priority servers, the three polices can deploy the same number of servers. As the percentage of high-priority servers increases, the number of servers that can be deployed by No Priority quickly falls to 3888 servers, which is the number of servers that are provisioned by their maximum power. This is because No Priority cannot differentiate server priority in power capping. In order to guarantee the performance for high-priority servers, priority-agnostic No Priority needs to make sure all the servers suffer from negligible power capping. We can see that the number of servers for Global Priority and Local Priority are higher but decrease as the percentage of high-priority servers increase. The decrease is because fewer percentage of low-priority servers are available to take power from during the worst-case scenario. For the most cases, Global Priority outperforms Local Priority till high-priority servers are 90% of the mix.

Second, we lowered $ServerPcapmin$ from 270W (minimum frequency of 1.2GHz) down to 160W (idle power) to gauge the effect of an extended capping range where individual server performance can be taken close to zero. In the normal case, since we would like all servers to have negligible capping, extending capping range (i.e., lowering $ServerPcapmin$) does not impact the number of servers that data center can support in the normal case. However, lowering $ServerPcapmin$ allows data center to accommodate more servers during the worst case (when low-priority servers can be capped to $ServerPcapmin$, while high-priory servers have negligible capping) - using 30% high-priority servers (same as for the main study), we can support 7290 servers for Global priority and 5832 servers for Local Priority. Since No Priority is priority agnostic, it cannot treat high-priority servers differently and therefore can support only 3888 servers (no impact from lowering $ServerPcapmin$).

Third, we studied the impact of the data center contractual budget. For normal operation, the additional capacity of the redundant feeds can be used to drive server loads even higher. Our above analysis used a contractual budget equal to the maximum consumption of one side of the data center. If the

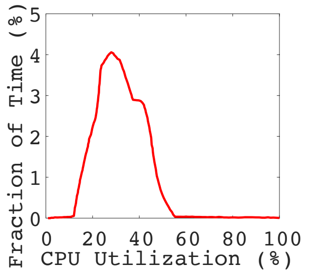


Figure 9. Distribution of average CPU utilization for a Google data center.

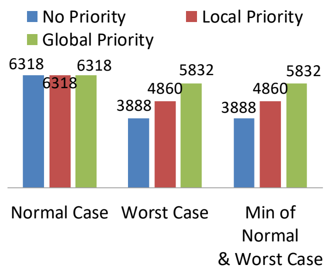


Figure 10. Total servers deployed (30% high-priority servers).

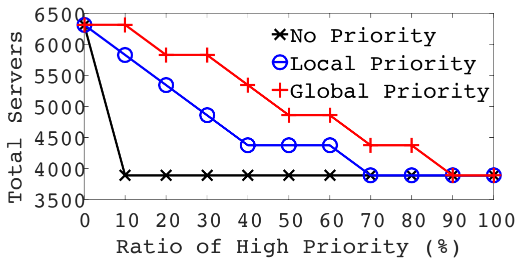


Figure 11. Total servers deployed under various ratios of high-priority servers.

contractual budget was higher, then more servers could be deployed for use during the normal case at the risk of deeper throttling required during failure events. This trade-off applies to all policies. For example, the number of servers can go up to 9720 for a budget of 3.3MW compared to 6318 for a budget of 2.1MW (Figure 10). Average data center load across these simulations came to 2.6MW (>2.52MW single feed capacity with 80% derating). If a feed fails while the actual load is higher than the capacity of the remaining feed, then servers will be capped. In this case, the average cap ratio may exceed 1% (our earlier criteria for worst case). When one feed fails, the different policies exhibit different extents of capping. No Priority will cap all classes equally (52% cap ratio), and the priority-aware schemes will cap the higher priority class less – 0.68% cap ratio for Global Priority and 4.2% for Local Priority (assuming a $ServerPcapmin$ of 160W).

VII. RELATED WORK

To our knowledge, our paper is the first work to (1) propose a power management system that can handle data centers with multiple power feeds; (2) design a global priority-aware power capping system that enables high-priority servers to borrow power from a low-priority server *anywhere* in the data center; and (3) propose a mechanism to make use of the stranded power that exists in redundant power infrastructure.

Server & Data Center Power Capping: Power capping first appeared in server products in 2006 [1]. Around the same time, Fan et al. observed that data centers rarely consumed their maximum peak power, and could allow up to 39% more servers in the same power infrastructure without throttling [2]. They recommended using power capping as a safety valve, using some amount of throttling to allow for the deployment of an even greater number of servers. Since then, several works on data center power capping [3, 4, 5, 10, 11, 14, 20, 32] have been proposed to effectively increase the server capacity of data centers. These works require server power capping [6-8] as the underlying mechanism to enable power capping for data centers. However, these server power capping mechanisms control only the sum of power consumption across all power supplies of a server and does not enforce separate power caps on individual power supplies. Therefore, they are inadequate to protect upstream power feeds in redundant power topologies (see Section III.A). As a result, these prior data center power capping methods [3, 4, 5, 10, 11, 14, 20, 32], which rely solely on traditional server power capping mechanisms and do not have the context of the redundant power topology, cannot safely control highly-available data centers with multiple power feeds, which is one of the important challenges we resolved in this paper. Moreover, we proposed a mechanism to effectively utilize the stranded power that is caused by the imbalanced load between different power feeds in redundant power infrastructure (see Section VI.D). This stranded power is different from what prior work [9] intended to utilize, which is caused by imbalanced load between different CBs in single-feed power infrastructure.

Priority-aware Capping: Prior works have included some notion of prioritizing budgets in power capping controllers [1][4][5]. However, such priorities are local to the individual controller. For example, in Dynamo, the workloads are known in advance and have assigned priorities. The priority mechanism works only at the leaf controller level, which at most covers “a few hundred” servers [5]. Our proposed solution provides the ability for multiple levels of the capping hierarchy to capture the priority of all child nodes, enabling the comparison of priorities across the entire data center for smarter capping decisions.

Other Knobs for Boosting Server Capacity: Kontorinis et al. [35] proposed using energy storage devices to shave peak power demand and allow an increase in server capacity. Wang et al. [36], Hsu et al. [37], Wallace et al. [38] proposed to efficiently utilize the power infrastructure capacity by workload scheduling to boost server capacity. These methods are orthogonal to ours and can be combined with our proposal. However, using only these methods may not be cost-effective to increasing the server capacity of a data center. Energy storage devices come with additional cost and space needs, and may need to be replaced after a certain number of charge/discharge cycles. In addition, an energy storage-only solution cannot handle power peaks that last longer than a few hours. On the other hand, power-aware workload scheduling puts additional requirements on workloads, such as requiring them to be short-lived [36] or repetitive [38], or bear a power consumption pattern lasting for several days [37]. In contrast, CapMaestro is designed for existing power infrastructures without imposing requirements on workload characteristics.

VIII. CONCLUSION

Power capping provides the means to actively limit load in the cloud and HPC data center, to avoid going over the rating limits of CBs, transformer capacity, or user-set contractual budget. We build CapMaestro, a distributed, fault-tolerant data center power management architecture that employs power capping and protects against oversubscription at every level of the power distribution hierarchy. CapMaestro includes novel controllers to manage power for servers with multiple power supplies. Our distributed algorithm respects both power limits in the hierarchy and server priority levels, while mitigating stranded power in data centers with redundant power feeds. We safely utilize the traditional gap between the provisioned data center power and the actual load of servers to support more servers within the existing power infrastructure. We evaluate a prototype of CapMaestro on real cloud servers to validate its guarantees, and simulate CapMaestro’s performance on a large-scale data center environment. We find that for a typical data center where 30% of randomly-selected servers are high-priority, CapMaestro supports 50% more servers than a data center without power capping, and 20% more servers than a data center that uses a state-of-the-art power capping technology that we modify to support redundant power feeds.

REFERENCES

[1] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, “Ensemble-level Power Management for Dense Blade Servers,” in Intl. Symp. on Computer Architecture (ISCA), 2006.

[2] X. Fan, W. Weber, and L. Barroso, “Power Provisioning for A Warehouse-Sized Computer,” in Intl. Symp. on Computer Architecture (ISCA), 2007.

[3] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No "Power" Struggles: Coordinated Multi-Level Power Management for the Data Center,” in Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2008.

[4] X. Wang, M. Chen, C. Lefurgy, and T. Keller, “SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers,” IEEE Trans. on Parallel and Distributed Systems, 2012.

[5] 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 Intl. Symp. on Computer Architecture (ISCA), 2016.

[6] C. Lefurgy, X. Wang and M. Allen-Ware, “Power Capping: A Prelude to Power Shifting,” Cluster Computing 11 (2008): 183-195.

[7] Intel, “Intel® Intelligent Power Node Manager 3.0 External Interface Specification Using IPMI,” Document Number 332200-001US, March 2015.

[8] Intel, “Intel® 64 and IA-32 Architectures Software Developer’s Manual,” Volume 3B: System Programming Guide, Part 2, Document number 253669-060US, Sept. 2016.

[9] L. Ganesh, J. Liu, S. Nath, and F. Zhao, “Unleash Stranded Power in Data Centers with RackPacker,” in Workshop on Energy-Efficient Design, 2009.

[10] A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, S. Sankar, “The Need for Speed and Stability in Data Center Power Capping,” in International Green Computing Conference (IGCC), 2012.

[11] X. Fu, X. Wang, and C. Lefurgy, "How Much Power Oversubscription is Safe and Allowed in Data Centers?", in International Conference on Autonomic Computing (ICAC), 2011.

[12] National Fire Protection Association, “National Electrical Code,” 2008.

[13] Rockwell Automation Inc., “Bulletin 1489 Circuit Breakers Selection Guide,” http://literature.rockwellautomation.com/idc/groups/literature/documents/sg/1489-sg001_-en-p.pdf, 2010.

[14] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, “Statistical Profiling-Based Techniques for Effective Power Provisioning in Data Centers,” in ACM European Conference on Computer Systems (Eurosys), 2009.

[15] K.J. Åström, and T. Hägglund, “Advanced PID Control,” ISA-The Instrumentation, Systems and Automation Society, 2006.

[16] M. Govindan, C. Lefurgy, and A. Dholakia, “Using On-line Power Modeling for Server Power Capping,” in Workshop on Energy Efficient Design, 2009.

[17] UL, “UL 489 Molded-Case Circuit Breakers, Molded-Case Switches, and Circuit Breaker Enclosures,” 13th Edition, October 14, 2016.

[18] Apache HTTP server, https://httpd.apache.org/.

[19] Apache HTTP server benchmarking tools, https://httpd.apache.org/docs/2.4/programs/ab.html.

[20] R. Azimi, M. Badiei, X. Zhan, N. Li, and S. Reda, “Fast Decentralized Power Capping for Server Clusters,” in IEEE Symposium on High-Performance Computer Architecture (HPCA), 2017.

[21] National Fire Protection Association, “National Electrical Code,” 2016.

[22] H. Lim, A. Kansal, and J. Liu, “Power Budgeting for Virtualized Data Centers,” in USENIX annual technical conference (USENIX ATC), 2011.

[23] D. Wang, S. Govindan, A. Sivasubramaniam, A. Kansal, J. Liu, and B. Khessib, “Underprovisioning Backup Power Infrastructure for Datacenters,” in Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2014.

[24] O. Sarood, A. Langer, A. Gupta, and L. Kale, “Maximizing Throughput of Overprovisioned HPC Data Centers under A Strict Power Budget,” in Intl. Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2014.

[25] H. Zhang and H. Hoffmann, “Maximizing Performance under A Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques,” in Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2016.

[26] Intel, Hewlett-Packard, NEC, Dell, “Intelligent Platform Management Interface (IPMI) Specification v2.0 rev. 1.1,” 2013.

[27] L. A. Barroso, J. Clidaras, and U. Hölzle. "The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines," in Synthesis lectures on computer architecture, 2013.

[28] M. Alian, A. H. M. O. Abulila, L. Jindal, D. Kim and N. S. Kim, "NCAP: Network-Driven, Packet Context-Aware Power Management for Client-Server Architecture," in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017.

[29] H. Yang, Q. Chen, M. Riaz, Z. Luan, L. Tang, and J. Mars, “PowerChief: Intelligent Power Allocation for Multi-Stage Applications to Improve Responsiveness on Power Constrained CMP,” in International Symposium on Computer Architecture (ISCA), 2017.

[30] Y. Li, D. Wang, S. Ghose, J. Liu, S. Govindan, S. James, E. Peterson, J. Siegler, R. Ausavarungnirun, and O. Mutlu, “SizeCap: Efficiently Handling Power Surges in Fuel Cell Powered Data Centers,” in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016.

[31] C. Li, R. Zhou, and T. Li, “Enabling Distributed Generation Powered Sustainable High-Performance Data Center,” in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2013.

[32] Z. Tang, H. Zhou, Y. Zhu, R. Tian and J. Yao, "Quantitative Availability Analysis of Hierarchical Datacenter under Power Oversubscription," IEEE International Conference on Smart Computing (SMARTCOMP), 2017.

[33] Supermicro, “Embedded BMC/IPMI User’s Guide Revision 2.0”, 2012.

[34] M. Muccini, W. Cook, “Power Consumption Reduction: Hot Spare,” Dell, white paper, 2012.

[35] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. Tullsen, T. S. Rosing, “Managing Distributed UPS Energy for Effective Power Capping in Data Centers”, in Intl. Symp. on Computer Architecture (ISCA), 2012.

[36] G. Wang, S. Wang, B. Luo, W. Shi, Y. Zhu, W. Yang, D. Hu, L. Huang, X. Jin, W. Xu, “Increasing Large-Scale Data Center Capacity by Statistical Power Control.”, in ACM European Conference on Computer Systems (Eurosys), 2016.

[37] C. Hsu, Q. Deng, J. Mars, L. Tang, “SmoothOperator: Reducing Power Fragmentation and Improving Power Utilization in Large-scale Datacenters”, in Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2018.

[38] S. Wallace, X. Yang, V. Vishwanath, W. E. Allcock, S. Coghlan, M. E. Papka, Z. Lan, “A Data Driven Scheduling Approach for Power Management on HPC Systems”, in Intl. Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2016.

[39] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, M. Schulz, “Finding the Limits of Power-Constrained Application Performance”, in in Intl. Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2015.

[40] D. A. Ellsworth, A. D. Malony, B. Rountree, M. Schulz, “Dynamic Power Sharing for Higher Job Throughput”, in Intl. Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2015.

APPENDIX

A. ARTIFACT DESCRIPTION

In our study, we have implemented CapMaestro as a real-system prototype running in the control plane of a proprietary cloud data center. Our implementation has two major components, called *manager* and *worker*. We run instances of the two components in the data center as independent virtual machines (VM). In this appendix, we describe the implementation details of these components within our power management system.

A.1 Manager Component

The manager component is responsible for ensuring that each controller in the power control framework is covered by a worker component. Periodically, the manager checks a database containing the inventory of the data center and its electrical connections (provisioned by the data center operator). Each entry of the database contains a list of controllers that are run within a single worker instance. The manager assigns the controllers of a newly discovered entry to a worker instance, and let the worker instance run the control operations described in Section IV. The manager also checks the health of each worker instance every 2 seconds. If a worker has failed, the manager removes the failed worker VM and assigns its work to a spare worker instance within a 2-second period. For reliability, multiple (typically three) copies of the manager VM are run.

A.2 Worker Component

The worker component is responsible for running the controllers that are assigned by manager. On startup, the worker receives a list of controllers. A worker communicates with other upstream or downstream workers to exchange metrics and budgets. This communication is on the order of milliseconds. Our system is flexible in terms of how to map controllers to the worker VMs and the number of layers in the worker hierarchy. A good mapping should be based on the number of servers deployed and the configuration of power delivery hierarchy in the data center. In a deployment of our prototype for a cloud data center, each rack of servers has a corresponding rack-level worker which consists of 6 CDU-level shifting controllers (2 power feeds x 3 phases) and all the capping controllers for servers within the rack. We also employ a room-level worker for the two power feeds to contain their RPP-level, transformer-level, and data center-level shifting controllers. For our real-system experiments in Section VI, our prototype is deployed over 4 servers. Here we use a single worker that consists of 4 capping controllers and two levels of shifting controllers. The controllers in the single worker faithfully executes all the details of our mechanisms.

Each capping controller reads sensors for the server under its control every second through IPMI [26]. The sensors include AC power monitors for the two power supplies and the *power cap throttling* level. Power cap throttling is an Intel Node Manager [7] metric that quantifies the percentage of server voltage/ frequency throttling.

Each capping controller averages the per-second readings over an 8-second interval for computing its metrics. Each capping controller sets the DC power cap for the server under its control every 8 seconds (i.e., control period is 8 seconds) based on budgets allocated to the server's power supplies. By averaging the readings for the metrics at this granularity, the resulting cap is a response to more stable changes yet fast enough to address failures in the infrastructure. The capping controller sends the power cap to the Node Manager through the server's baseboard management controller [33] via IPMI [26]. Node Manager then ensures that the server power is within the cap in 6 seconds.

Each capping controller computes the power demand ($ServerDemand$) for the server using a regression method [16]. The capping controller obtains the server power consumption and power cap throttling over the last 16 seconds and builds a model relating server power and throttling to estimate the server power at 0% throttling as the $ServerDemand$. If power is measured during an interval when power cap throttling is 0% (i.e., full performance), then the measured power is used instead of the forecast. Alternatively, any suitable approach to estimate demand could be used.

B. POWER SPLIT BETWEEN POWER SUPPLIES

A server with multiple power supplies may not split its power consumption equally across its power supplies. Figure 12 shows how the power is split between the two power supplies of our servers (Server SA, SB, SC, and SD) across a range of power loads (idle to full). We observe that there can be as much as 15% mismatch in the power consumed by the two supplies.

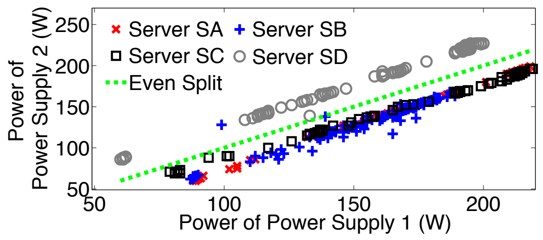


Figure 12. Power Split for Servers with 2 power supplies.

C. PROPERTY PROOFS FOR ALGORITHM

In this section, we prove the global priority-aware power capping algorithm achieves the following properties for both data centers with a single power feed and data centers with redundant power feeds:

- **Theorem 1:** Guaranteed Safety

The algorithm ensures the safety condition: (1) the data center contractual budget and all the power limits (such as the power rating of CBs and the power capacity of transformers) are not exceeded by the total power budget assigned to the servers under them[7]; and (2) each server receives a power budget no less than the server's $Pcapmin$ at each power feed[8].

[7] Our capping controller ensures that the power consumption of each power supply of a server does not exceed the power supply's power

- **Theorem 2:** Respecting Server Priority

After allocating the power budgets for each server using the proposed algorithm, it is impossible to find a server being throttled to lower performance state due to an enforced power cap, such that we can increase its power budget and decrease the power budgets of other servers with lower priorities at either one or multiple power feeds without violating the safety condition[9].

C.1 Definitions and Notations

In the control framework, we define priority level 1 as the lowest priority, and priority level N as the highest priority. Each shifting controller has metrics for every priority level between 1 and N. However, if there is no server with a certain priority level (j) under the shifting controller, the corresponding metrics for that priority level at the shifting controller ($Pcapmin(j)$, $Demand(j)$, $Request(j)$) will be zero. Each capping controller has metrics for a single priority, which is the priority for the server governed by the capping controller.

In the design of data center, we need to ensure that every power limit should be greater than or equal to the total $Pcapmin$ of all the servers under it, otherwise there will be safety issue. This is the fundamental requirement for designing data center - any data center equipped with power capping system (including ours) should follow. Therefore,

$$limit \geq \sum_j Pcapmin(j)$$

where $limit$ is the power limit that is protected by a shifting controller; $Pcapmin(j)$ is the minimum power budget for priority j at the shifting controller.

C.2 Proof for Theorem 1

We will first prove Lemmas 1.1, 1.2, and 1.3, before using them to prove Theorem 1.

Lemma 1.1: At any shifting controller, the total requested power by the controller itself for all the priorities is less than or equal to the power limit of the shifting controller. i.e., $\sum_{j\geq 1} Request(j) \leq limit$, where $Request(j)$ is the requested power for priority j at the shifting controller, $limit$ is the power limit of the shifting controller.

Proof for Lemma 1.1: As defined in Section 4.3.1,

$$Request(j) = min\left\{limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l)\ , \sum_i Request_i\,(j)\right\}$$

Therefore,

$$Request(j) \leq limit - \sum_{h>j} Request\,(h) - \sum_{l<j} Pcapmin\,(l)$$

Hence,

$$\sum_{h\geq j} Request(h) \leq limit - \sum_{l<j} Pcapmin\,(l)$$

Let $j = 1$, then we have

$$\sum_{h\geq 1} Request(h) \leq limit$$

Equivalently,

$$\sum_{j\geq 1} Request(j) \leq limit$$

Lemma 1.1 is proved.

Lemma 1.2: At any shifting controller or capping controller, the requested power for a priority is greater than or equal to the minimum power budget for the priority, i.e., $Request(j) \geq Pcapmin(j)$.

Proof for Lemma 1.2:

Use mathematical induction to prove the lemma.

Assign a number n to each level of the hierarchy in the control tree (please note the control tree is balanced). The leaf nodes in the control tree are assigned the smallest number; the top node in the control tree, which corresponds to the per-phase contractual budget for a power feed, is assigned the largest number. For example, for a data center shown in Figure 1 in the paper, shifting controllers corresponding to per-phase contractual budget, RPP, and CDU is assigned a level number of 4, 3, and 2 respectively. The capping controllers, which correspond to servers, are assigned a level number of 1.

Step A for proving Lemma 1.2

For any controller with a level number $n = 1$ (i.e, capping controllers) with priority j,

As defined in Section 4.3.1,

$$Request(j) = Demand(j)$$

$$Demand(j) = r \times max\{ServerDemand,\ ServerPcapmin\}$$

$$Pcapmin(j) = r \times ServerPcapmin$$

where r is the fraction of the server load borne by a power supply of the server.

Therefore,

$$\begin{aligned} Request(j) &= Demand(j) \\ &= r \times max\{ServerDemand,\ ServerPcapmin\} \\ &\geq r \times ServerPcapmin \\ &= Pcapmin(j) \end{aligned}$$

budget. Therefore, by ensuring the data center contractual budget and the power limits of circuit breakers and transformers are not exceeded by the total power budget assigned to the servers under them, a power capping algorithm can guarantee that they will not be exceeded by the power consumption under them.

8 This implies the power cap for the server will be greater than or equal to the $ServerPcapmin$ of the server, and therefore it is enforceable, as shown in the proof.

9 This is a formal and equivalent interpretation for the claim that "the proposed algorithm allows servers with high priority to always be throttled after servers with lower priorities, as long as the power limits in the data center allow."

Hence, Lemma 1.2 holds for any controller with a level number $n = 1$.

Step B for proving Lemma 1.2

Assuming Lemma 1.2 holds for any controller with a level number $n \leq k$, let's prove the lemma holds for any controller with a level number $n = k + 1$ (i.e., a shifting controller).

As Lemma 1.2 holds for any controller with a level number $n \leq k$, for any child node (with an index i) of a controller with a level number $n = k + 1$,

$$Request_i(j) \geq Pcapmin_i(j)$$

Therefore, if we sum over all the child nodes of this controller, we will have

$$\sum_i Request_i(j) \geq \sum_i Pcapmin_i(j)$$

As defined in Section 4.3.1,

$$Pcapmin(j) = \sum_i Pcapmin_i(j)$$

Hence,

$$\sum_i Request_i(j) \geq Pcapmin(j) \qquad (1)$$

As defined in Section 4.3.1,

$$Request(j) = min\left\{ limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l)\;, \sum_i Request_i\,(j) \right\}$$

Hence,

$$Request(j) \leq limit - \sum_{h>j} Request\,(h) - \sum_{l<j} Pcapmin\,(l), \qquad \text{for any } j$$

For $\boldsymbol{1 \leq j \leq N - 1}$ (priority level N is the highest priority, and priority level 1 is the lowest priority), using $(j + 1)$ to substitute j in the above inequality, we have

$$Request(j+1) \leq limit - \sum_{h>j+1} Request\,(h) - \sum_{l<j+1} Pcapmin\,(l)$$

Therefore,

$$Request(j+1) \leq limit - \sum_{h>j} Request\,(h) + Request(j+1) - \sum_{l<j} Pcapmin\,(l) - Pcapmin(j)$$

Subtract $Request(j + 1)$ from both sides and solve for $Pcapmin(j)$, we have

$$Pcapmin(j) \leq limit - \sum_{h>j} Request\,(h) - \sum_{l<j} Pcapmin\,(l), \qquad \text{for } 1 \leq j \leq N-1 \qquad (2)$$

For $\boldsymbol{j = N}$, as the power limit should be greater than or equal to the total $Pcapmin(l)$ at the controller, we have

$$limit \leq \sum_{l \leq N} Pcapmin(l)$$

Hence,

$$Pcapmin(N) = \sum_{l \leq N} Pcapmin\,(l) - \sum_{l<N} Pcapmin\,(l) \leq limit - \sum_{l<N} Pcapmin(l)$$

Further as N is the highest priority,

$$\sum_{h>N} Request(h) = 0$$

We have

$$Pcapmin(N) \leq limit - \sum_{l<N} Pcapmin\,(l) = limit - \sum_{h>N} Request(h) - \sum_{l<N} Pcapmin\,(l) \qquad (3)$$

Based on (2) and (3), we will have

$$Pcapmin(j) \leq limit - \sum_{h>j} Request\,(h) - \sum_{l<j} Pcapmin\,(l), \qquad \text{for any } j \qquad (4)$$

If $limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l) \leq \sum_i Request_i\,(j)$,

$$Request(j) = min\left\{ limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l)\;, \sum_i Request_i\,(j) \right\} = limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l)$$

Further based on equation (4),

$$Request(j) = limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l) \geq Pcapmin(j)$$

If $limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l) \geq \sum_i Request_i\,(j)$,

$$Request(j) = min\left\{ limit - \sum_{h>j} Request(h) - \sum_{l<j} Pcapmin\,(l)\;, \sum_i Request_i\,(j) \right\} = \sum_i Request_i\,(j)$$

Further based on (1),

$$Request(j) = \sum_i Request_i\,(j) \geq Pcapmin(j)$$

In either case,

$$Request \geq Pcapmin(j)$$

Therefore, Lemma 1.2 holds for any controller with a level number $n = k + 1$.

Hence, based on Step A and Step B for proving Lemma 1.2, we conclude that for any shifting controller and capping controller, $Request(j) \geq Pcapmin(j)$. Lemma 1.2 is proved.

Lemma 1.3: At any shifting controller, the constraint (the maximum power budget that is safe to allocate to this controller) is less than or equal to the power limit of the controller, i.e., $Constraint \leq limit$.

Proof for Lemma 1.3: As defined in Section 4.3.1, at any shifting controller,

$$Constraint = min\left\{limit,\ \sum_i Constraint_i\right\}$$

Therefore,

$$Constraint \leq limit$$

Lemma 1.3 is proved.

Proof for Theorem 1: At Step 1 of the budgeting phase, each child controller (no matter whether it is a shifting controller or capping controller) receives a budget of $Pcapmin(j)$ at each of its priority j. At later steps (Step 2, 3, and 4), each child controller may receive additional non-negative power budget (Lemma 1.2 guarantees the power budget assigned at Step 2 is non-negative, as $Request(j) \geq Pcapmin(j)$). Therefore, every server in the data center must at least receive its $Pcapmin(j)$ as budget from each of its working power supply. In other words, if we use $Budget_1$, …, $Budget_M$ to denote these power budgets (the server has M working power supplies), we have

$$Budget_i \geq Pcapmin(j)_i, \qquad i = 1, \ldots, M$$

here $Pcapmin(j)_i$ is the $Pcapmin(j)$ for the power supply i.

As defined in Section 4.3.1,

$$Pcapmin(j)_i = r_i \times ServerPcapmin, \qquad i = 1, \ldots, M$$

where r_i is the ratio of power supply load at power supply i, we can set a power cap (Cap) to the server that respects its power budgets and satisfies

$$\begin{aligned} Cap &= min\left\{\frac{Budget_1}{r_1}, \ldots, \frac{Budget_N}{r_N}\right\} \\ &\geq min\left\{\frac{Pcapmin(j)_1}{r_1}, \ldots, \frac{Pcapmin(j)_M}{r_M}\right\} \\ &\geq ServerPcapmin \end{aligned}$$

Therefore, the power cap for each server is greater than or equal to the $ServerPcapmin$ of the server, and thus it is enforceable.

Our algorithm makes sure that: at each shifting controller, the power budget that the controller receives is greater than or equal to the total power budget that the controller assigns to its child controllers[10]. Therefore, at any shifting controller, the total power budget received by the controller is greater than or equal to the total power budget assigned to the servers under the shifting controller. Hence, the data center contractual budget (which is the power budget received by the top shifting controller) is not exceeded by the total power budget of the servers in the data center. And due to the same reason, in order to prove each power limit in the data center (which is protected by other shifting controllers) is not exceeded by the total power budget assigned to the servers under it, we just need to prove each power limit is not exceeded by the power budget received by the shifting controller that protects the power limit (because the power budget received by this shifting controller is greater than or equal to the total power budget assigned to the servers under the power limit), i.e., $Budget \leq limit$.

At any shifting controller (including the top shifting controller), the power budget allocated to a child controller at each priority j at Step 1 of the budgeting phase is $Pcapmin(j)$. At Step 2 and 3 of the budgeting phase, the child controller may receive additional power budget up to $(Request(j) - Pcapmin(j))$. Therefore, the aggregate budget that the child controller receives at each priority j at Step 1, 2 and 3 is at most $Request(j)$. If we further sum over all the priorities of the child controller, the total budget that the child controller receives at Step 1, 2, and 3 is at most $\sum_j Request(j)$. Further based on Lemma 1.1, since $\sum_{j\geq 1} Request(j) \leq limit$, the total power budget that the child controller receives at Step 1, 2, and 3 does not violate the power limit of the child controller. At Step 4 of the budgeting phase, the child controller may receive extra power budget. However, if the child controller receives the extra power budget, the total budget that the child controller receives at Step 1, 2, 3, and 4 is up to the $Constraint$ of the child controller. Based on Lemma 1.3, $Constraint \leq limit$. Therefore, the total budget that the child controller receives at Step 1, 2, 3, and 4 is no larger than $limit$. In conclusion, the child controller receives a total power budget that does not exceed the power limit of the child controller.

Hence, Theorem 1 is proved.

C.3 Proof for Theorem 2

We will first prove Lemma 2.1 and Corollary 2.2, and then use them to prove Theorem 2.

Lemma 2.1: At any shifting controller, the aggregate power budget received by the controller at Step 1, 2, and 3 of the budgeting phase for a priority, is equal to the total power budget that this controller assigns to its child controllers for this priority at Step 1, 2, and 3 of the budgeting phase.

Proof for Lemma 2.1: Use proof by contradiction. Let's make the following assumption: Suppose at a shifting controller, priority j is the highest priority such that the aggregate power budget for priority j that this controller receives at Step 1, 2, and 3 (i.e., $Budget(j)$) is not equal to the total power budget that this controller assigns to its child controllers for this priority at Step 1, 2, and 3 (i.e., $\sum_i Budget_i(j)$; $Budget_i(j)$ is the aggregate power budget that this controller assigns to its child controller i for priority j at Step 1, 2, and 3). In other words,

$$Budget(j) \neq \sum_i Budget_i(j)$$

Based on how the power budget is assigned at Step 1, 2, and 3 of the budgeting phase, it is obvious that

$$Budget(j) \leq Request(j)$$

Based on the definition of $Request(j)$, it is obvious that

[10] This will be true if the data center contractual budget and all the power limits are greater than or equal to the total minimum power budget of the servers under them, which is the fundamental requirement for the data center design equipped with power capping system.

$$Request(j) \le \sum_i Request_i(j)$$

Therefore,

$$Budget(j) \le \sum_i Request_i(j)$$

Further because for any priority higher than j,

$$Budget(k) = \sum_i Budget_i(k), \qquad \text{for any } k > j$$

we can realize the proposed algorithm is able to distribute $Budget(j)$ fully to the child controllers of this shifting controller as their budgets for priority j at Step 1, 2, and 3. In other words,

$$Budget(j) \le \sum_i Budget_i(j)$$

Since

$$Budget(j) \ne \sum_i Budget_i(j)$$

Hence,

$$Budget(j) < \sum_i Budget_i(j)$$

Therefore, when the shifting controller assigns $Budget_i(j)$, it must borrow some power from the power budgets that it receives for lower priorities (i.e., $Budget(k)$, for $k < j$). Hence, at some lower priority l ($l < j$), the shifting controller must receive a budget larger than its $Pcapmin(l)$. As the budgeting phase always tries to match budget with requested power from high priority to low priority, this implies that

$$Budget(k) = Request(k), \qquad \text{for } k \ge j$$

If $Request(j) = \sum_i Request_i(j)$, because

$$Budget_i(j) \le Request_i(j)$$

we have

$$\sum_i Budget_i(j) \le \sum_i Request_i(j) = Request(j) = Budget(j)$$

which contradicts

$$Budget(j) < \sum_i Budget_i(j)$$

If $Request(j) < \sum_i Request_i(j)$, as defined in Section 4.3.1,

$$Request(j) = limit - \sum_{k<j} Pcapmin\,(k) - \sum_{k>j} Request(k)$$

As

$$Budget(k) = Request(k), \qquad \text{for } k \ge j$$

we have

$$Budget(j) = limit - \sum_{k<j} Pcapmin\,(k) - \sum_{k>j} Budget(k)$$

Therefore,

$$\sum_{k\ge j} Budget(k) + \sum_{k<j} Pcapmin(k) = limit$$

As at a priority l ($l < j$), the shifting controller receives a budget larger than its $Pcapmin(l)$, we have

$$\sum_k Budget(k) > \sum_{k\ge j} Budget(k) + \sum_{k<j} Pcapmin(k) = limit$$

which contradicts the safety condition.

Therefore, the assumption we made in the beginning of this proof does not hold. And hence, Lemma 2.1 is proved.

Corollary 2.2: At any shifting controller, the aggregate power budget received by the controller at Step 1, 2, and 3 of the budgeting phase for a priority, is equal to the total power budget that the servers with this priority under this controller received (from the power feed that this controller resides) at Step 1, 2, and 3 of the budgeting phase.

Proof for Corollary 2.2: At any shifting controller, we can recursively apply Lemma 2.1 on this shifting controller and any shifting controllers below it (in the control tree where this shifting controller resides), and then we can see Corollary 2.2 holds.

Proof for Theorem 2: Use proof by contradiction. Let's make the following assumption: Suppose there are two servers - SA and SB. SA has a high priority A. SB has a lower priority B. After the algorithm allocates power budgets to the servers in the data center, SA is under power capping (i.e., throttled). Assume we can increase the power budget for SA and decrease the power budgets for SB and potentially other servers with lower priorities at one or multiple power feeds, without violating the safety condition.

Then let's look at the shifting controller under which both SA and SB resides. We call this controller as Controller C. Let's assume the total power budget for servers with priority k under this controller is $Budget(k)$ after using our algorithm to make budgets, and is $Budget'(k)$ after adjusting the server power budgets on top of the budgeting decision made by our algorithm ("adjusting the server power budgets" refer to increasing the power budgets for SA and decrease the power budgets for SB and potentially other servers with lower priorities). Then,

$$Budget'(A) > Budget(A)$$
$$Budget'(k) = Budget(k), \text{ for any } k > A$$

In the case that SB receives a power budget greater than its $Pcapmin(B)$ at Step 1, 2, and 3 of the budgeting phase, the priority B of Controller C must receive a power budget greater than the $Pcapmin(B)$ of Controller C at Step 1, 2, and 3 of the budgeting phase. Otherwise, the priority B of Controller C will receive a power budget equal to the $Pcapmin(B)$ of Controller C at Step 1, 2, and 3. Based on Corollary 2.2, as the power budget received by the priority B of Controller C at Step 1, 2, and 3 is equal to the total power budget received by servers with priority B under Controller C (from the power feed that Controller C resides) at Step 1, 2, and 3, the total power budget received by the servers with priority B under Controller C at Step 1, 2, and 3 is equal to the $Pcapmin(B)$ of Controller C, which is further equal to the total $Pcapmin(B)$ of these servers. Further as all the servers under Controller C receive a power budget no less than their $Pcapmin(B)$ at Step 1, 2, and 3, every server with priority B under Controller C must receive a budget equal to its $Pcapmin(B)$ at Step 1, 2, and 3, which contradicts with the fact that SB receives a power budget greater than its $Pcapmin(B)$ at Step 1, 2, and 3. Therefore, the priority B of Controller C must receive a power budget greater than the $Pcapmin(B)$ of Controller C at Step 1, 2, and 3 of the budgeting phase.

As our algorithm always prioritize the power allocation for priority A over priority B at Step 1, 2 and 3 of the budgeting phase, we can realize that Controller C must receive a power budget for priority A that is equal to the requested power of priority A ($Request(A)$) at Step 1, 2 and 3.

In the case that SB receives a power budget equal to its $Pcapmin(B)$ at Step 1, 2 and 3 of the budgeting phase, it must receive some extra power at Step 4, otherwise its total power budget received at Step 1, 2, and 3 will not be greater than its $Pcapmin(B)$. Therefore, Controller C must receive some extra power budget from its parent controller at Step 4. Otherwise, as the power budget received by Controller C for each priority at Step 1, 2, and 3 is always equal to the total power budget allocated for the servers with the same priority at Step 1, 2 and 3 (based on Corollary 2), SB will receive no extra power budget at Step 4, which is a contradiction. As Controller C receives some extra power budget at Step 4, the priority A of Controller C must receive a power budget equal to its $Request(A)$ at Step 1, 2, and 3 of the budgeting phase.

In either case, priority A at Controller C must receive a power budget that is equal to its $Request(A)$ at Step 1, 2, and 3 of the budgeting phase. Therefore, for any priority k higher than A at Controller C, it must also receive a power budget that is equal to its $Request(k)$ at Step 1, 2 and 3 of the budgeting phase.

Therefore, we have (please note $Budget(k)$ captures the power budget that priority k of Controller C receives at Step 1, 2, 3, and 4):

$$Budget(k) \geq Request(k), \quad \text{for any } k \geq A \tag{5}$$

As

$$Budget'(A) > Budget(A)$$

$$Budget'(k) = Budget(k), \quad \text{for any } k > A$$

We have

$$Budget'(A) > Request(A), \tag{6}$$

$$Budget'(k) \geq Request(k), \quad \text{for any } k > A, \tag{7}$$

In order to satisfy the safety condition,

$$Budget'(k) \geq Pcapmin(k), \quad \text{for any } k < A, \tag{8}$$

If $limit - \sum_{k>A} Request\,(k) - \sum_{k<A} Pcapmin\,(k) \leq \sum_i Request_i(A)$ ($Request_i(A)$ is the requested power for priority A at the child controller i of controller C), then

$$Request(A) = limit - \sum_{k>A} Request\,(k) - \sum_{k<A} Pcapmin\,(k)$$

Therefore, further based on equation (6)(7)(8),

$$\sum_k Budget'(k) > \sum_{k\geq A} Request(k) + \sum_{k<A} Pcapmin\,(k) = limit$$

which violates the safety condition.
Therefore,

$$limit - \sum_{k>A} Request\,(k) - \sum_{k<A} Pcapmin\,(k) > \sum_i Request_i(A)$$

And hence,

$$Request(A) = \sum_i Request_i(A)$$

Priority A at Controller C receives a power budget equal to the $Request(A)$ of Controller C at Step 1, 2, and 3 of the budgeting phase. This budget is fully given to the child controllers of Controller C as budgets for priority A at Step 1, 2, and 3 (based on Lemma 2.1). As each child receives a budget for priority A up to $Request_i(A)$ at Step 1, 2, and 3 of the budgeting phase, and $Request(A) = \sum_i Request_i(A)$, we can realize that each child receives a budget for priority A equal to its $Request_i(A)$ at Step 1, 2, and 3 of the budgeting phase.

Let Controller C1 be the child controller of Controller C that contains server SA under it. Assume the total power budget for servers with priority k under this controller is $Budget_{C1}(k)$ after using our algorithm to make power budgets, and is $Budget'_{C1}(k)$ after adjusting the server power budgets on top of the budgeting decision made by our algorithm. Then we have

$$Budget'_{C1}(A) > Budget_{C1}(A)$$

$$Budget'_{C1}(k) = Budget_{C1}(k), \quad \text{for any } k > A$$

As priority A of C1 receives a budget equal to the $Request_{C1}(A)$ of Controller C1 at Step 1, 2, and 3 of the budgeting phase, and as our algorithm allocates power from high priority to low priority at Step 1, 2, and 3 of the budgeting phase, we have

$$Budget_{C1}(k) \geq Request_{C1}(k), \quad \text{for any } k \geq A, \tag{9}$$

Therefore,

$$Budget'_{C1}(A) > Request_{C1}(A), \tag{10}$$

$$Budget'_{C1}(k) \geq Request_{C1}(k), \quad \text{for any } k > A, \tag{11}$$

In order to satisfy the safety condition,

$$Budget'_{C1}(k) \geq Pcapmin_{C1}(k), \quad \text{for any } k < A, \tag{12}$$

Please note equation (9), (10), (11), and (12) are the same as equation (5) ,(6), (7), and (8), except that equation (9), (10), (11), and (12) characterize the relations at Controller C1. Therefore, based on the similar reasoning process, we can realize that at Controller C2, which is the child controller of Controller C1 that contains server SA under it, we have the following relations,

$$Budget_{C2}(k) \geq Request_{C2}(k), \quad \text{for any } k \geq A, \tag{13}$$

$$Budget'_{C2}(A) > Request_{C2}(A), \tag{14}$$

$$Budget'_{C2}(k) \geq Request_{C2}(k), \quad \text{for any } k > A, \tag{15}$$

$$Budget'_{C2}(k) \geq Pcapmin_{C2}(k), \quad \text{for any } k < A, \tag{16}$$

Apply this reasoning process recursively on the child controllers that contains server SA under them, eventually we can prove that at server SA,

$$Budget_{SA}(k) \geq Request_{SA}(k), \quad \text{for any } k \geq A$$

However, as we know that SA is under power capping, so

$$Budget_{SA}(A) < Demand_{SA}(A) \leq Request_{SA}(k)$$

Hence, there is a contradiction.

Therefore, the assumption we made in the beginning of this proof does not hold. And hence, Theorem 2 is proved.