Pelican: **P**ow**e**r Schedu**l**ing for QoS **i**n Large-s**c**ale Dat**a** Centers with Heteroge**n**eous Workloads

Bing Luo[*], Wei Chen[†], Xingxing Liu[†], Xiaozhong Li[†] Lifei Zhang[†] and Weisong Shi[*]
[*] Computer Science, Wayne State University, Detroit, MI, USA
Email: {bing.luo,weisong}@wayne.edu
[†] Baidu Inc., Beijing, China
Email: {chenwei24,liuxingxing01,lixiaozhong,lifeizhang}@baidu.com

***Abstract*—Given the high cost of building and operating large-scale data centers in terms of the power budget, it is crucial to maximize available power resources for overall quality of service(QoS). In this paper, we present a power scheduling system, Pelican, for large-scale data centers to increase overall QoS by over-provisioning the number of servers for heterogeneous workloads. We adopt a two-level design that separate power scheduling mechanism and power controlling policy without affecting the already over-complicated job scheduler. Our power controller leverage both priority and performance as key indicators to control the power of a server, which improves QoS while significantly reducing the risk of power violations.**

We implemented and deployed our Pelican in a real over-provisioned production data center at Baidu Inc. Our evaluations show that we are able to utilize 12.7% more computing resources without any violation and achieve 11.5% offline service throughput gain while not affecting the latency of online services so as to improve overall QoS.

***Index Terms*—power control, data center, scheduling**

I. Introduction

The rapid adoption of Internet of Things and edge computing, coupled with surge growth of numerous cloud services, have been accelerating the growth of data center demand worldwide. To build and operate data centers is extremely expensive in terms of the power budget. The industry average capital expenditures (CapEx) of IT critical power is about 10,000 to 25,000 USD per kilowatt [1] and the operational expenditures(OpEx) can reach 1,000 to 3,000 USD per kilowatt year in our data center. Given the tremendous expense, it is crucial to fully utilize the power capacity of data centers to reduce the Total Cost of Ownership(TCO) and improve Quality of Services(QoS).

Conservative server provisioning along with the diurnal pattern [2], [3], unfortunately, lead to typical low power utilization in modern data centers [4]–[7]. In order to take this opportunity to improve QoS, Wang *et al.* provisioned extra servers and implemented a statistical power control system [8]. By proactively scheduling fewer jobs to rows with high power utilization, it showed the possibility to improve throughput per provisioned watt while preventing clusters from power outage. The solution, however, is mainly suitable for offline workloads. Furthermore, scheduling a task to a different server do not affect its performance only when the task is location independent. In practical, many offline workloads require to run on the server where data is located because of the huge data size. This limits the type of workloads.

Combining online and offline services on the same set of servers can obviously increase the power utilization but also introduce more challenges. **1)** The number of online services tasks is purely depending on user input, which data center manager cannot control. Large power fluctuations are common and the duration for the increase in power is much shorter than offline workloads. **2)** To make heterogeneous workloads work harmoniously, the complexity of job scheduler is ever increasing. Furthermore, we may rethink the relationship between job scheduler and power management.

To address the above limitations and challenges, we investigate the behavior of power and task in data centers and present Pelican, a new power scheduling system for large-scale data centers with heterogeneous workloads. Instead of moving tasks on spatial dimension, we tried to move tasks on temporal dimension. Based on the current power of a rack, we will find an optimized power budget for each server and by limiting resources for tasks so as to control the power of a server. In this way, our system can control power resource without modifying the task assignment. Our work is evaluated in a production data centers in Baidu, the largest search engine and one of largest cloud service providers in China with millions of servers running billions of tasks per day.

To summary, our major contributions include the following:

- A new power scheduling system, Pelican, is proposed for QoS in large-scale data centers with fixed power budget and heterogeneous workloads by improving power utilization on a large scale;
- A two-level design is adopted to separate overall power scheduling and local power controlling for each server, which makes our power scheduling system more flexible and efficient for heterogeneous workloads;
- A simple and effective method is introduced that leverage task priority and heterogeneous workload property to improve QoS without modifying the task assignment;
- A large-scale empirical evaluation of Pelican is conducted against production workloads in a real data center to show the effectiveness and performance of our system.

The rest of the paper is organized as following: Section II introduces the background and our observations. We introduce our system design and implementation of Pelican in Section III and evaluate the system performance empirically in Section IV. We then discuss related works in Section V and followed by a conclusion in Section VI.

II. Review and Observation

As shown in [8], low average power utilization, especially at a larger scale, in the data center along with conservative server provisioning is one of the main opportunities for over-provisioning. Here, we are going to provide background information as well as review our data center power architecture, provisioning, which lead to our design.

A. Rack power

A server does not have hard power budget as long as rack power distribution unit(PDU) can supply enough power but maximum power of each model of server is measured for provisioning. Following the definition in [8], we called it the *rated power*, or measured maximum power draw from equipment. Rack level-power, however, is limited by both physical limits and limits from row level supply. If the power of a rack exceeds its power budget, we call it a *power violation*.

The overall power utilization of data centers in Baidu, Inc. is about 70% to 80%. In order to improve power utilization, some data centers have provisioned more server. To ensure safety, the data center operator chooses a power limit that is lower than physical power limit. And a rack level DVFS is deployed to those data centers to enforce the power limit. We monitored the power of a rack for 31 days and found out that the rack power exceeded rated power for 2% of the time. The maximum normalized power can reach 1.06. As a result, we need a better power management solution to improve QoS for over-provisioned racks with heterogeneous workloads.

B. Power controlling

Without modifying existing software and hardware on a server, there are two major ways to control power. One way is cooperating with job scheduler to change the placement of jobs so as to move the power in the spatial dimension. The other way is to control the computing resources assigned to tasks so as to move the power in the temporal dimension. The power variations in both temporal (over time) and spatial (across different racks) makes it possible to do either choice. In this paper, we choose to use the second way. One of the reason is that our services are location sensitive so that power controlling should not influence job placement. The most difficult part for the second way is how we can reduce performance impact. If we can somehow ”move” part of power usage of long run and resource hungry tasks during busy time to idle time, then we can make less performance impact on short and latency sensitive tasks. To do so, there are two questions. The first question is how to select proper tasks. In a heterogeneous workloads environment, offline workloads are usually much longer than online workloads and also consumes more power. Short tasks finished quickly so that we have no chance to do such operation. On the other hand, online workloads are more latency sensitive while offline workloads are more focus on throughput. As a result, offline workloads are preferred. Within offline tasks, we can choose lower priority instead of estimating the length of execution, which is much easier to obtain. In addition, the priority of offline workload is lower than online workload so that priority could be a good indicator to choose tasks. The second question is how can we ”move” the power. Generally, we can achieve it by controlling computing resources assigned to a task. To make the controller applicable to all servers in our data center, as well as keep our controller simple, we decide to use core binding to control the power of a specific task. Because core management is a common part when handling heterogeneous workloads, for example, some cores are reserved only for online workloads. Core binding can also react in a short time period and it is widely supported both by OS and hardware in all servers in our data centers.

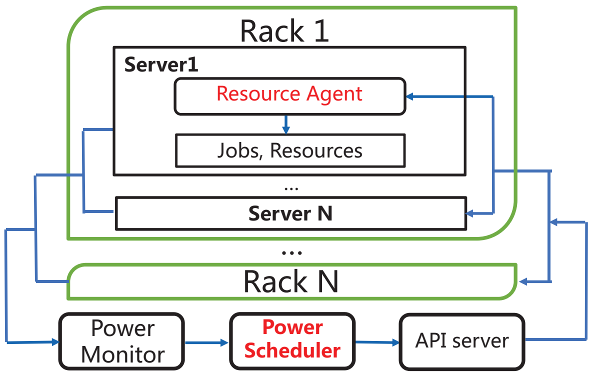


Fig. 1. System Architecture of Pelican.

III. Pelican Design and Implementation

A. Architecture

The overall architecture of Pelican is shown in Figure 1. The power scheduler implements most functionality of Pelican, which is responsible to manage one or multiple racks. At the beginning of each time interval(represented by T), the scheduler reads power data from our centralized power monitor, computes the power budget for each server, and deliver power budget through budget API to resource agent on each server. The resource agent then limit power usage by process management and resource management.

B. Power monitoring

We implement our own power monitor, which collects server-level power utilization, among other metrics through the intelligent platform management interface (IPMI). The power monitor samples the power from each server every time period. Our power monitoring service remains stateless for easy recovery. In our experiment, the monitoring cycle is ten seconds, which we believe is a good trade-off between measurement accuracy and monitoring overhead.

C. Resource agent and budget API

We use the budget API to indirectly control the resource assigned to tasks running on servers and leave power controlling policy to resource agents on servers so that different cluster could have various policies depending on workload and type of service. Since local server knows exact types and priorities of tasks running on it, it is appropriate to do local resource management instead of managing by power scheduler.

A centralized API server is implemented to act as an adapter to deliver power budget commands to a specific server. In this way, the scheduler does not need to locate and communicate with individual servers in the data center.

D. Scheduler

With API server, we implement a scheduler that periodically adjusts the power budget of servers so that the total power of a rack stays under the target power specified by data center operators. A scheduler can manage several racks at the same time. The decision process is independent and identical based on different inputs from racks. So here we discuss the scheduling method for one rack.

Algorithm 1 shows the overall schedule logic. Data center operator first decides a fixed power budget B_r for a rack so that our goal is to limit overall rack power less than it. Then a power threshold P_t is determined based on the change of power of a rack within a time period (See details in Section III-E). At each interval (ten seconds in our implementation), the scheduler obtains power utilization information from the power monitor. If current rack power exceeds power threshold, we will compute the power budget for each server in a rack using the schedule model, which we will discuss in Section III-E. The goal here is to keep rack power under power threshold. However, even if the rack power is lower than the power threshold, we should not stop controlling immediately because the observed rack power is a result of working resource agent. On the other hand, rack power in the next interval may exceed the power threshold again causing resource agent to be on and off frequently. That's why a safe power threshold P_s is introduced to determine if the rack power is safe enough. P_s should be less than P_t and is also determined based on the change of power but we will tune this value in Section IV-E. Only if current rack power is less than safe power threshold, the scheduler will notify resource agent to stop resource limit. Finally, we send power budget command to API server to apply on each server. To maximize the power utilization, P_t and P_s should be as close to B_r as possible and they are both related to future rack power. Instead of implementing a predictor, we show how we use a heuristic method to estimate P_t and P_s in the later part of this section.

The scheduling cycle matches the power monitoring cycle. Note that the scheduler cannot monitor or control any power fluctuation within a time interval, 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.

The scheduler is designed to be stateless, as a result, we can easily switch to a replica if any scheduler fails.

```
Input:
  – $P_r^k$: Current power of rack $k$
  – $P_t^k$: Threshold of rack $k$
  – $P_s^k$: Safe rack power of rack $k$
  – $N$: The number of racks
  – $P_i^k$: Current power of server $i$ in rack $k$
  – $D_i^k$: Dynamic power of server $i$ in rack $k$
  – $n_k$: The number of servers in rack $k$
1:  procedure POWER SCHEDULING
2:      for $k \leftarrow 1, N$ do
3:          if $P_r^k > P_t^k$ then
4:              $P_{rest} \leftarrow P_r^k - P_t^k$
5:              Sort servers by $D_i^k$ in decreasing order
6:              for $i \leftarrow 1, n_k$ do
7:                  if $P_{rest} \geq D_i^k \cdot R_{max}$ then
8:                      $B_i^k = P_i^k - D_i^k \cdot R_{max}$
9:                  else if $P_{rest} > 0$ then
10:                     $B_i^k = P_i^k - P_{rest}$
11:                 else
12:                     $B_i^k = P_i^k$
13:                 Send $B_i^k$ to budget API
14:         else if $P_r^k < P_s^k$ then
15:             Send stop command to budget API
        return
```

Algorithm 1: Power scheduling algorithm

E. Computing the power budget for servers

In order to avoid power violation due to a sudden power surge, we need to leave a safety margin. P_t and P_s describe the margin, as it determines when to turn resource agent on and off. The change of power is basically affected by the temporal variation of workload. We use a data-driven approach to estimate them. We would like to keep P_t and P_s close to rack power budget.

We monitor the power of all racks 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.

Our estimation is conservative as we are preparing for almost the largest change in observed history. We can use a better online power prediction model to get a better estimation, which we leave for future work.

To determine the budget for each server at the begin of a time interval is the most important task for the scheduler. We want to limit enough power usage to avoid power violations, and in the meantime, reduce overall negative performance impact on a rack.

Suppose P_i is the current power of i-th server in the rack and I_i is the idle power of the server. Then an effective power

Symbol	Description
P_r	The current overall rack power.
P_t	The rack power threshold.
P_i	The power of i-th server in the rack.
I_i	The idle power of i-th server.
D_i	The dynamic power of i-th server.
Δ_i	The expected reduced power by resource limitation.
R_i	The expected power reduce ratio.
R_{max}	The maximum allowed power reduce ratio.
B_i	The result power budget for i-th server.
P'_r	The expected rack power in next time interval.

TABLE I
KEY NOTATIONS IN PROBLEM FORMULATION. ALL POWER METRICS USED IN THE PROBLEM FORMULATION IS NORMALIZED TO P_M.

budget B_i should belong to $[P_i, I_i]$, where the maximum power of a server can possibly reduce is the dynamic power of the server $D_i = P_i - I_i$. So if we decide the value of B_i, the expected reduced power is $\Delta_i = P_i - B_i$. Thus, the expected rack power $P'_r = P_r - \sum_i \Delta_i$ Now we introduce expected power reduce ratio as $R_i = \Delta_i / D_i$. This ratio basically compares the expected reduced power to its current dynamic power, which quantifies the impact for current control. Obviously, the range of R_i is $[0, 1]$. To limit performance impact of control, our goal is to minimize average of R_i, which we call it impact ratio $R = \sum_i R_i / n$. Furthermore, we have an upper bound on ratio R_i, denoted as R_{max}.

Therefore, we formulate the Power Scheduling Problem (**PSP**) as:

$$\begin{aligned} \min \quad & R = \textstyle\sum_i R_i / n & (1) \\ \text{s.t.} \quad & 0 \le R_i \le R_{max} & (2) \\ & P'_r \le P_t, & (3) \\ & P'_r = P_r - \textstyle\sum_i \Delta_i & (4) \\ & B_i = P_i - R_i \cdot D_i & (5) \end{aligned}$$

where B_i is the result power budget for each server.

Table I summarizes key notations we used in the problem formulation.

PSP problem is a typical linear programming (LP) problem. There are many methods and tools to compute the solution of this LP problem if a solution exists. The solution of PSP problem, however, is special so that we can obtain the solution much easier instead of solving the general problem directly. Without the loss of generality, we assume that servers are sorted by D_i in decreasing order. Then the solution will be:

$$B_i = \begin{cases} P_i - R_{max} \cdot D_i, & \text{if } i < k \\ P_i - (P_r - P_t - \sum_{i=0}^{k} R_i \cdot D_i), & \text{if } i = k \\ P_i, & \text{otherwise} \end{cases} \quad (6)$$

where

$$k = \sup_{x \in \mathbb{Z}} \left(\sum_{i=0}^{x} R_i \cdot D_i \le P_r - P_t \right) \quad (7)$$

So basically the scheduler will limit power budget for servers with higher dynamic power.

F. Resource agent

As we discussed in Chapter II, the local resource agent limit power usage by leveraging core binding to control resources of tasks in our implementation. When a resource agent received a power budget command, it will control the number of binding cores in order to control the server power under a specified power budget until it receives a stop command. The resource agent reads power information directly from the server and the controlling cycle in our implementation is $1s$. Suppose the resource agent on server i receive a power budget B_i, then we can calculate the expected power reduce ratio $R_i = \frac{P_i - B_i}{P_i - I_i}$ where P_i, I_i are defined in Section III-D. Note that resource agent works on a higher frequency than power scheduler so that B_i is fixed within a power scheduler cycle but P_i may change over time. As a result, we need to calculate R_i every Resource agent control cycle. Intuitively, the number of cores to unbind is $R_i \cdot N_c$ where N_c is the number of running cores. However, the number of cores does not always linear related to power consumption. They are just positive correlated. So we need to gradually control the number of cores binding to tasks to both increase control accuracy and reduce chattering. On the other hand, we also need to ensure that the resource agent can control the power within the given time. So we adopt a simple linear closed loop control system to unbind $R_i \cdot N_c / C$ cores, where C is convergence coefficient. We need to trade off between the impact of changing binding cores and convergence time by tuning C and the results are shown in SectionIV-B.

After we calculate the number of cores to operate, resource agent will choose task with lower priority to unbind. In addition, R_i here is different from power scheduler that is not always positive because server power may be reduced below power budget. Negative R_i means release unbinded cores or binding more cores to tasks and the order to choose task is reversed.

The resource agent may stop partial of low priority tasks depends on the power budget required but it will never stop all the task on a server as we have a limit on expected power reduce ratio introduced in Section III-D.

IV. EVALUATION

A. Experimental setup

1) Cluster setup and workloads: The experiment platform is several racks which contain more than 200 homogeneous servers(with 56 cores) in a real over-provisioning production cluster. By real over-provisioning, we mean that the measured maximum power can be higher than the designed power budget. The over-provisioning ratio is 12.7% (this is measured by turning off DVFS and providing extra power source) and the rack is protected by DVFS and UPS. Given the hardware configuration, our goal is to utilize the existing hardware to improve QoS and significantly reduce power violation.

All servers in these racks are part of a datacenter-wide resource pool that is managed by a single job scheduler. Resource agent and power monitor are deployed on every

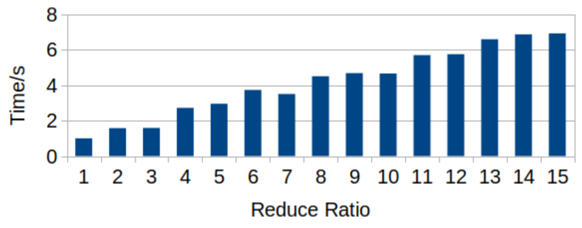


Fig. 2. Average time for resource agent to reduce power under various workloads and given different reduce ratio.

Workload	Light		Heavy	
Group	Exp	Ctr	Exp	Ctr
R_{mean}	0.01%	0%	1.7%	0%
R_{max}	6.31%	0%	7.12%	0%
P_{mean}	0.921	0.921	0.972	0.975
P_{max}	0.992	1.036	0.998	1.048
$Violations$	0	21	0	759

TABLE II
CONTROLLER EFFECTIVENESS UNDER LIGHT / HEAVY WORKLOAD. THE EXPERIMENT RUNS FOR 24 HOURS AND THE MEASUREMENTS ARE TAKEN EVERY 10S. R_{mean} AND R_{max} ARE THE MEAN/MAX IMPACT RATIO. P_{mean} AND P_{max} ARE THE MEAN/MAX POWER DRAW. $Violations$ IS THE TOTAL NUMBER OF POWER VIOLATIONS.

server and a central power controller is responsible for manage all racks with independent decisions.

Three workloads involved in our evaluation: web service, MPI service, and Map-reduce. The first one is online service and the last one is offline service. MPI service can be either online or offline service depends on individual task but MPI task requires more on stability. Thus it always have a higher priority than Map-reduce. And Web service always have a higher priority than MPI service. These are all representative workloads in our data center.

2) Key performance metrics: First key performance metric is the number of power violations. Safety is the first priority when considering over-provisioning. Users may choose to allow a few power violations, and small violation number shows the effectiveness of Pelican; Since our monitoring cycle is 10 seconds, the total time of violations is estimated as the production of violation number and monitoring cycle. Second is QoS. For online services, we focus on latency. And for offline services, we measure throughput. Our goal is to improve the throughput of offline workloads while not affecting the latency of online services. Third is the impact ratio (R). Besides the QoS, we hope to reduce the impact of operations on the number of servers. Obviously, a smaller average impact ratio is better;

B. The effectiveness of resource agent

Before we evaluate Pelican, We need to verify if the resource agent can work as expected given power budget from API server. To determine the convergence coefficient C described in Section III-F, let's first consider an extreme situation that all 56 cores are running but only one core with the highest priority consumes 100% of power and the power budget is 0% of dynamic power. Since scheduler cycle is 10s and resource agent cycle is 1s, resource agent has 10 times to control the resource and reduce power. In this case, C should be equal or less than 5.6 in order to unbind the last target core. Then, let's consider the other extreme situation that the server is running at maximum power and all cores consume the same amount of power. In this case, since the maximum allowed power reduce ratio in our experiment is set to over-provisioning ratio, 12.7%, we need to unbind 12.7% of 56 cores, which is 7.11 cores. Even we unbind 1 core every second, we are able to unbind 8 cores within 10 seconds. So we tested the resource agent with various workload when C is 5 6. We found that the effectiveness of the resource agent is not very sensitive to C so we set C to 5.6. Figure 2 shows the average time for resource agent to reduce power under various workloads and given different reduce ratio. We can see that the resource agent can always control the power under target value within a scheduling cycle(< 10s).

C. The effectiveness of Pelican's control

To evaluate the effectiveness of our system, we first mirror production MPI and MapReduce workloads to two groups of racks: experiment and control groups (with Pelican turned off). We use the scale-down method in [8] to observe power violation while keeping the physical devices safe. Also, we turn DVFS off so that our results can reflect real power change. We conduct our experiment for 24 hours using two extreme types of workloads in our cluster: heavy and light.

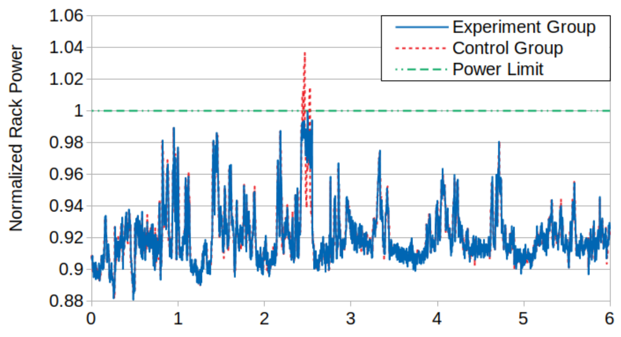
Table II shows a few performance metrics of Pelican. Under the heavy workload, there are 759 power violations in the control group without any power control, while there is no violation using Pelican's control. This proved the effectiveness of Pelican's power control ability.

The vibration of our workload is very high especially under heavy workloads. if we plot 24-hour power change, we can not see any detail so we plot 6 hours power change for both situations to show typical rack power utilization.

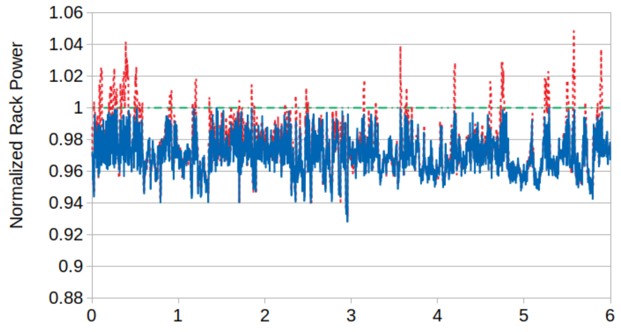
Figure 3(a) shows the light workload situation where the power draws mostly under the power limit. Even in this situation, we can see sharp power increases from time to time and several power violations, but overall, very few control actions are triggered. We can see a very high power increase happened at about 2.5h. In contrast, the heavy workload case Pelican control triggered quite often because the average power draw is very close to the power limit as shown in Figure 3(b). Besides the effectiveness test, one of our main goals is to figure out the appropriate rack threshold that can ensure safety. And then we apply the threshold to the same test on the production environment without scaling down for 220 hours with no power violence.

D. DVFS approach comparison

As we mentioned in Chapter II, DVFS is another widely used way to control power resource for tasks but it can cause overall performance disturbance, which is not suitable for our situation. In this section, we compare the QoS and demonstrate



(a)



(b)

Fig. 3. Power utilization under light (a) and heavy (b) workload. Pelican is deployed on Experiment group. And the control group is the base line for comparison.

our advantage under same over-provisioning ratio. The DVFS approach limit the power of each server to provisioned power when the overall power of a rack is close to the power limit. It is also threshold based approach and we choose the lowest threshold that can achieve same safety requirements as Pelican can.

We deploy Web service along with MPI and MapReduce on two racks with the same over-provision ratio configuration running the same workload trace. As we mentioned in workload description, Web service tasks are always on online services and MapReduce tasks are offline workloads but MPI can be both depending on the applications. We compare QoS of such cluster under DVFS and under Pelican respectively, i.e. compare latency on different levels for online tasks and throughput distribution for offline tasks.

Figure 4 shows the result of the median, 99.5th percentile and 99.9th percentile latency. Note that the y-axis is logarithmic. DVFS reduces the performance of online tasks almost doubling in terms of 99.5th percentile and 99.9th percentile latency. And DFVS also increase the median latency by about half compared with our system.

Figure 5 shows the results for throughput. We can see that the distribution of resulted throughput is similar. This is because DVFS can response faster but our method may provide more cores after power budget control. However, we still improve 1.43% overall throughput compared to DVFS method while achieving much better latency for online work-

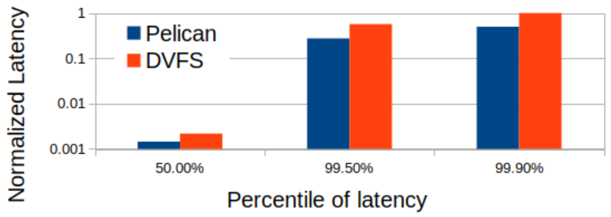


Fig. 4. Latency comparison using either DVFS or Pelican as power controller. Latency normalized to the Latency throughput in both case.

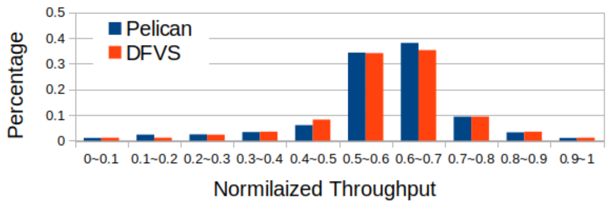


Fig. 5. Throughput distribution using either DVFS or Pelican as power controller. Throughput normalized to the maximum throughput in both case.

loads. This is mainly because we limit power usage on low priority and longer running offline workloads and not affecting online services while DVFS slows down overall performance of server causing longer task queue that significantly increase the latency of online tasks. This also proves that our method successfully schedules power on the temporal dimension.

E. Tuning for QoS

Our goal in this work is to improve overall QoS by utilizing provisioning more server given a fixed power budget. Specifically, we would like to keep safety first, then ensure that the latency of online services is not affected and last improve throughput for offline services.

We have performed a heavy workload evaluation and production test to determine rack threshold P_t mainly for safety. Here we present our evaluation on safe rack power P_s which determines when to stop resource agent control. Low P_s can ensure that it is safe to release more computing resources but it may also cause low performance. On the other hand, high P_s results in more computing power to be used as soon as possible but may also lead to surge power increase so that online services are also involved in resource control.

Thus, we run different P_s from 0.80 to 0.97(the safe threshold is 0.98) under varying production workload. The workload input is from production input that we can not control so we show different workload in different cases. Table III shows the representative results. Bold rows represent results under typical workloads. G_t is the throughput gain. R_{mean} is the average impact ratio. P_{mean} and P_{max} are the mean and maximum power. P_{max} can exceed, for example in #12, because it is the power of the control group.

First, we notice that the throughput gain G_t is positive related to P_s, especially in typical cases #4, #6, #9, and

#	P_s	P_{mean}	P_{max}	R_{mean}	$Latency$	G_t
1	0.8	0.936	1.031	0.0069%	1.0	9.7%
2		0.901	1.062	0.032%	1.0	9.30%
3		**0.925**	**1.011**	**0.067%**	**1.0**	**9.7%**
4	0.90	0.930	0.994	0%	1.0	10.50%
5		0.864	0.998	0%	1.0	10.10%
6		**0.923**	**1.006**	**0.06%**	**1.0**	**10.31%**
7	0.95	0.939	1.018	0.07%	1.0	11.2%
8		0.894	1.003	0.011%	0.995	13.5%
9		**0.929**	**1.013**	**0.074%**	**1.0**	**11.4%**
10	0.96	0.930	1.049	0.81%	1.0	11.6%
11		0.868	0.926	0%	1.0	10.8%
12		**0.925**	**1.021**	**0.078%**	**0.973**	**11.90%**

TABLE III
QOS UNDER DIFFERENT SAFE RACK POWER P_s AND WORKLOAD CONDITION. BOLD ROWS REPRESENT RESULTS UNDER TYPICAL WORKLOADS. G_t IS THE THROUGHPUT GAIN. P_{mean} AND P_{max} ARE THE MEAN AND MAXIMUM POWER FOR THE CONTROL GROUP, WHICH ARE GOOD INDICATORS OF THE POWER DEMAND. R_{mean} IS THE AVERAGE IMPACT RATIO.

#12. But it is also related to high power demand. For example, in #11, the overall power demand is low, which cause lower throughput gain. Then we found that R_{mean} is sensitive to P_s when P_{mean} is high, shown in #1, #4, #7, and #10. Also, the latency is always not affected when P_s is low.

Overall, we find that 0.95 is an effective choice. The latency in the typical case is not affected and the throughput increases by 11.4%. The increment is higher than other choices except when P_s is 0.96. But the latency is reduced in the typical case when P_s is 0.96. As a result, we choose 0.95 for P_s considering safety and QoS.

V. Related Work

The major risk of using over-provisioning is power outage so power scheduling becomes the core part of using it. It can be classified into two directions: scheduling on the spatial dimension like power-aware workload scheduling and scheduling on the temporal dimension like task resource management [9], [10].

Ideal power scheduling on temporal dimension reduces power draw of a task when the power budget is not enough and assigning more resources after there are available power resources in order do not affect the performance of the task. Sharma *et al.* proposed a real-time power management protocol in the Linux kernel mainly designed for web service [11]. Lo *et al.* proposed a feedback system that learns from request latency statistic and adjusts hardware limits to provide just fast enough server power for data center [12]. Luo *et al.* designed a general platform for back-end workloads to achieve energy proportionality by finding the best hardware configuration for given workload and service properties [13]. Zheng *et al.* explored the combination of power capping using use four kinds of DVFS algorithm and power shaving by UPS batteries [14]. On the other hand, utilizing other configurable hardware or software resource can be found in recent works. Sun *et al.* proposed performance-equivalent resource configuration(PERC) to reduce power usage while keeping the same performance [15]. Kontorinis *et al.* introduced an architecture that equips each server with a UPS so that the server can discharge the battery hen power budget is low and charge the battery otherwise [16].

Wrong task placement is one of the reasons why the power budget of a subsystem is not enough while global power budget is still enough. Scheduling task for energy or power has been widely studied. One of the simple ideas is place tasks together and turn off unused servers, which is called server consolidation. By reducing the idle power, researches tried to improve the overall energy efficiency of a data center [17]–[26]. PowerNap [27] and Anagnostopoulou *et al.* [28] leveraged hardware power states to change the power usage of a server according to the current workload. One key problem to do so is the long transition time [29] so SLAs or QoS is usually taken into consideration.

The core reason for wrong task placement is usually considered as the fault of Job scheduler. Nevertheless, many researchers tried to integrate power management into the job scheduler or combine power scheduler with QoS-aware cluster management systems [30]–[32]. Yao *et al.* proposed a framework called TS-BatPro to rearrange batching jobs to save energy for multi-core servers in data center [33]. It studies the performance and power characteristics of a server and schedules global batching jobs based on the model. Tesfatsion *et al.* introduced a dynamic resource management and scheduling system to improve energy efficiency for cloud services [34]. Cheng *et al.* proposed heterogeneity-oblivious task assignment method, E-Ant, which can improve overall energy efficiency without hurting the performance of a heterogeneous Hadoop cluster [35]. Petrucci *et al.* proposed a QoS-aware task management solution that learns from existing tasks map efficient device to meet the QoS requirements [2]. These systems work well on its own but it is hard to integrate into real data center job scheduler because there have been tremendous factors for it to decide where to place task.

VI. Conclusion

In this work, we investigate opportunities and challenges of improving QoS for large-scale data center with heterogeneous workloads by over-provisioning. While power controlling is still the key to this problem, heterogeneous workloads requires faster response time and the ability to deal with the temporal power management. We design and implement Pelican in our real over-provisioned production cluster then empirically demonstrate the feasibility of scheduling power budget within a rack without affecting task placement to utilize computing resources but prevent power outage. We adopt a two-level design that separate overall power scheduling and local power controlling for each server, which allows resource agent to maximize its ability to access local task information and react to large variation quickly while cooperate with other server with simple interface provided by power scheduler. This makes our power scheduling system more flexible and efficient for heterogeneous workloads.

Acknowledgment

We would like to thank people at Baidu Inc. Zehui Song, Sizhe Lv for their help in Pelican's implementation. Yiming Luo, Jianyong Hao, Ming He for their help in system testing. This work is in part supported by National Science Foundation (NSF) grant CNS-1561216.

References

[1] S. Ren, "Managing power capacity as a first-class resource in multitenant data centers," *IEEE Internet Computing*, vol. 21, no. 4, pp. 8–14, 2017.

[2] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Moss, J. Mars, and L. Tang, "Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 246–258.

[3] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 319–330. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000103

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

[5] 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*. IEEE, 2012, pp. 488–499.

[6] 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)*, vol. 47, no. 2, p. 33, 2014.

[7] C.-H. Hsu, Q. Deng, J. Mars, and L. Tang, "Smoothoperator: Reducing power fragmentation and improving power utilization in large-scale datacenters," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: ACM, 2018, pp. 535–548. [Online]. Available: http://doi.acm.org/10.1145/3173162.3173190

[8] G. Wang, S. Wang, B. Luo, W. Shi, Y. Zhu, W. Yang, D. Hu, L. Huang, X. Jin, and W. Xu, "Increasing large-scale data center capacity by statistical power control," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: ACM, 2016, pp. 8:1–8:15. [Online]. Available: http://doi.acm.org/10.1145/2901318.2901338

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

[10] 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)*, vol. 46, no. 4, p. 47, 2014.

[11] 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*. IEEE, 2003, pp. 63–72.

[12] 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*. IEEE Press, 2014, pp. 301–312.

[13] B. Luo, S. Wang, W. Shi, and Y. He, "ecope: Workload-aware elastic customization for power efficiency of high-end servers," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 237–249, April 2016.

[14] W. Zheng, K. Ma, and X. Wang, "Hybrid energy storage with supercapacitor for cost-efficient data center power shaving and capping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1105–1118, April 2017.

[15] F. Sun, H. Li, Y. Han, G. Yan, and J. Ma, "Powercap: Leverage performance-equivalent resource configurations for power capping," in *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, Nov 2016, pp. 1–8.

[16] 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 *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 488–499.

[17] 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*. IEEE, 2006, pp. 418–428.

[18] 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*. ACM, 2009, pp. 29–38.

[19] 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*, vol. 44, no. 1, pp. 76–80, 2010.

[20] 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*. ACM, 2010, pp. 1–6.

[21] 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*, vol. 180. Barcelona, Spain, 2001, pp. 182–195.

[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*, vol. 41, no. 3. ACM, 2013, pp. 96–107.

[23] 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*. ACM, 2012, pp. 211–224.

[24] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.

[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*, vol. 12, no. 1, pp. 1–15, 2009.

[26] 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*, vol. 10. San Diego, California, 2008, pp. 1–5.

[27] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating server idle power," in *ACM Sigplan Notices*, vol. 44, no. 3. ACM, 2009, pp. 205–216.

[28] 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)*, vol. 8, no. 4, p. 31, 2012.

[29] 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*. IEEE, 2011, pp. 319–330.

[30] 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*, vol. 35, no. 5. ACM, 2001, pp. 103–116.

[31] 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*. USENIX Association, 2009, pp. 28–28.

[32] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 77–88, 2013.

[33] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "Ts-batpro: Improving energy efficiency in data centers by leveraging temporal-spatial batching," *IEEE Transactions on Green Communications and Networking*, pp. 1–1, 2018.

[34] S. K. Tesfatsion, E. Wadbro, and J. Tordsson, "Perfgreen: Performance and energy aware resource provisioning for heterogeneous clouds," in *2018 IEEE International Conference on Autonomic Computing (ICAC)*, Sep. 2018, pp. 81–90.

[35] D. Cheng, X. Zhou, P. Lama, M. Ji, and C. Jiang, "Energy efficiency aware task assignment with dvfs in heterogeneous hadoop clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 70–82, Jan 2018.