Increasing Large-Scale Data Center Capacity by Statistical Power Control

Guosai Wang[†], Shuhao Wang[†], Bing Luo[‡], Weisong Shi[‡],
Yinghang Zhu[^], Wenjun Yang[^], Dianming Hu[^], Longbo Huang[†], Xin Jin[*], Wei Xu[†]

[†]Institute for Interdisciplinary Information Sciences, Tsinghua University, China
[‡]Department of Computer Science, Wayne State University, USA
[^]Baidu, Inc., China
[*]Deparement of Computer Science, Princeton University, USA

Abstract

Given the high cost of large-scale data centers, an important design goal is to fully utilize available power resources to maximize the computing capacity. In this paper we present Ampere, a novel power management system for data centers to increase the computing capacity by over-provisioning the number of servers. Instead of doing power capping that degrades the performance of running jobs, we use a statistical control approach to implement dynamic power management by indirectly affecting the workload scheduling, which can enormously reduce the risk of power violations. Instead of being a part of the already over-complicated scheduler, Ampere only interacts with the scheduler with two basic APIs. Instead of power control on the rack level, we impose power constraint on the row level, which leads to more room for over provisioning.

We have implemented and deployed Ampere in our production data center. Controlled experiments on 400+ servers show that by adding 17% servers, we can increase the throughput of the data center by 15%, leading to significant cost savings while bringing no disturbances to the job performance.

Categories and Subject Descriptors C.0 [*Computer Systems Organization*]: General - System architectures; C.4 [*Computer Systems Organization*]: Performance of Systems - Design studies, Measurement techniques, Modeling techniques.

Keywords power provisioning, data center, power control, scheduling

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, contact the Owner/Author(s). Request permissions from permissions@acm.org or Publications Dept., ACM, Inc., fax +1 (212) 869-0481.

EuroSys '16, April 18 - 21, 2016, London, United Kingdom
Copyright © 2016 held by owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4240-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2901318.2901338

1. Introduction

Data centers, or warehouse-sized computers, are becoming the typical way to host interactive Internet applications such as search engines, social networks and e-commerce sites, all of which have strict Service Level Agreements (SLAs), especially on the latency. The same set of machines is also used to run batch jobs such as analyzing the data generated from these services, and the overall job throughput is important for these tasks [5]. Data centers are extremely expensive to build. The industry average building cost is about 10,000-20,000 USD per kilowatt [24].

Given the high cost, it is important to fully utilize the capacity of data centers to reduce the Total Cost of Ownership (TCO). There are multiple factors constraining the *capacity* of a data center. The data center is designed with a *space budget* and *power budget*. With the adoption of high-density rack designs such as Scorpio [45] and Open Compute Project [36], the capacity of a modern data center is usually limited by the power budget. The power budget of a data center is statically partitioned into dozens of row-level Power Distribution Units (PDUs) and then into about 20 racks per row.

Despite of the power budget in the design specifications, what matters the most to the software applications is the *computation capacity* of a data center, i.e. how much computation resources (CPU cycles, memory, storage space) the data center can provide without violating the SLAs. In a data center with fixed power budget, increasing computation capacity can result in significant cost savings, which is the goal of this work.

Unfortunately, the power budget in a data center is often not fully utilized. As studies [14, 24, 32] point out, the typical power utilization is around only 60% of the provisioned power, which effectively doubles the cost of data centers. We found similar utilization numbers in our own data centers.

The main reason for the under-utilization is conservative server provisioning. People commonly ensure that the sum

of the *rated power*[1] of all equipment does not exceed the power budget. However, with modern power saving mechanisms, actual power draw from a server depends on its utilization, which seldom reaches the peak level [4, 16, 31]. The low utilization is due to the variations in workload, as well as trying to guarantee the SLAs for latency-critical services. In other words, servers are provisioned according to the worst case power draw, while they operate at the average case.

We call the difference between the rated power and the actual power the *unused power*. Unused power and the under-utilized rack space provide an opportunity to add extra servers to a data center. However, operators hesitate to use this power due to the concern that power draws exceeding the power budget can lead to outage. We call the situation of exceeding the power budget a *power violation*. Existing power management technologies such as power capping [11] alleviate the damage of power violations, but as we will show in Section 4.3, these approaches may affect the SLAs, preventing operators from adopting them. We define the *over-provisioning ratio* r_O as $P/P_M - 1$, where P is the total equipment rated power and P_M is the total provisioned power budget (we use the terms *power limit* and provisioned power budget interchangeably in this paper).

It is non-trivial to cultivate the unused power, due to static power partitions and high variations of workload [14]. As we will show in Section 2.2, we have a better opportunity to take advantage of the unused power at a larger scale, but the physical capacity of the PDUs limits the power partitioning scheme. We show that using our indirect control, we can virtually consolidate unused power at a larger scale.

Note that a higher r_O does not mean higher computation capacity. As an extreme case, if the sum of the minimal idle power of all equipment reaches the power limit, we cannot run any computation. Similar to [15], we use the metric *Throughput per Provisioned Watt* (TPW) (Section 4.1 provides more details) to capture the gain in capacity under a given power budget.

In this paper, we present Ampere, a new approach to improve TPW by provisioning extra servers. Ampere keeps the total power under the budget and brings zero performance disturbance to the existing jobs. Different from existing approaches that react to an over-committing event by capping the power draw, Ampere proactively reduces power violations by driving the workload to other less utilized rows and consolidating the unused power scattered among them. The approach is conceptually simple: by scheduling fewer jobs to rows with less unused power or letting them wait in the scheduler queue, we can reduce the amount of power increase and prevent power violations. However, we face several challenges implementing this scheme into a production data center.

First, the data center job scheduler is a very complicated system. Different data centers use different scheduling policies to achieve diverse goals. It is hard to have a general solution that directly adds power scheduling into these already-over-complicated policies.

To solve this problem, we limit our interface with the job scheduler to two simple operations: `freeze` a server and `unfreeze` a server. Freezing a server advises the scheduler not to assign new jobs to the server (the existing jobs are unaffected), while unfreezing does the opposite. By controlling the number of frozen servers on a row, we can *statistically* affect the number of jobs scheduled there without changing the scheduling policy.

The second challenge is how to estimate the number of servers we need to freeze. The goal is to freeze as few machines as possible to minimize the negative impact on scheduling and overall computation capacity, while keeping the row-level power below the budget.

We solve this challenge with a *data-driven* approach. We collect data from production data centers, and derive a statistical model on the effects of freezing servers on the power consumption. We compute the number of servers to freeze at each time interval based on the model. Obviously the model is not perfect, and we use Receding Horizon Control (RHC) techniques [26] to continuously adjust the number of frozen servers to compensate for the inaccuracy of the model.

We have deployed Ampere in a production data center with tens of thousands of servers running millions of jobs per day. In this paper, we present results on a number of controlled experiments with 400+ machines to demonstrate the performance of Ampere. We demonstrate that even with a conservative strategy, we can add 17% more servers into the fleet and get a 15% improvement in the effective computation capacity comparing to the provisioning based on rated power without any violation. This ratio translates to tens of thousands of extra server spaces across our fleet.

In summary, our major contributions are

- A new system, Ampere, that increases the computation capacity in large-scale data centers with fixed power budget by cultivating the otherwise unused power at a large scale;
- A new effective method to indirectly control the data center power by statistically influencing the amount of jobs scheduled to a row;
- A simple yet powerful interface to connect the power controller with the job scheduler, without modifying the job scheduler logic;
- A large-scale empirical evaluation of Ampere in a real data center with production workload, and detailed performance measurement using controlled experiments.

[1] Following the definition in [14], we use the rated power, or the measured maximum power draw from equipment, instead of the name plate power that is often higher.

In the remainder of the paper, Section 2 introduces the background on data center power characteristics. We show our system design, focusing on the controller model in Section 3 and evaluate the system performance empirically in Section 4. We review the related work in Section 5 and conclude in Section 6.

2. Background on Data Center Power Provisioning

In this section, we provide some background information on the data center power supply architecture, job management and some important observations of data center power utilization, which lead to the Ampere design.

2.1 Data center power provisioning and job scheduling

Row-level power provisioning. The power budget of a data center is normally partitioned into a number of PDUs, each of which serves about 20 racks. Each rack has a power budget of 8-10KW. As the typical rated peak power of a server is about 250W, we can have 40 servers per 10KW rack. This translates to 800 servers per PDU, which we call a *row* of servers. The partitioning is due to the physical limits of commercial power equipment, such as Uninterruptible Power Supply (UPS) and PDUs.

Servers are provisioned according to the power budget at each level. The provisioning is often based on the rated power and we will show that it leads to significant under-utilization of the provisioned power budget.

Power capping. The row-level power budget is enforced by physical circuit breakers (fuses) in each PDU, in order to protect the PDU from overloading. Since it would cause catastrophic service disruptions to cut down the power of hundreds of servers at the same time, power capping is used when the total power utilization of servers in a row is over the row budget. Power capping uses Dynamic Voltage and Frequent Scaling (DVFS) features provided by modern server hardware, and slows down the servers to reduce the power draw [17]. The recently proposed DVFS technique, running average power limit (RAPL) [11], reacts in a very short period of time ($<$ 1ms) to avoid triggering the circuit breaker at the higher level. The downside of DVFS is that it slows down a server without informing applications or the scheduler, which may cause unpredictable performance disturbances and SLA violations (details in Section 4.3). We have DVFS enabled in our data centers, but using Ampere, we dramatically reduce the cases where DVFS is triggered.

Job scheduling. Independent of the statically-partitioned power budget, jobs in a data center are scheduled by a centralized scheduler using the entire data center as a single resource pool. Data center job schedulers, such as Borg [51], Omega [44], Mesos [18] and YARN [48] track the utilization of various resources including CPU, memory and storage, and allocate them to different applications. Modern job schedulers allow complex and application-specific scheduling policies, and apply advanced optimization algorithms to achieve a variety of scheduling objectives.

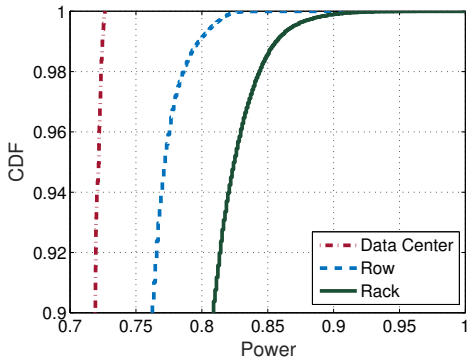


Figure 1. The CDF of the power utilization normalized to the provisioned power budget on rack, row and data center levels.

The scheduler in our data center is a custom system similar to Omega [44]. It is a two-level scheduler. The low level tracks the status of resources, bundles them into abstract resource containers and provides the containers to the upper level. The upper level is application-specific and decides how to efficiently allocate containers to jobs.

`Freeze` and `unfreeze` are two APIs provided by the lower level of the job scheduler. `Freeze` makes a server unavailable (frozen), so that the lower level can no longer add it to the candidate list. On the contrary, `unfreeze` makes a frozen server available again. In Ampere, these APIs enable us to control power indirectly by workload scheduling. We believe both APIs are simple enough to implement in any scheduler, making our approach generally applicable.

2.2 Characteristics of data center power utilization

We have the following important observations of data center power utilization, which directly lead to our design.

First, the average power utilization is low in the data center. Specifically, the utilization is lower at a larger scale. Figure 1 shows the cumulative distribution function (CDF) of the observed power utilization in one of our production data centers for a week at rack, row and data center levels. The servers are provisioned based on the rated power. We can see that the average power utilization at the data center level is only 70%, wasting almost one third of the available power budget. We emphasize that the under-utilization is not due to lack of workload demand, as there are often jobs waiting in the scheduler queue and the company is still building out new data center facilities to meet the increasing demand.

Intuitively, like in many systems with statistical multiplexing, it would be desirable to consolidate, rather than statically partition the power at the data center level as a single

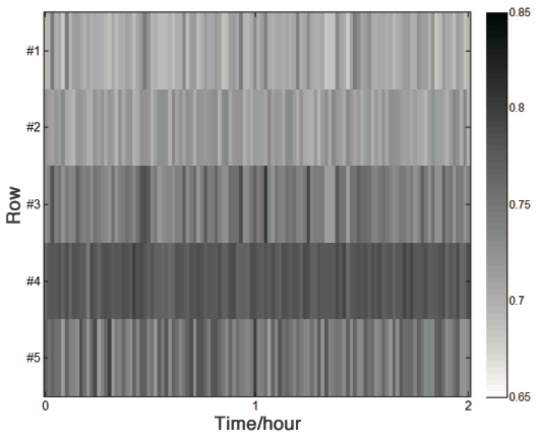


Figure 2. Row power of five randomly chosen rows during a two-hour period. The grayscale represents the power utilization. We can see both temporal and spatial variations.

pool [14]. Our system helps to indirectly achieve the consolidation.

To show the source of the unused power, we provide a formal notation of the power at level $X \in \{\text{row, rack}\}$. Assume that there are n homogenous servers at X level and the rated power of each server is P_m, and P_M is the provisioned power budget of X, as defined in Section 1, then we have $P_M = nP_m$. At runtime, it is unlikely that all servers are at their rated power simultaneously, and the total runtime power of the level will generally be lower than nP_m. Thus we define the unused power $\overline{P}_t^X$ of level X at time t as

$$\overline{P}_t^X = P_M - \sum_i^n P_{it} \quad (1)$$

where P_{it} is the realtime power of the i-th server in level X at time t. Obviously the unused power at row level is always no lower than that at rack level, as $\overline{P}_t^{row} = \sum_{rack \in row} \overline{P}_t^{rack}$.

The second observation is that there are large variations on power utilization at the row level. The variations are both temporal (over time) and spatial (across different rows), and Figure 2 shows the variations. We see that the power draw across different rows is highly unbalanced. The reason for this imbalance is that different rows mainly focus on running different sets of products. Also the power across these rows shows weak correlations over time (80% of the correlation coefficients are under 0.33). The variations and imbalance in workload provide us with opportunities to dynamically schedule power to where it is required.

3. Ampere Design and Implementation

In this section, we first discuss the important design choices and an overview of the Ampere architecture. Then, we focus on the controller, introducing the variable under control and its effects on the power, as well as the control algorithm. Finally we provide details about our controller model.

3.1 Design choices and rationales

We have the following four important design choices.

Managing power at the row level. We decide to manage power at the row level because (1) it matches the row-PDU physical fuse configuration in our hardware; (2) there is a larger amount of unused power at the row level than at the rack level, as discussed in Section 2.2; (3) there are abundant servers and tasks at row level, enabling our probabilistic control mechanism; (4) we can leverage the unbalanced power draw across different rows, and statistically direct jobs to different rows with optimal power conditions.

Minimal interface with the scheduler. To implement power-aware scheduling, one straightforward design would be making the scheduler power distribution aware. However it is not practical mainly due to the complexity of incorporating the information into different scheduling policies, especially those application-specific schedulers.

Thus, Ampere does not read any data from the scheduler and only requires the `freeze`/`unfreeze` interface. This enables Ampere to easily integrate with different schedulers and scheduling policies.

Power control with statistical influence on new job placement. Using the `freeze`/`unfreeze` API, we can affect the probability of placing new jobs to specific rows, and thus control the power usage. This has no impact on the performance of existing jobs. Furthermore, by driving away job assignments from a row, we leave the choice of where to put such jobs to the scheduler, allowing it to take advantage of the existing policies. This is equivalent to creating a virtual pool of unused power for the scheduler.

Using simple system model and tolerating inaccuracy with control. We use data-driven predictive models to characterize the potential impact on realtime power of our control activities. Given the statistical nature of our control input, we observe high variations on the effects of the control input. Instead of demanding a very precise model as most power capping approaches do, we use RHC to periodically obtain optimized control decisions and correct the random errors in our system model.

3.2 Ampere architecture

Figure 3 shows the architecture of Ampere. An in-house developed power monitor collects and aggregates the power utilization at the server, rack and row level. The centralized controller implements most functionality of Ampere. For each minute, the controller reads the data from the database, computes the number of servers to freeze in the next time period, and uses the `freeze`/`unfreeze` interface to advise the scheduler to freeze them. The data center operator can set a control target for the maximum allowed power budget, which can be lower than the physical limit, to provide an

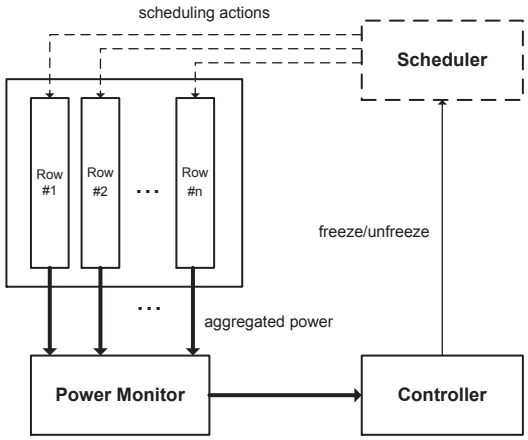


Figure 3. System architecture of Ampere.

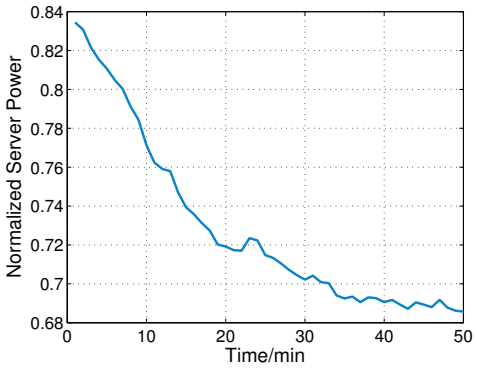


Figure 4. Power drops over time when a server is frozen. The figure shows the average power normalized to the rated power of about 80 servers at different time points after being frozen. The noises on the curve are due to the randomness of the workload on the servers.

extra safety margin. Note that the controller is stateless, and thus if the controller fails, we can easily switch to a replacement.

3.3 Power monitoring

We implement our own power monitor, which collects server-level power utilization, among other metrics through the intelligent platform management interface (IPMI). We leverage our in-house streaming computation framework to aggregate the data to provide the row-level power. We store the history data in a MySQL database and export a RESTful API for efficient query against these data. The power monitor samples the power from every server at every minute, and stores the numbers in a time series database. We rely on the time series database to provide data persistence and failover. Our power monitoring service remains stateless for easy recovery. We believe one minute is a good tradeoff between measurement accuracy and monitoring overhead.

3.4 Interface to the job scheduler

We use the `freeze`/`unfreeze` API to indirectly control job scheduling. When we freeze a server, it has two effects: (1) the power of frozen servers will go down over time, because the existing jobs will finish; (2) there will be statistically fewer jobs scheduled to the row with frozen servers, so the power increase will slow down.

To examine the first effect, we randomly select a group of about 80 servers with relatively high power utilization, freeze them for a period of time, and observe their power drop. Figure 4 displays the average power change over time. We can see the power gradually drops to the minimum (close to the idle power) after about 35 minutes.

As for the second effect, the number of jobs scheduled to a row is roughly proportional to the number of available servers of the row, assuming that there are multiple rows with different workloads. As we have discussed in Section 2.2, the assumption is generally true for our case. Thus, freezing a percentage of servers will likely reduce the number of *new* jobs assigned to the row, lowering the power increase during the next time period.

These two effects impact the row-level power jointly. We define the *freezing ratio* u_t as the percentage of frozen servers in the total number of servers in a row at a given time t. We want to identify a function $f(u_t)$ given a specific over-provisioning ratio r_O to quantify the effect of freezing u_t servers.

We empirically identify the impact of u_t on the row power using a controlled experiment. We will describe the detailed setup in Section 4.1. We denote the power of the control group and the experiment group at time t as P_t^C and P_t^E respectively. We set up the experiment so that $P_t^C = P_t^E$ without power control. In other words, the only cause of the difference between P_t^C and P_t^E is the control input u_t. With this setup, we can express $f(u_t)$ as $f(u_t) = P_{t+1}^C - P_{t+1}^E$.

In order to collect data to evaluate $f(u_t)$ using a regression model, we set u_t to a variety of different values over a period of 24 hours, and measure the power of the experiment group and the control group in the controlled experiment on a cluster with about 400 servers. Figure 5 shows the measurement result. We approximate $f(u_t)$ using a linear function $y = k_r x$ where k_r is a parameter dependent on r_O. We can use this simple model because our RHC mechanisms can help correct errors in the model over time.

We also want to point out that the linearity assumption of $f(u_t)$ helps us simplify our controller model greatly, as we will discuss in Section 3.6.

3.5 Controller

With the `freeze`/`unfreeze` API, we implement a controller that periodically adjusts the power draw of a single row so that it stays under the power budget.

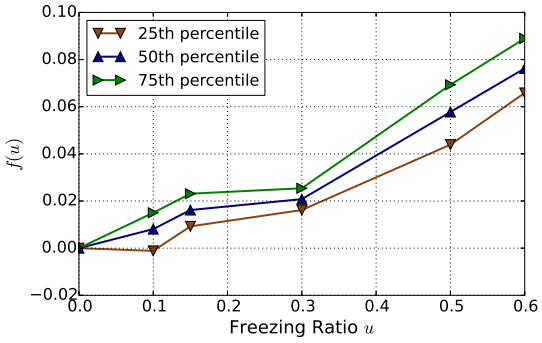


Figure 5. The effects of freezing ratio u on the power change $f(u)$. We plot the 25th, 50th and 75th percentile for $f(u)$ under different u. $f(u)$ is r_O-dependent, so we plot the normalized values.

Algorithm 1 shows the control logic. At each interval (one minute in our implementation), we obtain the power utilization from the power monitor, and compute the unused power (as defined in Section 2.2). Then we compute how many servers of each row to `freeze`, using the control model we will discuss in Section 3.6. Finally, we select a set of servers to `freeze` or `unfreeze`.

We prefer to freeze servers with highest power draw mainly because servers with lower power utilization may have more computation capacity left and thus freezing them may result in a higher cost. To avoid freezing and unfreezing a server too frequently, we introduce the r_{stable} parameter to the algorithm. The algorithm will only `unfreeze` a server and `freeze` another one if the server's power drops by at least $(1 - r_{stable})$. We find that the value of r_{stable} does not affect the performance much, and we choose $r_{stable} = 0.8$ in all of our experiments.

We take a control action every minute, which is an interval matching our power monitoring frequency. Note that the controller cannot monitor or control any power fluctuation within a minute, imposing a risk of short-term power violations. This is why we still have DVFS-based hardware power capping on as a safety-net against these rare cases.

3.6 Computing the percentage of frozen servers

The most important decision of the controller is the number of servers to freeze. We want to freeze enough servers to avoid power violations, and in the mean time, freeze as few as possible to minimize the negative impact on computation capacity. We discuss how we obtain the number of servers to freeze in this section. We first formulate the problem of computing the optimal number of servers to freeze in a general form of a receding horizon control (RHC) problem [26], and then use heuristics based on data-driven observations to reduce the RHC problem to a simplified version. Table 1 summarizes the key notations we use in the problem formulation.

Input:
- P_M: Power limit
- $r_{threshold}$: Threshold ratio
- r_{stable}: Stable ratio
- P_k: Current power of row k
- p_s: Current power of server s
- $S_f[k]$: The set of frozen servers at row k
- n_k: The number of servers in row k
- $F(\cdot)$: The function from row power to freezing ratio

Output: Updated $S_f[k]$ for each row k

procedure POWER CONTROLLING
  for $k \leftarrow 1, N$ **do** ▷ N is the number of rows
    create set S ▷ candidate servers for freezing
    if $P_k/P_M > r_{threshold}$ **then**
      $n_{freeze} \leftarrow \lfloor F(P_k/P_M) \cdot n_k \rfloor$
      $S \leftarrow n_{freeze}$ servers with highest power
      $p_{threshold} \leftarrow r_{stable} \cdot \min_{s \in S} p_s$
      for all s s.t. s is in row k, $s \notin S$ **do**
        if $p(s) > p_{threshold}$ **then**
          $S.add(s)$ ▷ for stability
      for all $s \in S_f[k] - S$ **do**
        unfreeze s, update $S_f[k]$
      if $|S_f[k]| > n_{freeze}$ **then**
        unfreeze arbitrary $|S_f[k]| - n_{freeze}$ servers, update $S_f[k]$
      else if $|S_f[k]| < n_{freeze}$ **then**
        freeze $n_{freeze} - |S_f[k]|$ servers with highest power in $S - S_f[k]$, update $S_f[k]$
    else
      unfreeze all servers, $S_f[k] \leftarrow \emptyset$
  return S_f

Algorithm 1: Power controlling algorithm

The general model. The idea of controlling power using RHC is as follows. At each time t, we calculate an optimal control $U_t = \{u_t, u_{t+1}, \cdots, u_{N-1}\}$ on a finite fixed horizon, starting at time t, say $[t, t + N - 1]$ (N is a parameter representing how much time ahead we want to predict). We only carry out the first control u_t, that is, freezing u_t servers.

Symbol	Description
U_t	The control actions obtained at time t.
u_t	The percentage of servers to be frozen at time t.
P_t	The normalized row power draw at time t.
P_M	The normalized provisioned row power budget (= 1.0).
E_t	The normalized power increase at time t.
$f(u_t)$	The relative reduction of power by the control u_t.
$C(U_t)$	The cost function of U_t.
k_r	The gradient of the function that fits $f(u_t)$.

Table 1. Key notations in problem formulation. All power metrics used in the problem formulation is normalized to P_M.

We repeat this computation at each time t, and the "horizon" of the control recedes as the time proceeds.

Suppose P_t is the row-level power at time t. We introduce E_t to denote the power demand increase, which is a predicted value that indicates the first order difference of row-level power. Say the predicted power for time $t+1$ is $P_{t+1}^{predict}$, then $E_t = P_{t+1}^{predict} - P_t$. The change of power is basically affected by the temporal variation of workload. Instead of implementing a predictor, we show how we use heuristic method to estimate E_t in the later part of the section.

We use the function $f(u_t)$, as we have described in Section 3.4, to model the effect of frozen servers on row power. Thus we have $P_{t+1} = P_t + E_t - f(u_t)$.

We use $C(U_t)$ to denote the cost function, which indicates the degradation of computing capacity or other performance metrics due to freezing servers. We use a simple linear combination of u_t and model the cost function as

$$C(U_t) = \sum_{t \le k \le t+N-1} u_k. \tag{2}$$

Therefore, we formulate the Power Control Problem (**PCP**) as

$$\begin{aligned}
\min \quad & C(U_t) = \textstyle\sum_k u_k && (3)\\
\text{s.t.} \quad & P_{k+1} \le P_M && (4)\\
& P_{k+1} = P_k + E_k - f(u_k), && (5)\\
& 0 \le u_k \le 1 && (6)\\
& k = t, \cdots, t+N-1 \text{ for (3)-(6).}
\end{aligned}$$

PCP is a typical RHC problem. Note that we do not need to assume $f(u_t)$ linear. Section 3.4 provides the method to empirically evaluate $f(u_t)$. Given a series of predicted E_k, there are many methods and tools to compute the solution of this RHC problem if a solution exists [1, 23, 26, 39].

Control model simplification. Instead of solving the general problem directly, as in our case, the function $f(u_t)$ is close to linear, we can reduce **PCP** to a much simpler problem. Using the empirically obtained $f(u_t)$ that can be approximated by a linear function $y = k_r x$, we get

$$f(u_t) = k_r u_t. \tag{7}$$

In this way, we replace Eq. (5) by

$$P_{k+1} = P_k + E_k - k_r u_t. \tag{8}$$

With the linear function $f(u_t)$ we define a simplified **PCP** (**SPCP**) as follows:

$$\begin{aligned}
\min \quad & C(u_t) = u_t && (9)\\
\text{s.t.} \quad & P_{t+1} \le P_M && (10)\\
& P_{t+1} = P_t + E_t - k_r u_t && (11)\\
& 0 \le u_t \le 1. && (12)
\end{aligned}$$

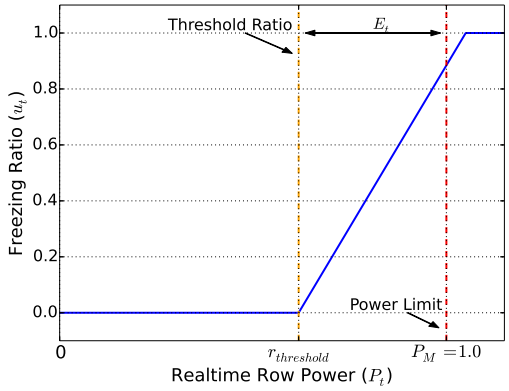


Figure 6. The control function F from P_t to u_t. The threshold $r_{threshold}$ is defined by $P_M - E_t$. Note the curve varies for different E_t and k_r.

This is a special case of RHC problem in that the distance to horizon is 1. Assuming there is a feasible solution, the constraints and the object function are all linear so it is very easy to obtain the optimal solution, which is

$$u_t = \max\{\min\{(P_t + E_t - P_M)/k_r, 1.0\}, 0\}. \tag{13}$$

Empirically, $E_t - 1.0 \cdot k_r \le 0$, which means that if all servers are frozen, the row-level power will not rise. Assuming **PCP** has feasible solutions (and thus **SPCP** also has feasible solutions), and denoting the optimal solution of **SPCP** as u'_{t_0} (Eq. (13)), we prove the following lemma:

Lemma 3.1. *$U'_{t_0} = \{u'_{t_0}, u'_{t_0+1}, \cdots, u'_{t_0+N-1}\}$ is the optimal solution of **PCP**, where u'_i ($i = t_0, \cdots, t_0+N-1$) is the optimal solution of **SPCP** in which $t = i$, $E_t = E_i$, and $P_t = P_{t_0} + \sum_{k<i}(E_k - k_r u'_i)$.*

Proof. Without the loss of generality, we assume that $E_k \ge 0$ ($k = t, \cdots, t+N-1$) and $P_t + \sum_k E_k > P_M$. We can easily verify U'_{t_0} is a feasible solution to **PCP**. Assuming that $U^*_{t_0} = \{u^*_{t_0}, \cdots, u^*_{t_0+N-1}\}$ is an optimal solution to **PCP**, by applying Eq. (8) iteratively for all k, we get

$$P_{t_0} + \sum_k E_k - k_r \sum_k u^*_k = P_{t_0+N} \le P_M. \tag{14}$$

Therefore

$$C(U^*_{t_0}) = \sum_k u^*_k \ge (P_{t_0} + \sum_k E_k - P_M)/k_r \ge C(U'_{t_0}). \tag{15}$$

Thus, U'_{t_0} is the optimal solution of **PCP**. □

This reduction greatly simplifies the problem: we only need to optimize the freezing ratio for a horizon at a distance of 1 at each time t.

In the optimal control strategy, the predicted power demand E_t defines a safety margin $[1.0 - E_t, 1.0]$. That is, if

the power P_t is below a threshold $r_{threshold} = 1.0 - E_t$, we do not need any control as there is unlikely a imminent power violation. However, if $P_t > r_{threshold}$, the closer the current power is to the power limit, the more servers we will freeze. We call $r_{threshold}$ the *threshold ratio*. Figure 6 shows the intuition and relationship among $r_{threshold}$, P_M and E_t.

Estimate the power change E_t**.** In order to avoid power violation due to a sudden power surge, we need to leave a safety margin. The estimated E_t determines the margin, as it indicates the expected power increment during the next minute. We use a data-driven approach to estimate E_t. We would like to keep E_t small to improve the power utilization.

We monitor the power of all rows in our data center for a long time, and collect the power increase for every minute. We discover that the distribution of power increase varies for different hours in a day, so we calculate the 99.5-percentile power increase for each hour and use the one matching the hour of t to estimate E_t. By experiments we find that Ampere's performance is not sensitive to E_t. Nevertheless, our E_t estimation is conservative as we are preparing for almost the largest change in observed history (99.5th percentile). We can use a better online power prediction model to get a better estimation, which we leave for future work.

4. Evaluation

In this section, we present the evaluation results from a production data center. We first introduce the experiment setup and characterize the workload. Then we show the effectiveness of Ampere and its advantage over existing power capping methods. Finally we evaluate the effects of various parameter choices in Ampere.

4.1 Experiment setup

In this section, we briefly introduce the cluster setup, the production workload properties and our controlled experiment setup that allows us to perform experiments on a production cluster.

4.1.1 Cluster setup and workload

We have implemented Ampere in one of our production data centers. Here we present results on a single row with 400+ homogeneous servers. All servers in this row are part of a datacenter-wide resource pool managed by a single job scheduler as described in Section 2.1. We only focus on the power draw from servers, as other devices like network switches consume only a negligible fraction of the total power.

All servers run production workload comprised of mainly batch jobs (e.g., Map-reduce tasks). We add interactive jobs to the cluster to evaluate the effectiveness of Ampere, and we show the results in Section 4.3. The durations of these batch jobs vary and Figure 7 shows the CDF of the job durations. The average job duration is about 9 minutes, and about 40% jobs finish in 2 minutes. The arrival rate of jobs in the cluster

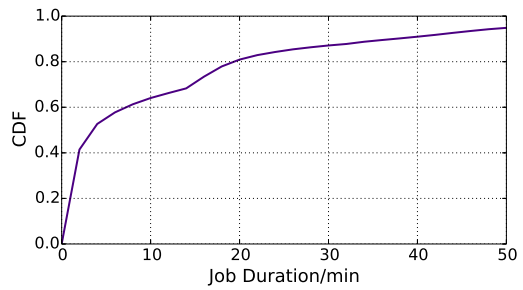


Figure 7. The CDF of batch job durations in the production cluster.

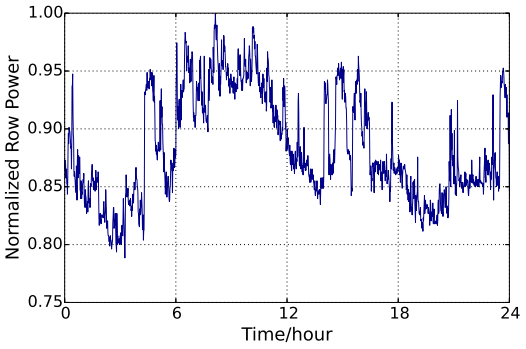


Figure 8. The power of a row in 24 hours, normalized to the maximum power. The values on the X-axis do not correspond to actual wall clock.

also varies a lot over time, and usually the rate is 400-600 jobs per minute. The variations of job durations and arrival rate make our probabilistic control more effective, as there is a good chance that some job will finish on some frozen machine, reducing the power utilization.

As we describe in Section 2.2, the row level power that is directly affected by the workload on the row, also varies significantly over time. Figure 8 shows the power utilization in a twenty-four hour period with a reading every minute. There are two observations:

1) At a larger time scale (hours), we observer high variations. This larger-scale variations leave us with room at many time periods to over-provision servers and keep the power below the daily peak.

2) At a smaller time scale (a few minutes), we can see many spikes and valleys on power utilization. It is hard to predict these spikes. To better characterize these spikes, we plot the CDF of the first order differences of the power (the power changes at 1-minute scale) in Figure 9. We can see that within a single minute, the power change is generally small (smaller than +-2.5% for 99% of the time), but it can be a change as large as 10%. We design the approach described in Section 3.6 to handle these relatively large spikes. Section 4.2 shows the results.

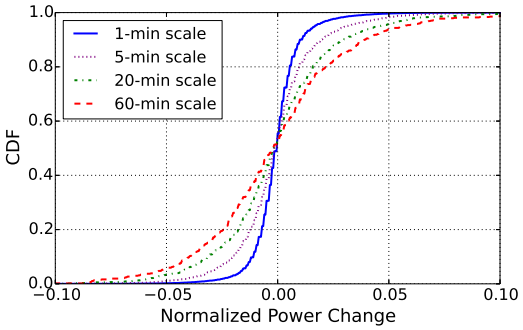


Figure 9. The CDF of the power changes of the control group on various time scales. For the k-minute scale, we compute a sequence of the maximum power for every k minutes, and then plot the CDF of the first order differences of the power sequence. The power changes are normalized to the provisioned power budget.

Considering some operational maintenance issues of the scheduler, we limit the maximum ratio of freezing servers to 50%. This limitation causes a few power violations, as we will show later in the section.

4.1.2 Controlled experiment design

As we cannot isolate a large number of servers to conduct trace-based experiments to compare the system performance with or without Ampere, we perform a controlled experiment. We partition the servers into two virtual groups, the *experiment group* and the *control group*. Specifically, we partition them based on the parity of the server IDs and thus a server is assigned to a group in a uniformly random way. Both groups accept jobs from the same scheduler and thus statistically they should have similar workload. We verify the fact by calculating the average power and the correlation coefficients of power of the experiment and control groups (with Ampere turned off) over five days. The difference between the average power is less than 0.46%, and the correlation coefficient of the power is 0.946. Thus, we can safely assume that any differences between these two groups are results of the control actions from Ampere.

We turn off the power capping so we can observe the real power demand. In order not to cause real-world power violation, we emulate the power violation events by scaling down the power budget of these servers. Consider we set the power budget to P'_M instead of the actual P_M, with N_r servers per rack, we can emulate the case where in each rack, $\lfloor P'_M/P_m \rfloor$ of the servers are designed to be provisioned, and the other $N_r - \lfloor P'_M/P_m \rfloor$ servers are over-provisioned. Thus we can calculate the over-provisioning ratio r_O as

$$r_O = N_r/\lfloor P'_M/P_m \rfloor - 1 = P_M/P'_M - 1. \quad (16)$$

We use the scaling-based emulation differently in our evaluations. In Section 4.2, we scale down the power budgets of both groups to compare the power of two over-provisioned groups and show the effectiveness of our control. In Section 4.4, we only scale down the power budget of the experiment group so that we can observe the impact on throughput. We emphasize that we only use the scale-based emulation to provide more insights into how Ampere works, and it is not part of the production system.

We set the over-provision ratio to 0.25, a very high value, in most of our experiments to demonstrate the effectiveness of Ampere under extreme conditions. As we will show later in Section 4.4, we choose $r = 0.17$ as the optimal value for real production.

4.1.3 Key performance metrics

We focus on the following three key performance metrics:

1) The number of power violations. Given the power capping mechanism (Section 1), the user may choose to allow a few power violations, and small violation number shows the effectiveness of Ampere;

2) The ratio of frozen servers (u_t). Obviously, a smaller frozen ratio can help minimize the impact on the overall performance of the cluster;

3) The gain in throughput per provisioned watt (TPW). We define TPW as:

$$\text{TPW} = \frac{\text{Total throughput during a time interval } T}{P_M \cdot T} \quad (17)$$

where P_M is the total provisioned power budget, and the throughput is the number of jobs accepted during the time period T. We simply choose the job count as the throughput indicator because with large number of jobs, each job has similar average resource requirements.

The *Gain in TPW*, G_{TPW}, is the increase of TPW by over-provisioning. We use this metric to evaluate the balance between the computation capacity gain from adding more servers, and the capacity loss from freezing some servers. We discuss the evaluation of G_{TPW} in Section 4.4.

4.2 The effectiveness of Ampere's control

We first provide evaluation results using two extreme types of workload in our cluster - heavy and light. We fix the over-provision ratio at 0.25 for both experiments in this section.

Table 2 shows a few performance metrics of Ampere. We observe a smaller maximum power draw from the experiment group. Meanwhile, under the heavy workload, in the control group without any power control, we observe 321 power violations, while we observe only one violation using Ampere's control, and this violation is due to the 50% freezing ratio limitation. These observations have proved the effectiveness of Ampere's power control ability.

Taking a closer look at the control actions with different workload, Figure 10 plots the power draw and control actions over a period of 24 hours under heavy and light workload. Figure 10(a) shows the light workload case, i.e., the power draw mostly under the power limit. In this case, we

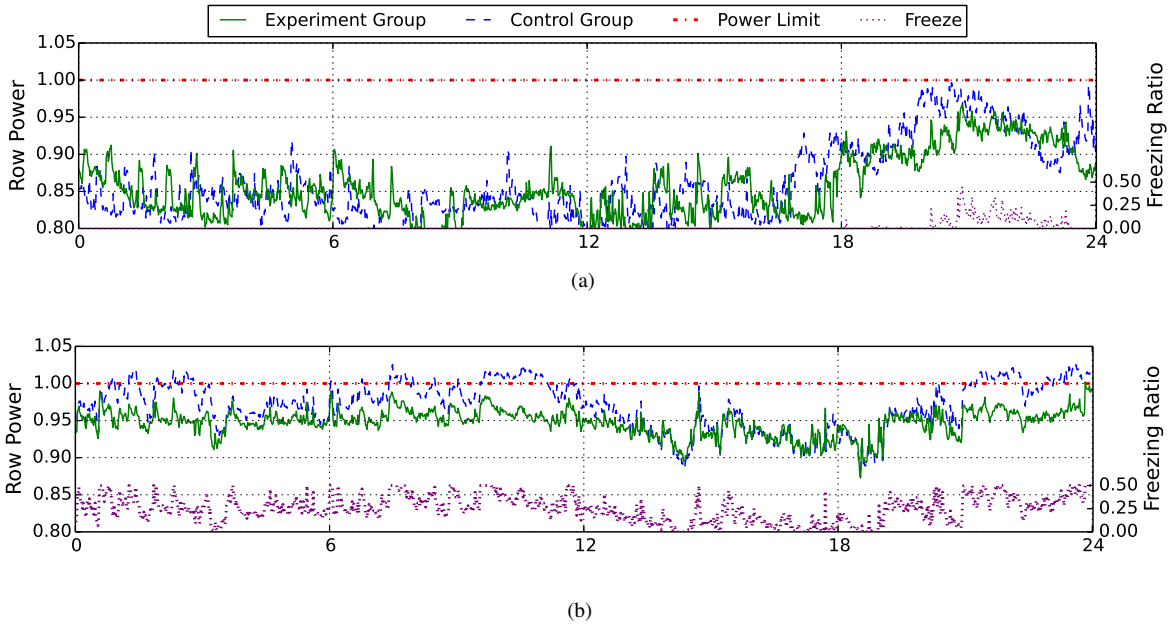


Figure 10. The freezing ratio u_t and its effects on the power utilization, on both light (a) and heavy (b) workload over 24 hours. The control group does not have power control and thus the power differences between the control group and experiment group are approximately the effects of the power control.

Workload	Light		Heavy	
Group	Exp	Ctr	Exp	Ctr
u_{mean}	1.5%	0%	24.7%	0%
u_{max}	44.1%	0%	50.0%	0%
P_{mean}	0.857	0.860	0.948	0.970
P_{max}	0.967	0.997	1.002	1.025
$Violations$	0	0	1	321

Table 2. Controller effectiveness under light / heavy workload. The experiment runs for 24 hours and the measurements are taken every minute. u_{mean} and u_{max} are the mean/max freezing ratio. P_{mean} and P_{max} are the mean/-max power draw. $Violations$ is the total number of power violations.

only take control actions occasionally, and thus cause little impact on the overall power or throughput. In contrast, Figure 10(b) shows the heavy workload situation, during which the power draw would exceed the power budget quite often without control. We can see from the figure (purple/dotted lines) that Ampere freezes a significant number of servers at many time periods and successfully avoids power violations. Table 2 shows the statistics for a 24 hour period for light workload and heavy workload.

Note that because we limit the maximum freezing ratio of servers to 50%, we get many saturation on the control input in Figure 10(b). This limitation effectively reduces how much we can react to a power surge and thus makes it more vulnerable to power violations. We will try to remove this scheduler limitation in our future work.

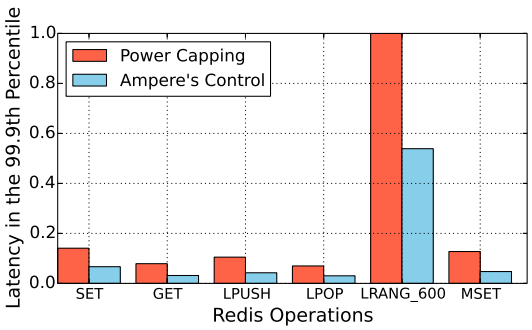


Figure 11. The 99.9th percentile normalized latency of various operations in the Redis-benchmark, using either power capping or Ampere as power controller.

4.3 Advantage over power capping approach

Power capping, as we have already mentioned, may cause performance disturbances. In this section, we compare the performance disturbances, and demonstrate that Ampere has big advantage in this aspect.

We deploy a Redis [41] cluster on a row with over-provision ratio r_O set to 25%. We repeatedly run a Redis-benchmark [42] on a number of clients located in another cluster that does not have any power control. We compare the performance of the Redis cluster under power capping and under Ampere, respectively. We report the 99.9th percentile latency observed on the client side. Figure 11 shows the results.

We can see power capping reduces the performance of the Redis, almost doubling the 99.9th percentile latency in almost all benchmarks. This is because Redis servers are CPU-bound, and reducing CPU frequency on a busy server slows down request processing, causing significant queuing effects that lead to longer latency. In comparison, with the control of Ampere, we rarely trigger power capping, and `freeze`/`unfreeze` operations do not affect existing jobs. Thus Ampere results in a much smaller latency for interactive applications like Redis.

Without the control of Ampere, power capping is very common in the cluster. To quantify this fact, we collect 8640 power utilization records (one per minute) on all servers over several days. Among these records, 1306 are over power budget. For each of these 1306 minutes, we check each individual server to see if it is power capped. Our data shows that on average, 54.34% servers are power capped for roughly 15% of the total time, which is quite unacceptable for latency-sensitive jobs. In comparison, Ampere has no impact on running jobs, and thus is applicable to both interactive and batch jobs.

4.4 Factors that affect the TPW

The goal of our work is to increase the computation capacity given a fixed power budget, and thus TPW is an important metric. Two parameters affect TPW, the over-provision ratio r_O and the throughput ratio r_T. The throughput ratio r_T is in turn affected by the workload. In this section, we provide some quantitative analysis on the r_O choice and its effect on TPW under different workload.

Over provision, workload and the gain in TPW. The increase on TPW is obvious from the number of extra servers provisioned. To evaluate the throughput loss due to control, we compare the number of jobs accepted in the experiment group $thru^E$ and the control group (with the same number of servers, but with Ampere turned off) $thru^C$ during the same time period. We define the throughput ratio r_T as $thru^E/thru^C$. Generally $r_T \leq 1.0$ as freezing servers reduces the throughput.

Given r_T and the over-provision ratio r_O, we estimate G_{TPW} by

$$G_{TPW} = \frac{TPW^E}{TPW^C} - 1 = \frac{thru^E/(P'_M * t)}{thru^C/(P_M * t)} - 1 = r_T \cdot (1 + r_O) - 1. \quad (18)$$

For example, if the over-provision ratio is 0.25, i.e. we add 25% more power budget (and thus 25% more servers) to the row. With sufficient power budget, we should get an increase of capacity by 25% and the throughput should increase by 25% with enough workload. The power control of Ampere reduces the throughput. We measure the loss by comparing the experiment group throughput with the control group. For example, if we observe a 10% decrease in throughput in the experiment group, then the overall TPW gain is $G_{TPW} = (1 - 0.1) * (1 + 0.25) - 1 = 0.125$.

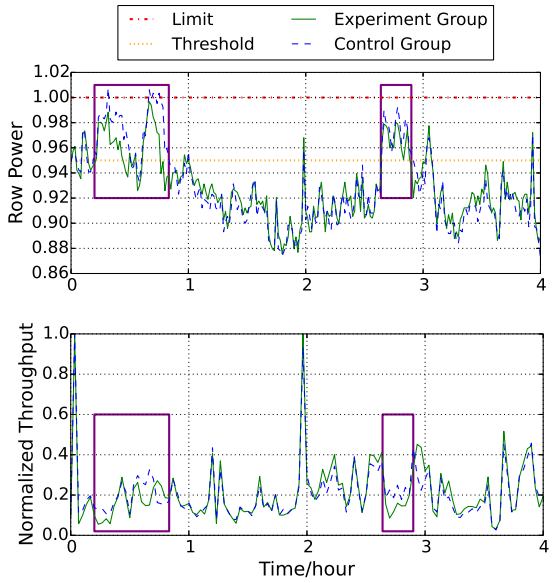


Figure 12. The effect of Ampere on power and throughput. The box highlights the effect: the control actions effectively reduces both the power and the throughput of the experiment row. Ampere only applies the control action when the power is above the threshold, leaving other regions unaffected.

Note that G_{TPW} is workload dependent. Under a light workload, adding servers just cause more servers to stay idle without any positive effects. With a fully utilized cluster already taking the entire power budget, we cannot run new jobs even with more servers. However, as we show in Section 2.2, when the workload shows high variation, we have plenty of opportunities to get a good G_{TPW}.

An intuitive example. We illustrate that TPW does not increase monotonically with the over-provisioning ratio r_O.

Figure 12 shows an example of how r_O affects TPW. During this experiment, we set $r_O = 0.25$ and run the experiment for four hours. The boxed area on the left shows the time period when power utilization is high. Comparing to the control group, we observe a throughput decrease by about 20% in the experiment group, as expected. Thus we have $r_T = 0.8$ and $G_{TPW} = (1 + 0.25) \times 0.8 - 1$, which is close to zero. Intuitively, as we are already using all the power without over-provisioning, adding more servers do not bring in extra capacity for jobs due to the power limit. It is even worse that the extra servers consume idle power, eventually hurting the overall throughput. Thus, we need to avoid this situation in production.

If we choose a smaller $r_O = 0.17$, under the same heavy workload in Figure 12, we are under the power budget most of the time, and thus r_T is close to 1.0, making TPW gain close to the over-provision ratio $G_{TPW} = (1 + 0.17) \times$

$1.0 - 1 = 0.17$, indicating that we can fully utilize the over-provisioned resource even at a high workload.

Of course, workload varies over time. During the four hour period shown in Figure 12, the average r_T is 0.95, a higher number than 0.8. Therefore, we can estimate that when $r_O = 0.25$, we can get a gain in TPW, $G_{TPW} = (1 + 0.25) * 0.95/1.0 - 1 = 0.19$, a better number than the case with high workload.

From the example, we see that both r_O and the average workload have a high impact on the gain in TPW.

Evaluation on different over-provision ratio and workload. Over an experiment period of 20 days, we run Ampere using different over-provisioning ratio under varying production workload. Table 3 captures representative results from 13 days. Note that the workload is from real production that we have no control over, and it is the reason why we show different workload in different cases. We have the following two observations.

First, with a given r_O, G_{TPW} is directly affected by the control input u_{mean}. It is intuitive that the more servers we freeze on average, the smaller the capacity is, and so is the G_{TPW}. A deeper analysis of Table 3 shows that u_{mean} is largely affected by the average power demand P_{mean} [2]. With similar P_{mean}, occasional spikes on P trigger large u_t at certain times, also affecting G_{TPW}. For example, #4 shows worse G_{TPW} mainly due to the high maximum workload.

Second, the over-provisioning ratio r_O also has a large impact on G_{TPW}. For example, #4 and #7 have very close P_{mean} (scaled by r_O), but r_T and G_{TPW} are both better with a smaller r_O. It is due to the same reason as we have discussed earlier: $r_O = 0.25$ is too high an over-provision ratio, causing u_t to be high quite often. As another extreme, $r_O = 0.13$ is too low, because G_{TPW} is upper bounded by r_O, the gain is only 0.13, making it a less attractive choice.

Choosing the optimal r_O**.** Given the observation above, we want to choose a moderate r_O. There are two metrics to consider: (1) We want to choose a r_O so that we can maximize G_{TPW} under the *typical* workload; (2) Choosing r_O is also a tradeoff between safety (fewer power violations) and performance (higher TPW).

In our experiments in Table 3, we find that 0.17 is a safe and effective choice. From our observation over a month, the 85th and the 95th percentile power is 0.909 and 0.924 (scaled to match r_O), which means most of the time G_{TPW} will be at least 15%. Thus both G_{TPW} and over-provisioning efficiency are relatively higher compared to other r_O choices. In conclusion, we choose 0.17 as our over-provisioning ratio considering safety, G_{TPW} and efficiency.

[2] We estimate the power demand using the control group power, which is unaffected by the control. To show the effect of over-provisioning, we normalize the power of the control group to the scaled power budget of the experiment group. That's why P_{max} may exceed 1.0 in some cases.

#	r_O	P_{mean}	P_{max}	u_{mean}	r_{thru}	G_{TPW}
1	0.25	0.903	1.028	0.019	0.953	19.70%
2		0.931	1.062	0.134	0.941	17.60%
3		**0.936**	**1.062**	**0.152**	**0.885**	**10.60%**
4		0.927	1.061	0.196	0.835	4.30%
5	0.21	0.786	0.913	0	1.0	20.70%
6		0.835	0.982	0.0016	1.0	20.70%
7		0.894	1.000	0.009	0.979	18.20%
8		**0.903**	**1.036**	**0.11**	**0.88**	**6.20%**
9	0.17	0.836	0.931	0	1.0	17%
10		0.839	0.926	0	1.0	17%
11		**0.908**	**0.992**	**0.07**	**0.984**	**14.90%**
12		0.938	1.004	0.12	0.904	5.50%
13	0.13	0.847	0.969	0	1.0	13%

Table 3. G_{TPW} under different over-provision ratio r_O and workload condition. P_{mean} and P_{max} are the mean and max power of the control group, respectively, which are good indicators of the power demand. u_{mean} is the average freezing ratio. Bold rows represent results under typical workload.

5. Related Work

Fan *et al.* did the first quantitative study on large-scale data center power consumption [14]. They showed that there were wide gaps between the average power utilization on rack, PDU, and cluster levels. They pointed out the potential opportunities of using power over-provisioning to increase data center capacity, and proposed a theoretical approach to implement over-provisioning with power capping mechanisms. Wang *et al.* [52] further characterized the power utilization. In particular, they focused on the power peaks and valleys. Many projects on power optimization, including ours, confirmed the observations in these work, and designed control mechanisms based on these observations.

There are two major approaches to manage power: using hardware features like DVFS, and using power-aware workload scheduling [35, 37]. We introduce related work in both categories and describe the uniqueness of our approach.

5.1 Controlling power by hardware power capping

Simple mechanisms directly control the hardware power states (sleep, off, on). PowerNap [33] and Anagnostopoulou *et al.* [2] target to minimize idle power and transition time within different power states. PowerNap turns several components into power saving states when the server is idle, and wakes them up upon a user request. Given the time it takes to switch between states, people have proposed different ways to minimize the impact to SLAs during transitions [34]. Liu *et al.* showed that it is possible to exploit the best power policy for a given SLA constraint on a single node, and proposed SleepScale, a runtime power management tool to efficiently apply power control [29].

More advanced mechanisms use hardware features like power scaling. Power capping by DVFS imposes an effective power control over CPU and DRAM [14]. The challenge is to lower the system speed while keeping the job SLA vi-

olations as few as possible. Sharma *et al.* implemented a feedback-based power management protocol that can provide some SLA guarantees when DVFS is enabled [46]. Lo *et al.* [30] proposed a more general version called PEGASUS, which works on a data center level. Raghavendra *et al.* proposed a theoretical power management proposal [40] with provable correctness, stability, and efficiency. This proposal is hard to implement in real data centers, as it requires highly coordinated control on both hardware and software layers, which is costly to achieve in realtime.

Ampere uses power capping differently. It only uses power capping as a safety-net in the highly unlikely events. In normal cases, we do not bring any disturbances to the existing jobs on the cluster.

5.2 Workload scheduling and power management

Many approaches use server consolidation. They transition idle servers into low-power or power-off states when the utilization is low [6, 9, 22, 25, 27, 28, 38, 43, 47, 53]. IBM proposed a realtime power management algorithm for parallel computers, which uses workload history to predict short-term workload fluctuations and then decides which servers to turn on and off [7]. Xu *et al.* [54] proposed a technique by adjusting the number of active nodes using workload information under certain time intervals. However, turning off servers is a complex process that requires process migration or restarts, and thus it is very hard to guarantee the SLA requirements [30]. Freezing servers in Ampere is different, as it just rejects new jobs and does not affect existing ones.

Researchers have also proposed ideas on how to integrate power management into the job scheduler to achieve better power usage pattern. Chase *et al.* used a dynamically reconfigurable switch to control the routing of requests, so they would use a server that could optimize energy consumption while satisfying the SLA [8]. Verma *et al.* built a power-aware application placement controller for high performance computing clusters [50]. Facebook built *Autoscale* to keep all active servers above a moderate CPU utilization, in order to achieve better power efficiency [3]. It was tightly coupled with the job scheduler, who chose a subset of servers as the *active pool* and automatically adjusted the pool size. On the other hand, Power aware scheduling policies can also be integrated into other QoS-aware cluster management systems [12, 13, 31, 49, 51, 55], which will furtherly complicate the design and implementation of the cluster schedulers.

Instead of a tight coupling with a scheduler designed for a simple workload pattern or a scheduler which has been over-complicated, our loose coupling with scheduler allows Ampere to integrate with complex data-center-level job schedulers with unknown custom policies and job patterns.

Govindan *et al.* used workload power profiles and implemented dynamical power provisioning on the PDU-level using DVFS [15]. They implemented their technique in a data center prototype and showed the improvement of the Computation per Provisioned Watt (CPW) for a few types of applications (e.g. TPC-W). While our work also leverage statistical multiplexing of jobs, Ampere does not require an *accurate* workload-power profile that is challenging to obtain in a data center running a complex mix of workload.

5.3 Commercial power management tools

Commercial data center power management solutions focus on power monitoring, visualization and reporting. Some tools provide an interface to implement power capping. At low level, most tools, including ours, use the IPMI specifications to communicate with the Baseboard Management Controller (BMC), to monitor power draw, among other information, from individual servers. The software tools integrate the power at rack, row or data center level.

Common tools include Intel DCM Energy Director [21], IBM PowerExecutive [20], HP Thermal Logic [19], and Cisco Unified Computing System [10]. Most of the tools are vendor specific, and do not handle data centers of our scale, and thus we build our own power monitoring solution.

6. Conclusion and Future Work

One of the biggest problems in our data centers is insufficient power budget given the ever increasing demand on computation. While it is economically attractive to provision more servers into an existing data center with a fixed budget, it is a hard tradeoff between the cost saving and other considerations such as power system safety, performance stability especially for interactive jobs, as well as the complexity it brings to integrate with existing system.

We design and implement Ampere, and empirically demonstrate the feasibility of using statistical control to indirectly manage the power utilization across a cluster. Specifically, as we use receding horizon control (RHC) to correct errors over time, we can achieve effective power control using statistical and inaccurate system models that are inexpensive to maintain in production. Also, using the simple `freeze`/`unfreeze` API, Ampere can be loosely coupled with complex job schedulers, which greatly simplifies the system implementation. For evaluation, we conduct controlled experiments with real production workload and provide detailed insights into the performance of the system. In production, we deploy Ampere to a data center, allowing us to provision 17% more servers, leading to a throughput gain of 15%.

There are two kinds of future work we are pursuing. First, on the power management side, we are exploring ways to schedule the jobs to different rows so that there can be a larger variance in power utilization across different rows, leading to more unused power to cultivate. Note that even with the improvement, we can still use the simple interface of Ampere. Second, we believe the simple statistical interface is a promising design to connect the low-level data center infrastructure to the higher-level software components

such as the job scheduler and even applications, and allows cross-layer optimizations. We are building a workload-sensitive cooling control system based on a similar interface.

Acknowledgments

We would like to thank our colleagues at Baidu Inc. Yuliang Liu, Bin Zhang, Hao Xu and Yongfeng Ji for help with Ampere's implementation, Xiaojing Wang for reviewing the drafts of this paper, and Jun Liu as well as Jiecheng Guo for insightful comments on the design of Ampere. We also thank the students of Tsinghua University Cheng Yang and Hao Xue for helpful comments on the early drafts of this paper. Finally, we thank the anonymous reviewers from EuroSys 2016 for their insightful comments, and our shepherd Jon Crowcroft for helping to shape the final version of the paper.

This research is supported in part by the National Natural Science Foundation of China Grants 61033001, 61361136003, 61532001, 61303195, China 1000 Talent Plan Grants, Tsinghua Initiative Research Program Grants 20151080475, a Microsoft Research Asia Collaborative Research Award, and a Google Faculty Research Award.

References

[1] acado. ACADO Toolkit. http://acado.github.io/.

[2] V. Anagnostopoulou, S. Biswas, H. Saadeldeen, A. Savage, R. Bianchini, T. Yang, D. Franklin, and F. T. Chong. Barely alive memory servers: Keeping data active in a low-power state. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 8(4):31, 2012.

[3] autoscale. Facebook Autoscale. https://code.facebook.com/posts/816473015039157/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale/.

[4] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, (12):33–37, 2007.

[5] L. A. Barroso, J. Clidaras, and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3): 1–154, 2013.

[6] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient idle desktop consolidation with partial VM migration. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 211–224. ACM, 2012.

[7] D. J. Bradley, R. E. Harper, and S. W. Hunter. Workload-based power management for parallel computer systems. *IBM Journal of Research and Development*, 47(5.6):703–718, 2003.

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 103–116. ACM, 2001.

[9] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *ACM SIGOPS Operating Systems Review*, 44(1):76–80, 2010.

[10] ciscoproduct. Cisco Unified Computing System. http://www.cisco.com/c/en/us/products/servers-unified-computing.

[11] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 189–194. IEEE, 2010.

[12] C. Delimitrou and C. Kozyrakis. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, 41(1):77–88, 2013.

[13] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[14] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[15] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 317–330. ACM, 2009.

[16] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[17] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17, 2012.

[18] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[19] hpproduct. HP Thermal Logic. http://h20621.www2.hp.com/video-gallery/us/en/4b547a2b6afff5fc7fc512069ad54d3928a124c9/r/video.

[20] ibmproduct. IBM PowerExecutive. https://www-01.ibm.com/marketing/iwm/tnd/demo.jsp?id=IBM+PowerExecutive+Power+Capping+Mar07.

[21] intelproduct. Intel DCM Energy Director. http://www.intel.com/content/dam/www/public/us/en/documents/articles/intel-dcm-energy-director-overview.pdf.

[22] C. Isci, S. McIntosh, J. Kephart, R. Das, J. Hanson, S. Piper, R. Wolford, T. Brey, R. Kantner, A. Ng, et al. Agile, efficient virtualization power management with low-latency server power states. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 96–107. ACM, 2013.

[23] jmpc. jMPC Toolbox. http://www.i2c2.aut.ac.nz/Resources/Software/jMPCToolbox.html.

[24] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing distributed UPS energy for effective power capping in data centers. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 488–499. IEEE, 2012.

[25] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized

computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

[26] W. H. Kwon and S. H. Han. *Receding Horizon Control: Model Predictive Control for State Models*. Springer Science & Business Media, 2006.

[27] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking (TON)*, 21(5):1378–1391, 2013.

[28] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen. GreenCloud: A new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, pages 29–38. ACM, 2009.

[29] Y. Liu, S. C. Draper, and N. S. Kim. SleepScale: Runtime joint speed scaling and sleep states management for power efficient data centers. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 313–324. IEEE, 2014.

[30] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st annual international symposium on Computer architecuture*, pages 301–312. IEEE Press, 2014.

[31] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 450–462. ACM, 2015.

[32] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos. Cloud computing: Survey on energy efficiency. *ACM Computing Surveys (CSUR)*, 47(2):33, 2014.

[33] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating server idle power. In *ACM Sigplan Notices*, volume 44, pages 205–216. ACM, 2009.

[34] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[35] S. Mittal. Power management techniques for data centers: A survey. *arXiv preprint arXiv:1404.6681*, 2014.

[36] opencomputeproj. Open Compute Project. http://www.opencompute.org/.

[37] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys (CSUR)*, 46(4):47, 2014.

[38] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on compilers and operating systems for low power*, volume 180, pages 182–195. Barcelona, Spain, 2001.

[39] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[40] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 48–59. ACM, 2008.

[41] redis. Redis. http://redis.io/.

[42] redisbenchmark. Redis-benchmark. http://redis.io/topics/benchmarks.

[43] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 418–428. IEEE, 2006.

[44] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 351–364. ACM, 2013.

[45] scorpio. Open Data Center Committy. http://www.opendatacenter.cn/.

[46] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 63–72. IEEE, 2003.

[47] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, volume 10, pages 1–5. San Diego, California, 2008.

[48] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[49] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica. The power of choice in data-aware cluster scheduling. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 301–316, 2014.

[50] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 28–28. USENIX Association, 2009.

[51] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, page 18. ACM, 2015.

[52] D. Wang, C. Ren, S. Govindan, A. Sivasubramaniam, B. Urgaonkar, A. Kansal, and K. Vaid. ACE: Abstracting, characterizing and exploiting peaks and valleys in datacenter power consumption. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):333–334, 2013.

[53] Z. Wang, N. Tolia, and C. Bash. Opportunities and challenges to unify workload, power, and cooling management in data centers. In *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, pages 1–6. ACM, 2010.

[54] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé. Energy-efficient policies for embedded clusters. In *ACM SIGPLAN Notices*, volume 40, pages 1–10. ACM, 2005.

[55] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers. *ACM SIGARCH Computer Architecture News*, 41(3):607–618, 2013.