SmoothOperator: Reducing Power Fragmentation and Improving Power Utilization in Large-scale Datacenters

Chang-Hong Hsu[†], Qingyuan Deng[§], Jason Mars[†], Lingjia Tang[†]

[†]University of Michigan, Ann Arbor
{hsuch, profmars, lingjia}@umich.edu

[§]Facebook, Inc.
qdeng@fb.com

Abstract

With the ever growing popularity of cloud computing and web services, Internet companies are in need of increased computing capacity to serve the demand. However, power has become a major limiting factor prohibiting the growth in industry: it is often the case that no more servers can be added to datacenters without surpassing the capacity of the existing power infrastructure.

In this work, we first investigate the power utilization in Facebook datacenters. We observe that the combination of provisioning for peak power usage, highly fluctuating traffic, and multi-level power delivery infrastructure leads to significant *power budget fragmentation* problem and inefficiently low power utilization. To address this issue, our insight is that heterogeneity of power consumption patterns among different services provides opportunities to re-shape the power profile of each power node by re-distributing services. By grouping services with asynchronous peak times under the same power node, we can reduce the peak power of each node and thus creating more power head-rooms to allow more servers hosted, achieving higher throughput. Based on this insight, we develop a *workload-aware service placement* framework to systematically spread the service instances with synchronous power patterns evenly under the power supply tree, greatly reducing the peak power draw at power nodes. We then leverage *dynamic power profile reshaping* to maximally utilize the headroom unlocked by our placement framework. Our experiments based on real production workload and power traces show that we are able to host up to 13% more machines in production, without changing the underlying power infrastructure. Utilizing the unleashed power headroom with dynamic reshaping, we achieve up to an estimated total of 15% and 11% throughput improvement for latency-critical service and batch service respectively at the same time, with up to 44% of energy slack reduction.

CCS Concepts • **Hardware → Enterprise level and data centers power issues**; • **Computer systems organization** → *Cloud computing*;

Keywords datacenter power management; power fragmentation; power and energy efficiency

ACM Reference Format:
Chang-Hong Hsu[†], Qingyuan Deng[§], Jason Mars[†], Lingjia Tang[†]. 2018. SmoothOperator: Reducing Power Fragmentation and Improving Power Utilization in Large-scale Datacenters. In *ASPLOS '18: 2018 Architectural Support for Programming Languages and Operating Systems, March 24–28, 2018, Williamsburg, VA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3173162.3173190

ASPLOS '18, March 24–28, 2018, Williamsburg, VA, USA

ACM ISBN 978-1-4503-4911-6/18/03…$15.00
https://doi.org/10.1145/3173162.3173190

1 Introduction

As cloud computing and Internet services become increasingly popular, datacenter loads are rapidly growing [1, 3]. The power budget supplied by the power delivery infrastructure in datacenters, however, often constrains the amount of server that can be added to handle the ever growing load. Indeed, the power budget [38, 51] has become one of the most contentious resources in datacenter management. Building new datacenter facilities and new power infrastructures would help alleviate the problem, but they are costly and time-consuming. Datacenter operators need new techniques to maximize utilization of the existing power infrastructure.

We investigate the power delivery infrastructure at Facebook production datacenters, and observe that the power provisioning in these datacenters faces two major challenges:
Challenge 1: Peak provisioning leads to low power budget utilization. For modern large-scale datacenters that mostly serve user-facing workloads, their total power consumption usually follows a *diurnal pattern* [35, 39], which reflects the highly fluctuating user activity level throughout a day. To ensure sustainability and safety, datacenter managers need to ensure that the peak aggregate power consumption can be accommodated under the given, fixed power budget supplied by power infrastructure. *Peak provisioning*, however, usually leads to highly underutilized power budgets during the rest of the day.

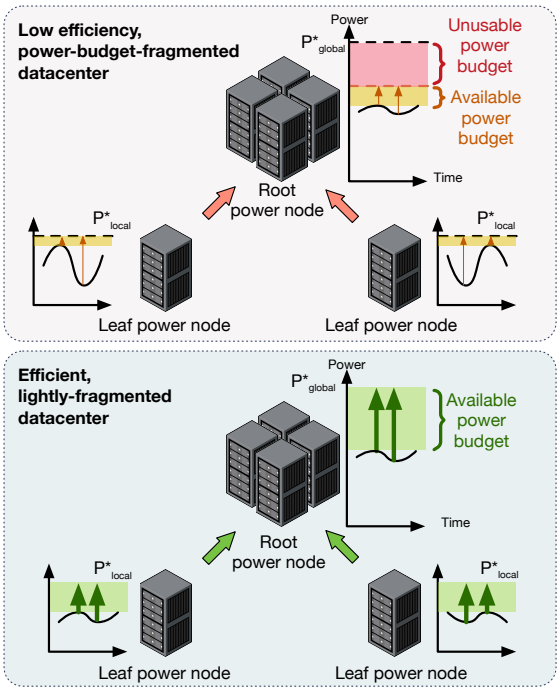

Figure 1. Grouping servers with synchronous power consumption patterns together often lead to rapid local peaks, which consume the local power budgets quickly and cause significant power budget fragmentation.

Challenge 2: Multi-level power delivery infrastructure leads to power budget fragmentation. Most modern datacenters deploy a multi-level power delivery infrastructure. These infrastructures have a tree-like structure, each node being a power delivery device, or we call a *power node*. Servers are directly supplied by lowest-level (*leaf*) power nodes, which are supplied by higher-level power nodes. Such hierarchical infrastructure improves reliability and is a common practice in large-scale datacenters. Unfortunately, it also leads to negative effects on power budget utilization because peak provisioning occurs at all levels of power nodes. **Power budget fragmentation** problem, as illustrated in Figure 1, frequently occurs in multi-level power infrastructures. For instance, in Figure 1, we show two simplified datacenters, each of which is equipped with a two-level power infrastructure, and the same set of servers and service instances. The only difference between these two datacenters is how service instances are placed under the leaf power nodes. In the first datacenter, servers having synchronous power consumption patterns are connected to the same leaf node. This creates rapid peaks with high amplitudes at the leaf nodes, which consume the local power budget quickly. In such a datacenter, while there is still an abundant amount of power headroom at the root node, there is no room to connect any more servers to these leaf power nodes. *Since servers can only be supplied by the leaf power nodes, if the power budget is highly fragmented at the lower level of power delivery infrastructure, the abundant power headroom at the root node can never be utilized, making the datacenter inefficient.* In the second datacenter, on the contrary, servers are mixed in a manner that service instances with synchronous power consumption patterns are spread out. Our insight is *when carefully spreading out synchronous service instances, rapid peaks at leaf power nodes are eliminated, and power headrooms at the leaf nodes are increased.* These increased local power headrooms allow more servers to be supplied, which improves the utilization of the power budget and the overall datacenter efficiency.

A large body of prior solutions, including power capping [6, 8–10, 14, 18, 23, 29, 40, 42, 50, 51], were proposed to solve Challenge 1. When applying these prior works in a datacenter with oblivious service placement, however, their potentials are largely limited by power budget fragmentation. In such a datacenter, instances of the same services are typically placed together. Since these instances reach their peaks around the same time, the corresponding leaf nodes that supply these latency-critical servers consume their power budgets much faster than the other leaf nodes. In this case, unfortunately, they need to be largely capped, even when there are still ample amounts of power headroom at other leaf nodes. A few techniques [16, 20, 28, 38] were proposed to address Challenge 2 in order to make power capping more efficient. These solutions, however, either require modifications to the power infrastructure [20, 28, 38], or due to the battery capacity can only handle peaks that span at most tens of minutes, making it unsuitable for Facebook type of workloads whose peak may last for hours. [16].

In this work, we aim to improve the efficiency of power usage in datacenters, allowing datacenters to achieve higher throughput without changing the existing power infrastructure. To this end, we propose **SmoothOperator**, a framework that analyzes the temporal heterogeneity of power consumption patterns and derives a highly power efficient service placement. SmoothOperator leverages a novel approach to systematically model and score the temporal heterogeneity among different services. Based on the analysis, SmoothOperator uses a clustering-based approach to identify instances that create high peaks when being placed together, and spreads them out across the datacenter.

This framework increases power headroom at all levels of the power delivery infrastructure, allowing hosting more servers and increasing the throughput of the datacenter. Meanwhile, we also observe that simply adding servers in a straightforward way can lead to resource underutilization and there are further opportunities to improve the throughput. We leverage a new type of disaggregate servers that decouple the compute and the storage components, recently deployed in production. We design a set of history-based server conversion and proactive throttling and boosting policies to fully utilize the newly-added servers to further increase the resource and power utilization.

Specifically, the contributions of this work include

1. We identify the *power budget fragmentation* problem in production datacenters: since the number of servers a power node can support depends on peak aggregate power of theses servers, placing the service instances that have synchronous power consumption patterns

under any power node leads to rapid peaks, degrading the overall power budget utilization. We then characterize and identify the opportunity for solving this problem based on real production power data.
2. We propose a *workload-aware service instance placement and remapping* framework to mitigate power budget fragmentation problem, utilizing previously wasted power budget at higher level power nodes. The key insight is that when service instances do not reach their peak power usage at the same time, the aggregate power consumption pattern has a lower peak value at the low-level power nodes, enabling improved utilization of the overall power budgets.
3. To further utilize the additional power headroom, we leverage storage-disaggregated servers [26] and design *dynamic power profile reshaping* techniques to improve throughput. The dynamic power profile reshaping technique is composed of a history-based server conversion policy and a proactive throttling and boosting policy. The results indicate that we could achieve additional throughput improvement without changing the underlying power delivery infrastructure.

We conduct our experiments based on data collected from three production datacenters. The results show that our workload-aware service instance placement improves the power budget utilization by up to 13%, allowing extra servers to be housed without modifying the existing power infrastructure. The server addition leads to throughput improvement for latency-critical services by 13%. By applying server conversion to the workload-aware placement, we achieve additional 8% throughput improvement for batch service. When we further apply proactive throttling and boosting, we achieve an estimated 15% and 11% throughput improvement for latency-critical services and batch services, respectively.

2 Power budget fragmentation and inefficiency

2.1 Multi-level power infrastructure in datacenters

Multi-level power infrastructure is commonly deployed in large-scale, production datacenters. At each level, the total power budget of a power node is shared among its children nodes. This type of tree-like multi-level power infrastructure is designed to avoid a single point of failure and for the convenience of management.

Facebook datacenters feature four-level power infrastructure, consistent with Open Compute Project specification [2, 51], as shown in Figure 2. Each datacenter is composed of several rooms, which we call *suites*. Servers are placed onto rows of racks, allocated into these suites. A suite is equipped with multiple top-level power nodes, i.e., main switching boards (MSBs), each of which supplies some second-level power nodes, switching boards (SBs), which further feeds to a set of reactive power panels (RPPs). Finally, the power is fed into racks, each composed of tens of servers. The power budget of each node is approximately the sum of the budgets of its children.

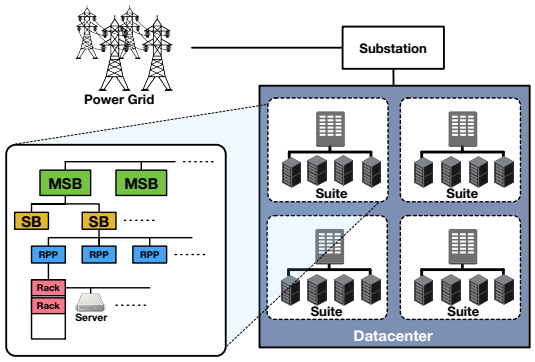

Figure 2. Multi-level power delivery infrastructure deployed in Facebook's datacenters.

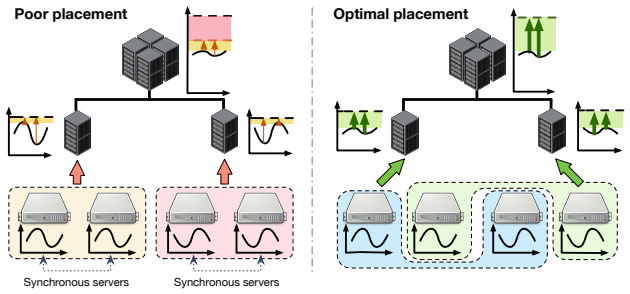

Figure 3. Careful service instance placement can further utilize power headroom for housing more servers, improving the overall computing capacity of the datacenter without changing the underlying power infrastructure.

2.2 Power budget fragmentation

Power budget fragmentation exists because servers hosting the services with synchronous power consumption patterns are grouped together under the same sub-tree of the multi-level power infrastructure. Such a placement creates rapid, unbalanced peaks with a high amplitude, which consumes the power budget of the supplying power node fast. When the aggregate power at a power node exceeds the power budget of that node, after a short amount of time, the circuit breaker is tripped and the power supply for the entire sub-tree is shut down. These local power limits, therefore, make it harder to efficiently utilize the total power budget that is supplied to the entire datacenter.

For example, in Figure 3, we have a simplified datacenter with a 2-level power infrastructure and 4 service instances to be placed under the leaf power nodes. We assume that service instance 1 and 2 have an identical (perfectly *synchronous*) power consumption pattern, and service instance 3 and 4 have perfectly out-of-phase patterns. To avoid tripping the circuit, the limiting factor of the number of servers that can be supplied under a power node is the peak power (maximum aggregated power across all servers supplied by the same node). When synchronous servers are placed together, as shown in the left sub-figure of Figure 3, it leads to a higher peak power at the supplying power node than the optimal placement. This higher peak curtails the number of servers

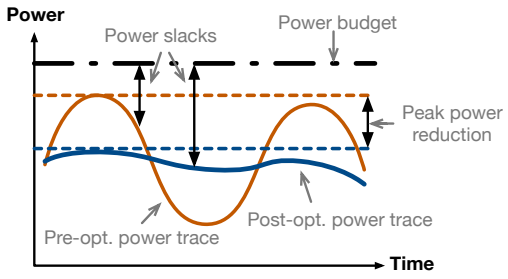

Figure 4. Power Slack, defined as the difference between the current power consumption at time t and the power budget. It quantified the power utilization efficiency.

that can be supplied by the datacenter, which indicates a lower power utilization.

In the following, we define two metrics to quantify the level of power budget fragmentation and inefficiency of power budget utilization. In this work, we focus on improving these two metrics:

1. **Sum of peaks:** Sum of the peaks of the power nodes in a datacenter is an important indicator of the level of power budget fragmentation. With the same set of service instances, poor placements can produce very a high peak power value at leaf nodes, indicating the peaks of the service instances are not evenly spread out. The sum of the power node peaks is, therefore, much larger than that in the optimal placement.
2. **Power slack and energy slack:** Power slack and energy slacks are indicators for power budget utilization. As illustrated in Figure 4, *power slack* is a measurement of the unused power budget at a point of time, and is defined as

$$P_{slack,t} = P_{budget} - P_{instant,t}, \quad (1)$$

where $P_{instant,t}$ is the instant power consumption of the power node at time t, and P_{budget} is a constant number representing the given power budget of this power node. The lower the power slack, the higher proportion of the power budget is utilized at that point of time. *Energy slack* is simply the integral of power slack over a timespan T.

$$E_{slack,T} = \int_T P_{slack,t} dt \quad (2)$$

A low energy slack means the power budget is highly utilized over the corresponding timespan.

In the following sections, we will use these two metrics to guide the development of our solutions and measure the quality of our result.

2.3 Peak heterogeneity in datacenters

Modern datacenters often provide a wide spectrum of services. Even in highly web-centric companies such as Facebook, to support a variety of data-centric features and the high visit rate, a significant proportion of servers in their production datacenters are provisioned to serve hundreds to thousands of internal services. Figure 5 presents the breakdown of 30-day average power consumption of the top 10 power consumer workloads measured in the three datacenters under study.

We observe that the power usage patterns are heterogeneous across these services. Such heterogeneity indicates abundant opportunities to mitigate the power budget fragmentation problem by grouping services with complementary power consumption patterns under the same power node. Figure 6 shows the diurnal patterns of three of the major services that are hosted in one of Facebook datacenters: **web**, **db**, and **hadoop**. The bands indicate the percentiles of the power reading among all the servers hosting that service. For example, the darkest band in the top-most sub-figure indicates the range between the 45th-percentile and the 55th-percentile power consumption readings among all the web servers at any given time. From this figure we show that servers hosting different services have very different power consumption patterns.

The web clusters serve the traffic coming directly from the users and hitting the production web site. This type of clusters, including web and cache servers, is the major part of Facebook datacenters, and is one of the largest consumers of the power budget. Because of its user-facing nature, the servers in these clusters have highly synchronous power patterns, which follow the typical user activity level. Meanwhile the latency requirement for this type of workload is high because it directly affects user experience.

The db clusters are the backend of the web and cache clusters. What distinguishes backend servers from these front-end servers is that query only hits db servers when the data needed are missing in the cache servers. Compared to front-end servers, these servers are more I/O bound, thus not exhibiting high power consumption even when the front-end servers are experiencing peak usage during daytime. However, these servers perform daily backup at night, which involves a lot of data compression. Therefore, as shown in Figure 6, while these servers also have predictable diurnal pattern, their peaks occur during the night time.

The hadoop clusters serve batch jobs that aim to improve the quality of data that are needed by the website or the company. Since users don't directly interact with this type of servers, these services are not bound by latency requirement; instead, they are optimized to provide high throughput. To achieve satisfactory throughput, the execution of this type of jobs highly relies on the job scheduler's policy, and the server are running at higher settings of CPU frequencies. Consequently, we find in Figure 6 that their power consumptions are constantly high and less relevant to the user activity level.

We see from the above that there are abundant opportunities to improve the datacenter power budget utilization if the datacenter operators take advantage of different characteristics of different workloads. Recognizing such opportunities, we design SmoothOperator to capture and leverage the peak heterogeneity identified among these production services, in order to mitigate the power budget fragmentation problem by intelligently grouping the servers with asynchronous power consumption patterns. We will introduce how we design SmoothOperator in Sections 3 and 4.

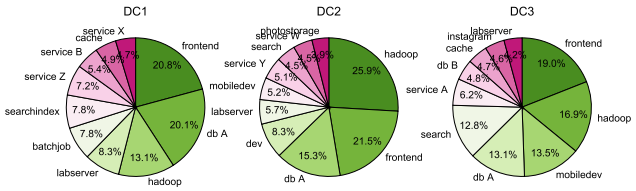

Figure 5. The breakdown of 30-day average power consumption of the top 10 power consumer workloads measured in three datacenters under study.

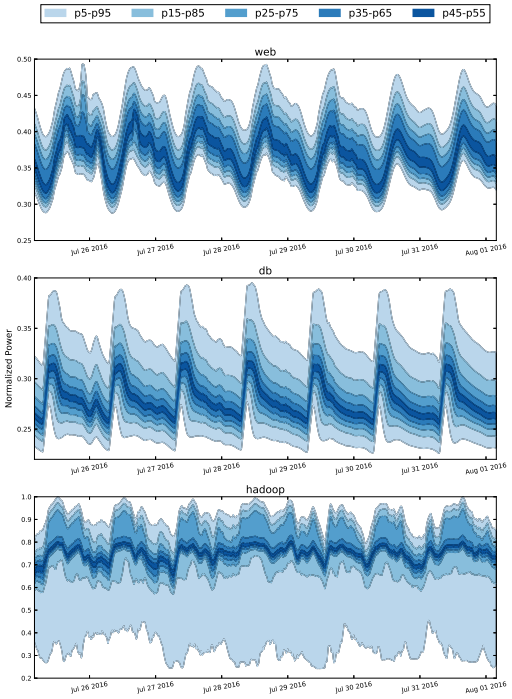

Figure 6. Servers serving different types of workload have very distinctive diurnal patterns. The bands indicate the percentiles of the power reading among all servers hosting one service type. Y axis is normalized to the maximum power reading observed on a single server in the datacenter.

3 Workload-aware service instance placement and remapping

In this section, we address the power budget fragmentation problem found in production datacenters, and introduce our *workload-aware service instance placement and remapping* framework. Our framework takes advantage of the service-level and service-instance-level heterogeneity, and spreads out service instances with synchronous power patterns under different power nodes.

3.1 Service and service instance

Facebook datacenters house thousands of web and data processing services. Similar to Microsoft datacenters[53], for major services, each service team manages its own *service* on a separate set of physical servers, and different major services do not share physical servers. A service is a collection of hundreds to thousands of *service instances*. Each of these service instances is a process that runs a copy of the service or a part of the service. For example, service such as Memcached runs on thousands of machines, and each Memcached process on a server is a service instance. Facebook deploys service instances as native processes instead of virtual machines. This policy not only reduces operational complexity, but also minimizes the variability caused by the interference due to co-scheduling and colocation.

3.2 Overview of placement framework

We illustrate our framework in Figure 7, which includes four major steps:

1. **Collect traces and extract representive traces** We first collect and construct multi-weeks of power traces for each service instance, which we call *instance power trace*, of the target datacenters ($\boldsymbol{PI}_i$ in Figure 7). We use these instance-level traces to further construct a *service power trace* ($\boldsymbol{PS}_i$) for each of the top power-consuming services. Each *service power trace* exhibits the representative temporal patterns of a service aggregated across all its instances. The service-level traces then serve as a set of bases that facilitates the evaluation of dissimilarity between instance-level power traces (Section 3.3).
2. **Calculating asynchrony scores** We then identify synchronous service instances, namely, the instances whose power consumption peak at the same time. To achieve that, we calculate a vector of *asynchrony scores* for each service instance based on the corresponding instance power trace and the service power traces of all the services (Section 3.4). We use the vectors of asynchrony scores to estimate the impact to the aggregated peak when two or more service instances are grouped together. This step transforms each server into a data point in a high dimensional space spanned by the asynchrony scores.
3. **Clustering** We then apply a clustering method (Section 3.5) on these service instances based on their asynchrony score vectors, identifying the ones with synchronous power consumption behavior.
4. **Placement** We place the service instances based on the clustering result, aiming to maximize the asynchrony score of each power node (Section 3.5).

After the initial application, our framework can be continuously applied to the datacenter to fine-tune the placement when power consumption patterns start to exhibit middle-term or long-term (e.g., in weeks or longer) shifts or changes.

Note that workload-aware service instance placement also provides benefits from the power safety aspect of datacenter. In the optimized placement, service instances that have highly synchronous behaviors are now spread out evenly across all the power nodes. When bursty traffic arrives, the sudden load change is now *shared* among all the power nodes. Such load sharing leads to a lower probability of high peaks aggregates at a small subset of power nodes, and therefore decreases the likelihood of tripping the circuit breakers inside certain heavily-loaded power nodes.

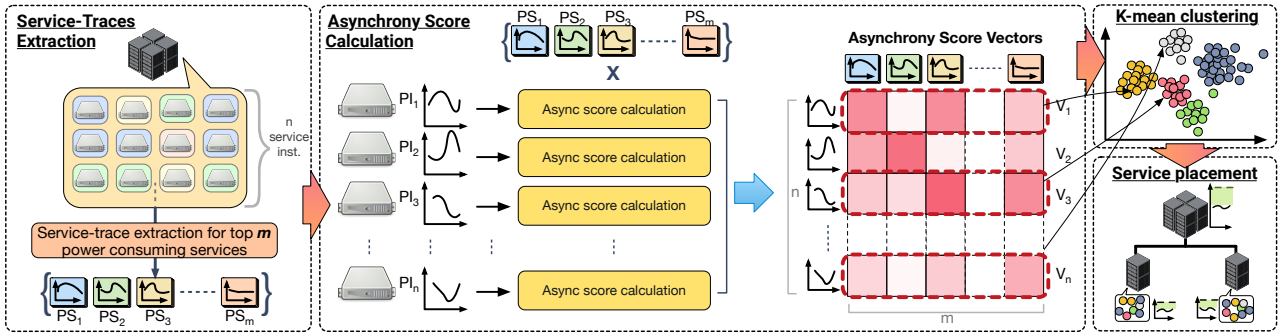

Figure 7. The overview of workload-aware service placement.

3.3 Constructing power traces

To be able to derive a workload-aware instance placement for datacenters, we need to capture the power consumption patterns of service and service instances. As shown in Section 2.3, services exhibit a diversity of power consumption patterns [17, 53] and such service-level power heterogeneity provides abundant opportunities for mitigating the fragmentation problem. In fact, in addition to service-level heterogeneity, we observe from the production data significant amounts of *instance-level heterogeneity*, even within the same service. Such heterogeneity usually stems from imbalanced accessing pattern or skewed popularity among different instances of a same service. SmoothOperator aims to capture both of these two types of heterogeneity by constructing *instance power traces* and *service power traces*. In the following, we introduce the details of these two types of power traces.

Constructing Instance Power Traces For every service instance in the datacenter, we construct a log of power readings, i.e., an *instance power trace (I-trace)*, to represent the corresponding service instance's history of power consumption. Each of these I-traces is a *time series*, which is a vector, containing seven days of the exact power reading recorded by the power sensor on the corresponding machine, one reading per minute.

$$\boldsymbol{PI}_{i,w} = \langle p_{i,t} : t \in T_w \rangle, \tag{3}$$

where $PI_{i,w}$ is the instance power trace of server instance i of week w, and $p_{i,t}$ is the power reading of instance i at time t, and T_w is the series of timestamps within week w during which the power readings are logged. We choose the length of 7 days because, in large-scale user-facing datacenters, user traffic has strong day-of-the-week activity patterns [5, 43]. Note that, since power traces are simply vectors, vector arithmetic can be directly applied.

SmoothOperator focuses on balancing the power load of the power nodes by investigating the service instances' middle-term (e.g., hourly, daily) to long-term (e.g., weekly or longer period) power consumption patterns. To prevent SmoothOperation from overfitting its decisions to any specific week in which significant unusual short-term variations exist (e.g., bursty traffics due to power failure of neighboring datacenters), we collect 2-3 weeks of I-traces for each service instance, and use vector arithmetic to calculate the *averaged instance power trace* for each service instance. That is,

$$\bar{\boldsymbol{PI}}_i = \frac{\sum_{w \in W} \boldsymbol{PI}_{i,w}}{|W|} \tag{4}$$

Each of these averaged I-traces remains to be a 7-day-long vector; each element of this averaged instance trace is the average of the power reading recorded at the same time-of-week across these multiple weeks.

Constructing Service Power Traces. We then construct *service power traces (S-traces)*, one for each of the top power-consumer services running in the datacenters. For service Y, we calculate the vector sum of the $\bar{PI}$'s of all of Y's instances, and divide the vector sum by the number of instances of service Y. This calculation is formulated as follows:

$$\bar{\boldsymbol{PS}}_Y = \frac{\sum_{service(i)=Y} \bar{\boldsymbol{PI}}_i}{|Y|}, \tag{5}$$

where $\bar{\boldsymbol{PS}}_Y$ is the service power trace of service Y, and $|Y|$ is the number of instances of service Y. These S-traces represent the most significant power consumption patterns observed in the datacenters; it means that, when randomly sampling any large enough group of service instances from the datacenter, the aggregate power trace of this group of service instances will be close to a linear combination of these top-consumer S-traces. When considering adding an extra service instance to a group of instances, we use these S-traces to evaluate whether the new instance's power consumption pattern will add significantly to the peak of the aggregate power trace of that group.

3.4 Asynchrony score function

To measure how the peaks of the power traces of a set of service instances spread out over time, we define a metric, namely *asynchrony score*. We use an *asynchrony score function* to evaluate the asynchrony score over a set of power traces M, which is defined as follows:

$$A_M = f(M) = \frac{\sum_{j \in M} peak(\bar{\boldsymbol{P}}_j)}{peak(\sum_{j \in M} \bar{\boldsymbol{P}}_j)}. \tag{6}$$

For example, if we want to evaluate whether two service instances a and b should be placed together, we would like to know if they peak asynchronously or not. Since an I-trace is a power trace, we can calculate asynchrony score function over two I-traces. This is done by calculating the

following ratio for $\bar{PI}_a$ and $\bar{PI}_b$, and the aggregate power trace $\bar{P}_{\{a,b\}} = \bar{PI}_a + \bar{PI}_b$:

$$A_{\{a,b\}} = f(\{a,b\}) = \frac{peak(\bar{PI}_a) + peak(\bar{PI}_b)}{peak(\bar{P}_{\{a,b\}})}, \quad (7)$$

where $A_{\{a,b\}}$ is the *asynchrony score between service instance a and service instance b*, and $peak(\bar{P}_j)$ is the peak value of the power trace $\bar{P}_j$. The lower the asynchrony score, the more overlapping the peaks of the component power traces, and therefore the *worse* the group is; the higher the score, the less the overlap. For example, in the poor placement case in Figure 3, each leaf node has a asynchrony score of 1.0. If we exchange server 2 and server 3, each of the leaf power nodes will have a asynchrony score close to 2.0.

For a set of power traces M, the lowest possible A_M is 1.0, meaning that every component power traces peaks at the same time. The highest asynchrony score, $|M|$, occurs when every instance has the same peak value p and the peak of the aggregate power trace of M is also p, meaning that the aggregation of this group of instances has zero impact on the peak. This scenario represents the most efficient use of the power budget.

3.5 Service instance placement

Our workload-aware service instance placement mechanism relies on iterative calculation of asynchrony scores to derive a placement of service instances that leads to more stable, less varying aggregated power consumption under power nodes. The resulting placement has a high asynchrony scores at all levels of power nodes, which maximizes the power headroom and mitigates fragmentation in the datacenter, as we described in Section 2.2. In the following, we dive into the details of our service instance placement mechanism.

Calculating asynchony score vectors for service instances. We first calculate a *asynchrony score vector* v_i for each service instance i. We do this by calculating the asynchrony score of the averaged I-trace of instance i against all the S-traces; each element of this asynchrony score vector which is an asynchrony score between the averaged I-trace of i and one of the S-traces, which we call an *instance-to-service (I-to-S) asynchrony score*. This vector evaluates how adding this service instance to a large mix of other instances potentially impacts the aggregate peak of this group.

We choose to use I-to-S asynchrony scores, instead of instance-to-instance (I-to-I) asynchrony scores, for the evaluation because of two reasons: 1) the pair-wise I-to-I asynchrony score calculation could take an unacceptable amount of time since there are tens to hundreds of thousands of service instances in a datacenter, and 2) after doing the embedding in the next step, these I-to-I asynchrony score vectors will span a sparse high-dimensional space ($> 10^4$), which can lead to overfitting and is not ideal for clustering [44].

Classifying and placing instances. Suppose B is the set of the top power-consumer services, we extract $|B|$ S-traces. Each service instance, hence, has an asynchrony scores vector of length $|B|$. We can then embed all the service instances as data points in a $|B|$-dimensional space spanned by the $|B|$ asynchrony scores. We then apply *k-means clustering* to these points to classify the service instances into highly-synchronous groups. An example of the clustered result is demonstrated in Figure 8. We then select service instances using a round-robin like heuristic and allocate the instances to the power nodes from the top level of the power infrastructure to the bottom.

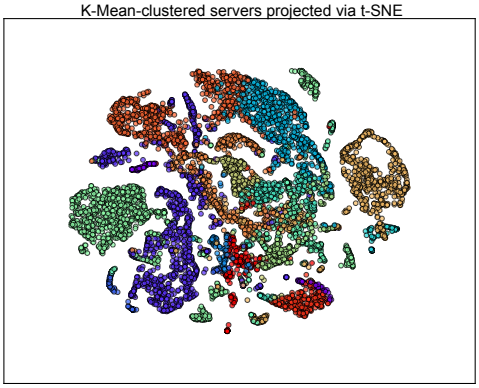

Figure 8. The production service instances in one of the suites of DC1 are embedded into the $|B|$-dimensional asynchrony-score space. k-means clustering is applied to classify asynchronous servers (shown in different colors). This figure shows the projected result onto a 2-dimensional space via t-SNE [33].

For example, assume that we want to place a set of service instances I into the datacenter. We start from the top level l_0 and want to allocate service instances to q second-level nodes. The first step is to extract $|B|$ S-traces out of these servers. For each server, we calculate one asynchrony score against each S-trace, and have $|B|$ asynchrony scores for each service instance in the end. Each server is then considered as a data point in the $|B|$-dimensional space spanned by the asynchrony scores. We then apply k-means clustering to these data points and obtain a set of h clusters, denoted as $C = \{c_1, c_2, c_3, ..., c_h\}$. We configure h to be a multiple of q. Each of these clusters have the same number of instances. For each second-level power node, we iterate through all the clusters, and assign $\frac{|c_j|}{q}$ service instances from cluster j to that power node, and so on, until all the service instances are assigned to the second level power nodes. The process repeats and terminates when all the service instances are assigned to the last-level power nodes.

3.6 Adapting to workload changes

In datacenters, traffic and workload can change over time. Short-term workload uncertainties such as power spikes caused by traffic bursts are handled by commonly deployed emergency measures such as power capping solutions [51], and is out of the scope of this paper. On the other hand, mid-term to long-term workload changes are what we care the most about. These changes are usually caused by the change of accessing patterns, which might gradually make

the power efficiency of the current deployment suboptimal. Although, according to the past data in Facebook datacenters, significant changes rarely occur within months, we want to be able to identify when the current placement becomes suboptimal and apply incremental adjustment to it. To address this issue, our framework continuously records the I-traces and the S-traces, and dynamically re-evaluate the severity of the fragmentation problem by monitoring the sum of peaks of power traces at each level of power infrastructure. When the placement becomes suboptimal with respect to the changing workload, we identify the power nodes with the most severe fragmentation problem (i.e., the node with the lowest asynchrony score or largest sum of peaks), and calculate a *differential asynchrony score* for every server. A differential asynchrony score is calculated between the I-trace of a service instance and the *averaged aggregate power trace* of a large group of server. We define

$$\bar{\boldsymbol{PA}}_{i,N} = \frac{\sum_{(j \in S_N \wedge j \neq i)} \bar{\boldsymbol{PI}}_j}{|S_N - 1|}$$

to be the averaged aggregate power trace of service instance i against power node N, where i is the service instance we are evaluating, and S_N represents the set of servers supplied by power node N; we define the differential asynchrony score of instance i against power node N to be

$$AD_{i,N} = \frac{peak(\bar{\boldsymbol{PI}}_i) + peak(\bar{\boldsymbol{PA}}_{i,N})}{peak(\bar{\boldsymbol{PI}}_i + \bar{\boldsymbol{PA}}_{i,N})},$$

we can then choose the service instance having the worst (lowest) differential asynchrony score, and swap it with some other service instance from another power node, if and only if that swap make the differential asynchrony scores higher at both of the two power nodes involved.

4 Exploiting power budgets with dynamic power profile reshaping

In the previous section, we explore how workload-aware service instance placement based on temporal power behaviors would help alleviating power budget fragmentation. Such mitigation of fragmentation allows datacenters to host more servers and improve throughput. In this section, we further explore the opportunities of achieving additional throughput increase by carefully utilizing the extra servers.

Based on our investigation of production power traces, we design a **proactive dynamic power profile reshaping** approach, which includes two steps: (1) *history-based server conversion* and (2) *history-based proactive throttling and boosting* to further utilize the power headroom. In the rest of the sections, we first discuss the challenges of fully utilizing the unleashed power budget achieved by service placement. We then show how we leverage server conversion with *storage disaggregated servers* [26, 27] to improve the throughput by keeping these extra servers well utilized at all times. Lastly, we show how proactive throttling and boosting intelligently manage power budget allocation, allowing us to deploy extra conversion servers inside datacenters.

In the rest of the section, we denote *latency-critical* workload using **LC**, and *non-latency critical, throughput-oriented* workload using **Batch**.

4.1 Challenges

To utilize the unleashed power headroom we can add extra service-specific servers. This approach, although can improve throughput for the specific service, leaves throughput opportunities on the table. For example, assuming that we add LC-specific servers to a datacenter to use the unleashed power headroom and accommodate extra traffic. During the off-peak hours, even with the increased traffic, however, the original set of LC servers are likely to be sufficient to handle the burden without hurting the QoS of LC servers. In other words, newly added LC-specific servers will be underutilized during those hours. To address the low utilization issue, we want the set of "extra servers" to be able to host batch services to further improve batch throughput during off-peak hours.

4.2 History-based server conversion

Conversion with storage-disaggregated servers Our insight is that the recently proposed storage-disaggregated servers [26, 27] is an ideal platform for solving the above issue. Storage-disaggregated servers are recently widely deployed in Facebook datacenters. In storage-disaggregated servers, the main storage components (i.e., Flashes) are separated from the compute counterparts (i.e., CPUs and memory). The compute nodes access the storage nodes over a high-bandwidth in-datacenter network instead of local PCIe links. This disaggregate approach incurs minimum overhead because the network access latency (microsecond-level) is small compared to disk access latency (millisecond-level) [26].

Benefits of server conversion Using storage-disaggregated servers, we can design server conversion to fully utilize the newly-added servers during both peak and off-peak hours. Server conversion switches the service a server hosts, between LC service and batch service, adaptively based on the load. Server conversion with storage-disaggregated servers offers several advantages. First, it allows us to accommodate the increased LC traffic at peak hours by hosting only LC service and improve the Batch throughput during off-peak hours. Second, these storage-disaggregated servers allow us to maintain a better overall resource utilization while maintaining the data availability. This is one key advantage of using storage-disaggregated servers. Because data reside in their dedicated storage nodes, and is intact and still accessible by other Batch servers even when the server is converted to LC servers during peak time. Third, the server conversion process is low overhead and does not require time-consuming data migration. Last but not least, because server conversion does not require any OS reboots, the OS is always running, meaning that even during conversion, the converted servers are still controllable by other runtime monitors, which ensures that the power safety is maintained during the conversion process.

Design of conversion policy Inspired by prior proposals [7, 32, 34] which leverage workload colocation on leaf machines to improve node-level resource utilization, we design a *server conversion policy* to allow LC and Batch workloads to safely share the available power budget throughout the hierarchy of the power infrastructure. We are also inspired

by previous works [53] and design our conversion policy by taking advantage of the clear patterns found in historical LC load data. The server conversion policy is designed to be applied on the set of conversion servers e_{conv} as follows: First, we learn the *guarded per-LC-server load level* from the historical data (training data), namely the load level of each server when LC achieves satisfactory QoS and define this load level as the *conversion threshold* (L_{conv}). During runtime, we continuously monitor the LC server load over each original set of LC servers. Based on the average load level, we distinguish two phases: a *Batch-heavy Phase* and a *LC-heavy Phase*. When the average LC server load over the original LC server is smaller than L_{conv}, this datacenter is in *Batch-heavy Phase*. When this average LC load increases to a level close to L_{conv}, our server conversion demands the conversion servers to be converted to LC instances, and we enter *LC-heavy Phase*. The threshold L_{conv} is also used to manage the load on each LC server. If any of the LC servers experiences a load higher than L_{conv}, then our server conversion process will stop sending queries to this server, and, instead, send the next query to other LC servers or a conversion server.

We can further maximize the throughput improvement by throttling the Batch clusters' power consumption during peak hours. Such throttling proactively creates additional power headroom during the peak hours, allowing us to house an additional set of conversion servers e_{th} in the datacenter. To achieve further throughput improvement, we augment the previous policy to leverage this set e_{th} of conversion servers. We monitor the load of the original set of LC servers and of the LC servers in e_{conv}, comparing the load with the same conversion threshold L_{conv}. In this augmented policy, the definitions of *LC-heavy Phase* and *Batch-heavy Phase* remain unchanged. One distinction between this augmented policy and the previous policy is that, when the average LC load over the original LC servers and the LC servers in e_{conv} approaches to L_{conv}, we now first throttle the Batch clusters, and then it starts to convert servers in e_{th} into LC servers. Another distinction is that, during *Batch-heavy Phase*, we *boost* the performance of Batch servers to compensate for the loss of throughput caused by the throttling.

5 Evaluation

5.1 Experimental setup

In this work, we conduct our experiments using the power traces measured in three of Facebook's largest datacenters. All of these three datacenters are power supplied by the multi-level power infrastructure described in Section 2. Each datacenter consists of four suites and tens of thousands of servers. For every server housed in these three datacenters, we measure and log three weeks of power trace. The averaged instance power traces constructed by taking the average of the first two weeks of instance power traces serve as our training data. The third week of power traces serve as our testing data. We derive SmoothOperator's power-efficient instance placement and power profile reshaping policies based on the training data, and evaluate the benefit of each these two components using the testing data. The time interval of a week serves well as the unit of evaluation because, in large-scale user-facing datacenters, user traffic has strong day-of-the-week activity patterns [5, 43].

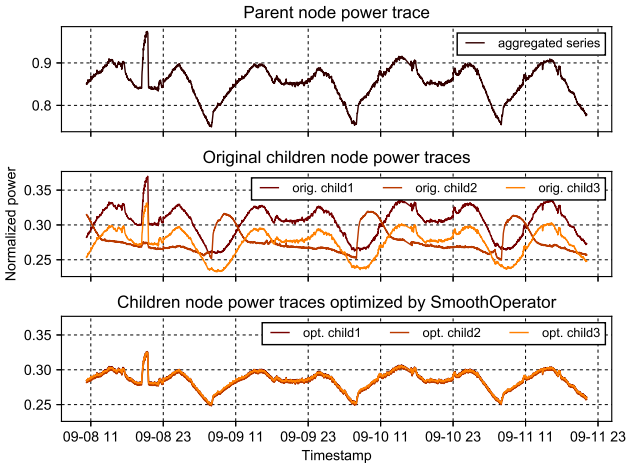

Figure 9. The comparison between the children power trace generated by the oblivious and workload-aware placement in a production suite of DC1. The workload-aware placement generates smoother power traces, greatly alleviating fragmentation. Power peaks are reduced at the child node thus more servers can be supported at each node.

5.2 Results

In this section, we present evaluation results that demonstrate the effectiveness of SmoothOperator on improving the efficiency of power usage and on achieving higher throughput in the three target datacenters. Specifically, we present the following two studies: 1) We show that SmoothOperator's workload-aware service instance placement framework reduces the peak power at different levels of power nodes, mitigating the power budget fragmentation problem and allowing datacenter operators to host more servers under the same power budget. 2) We show that, with SmoothOperator's dynamic power profile reshaping policy, we improve the power utilization during non-peak hour, which further improves service throughputs.

5.2.1 Peak power reduction by workload-aware service placement

We start with the study that shows how SmoothOperator's workload-aware service instance placement reduces the peak power and mitigates the power budget fragmentation problem. As we described in the previous sections, the distinctive diurnal patterns of services and the service-level and instance-level heterogeneity provide abundant opportunities for improving the efficiency of power utilization in these datacenters.

Figure 9 presents how our framework reduces fragmentation using production power traces. In this figure, we apply SmoothOperator's workload-aware instance placement to the sub-tree rooted at a middle-level power node N, including node N and all the descendent power nodes and the service instances supplied by N. We demonstrate in the top

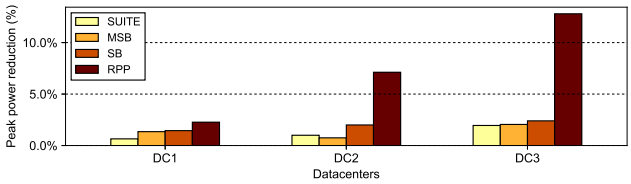

Figure 10. The peak-power reduction achieved at each level of the power infrastructure in the three datacenters under study. There is significant peak reduction at RPP level, which directly translates to the percentage of extra servers that can be hosted.

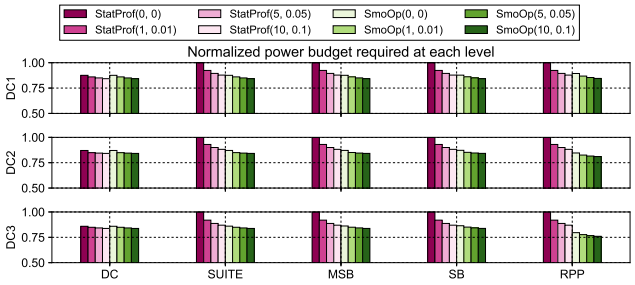

Figure 11. Required power budget achieved by previous work and SmoothOperator. StatProf(u, δ) refers to the result of previous work with a degree of under-provisioning u and a degree of overbooking δ. SmoOp(u, δ) refers to the SmoothOperator counterpart.

graph in Figure 9 the power trace N. In the middle graph, we show the power traces of the children power nodes of N, one for each child, *before* we apply the workload-aware service instance placement on N. In the bottom graph, we show the power traces of N's children nodes *after* we apply our workload-aware service instance placement to N. Note that the power trace at N is not changed by SmoothOperator because, in the example, our placement policy does not move service instances into or out of the subtree rooted at N. This figure shows that SmoothOperator, with the placement step alone, makes the power traces of the children power nodes less varying and more balanced, and reduces the peak power of the children power nodes.

Peak power reductions achieved at different levels of power infrastructure We present the peak power reduction at all levels in all three datacenters in Figure 10. Recall from Section 2 that, for a same power delivery tree, the sum of peak powers of power nodes at a certain level is an important indicator of the severity of power budget fragmentation at that level. As shown in Figure 10, we find our workload-aware service instance placement can reduce the RPP-level peak power by 2.3%, 7.1% and 13.1% for the three datacenters, respectively. At higher levels, SmoothOperator achieves less significant reduction. The reason is that each of these higher level power nodes *indirectly supply* a group of up to thousands of service instances with higher degree of heterogeneity within them than the leaf nodes. Note that, however, the leaf power nodes suffer from fragmentation significantly. In other words, peak power reductions at the lowest level are of the utmost importance, as servers can only be directly power supplied by the low-level power nodes. These reductions translate to the proportion of extra servers allowed to be housed under the same power infrastructure.

While we can extract power headroom in DC3, the benefit we get in DC1 is smaller. The reasons are two-fold: First, the degree of heterogeneity among instances power traces found is DC1 is much smaller than that in DC3. Second, due to the lower degree of heterogeneity, the baseline (original) placements in DC1 suites are more balanced compared to DC3. For DC3, synchronous service instances are largely placed under the same sub-trees of the power infrastructure in the original placement, allowing us to achieve improvement.

We compare our approach to a previous work [20]. This work aims to optimize the provisioning of the capacity of power nodes in datacenters. It models power pattern of instances and power nodes as cumulative distribution functions (CDFs), and relies on leveraging these probability distributions to aggressively under-provision and overbook power nodes.

For example, to power supply a set of service instances M, this previous work models the power profile of each instance i in M as a c_i and defines a *degree of under-provisioning* u. The budget of the power node that supplies M will be set to $\sum_{i \in M} c_{i,u}$, where $c_{i,u}$ denotes the $(100-u)$-th percentile power of instance i's power profile c_i. This work also recognizes that, at datacenter-level, they can take advantage of the heterogeneity among the instances with overbooking, and defines a *degree of overbooking* δ, which further reduces the datacenter-level provisioning requirement. Suppose the datacenter-level power capacity was $\sum_{i \in dc} c_{i,u}$; with δ, the capacity can be further reduced to $\sum_{i \in dc} c_{i,u}/(1+\delta)$.

The comparison can be found in Figure 11, which shows the power budget provisioning required by the three datacenters after applying the two approaches. Since the under-provisioning and overbooking techniques used in the prior work is independent with our techniques, in our experiment, we also add several configurations of SmoothOperator (`SmoOp`) in which under-provisioning and overbooking are allowed. We denote `StatProf(`u`, `δ`)` as the configuration of previous work with a degree of under-provisioning u and a degree of overbooking δ, and `SmoOp(`u`, `δ`)` as the SmoothOperator counterpart. Note that, with such notation, SmoOp(0, 0) represents using SmoothOperator alone without any under-provisioning nor overbooking.

We can see that SmoOp(0, 0) achieves >12% of reduction in the required power budget provisioning in all cases, and it achieves higher level of improvement over the prior work as we go down the hierarchy of power infrastructure. When compared to StatProf, across all the levels, SmoOp(0,0) almost always outperforms or is on par with the most ambitious StatProf (i.e., StatProf(10, 0.1)). If we allow SmoothOperator to under-provision and overbook, across all levels and datacenters, it always requires less power budget provisioned than the StatProf counterpart, by up to >10%. For instance, in DC3, while StatProf(10, 0.1) achieves only 13% of reduction, SmoOp(0, 0) achieves 20% and SmoOp(10, 0.1) achieves 24%.

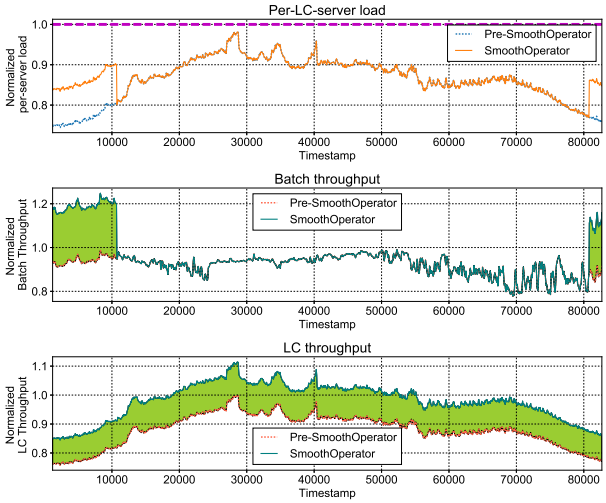

Figure 12. Server conversion's impact on per-LC-server load, LC and Batch throughput.

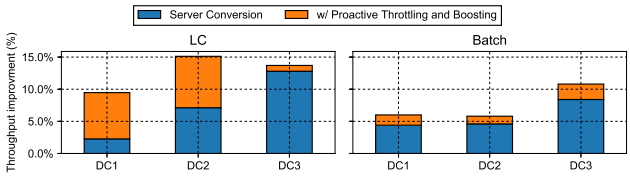

Figure 13. The breakdown of throughput improvement of LC and Batch services.

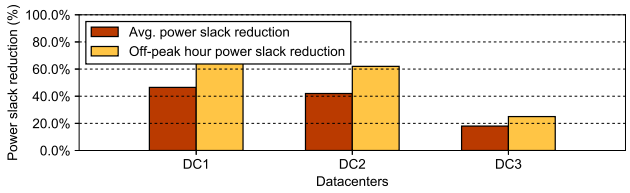

Figure 14. Average power slack reduction and off-peak phase power slack reduction achieved at three of Facebook's production datacenters.

These results show that, by intelligently leveraging temporal heterogeneity among instance power traces, SmoothOpeartor lowers the peak of the power profiles at power nodes, *creating* the extra headroom that was not available to the prior work. Moreover, since SmoOp(0, 0) always outperforms StatProf(10, 0.1), we conclude that the benefit we get from SmoothOperator does not need to rely on probabilities, implying higher level of safety guarantee at power nodes.

5.2.2 Benefit of dynamic power profile reshaping

In this subsection, we present evaluation results demonstrating the benefit of using dynamic power profile reshaping to utilize the power budget unleashed by SmoothOperator's workload-aware service instance placement.

Impact of server conversion We present a segment of our experiment (Figure 12) to highlight the server conversion's impact on Batch throughput, LC throughput, and the load per LC server. In this example, the DC optimized by the workload-aware service instance placement has 11% of extra power headroom to accommodate extra traffic. If we add only LC-specific servers, this datacenter can achieve 11% extra LC throughput. If we use conversion servers to fill in the power gap, we gain extra benefit for Batch services. As shown in the top subgraph in Figure 12, during Batch-heavy Phase, the per-server load for LC-servers is low and the original set of LC servers are underutilized. During this time, we do not need extra computing power to handle the incoming LC queries. As long as the LC servers are under a specified guarded load level, the added conversion servers can be converted to Batch service instances to perform Batch workload, as shown in the middle subgraph of Figure 12. On the other hand, during LC-heavy phase, the original set of LC servers do not have the capacity to handle the increased LC traffic. This is when conversion servers need to kick in to help reduce the per-server load for LC servers. In the top subgraph of Figure 12 we see such an impact.

Figure 13 presents the throughput improvement achieved by both server conversion and proactive throttling and boosting policies throughout the entire experiments. It shows that, with server conversion alone, we are able to utilize the 13% unlocked power budget to trade for up to 13% LC throughput plus 8% Batch throughput at the same time.

Impact of proactive throttling and boosting In this section, we evaluate the benefit we obtain from applying proactive throttling to the Batch instances. The throttling and boosting policy proactively throttles the Batch services to increase the power budget allocation for LC services during LC-heavy Phase; during the Batch-heavy Phase, it only increases the allocation and boosts the performance of Batch instances, but does not throttle the LC services.

Figure 13 demonstrates the extra throughput improvement achieved by the proactive throttling and boosting policy on top of server conversion. The improvement of Batch throughput here is small, 1.6%, 1.2%, and 2.4%, for the three datacenters, respectively. This is as expected because the gain brought by the extra conversion server barely compensates the loss caused by aggressive throttling during LC-heavy Phase. On the other hand, we largely improve the capacity of the LC service in 2 of the 3 datacenters during the peak hours. The extra improvement of LC throughput is 7.2%, 8%, and 1.8% for DC1, DC2, and DC3, respectively, which translates to a capacity gain that can accommodate multi-millions extra queries per second.

Power slack reduction In Figure 14, we present the result of average power slack reduction and off-peak hour power slack reduction of the three datacenters. The reduction of power slack means that the power budget available during off-peak hours is used to do more work. From Figure 14 and Figure 13 we find that the benefit gained in DC3 is smaller compared to the other two datacenters. The reason of this is that DC3 has a large proportion of LC service instances among the top power consumers compared to the Batch type counterpart. Therefore fewer Batch service instances can be

Table 1. Comparison between SmoothOperator and prior approaches for improving datacenter power efficiency

	Power Routing [38]	Stat. Multiplexing [20]	DistributedUPS [28]	SmoothOperator
Using temporal information			✓	✓
Using existing power infra.		✓		✓
Automated process	✓	✓	✓	✓
Balancing local peaks	✓			✓
Proactive planning				✓

throttled for LC to borrow power budget from, limiting the improvement of Batch throughput. In other two datacenters, power slack reduction are achieved. As shown in Figure 14, the dynamic power profile reshaping achieves 44%, 41%, and 18% average power slack reduction, respectively in the three datacenters.

6 Related Work

Aside from prior proposals focusing on improving node-level power and energy efficiency [24, 31, 32], there has been a large body of previous works addressing power and energy efficiency problems in datacenters from a system architecture's perspective.

One class of solutions tries to address the power bottleneck by reducing the power consumption of computation. Power management problems at server level and small cluster level have drawn significant attention [8, 11, 13, 15, 19, 25, 30, 36, 37, 40, 41, 45–47]. A popular approach among them relies on using virtual machines (VMs) to achieve work consolidation and performance isolation among different workloads, and treats VM consolidation as a resource allocation problem [8, 15, 30, 37, 40, 47]. However, due to strict latency requirement and simplicity of management, large-scale production datacenters, such as Facebook's and Microsoft's [53], deploy their datacenter without virtualization in a more autonomous manner. In this type of datacenters, each service team deploy their service on their own, separate set of physical servers, and different major services do not share physical servers. This consolidation-based approach is not directly applicable in this type of datacenters.

Recently, researchers started to consider intelligently charging and discharging energy storage devices (ESDs) to enable temporary excessive power draw at power nodes during power emergencies [4, 21, 22, 28, 49, 52]. The problem of this type of approach is similar to power-capping-based approaches, in that the unbalanced peaks across the datacenter can make the sharp peaks at some of the nodes deplete the ESDs quickly, while at some power nodes the power draw never entails the extra capacity.

There are other proposals of flexible power infrastructure focusing on temporarily extending (or reallocating) the power budget among power nodes [12, 38] to adapt to dynamism of power consumption. Among them, power routing [38] suggests dynamically connecting rows and racks of servers to different power nodes, in order to balance the load among power nodes. Dual-corded power supply, however, only provides limited flexibility (degree of 2) for power routing purpose. To maximize the flexibility, normal fault-tolerant dual-corded power supply architecture must be expanded to enable richer connectivity. The costly upgrade to the infrastructure and a significant change to the power supply topology required by their solution can further lead to long service down time during the installation and setup process.

Several prior works [14, 20, 48] showed that by statistically multiplexing the probability distributions (PDFs) of power of different workloads, datacenters can overbook and/or under-provision safely. These works, however, do not take advantage of time-series power information of workloads. Our insight is that, the strong diurnal patterns and the asynchronous power behavior across workloads provide significant opportunities for de-fragmenting power budget. Also, their techniques degrade the performance of user-facing services significantly during the peak time, which is not ideal for user-facing datacenter.

In fact, SmoothOperator and a lot of the works mentioned in this section are not mutually exclusive. The workload-aware service instance placement framework is complementary to many of the prior solutions.

7 Conclusion

In this work, we investigate three of Facebook's datacenters and aim to solve the power budget fragmentation problem found in them. We leverage the knowledge of the temporal heterogeneity of power consumptions of different workloads, and apply our workload-aware service instance placement technique to unlock the wasted power budget, and evaluate the effect of server-conversion-based dynamic power profile reshaping runtime in production environment. Our result shows that we are able to host up to 13% more servers in production environment by applying our placement technique, without making modification to the underlying power infrastructure. To this end, we reduce up to 44% of average power slack inside datacenters. Without incurring significant burden on latency-critical servers, we achieve up to 13% plus 8% throughput improvement for latency-critical service and batch service, respectively, by using server conversion alone; or 15% and 11% throughput improvement with the help of DVFS.

8 Acknowledgment

We thank our anonymous reviewers for their constructive feedback and suggestions. We also thank Dr. Md E. Haque, Mr. Yunqi Zhang, and Mr. Pai-Shun Ting for their time and valuable input. This work was sponsored by Facebook, the National Science Foundation (NSF) under grants IIS-VEC1539011, and NSF CAREER SHF-1553485.

References

[1] 2015. Earnings Buzz: Twitter Inc (TWTR), Facebook Inc (FB), LinkedIn Corp (LNKD). (2015). http://www.ibtimes.com/earnings-buzz-twitter-inc-twtr-facebook-inc-fb-linkedin-corp-lnkd-2028461.

[2] 2016. Open Compute Project. (2016). http://www.opencompute.org.

[3] 2017. Facebook Newsroom. (2017). http://newsroom.fb.com/company-info/.

[4] Baris Aksanli, Eddie Pettis, and Tajana Rosing. 2013. Architecting Efficient Peak Power Shaving Using Batteries in Data Centers.

[5] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 267–280.

[6] Arka A Bhattacharya, David Culler, Aman Kansal, Sriram Govindan, and Sriram Sankar. 2013. The need for speed and stability in data center power capping. *Sustainable Computing: Informatics and Systems* 3, 3 (2013), 183–193.

[7] Alex D. Breslow, Ananta Tiwari, Martin Schulz, Laura Carrington, Lingjia Tang, and Jason Mars. 2013. Enabling Fair Pricing on HPC Systems with Node Sharing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*.

[8] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. 2001. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review* 35, 5 (2001), 103–116.

[9] Hao Chen, Can Hankendi, Michael C Caramanis, and Ayse K Coskun. 2013. Dynamic server power capping for enabling data center participation in power markets. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 122–129.

[10] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: memory power estimation and capping. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*. IEEE, 189–194.

[11] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F Wenisch, and Ricardo Bianchini. 2012. Coscale: Coordinating cpu and memory system dvfs in server systems. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 143–154.

[12] W. El-Essawy, A. P. Ferreira, J. C. Rubio, T. Keller, K. Rajamani, and M. Ware. 2011. Enabling Real-Time Data Center Energy Management. (2011).

[13] Songchun Fan, Seyed Majid Zahedi, and Benjamin C Lee. 2016. The computational sprinting game. In *ACM SIGOPS Operating Systems Review*, Vol. 50. ACM, 561–575.

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

[15] Shuo Fang, Renuga Kanagavelu, Bu-Sung Lee, Chuan Heng Foh, and Khin Mi Mi Aung. 2013. Power-Efficient Virtual Machine Placement and Migration in Data Centers.

[16] Xing Fu, Xiaorui Wang, and Charles Lefurgy. 2011. How Much Power Oversubscription is Safe and Allowed in Data Centers. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11)*. ACM, New York, NY, USA, 21–30. https://doi.org/10.1145/1998582.1998589

[17] A. Gandhi, Yuan Chen, D. Gmach, M. Arlitt, and M. Marwah. 2011. Minimizing data center SLA violations and power consumption via hybrid resource provisioning. In *Green Computing Conference and Workshops (IGCC), 2011 International*.

[18] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, Jeffrey O Kephart, and Charles Lefurgy. 2009. Power capping via forced idleness. (2009).

[19] Lakshmi Ganesh, Jie Liu, Suman Nath, and Feng Zhao. 2009. Unleash stranded power in data centers with RackPacker. *Workshop on Energy-Efficient Design (WEED)* (2009).

[20] Sriram Govindan, Jeonghwan Choi, Bhuvan Urgaonkar, Anand Sivasubramaniam, and Andrea Baldini. 2009. Statistical profiling-based techniques for effective power provisioning in data centers. In *Proceedings of the 4th ACM European conference on Computer systems (EuroSys)*. 317–330.

[21] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. 2011. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *ACM SIGARCH Computer Architecture News*.

[22] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. 2012. Leveraging Stored Energy for Handling Power Emergencies in Aggressively Provisioned Datacenters. In *ACM SIGARCH Computer Architecture News*.

[23] Can Hankendi, Sherief Reda, and Ayse Kivilcim Coskun. 2013. vCap: Adaptive power capping for virtualized servers. In *ISLPED*.

[24] Chang-Hong Hsu, Yunqi Zhang, Michael A Laurenzano, David Meisner, Thomas Wenisch, Jason Mars, Lingjia Tang, and Ronald G Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 271–282.

[25] Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 93.

[26] Ana Klimovic, Christos Kozyrakis, Eno Thereksa, Binu John, and Sanjeev Kumar. 2016. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys)*.

[27] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2017. ReFlex: Remote Flash? Local Flash. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 345–359.

[28] Vasileios Kontorinis, Liuyi Eric Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean M Tullsen, and Tajana Simunic Rosing. 2012. Managing distributed ups energy for effective power capping in data centers. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*. IEEE, 488–499.

[29] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. 2008. Power capping: a prelude to power shifting. *Cluster Computing* 11, 2 (2008), 183–195.

[30] Harold Lim, Aman Kansal, and Jie Liu. 2011. Power Budgeting for Virtualized Data Centers. In *2011 USENIX Annual Technical Conference (USENIX ATC'11)*.

[31] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards energy proportionality for large-scale latency-critical workloads. In *ACM SIGARCH Computer Architecture News*, Vol. 42. IEEE Press, 301–312.

[32] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: improving resource efficiency at scale. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 450–462.

[33] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9 (2008).

[34] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 248–259.

[35] David Meisner, Christopher M Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F Wenisch. 2011. Power management of online data-intensive services. In *International Symposium on Computer Architecture (ISCA)*.

[36] David Meisner, Christopher M Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F Wenisch. 2011. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 319–330.

[37] Ripal Nathuji and Karsten Schwan. 2007. Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems. In *ACM SIGOPS Operating Systems Review*.

[38] Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F. Wenisch, and Jack Underwood. 2010. Power routing: dynamic power provisioning in the data center. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*.

[39] Vinicius Petrucci, Michael A Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. 2015. Octopus-man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers. In *International Symposium on High Performance Computer Architecture (HPCA)*.

[40] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, Vol. 36. ACM, 48–59.

[41] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey Chase. 2006. Ensemble-level power management for dense blade

servers. In *ACM SIGARCH Computer Architecture News*, Vol. 34. IEEE Computer Society, 66–77.

[42] Sherief Reda, Ryan Cochran, and Ayse K Coskun. 2012. Adaptive power capping for servers with multithreaded workloads. *IEEE Micro* 5, 32 (2012), 64–75.

[43] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, Vol. 45.

[44] Michael Steinbach, Levent Ertöz, and Vipin Kumar. 2004. The challenges of clustering high dimensional data. In *New directions in statistical physics*. Springer, 273–309.

[45] Balaji Subramaniam and Wu-chun Feng. 2015. Towards energy-proportional computing using subsystem-level power management. *arXiv preprint arXiv:1501.02724* (2015).

[46] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. 2013. Crank it up or dial it down: coordinated multiprocessor frequency and folding control. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 210–221.

[47] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*.

[48] Di Wang, Chuangang Ren, Sriram Govindan, Anand Sivasubramaniam, Bhuvan Urgaonkar, Aman Kansal, and Kushagra Vaid. 2013. ACE: Abstracting, characterizing and exploiting datacenter power demands.

[49] Di Wang, Chuangang Ren, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Hosam Fathy. 2012. Energy Storage in Datacenters: What, Where, and How Much?. In *ACM SIGMETRICS Performance Evaluation Review*.

[50] Xiaorui Wang, Ming Chen, Charles Lefurgy, and Tom W Keller. 2009. SHIP: Scalable hierarchical power control for large-scale data centers. In *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*. IEEE, 91–100.

[51] Qiang Wu, Qingyuan Deng, Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. 2016. Dynamo: Facebook's Data Center-Wide Power Management System. In *International Symposium on Computer Architecture (ISCA)*.

[52] Jianguo Yao, Xue Liu, Wenbo He, and Ashikur Rahman. 2012. Dynamic Control of Electricity Cost with Power Demand Smoothing and Peak Shaving for Distributed Internet Data Centers.

[53] Yunqi Zhang, George Prekas, Giovanni Matteo Fumarola, Marcus Fontoura, Inigo Goiri, and Ricardo Bianchini. 2016. History-Based Harvesting of Spare Cycles and Storage in Large-Scale Datacenters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.