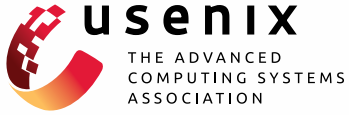


CloudPowerCap: Integrating Power Budget and Resource Management across a Virtualized Server Cluster

Yong Fu, *Washington University in St. Louis;* Anne Holler, *VMware;*
Chenyang Lu, *Washington University in St. Louis*

https://www.usenix.org/conference/icac14/technical-sessions/presentation/fu

This paper is included in the Proceedings of the
11th International Conference on Autonomic Computing (ICAC '14).

June 18–20, 2014 • Philadelphia, PA

ISBN 978-1-931971-11-9

Open access to the Proceedings of the
11th International Conference on
Autonomic Computing (ICAC '14)
is sponsored by USENIX.

CloudPowerCap: Integrating Power Budget and Resource Management across a Virtualized Server Cluster

Yong Fu
Washington University in St. Louis

Anne Holler
VMware

Chenyang Lu
Washington University in St. Louis

Abstract

In many data centers, server racks are highly underutilized due to maintaining the sum of the server nameplate power below the power provisioned to the rack. The root cause of this rack underutilization is that the server nameplate power is often much higher than can be reached in practice. Although statically setting per-host power caps can ensure the sum of the servers' maximum power draw does not exceed the rack's provisioned power, it burdens the data center operator with managing the rack power budget across the hosts. In this paper we present CloudPowerCap, a practical and scalable solution for power cap management in a virtualized cluster. CloudPowerCap, closely integrated with a cloud resource management system, dynamically adjusts the per-host power caps for hosts in the cluster to respect not only the rack power budget but also the resource management system's constraints and objectives. Evaluation based on an industrial cloud simulator demonstrates effectiveness and efficacy of CloudPowerCap.

1 Introduction

In many datacenters, server racks are as much as 40 percent underutilized [7]. Rack slots are intentionally left empty to keep the sum of the servers' nameplate power below the power provisioned to the rack, and the servers that are placed in the rack cannot make full use of the rack's provisioned power. The root cause of this rack underutilization is that a server's peak power consumption is in practice often significantly lower than its nameplate power [5]. This server rack underutilization can incur substantial costs. In hosting facilities charging a fixed price per rack, which includes a power charge that assumes the rack's provisioned power is fully consumed, paying a 40 percent overhead for rack underutilization is nontrivial. And in a private datacenter, the amortized capital costs for the infrastructure to deliver both the racks' provisioned power and the cooling capacity to handle the racks' fully populated state comprises 18 percent of a datacenter's total monthly costs [10]. If that infrastructure is 40 percent underutilized, then 7 percent of the data center's monthly costs are wasted for this reason.

Due to the significant cost of rack underutilization, major server vendors are now shipping support for per-host power caps, which provide a hardware or firmware-enforced limit on the amount of power that the server can draw [12, 4, 13]. These caps work by changing processor power states [11] or by using processor clock throttling, which is effective since the processor is the largest consumer of power in a server and its activity is highly correlated with the server's dynamic power consumption [5, 12]. Using per-host power caps, data center operators can set the caps on the servers in the rack to ensure that the sum of those caps does not exceed the rack's provisioned power. While this approach improves rack utilization, it burdens the operator with manually managing the rack power budget allocated to each host in a rack. In addition, it does not lend itself to flexible allocation of power to handle workload spikes or to respond to the addition or removal of a rack's powered-on server capacity.

Many data centers use their racked servers to run virtual machines (VMs). Several research projects have investigated power cap management for virtualized infrastructure [21, 18, 19, 16, 28, 3]. While this prior work has considered some aspects of VM Quality-of-Service (QoS) in allocating the power budget, it has not explored operating in a coordinated fashion with a comprehensive resource management system for virtualized infrastructure. Sophisticated cloud resource management systems such as VMware Distributed Resource Scheduler (DRS) support admission-controlled resource reservations, resource entitlements based fair-share scheduling, load-balancing to maintain resource headroom for demand bursts, and respect for constraints to handle user's business rules [9]. However the operations of cloud resource management systems aforementioned can be compromised if coordination is not carefully considered in design of the integrated power cap management system. For example, changing host power cap naively may affect resource to VMs, impacting end-users' Service-Level Agreements (SLAs), fairness, robustness and peak performance. Similarly, if the resource management system consolidates VMs and powers off unneeded hosts to save power, the host power cap setting system needs to be aware of that activity or it may cause the power budget to be inefficiently allocated to hosts, impacting the amount of powered-on computing capacity available for a given power budget.

This paper presents CloudPowerCap, an autonomic computing approach to power budget management in a virtualized environment. CloudPowerCap manages the power budget for a cluster of virtualized servers, dynamically adjusting the per-host power caps for servers in the cluster. It allocates the power budget in close coordination with a cloud resource management system, operating in a manner consistent with the system's resource management constraints and goal of ensuring VMs receive

the resources to which they are entitled. To facilitate interoperability between power cap and resource management, CloudPowerCap maps a servers power cap to its CPU capacity and coordinates with the resource management system through well defined interfaces and protocols. The integration of power cap and resource management results in the following novel capabilities in cloud management.

- **Constraint satisfaction via power cap reallocation:** Dynamic power cap reallocation enhances the system's capability to satisfy VM constraints, including resource reservations and business rules.
- **Power-cap-based entitlement balancing:** Power cap redistribution provides an efficient mechanism to achieve entitlement balancing among servers. Power-cap-based entitlement balancing can reduce or eliminate *real* migrations of VMs and associated overhead.
- **Power cap redistribution for power management:** CloudPowerCap can redistribute power caps among servers to handle server power-off/on state changes caused by dynamic power management. Power cap redistribution reallocates the power budget freed up by powered-off hosts, while reclaiming budget to power-on those hosts when needed.

We have implemented and integrated CloudPowerCap with VMware Distributed Resource Scheduler (DRS). Evaluation based on an industrial cloud simulator demonstrated the efficacy of integrated power budget and resource management in virtualized server clusters.

2 Motivation

In this section, we motivate the problem CloudPowerCap is intended to solve. We first describe the power model mapping a host's power cap to its CPU capacity, which enables CloudPowerCap to integrate power cap management with resource management in a coordinated fashion. We next discuss some trade-offs in managing a rack power budget. After a brief introduction of the resource management model, we then provide several examples of the value of combining dynamic rack power budget management with a cloud resource management system.

2.1 CloudPowerCap Power Model

The power model adopted by CloudPowerCap maps the power cap of the host to the CPU capacity of the host, which is in turn managed by a resource management system directly. A host's power consumption $P_{consumed}$ is commonly estimated by its CPU utilization U and the idle P_{idle} and peak P_{peak} power consumption of the host via a linear function, which is validated by real-world workloads in previous measurements and analysis [17, 5],

$$P_{consumed} = P_{idle} + (P_{peak} - P_{idle})U. \quad (1)$$

The power P_{idle} represents the power consumption of the host when the CPU is idle. P_{idle} intentionally includes the power consumption of the non-CPU components, such as networking and memory, whose power draw currently does not vary significantly with utilization. We note that in enterprise virtualized data centers, either the local disks are kept busy enough not to spin down or (more typically) shared storage is employed. The power P_{peak} represents the power consumption of the host when the CPU is 100% utilized at its maximum CPU capacity C_{peak}, with the CPU utilization U expressed as a fraction of the maximum capacity.

Equation (1) is an upper-bound estimation of $P_{consumed}$ if a host power management technology such as dynamic voltage and frequency scaling (DVFS) is used since DVFS can deliver amount of CPU capacity at a lower power consumption. For example, DVFS could deliver the equivalent of 50 percent utilization of a 2 GHz processor at lower power consumption by running the processor at 1 GHz with 100 percent utilization. Computing $P_{consumed}$ as an upper bound is desirable for the resource management use case, to ensure sufficient power budget for worst case.

Considering upper-bound power estimation from Equation (1), for a host power cap P_{cap} set below P_{peak}, we could solve for the *lower-bound* of the CPU capacity C_{capped} corresponding to P_{cap}, i.e., the host's effective CPU capacity limit which we refer to as its *power-capped capacity*. In this case, we rewrite Equation (1) as:

$$P_{cap} = P_{idle} + (P_{peak} - P_{idle})(C_{capped}/C_{peak}). \quad (2)$$

and then solve for C_{capped} as:

$$C_{capped} = C_{peak}(P_{cap} - P_{idle})/(P_{peak} - P_{idle}). \quad (3)$$

2.2 Managing a Rack Power Budget

To illustrate some trade-offs in managing a rack power budget, we consider the case of a rack with a budget of 8 KWatt, to be populated by a set of servers. Each server has 34.8 GHz CPU capacity comprising 12 CPUs, each running at 2.9 GHz, along with the other parameters shown in Table 1.

CPU	Memory	Nameplate	Peak	Idle
34.8 GHz	96 GB	400 W	320 W	160 W

Table 1: The configuration of the server in the rack.

Given the power model presented in the previous section and the servers in Table 1, the rack's 8 KWatt power budget can accommodate various deployments including those shown in Table 2. Based on the 400 Watts nameplate power, only 20 servers can be placed in the rack. Instead setting each server's power cap to its peak attainable power draw of 320 Watts allows 25 percent

more servers to be placed in the rack. This choice maximizes the amount of CPU capacity available for the rack power budget, since it best amortizes the overhead of the servers' powered-on idle power consumption. However, if memory may sometimes become the more constrained resource, setting each server's power cap at 250 Watts to allow 32 hosts to be placed in the rack significantly increases the memory available for the given power budget. By dynamically managing the host power caps of the servers in Table 1, trade-offs between CPU and memory capacity are illustrated in Table 2.

Power Cap(W)	Servers	CPU		Memory	
		Total(GHz)	Ratio	Total(GB)	Ratio
400	20	696	1.00	1920	1.00
320	25	870	1.25	2400	1.25
285	28	761	1.09	2688	1.40
250	32	626	0.90	3072	1.60

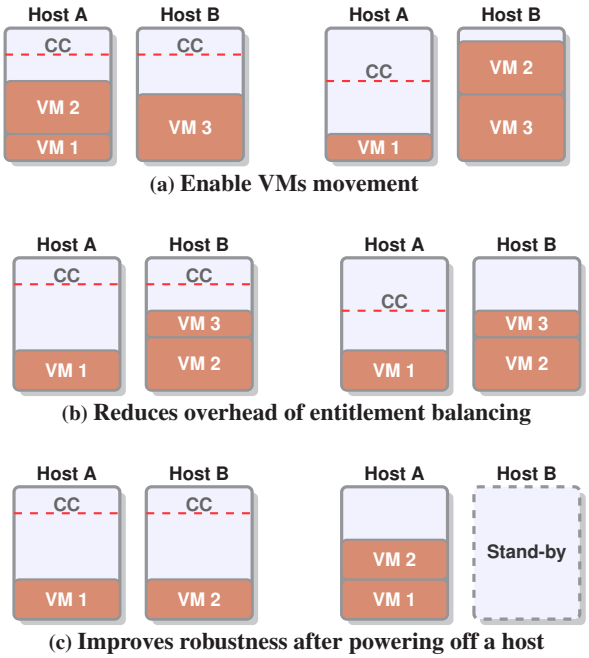
Table 2: Trade-offs between CPU and memory with different power caps

2.3 Powercap Distribution Examples

The cloud resource management system with which CloudPowerCap is designed to interoperate computes each VM's entitled resources and handles the ongoing location of VMs on hosts so that the VMs' entitlements can be delivered while respecting constraints, providing fair resource allocation by entitlement balancing, and optionally reducing power consumption.

In this section we use several scenarios to illustrate how CloudPowerCap can redistribute host power caps to support cloud resource management, including enabling VM migration to correct constraint violations, providing spare resource headroom for robustness in handling bursts, and avoiding migrations during entitlement balancing. In these scenarios, we assume a simple example of a cluster with two hosts. Each host has an uncapped capacity of 2x3GHz (two CPUs, each with a 3GHz capacity) with a corresponding peak power consumption of 600W (values chosen for ease of presentation).

Enforcing constraints: Host power caps should be redistributed when VMs are placed initially or relocated, if necessary to allow constraints to be respected or constraint violations to be corrected. For example, a cloud resource management system would move VM(s) from a host violating affinity constraints to a target host with sufficient capacity. However, in the case of static power cap management, this VM movement may not be feasible because of a mismatch between the VM reservations and the host capacity. As shown in Figure 1a, host A and B have the same power cap of 480 W, which corresponds to a power-capped capacity of 4.8 GHz. Host A runs two VMs, VM 1 with reservation 2.4 GHz and VM 2 with reservation 1.2 GHz. And host B runs only one 3 GHz reservation VM. When VM 2 needs to be colocated with VM 3 due to a new VM-VM affinity rule between the two VMs, no target host in the cluster has sufficient power-capped capacity to respect their combined reservations. However, if CloudPowerCap redistributes the power caps of host A and B as 3.6 GHz and 6 GHz respectively, then VM 2 can successfully be moved by the cloud resource management system to host B to resolve the rule violation in the cluster. Note that host A's capacity cannot be reduced below 3.6 GHz until VM 1's migration to host B is complete or else the reservations on host A would be violated.



(a) Enable VMs movement

(b) Reduces overhead of entitlement balancing

(c) Improves robustness after powering off a host

Figure 1: Power cap distribution scenarios. Left-hand figures correspond to hosts' status before distribution; right-hand figures show hosts' status after. Power-capped capacity is not shown when the power cap of the host equals its peak power. (CC: Power-capped capacity)

Enhancing robustness to demand bursts: Even when VM moves do not require changes in the host power caps, redistributing the power caps can still benefit the robustness of the hosts to handling VM demand bursts. For example, suppose as in the previous example that VM 1 needs to move from host A to host B because of a rule. In this case, a cloud resource management system can move VM 1 to host B while respecting the VMs' reservations. However, after the migration of VM 1, the *headroom* between the power capped capacity and VMs' reservations is only 0.6 GHz on host B, compared with 2.4 GHz on host A. Hence, host B can only accommodate as high as a

15% workload burst without hitting the power cap while host A can accommodate 100%, that is, host B is more likely to introduce a performance bottleneck than host A. To handle this imbalance of robustness between the two hosts, CloudPowerCap can redistribute the power caps of host A and B as 3.6 GHz and 6 GHz respectively. Now both hosts have essentially the same robustness in term of *headroom* to accommodate workload bursts.

Reduce overhead of VM migration: Before entitlement balancing, power caps should be redistributed to reduce the need for VM migrations. Load balancing of the resources to which the VMs on a host are entitled is a core component of cloud resource management since it can avoid performance bottlenecks and improve system-wide throughput. However, some migrations of VMs for load balancing are unnecessary. As shown in Fig 1b, the VM on Host A has an entitlement of 1.8 GHz while the VMs on host B have a total entitlement of 3.6 GHz. The difference in entitlements between host A and B are high enough to trigger entitlement balancing, in which VM 3 is moved from host B to host A. After entitlement balancing, host A and B have entitlements of 3 GHz and 2.4 GHz respectively, that is, the workloads of both hosts are more balanced. However, VM migration has an overhead related to copying the VM's CPU context and memory content between the hosts involved, inducing some latency [23]. In contrast, changing a host power cap only involves issuing a simple baseboard management system command which completes in less than one millisecond [12]. Hence, instead of entitlement balancing, CloudPowerCap can perform the cheaper action of redistributing the power caps of hosts A and B, increasing host B's power capped capacity to 6 GHz after decreasing host A's power capped capacity to 3.6 GHz, which also results in more balanced entitlements for host A and B. In general, the redistribution of power caps before entitlement balancing, called *powercap based entitlement balancing*, can reduce or eliminate the number of VM migrations for load balancing and the overhead of the migrations.

Adapting to host power on/off: Power caps should be redistributed when cloud resource management powers on/off host(s) to improve cluster efficiency. A cloud resource management system detects when there is ongoing under-utilization of cluster host resources leading to *power-inefficiency* due to the high host idle power consumption, and it consolidates workloads onto fewer hosts and powers the excess hosts off. In the example shown in Figure 1c, host B can be powered off after VM 2 is migrated to host A. However, after host B is powered-off, it does not consume power and hence does not need its power cap. Therefore the utilization of host A can be increased due to migrated VM 2, which impacts the capacity *headroom* of host A. Power cap redistribution after powering off host B can increase the power cap of host A to 6 GHz, allowing the *headroom* of host A to increase to 3 GHz and hence increase system robustness and reduce the likelihood of resource throttling. Similarly, powercap redistribution can improve robustness when resource management powers on hosts.

3 CloudPowerCap Design

CloudPowerCap is designed to provide power budget management to existing resource management systems, in such a way as to support and reinforce such systems' design and operation. Such resource management systems are designed to satisfy VMs' resource entitlements subject to a set of constraints, while providing balanced headroom for demand increases and, optionally, reduced power consumption. CloudPowerCap improves the operation of resource management systems, via power cap allocation targeted to their operation.

Existing resource management systems typically involve nontrivial complexity. Fundamentally reimplementing them to handle hosts of varying capacity due to power caps would be difficult and the benefit of doing so is unclear, given the coarse-grained scales at which cloud resource management systems operate. In CloudPowerCap, we take the practical approach of introducing power budget management as a separate manager that coordinates with an existing resource management system such that the existing system works on hosts of *fixed* capacity, with specific points at which that capacity may be modified by CloudPowerCap in accordance with the existing system's operational phase. Our approach therefore enhances modularity by separating power cap and resource management, while coordinating them effectively through well defined interfaces and protocols, as described below. Due to practical and modular design, CloudPowerCap is the same scalablity of the clous resource management it integrated with.

Since the aim of CloudPowerCap is to enforce the cluster power budget while dynamically managing hosts' power caps by closely coordinating with the cloud resource management system, CloudPowerCap consists of three components, as shown in Figure 2, corresponding to the three major functions of the cloud resource management system. The three components, corresponding to main components in DRS, execute step by step and work on two-way interaction with components in DRS.

Powercap Allocation: During the powercap allocation phase, potential resource management constraint correction moves may require redistribution of host power caps. Because CloudPowerCap can redistribute the host power caps, the cloud resource management system is able to correct more constraint violations than would be possible with statically-set host power caps.

Powercap-based Entitlement Balancing: If the resource management system detects entitlement imbalance over the user-set threshold, powercap based entitlement balancing first tries to reduce the imbalance, by redistributing power caps without actually migrating VMs

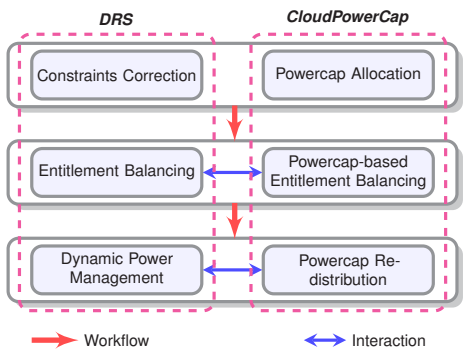


Figure 2: Structure and two-way interaction of CloudPowerCap working with DRS and DPM.

between hosts. This is valuable because redistributing power caps, which is less than 1 millisecond [12], is much faster than VM migration [26]. Powercap-based entitlement balancing may not be able to fully address imbalance due to inherent physical host capacity limits. If powercap balancing cannot reduce the imbalance below the imbalance threshold, the resource management entitlement balancing can address the remaining imbalance by actual VM migration.

Powercap Redistribution: If the resource management system powers on a host to match a change in workload demands or other requirements, CloudPowerCap performs a two-pass power cap redistribution. First it attempts to re-allocate sufficient power cap for that host to power-on. If that is successful and if the system selects the host in question after its power-on evaluation, then CloudPowerCap redistributes the cluster power cap across the updated hosts, to address any unfairness in the resulting power cap distribution. Powered-off host powercap redistribution improves the availability of power for the remaining powered-on hosts, allowing them to be more responsive to workload bursts. And when powered-on hosts can handle bursts without powering on additional hosts, this redistribution improves power efficiency. We note that the effect of off/on a host due to fault and recovery by high availability (HA) is similar to powering on/off hosts by a power management system. Hence, powercap redistribution can be adapted to work with HA.

4 CloudPowerCap Implementation

We implemented CloudPowerCap to work with the VMware Distributed Resource Scheduler (DRS) [24] along with its optional Distributed Power Management (DPM) [25] feature. CloudPowerCap could also complement some other distributed resource management systems for virtualization environments. In this section, we first present an overview of DRS and then detail the design of each CloudPowerCap component and its interaction with its corresponding DRS component. Due to space restriction, the implementation of Powercap redistribution is not presented in this paper. Please refer to the technical report for details [6].

4.1 DRS Overview

VMware DRS performs resource management for a cluster of ESX hypervisor hosts. By default, DRS is invoked every five minutes. At the beginning of each DRS invocation, DRS runs a phase to generate recommendations to correct any cluster constraint violations by migrating VMs between hosts. Examples of such corrections include evacuating hosts that the user has requested to enter maintenance or standby mode and ensuring VMs respect user-defined affinity and anti-affinity business rules. DRS next performs entitlement balancing. The load metric of each host in DRS is *normalized entitlement*, which is defined as the sum of the per-VM entitlements for each VM running on the host divided by the capacity of the host. DRS chooses to migrate the VM that reduces imbalance most and the move-selection step repeats until either the load imbalance is below a user-set threshold or the number of moves generated in the current pass hits a configurable limit based on an estimate of the number of moves that can be executed in five minutes. Finally DRS optionally runs dynamic power management, which opportunistically saves power by dynamically right-sizing cluster capacity to match recent workload demand, while respecting the cluster constraints and resource controls.

4.2 Powercap Allocation

Powercap Allocation redistributes power caps if needed to allow DRS to correct constraint violations. DRS's ability to correct constraint violations is impacted by host power caps, which can limit the available capacity on target hosts. However, as shown in Fig 1a, by increasing the host power cap, the DRS algorithm can be more effective in correcting constraint violations by redistributing the cluster's unreserved power budget.

CloudPowerCap and DRS work in coordination, as shown in Figure 3, to enhance the system's capability to correct constraints violations.

1) Powercap Allocation first calls *GetFlexiblePower* to get *flexiblePower*, which is a special clone of the current cluster snapshot in which host power cap of each host is set to its reserved power cap, i.e., the minimum power cap needed to support the capacity corresponding to the reservations of the VMs currently running on that host.

2) The *flexiblePower* is used as a parameter to call *ConstraintsCorrection* function in DRS, which recommends VM migrations to enforce constraints and update hosts' reserved power caps for the new VM placements after the recommended migrations.

3) As a result of performing *ConstraintsCorrection*, DRS generates VM migration actions to correct con-

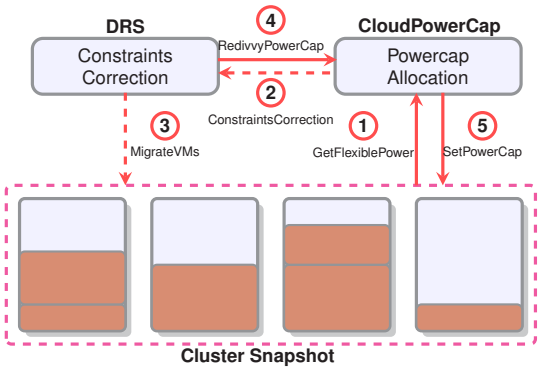


Figure 3: Coordination between CloudPowerCap and DRS to correct constraints. Solid arrows indicate invocations of CloudPowerCap functions while dashed arrows indicate invocations of DRS functions.

straints. Note that when applying VMs migration actions on hosts in the cluster, dependencies are respected between these actions and any prerequisite power cap setting actions generated by CloudPowerCap.

4) If some constraints are corrected by DRS, the power caps of source and target hosts may need to be reallocated to ensure fairness. For this case, *RedivvyPowerCap* of CloudPowerCap is called to redistribute the power cap.

5) Finally Powercap Allocation generates actions to set the power cap of hosts in the cluster according to the results of *RedivvyPowerCap*.

The key function in Powercap Allocation is *RedivvyPowerCap*, in which the unreserved power budget is redistributed after the operations for constraint violation correction. The objective of RedivvyPowerCap is to distribute the cluster power budget according to *proportional resource sharing* [27] for maintaining fairness of unreserved power budget distribution across hosts after the constraint correction. RedivvyPowerCap may introduce new opportunity to move VMs to correct constraints after redistributing the unreserved power budget. However, for simplicity, Powercap Allocation stops after invoking RedivvyPowerCap only once.

4.3 Entitlement Balancing

Entitlement balancing is critical for systems managing distributed resources, to deliver resource entitlements and improve the responsiveness to bursts in resource demand. For resource management systems like DRS without the concept of dynamic host capacity, entitlement balancing achieves both of these goals by reducing imbalance via migrating VMs between hosts. However, with dynamic power cap management, CloudPowerCap can alleviate imbalance by increasing the power caps of heavy loaded hosts while reducing the power caps of lightly loaded hosts rather than actually migrating VMs between those hosts as shown in Figure 1b. Considering the almost negligible overhead of power cap reconfiguration comparing to VM migration, Powercap-based Entitlement Balancing is preferred to DRS entitlement balancing when the cluster is imbalanced. However, because power cap adjustment has a limited range of operation, Powercap-based Entitlement Balancing may not fully eliminate imbalance in the cluster.

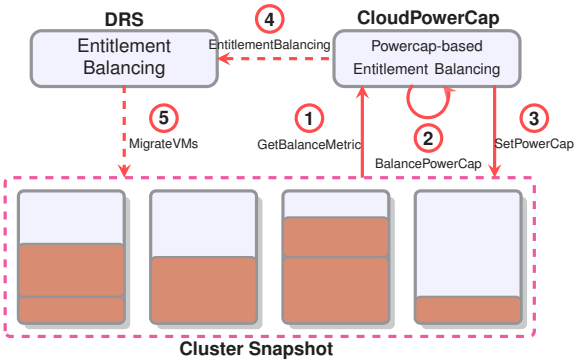


Figure 4: Work flow of Powercap-based Entitlement Balancing and its interaction with DRS entitlement balancing. Solid arrows indicate invocations of CloudPowerCap functions while dashed arrows indicate invocations of DRS functions.

The process of powercap based entitlement balancing and its interaction with DRS load balancing are shown in Figure 4.

1) To acquire the status of entitlement imbalance of the cluster, Powercap-based Entitlement Balancing first calculates the DRS imbalance metric for the cluster.

2) Then Powercap-based Entitlement Balancing tries to reduce the entitlement imbalance among hosts by adjusting their power caps in accordance with their normalized entitlements.

3) If Powercap-based Entitlement Balancing is able to impact cluster imbalance, its host power cap redistribution actions are added to the recommendation list, with the host power cap reduction actions being prerequisites of the increase actions.

4) If Powercap-based Entitlement Balancing has not fully balanced the entitlement among the hosts, DRS entitlement balancing is invoked on the results of Powercap-based Entitlement Balancing to reduce entitlement imbalance further.

5) DRS may generate actions to migrate VMs.

The key function *BalancePowerCap* was developed along the lines of *progressive filling* to achieve max-min fairness [2]. The function progressively increases the host power cap of the host(s) with highest normalized entitlement while progressively reducing the host power cap of the host(s) with lowest normalized entitlement . This

process is repeated until either the DRS imbalance metric crosses the balance threshold or any of the host(s) with highest normalized entitlement reach their peak capacity and hence further reduction in overall imbalance is limited by those hosts.

4.4 Powercap Redistribution

Powercap Redistribution responds to DPM dynamically powering on/off hosts. When CPU or memory utilization becomes high, DPM powers on hosts and Powercap Redistribution ensures that sufficient power cap is assigned to the powering-on host. On the other hand, when both CPU and memory utilization are low for a sustained period, DPM may consolidates VMs onto fewer hosts and powers off the remaining hosts to save energy. In this case, Powercap Redistribution distributes the power caps of the powered-off hosts among the active hosts to increase their capacity.

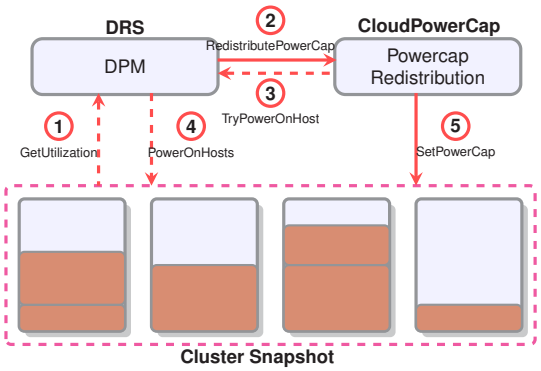


Figure 5: Coordination between CloudPowerCap and DRS and DPM in response to power on/off hosts. Solid arrows indicate to invoke CloudPowerCap functions while dashed arrows indicate to invoke DRS functions.

The coordination between Powercap Redistribution and DPM when DPM attempts to power on a host is depicted in Figure 5.

1) If there is sufficient unreserved cluster power budget to set the target host's power cap to peak, the host obtains its peak host power cap from the unreserved cluster power budget and no power cap redistribution is needed.
2) If the current unreserved cluster power budget is not sufficient, *RedistributePowerCap* is invoked to allow the powering-on candidate host to acquire more power from those hosts with lower CPU utilization.
3) DPM decides whether to power on the candidate host given its updated power cap after redistribution and its ability to reduce host high utilization in the cluster.
4) If the host is chosen for power-on, the normal DPM function is invoked to generate the action plan for powering on the host.
5) If DPM decides to recommend the candidate power-on, any needed host power cap changes are recommended as prerequisites to the host power-on.

4.5 Implementation Details

We implemented CloudPowerCap on top of VMware's production version of DRS. Like DRS, CloudPowerCap is written in C++. The entire implementation of CloudPowerCap comprises less than 500 lines of C++ code, which demonstrates the advantage of instantiating power budget management as a separate module that coordinates with an existing resource manager through well-defined interfaces.

As described previously in this section, DRS operates on a snapshot of the VM and host inventory it is managing. The main change we made for DRS to interface with CloudPowerCap was to enhance the DRS method for determining a host's CPU capacity to reflect the host's current power cap setting in the snapshot. Other small changes were made to support the CloudPowerCap functionality, including specifying the power budget, introducing a new action that DRS could issue for changing a host's power cap, and providing support for testability.

During CloudPowerCap initialization, for each host, the mapping between its current power cap and its effective capacity is established by the mechanisms described in Section 2.1. For a powered-on host, the power cap value should be in the range between the host's idle and peak power. When computing power-capped capacity of a host based on the power model (3), it is important to ensure that the capacity reserved by the hypervisor on the host is fully respected. Hence, the power-capped capacity $C_{mcapped}$ managed by the resource management system, i.e., managed capacity, is computed as:

$$C_{mcapped} = C_{capped} - C_H, \tag{4}$$

where the power-capped raw capacity C_{capped} is computed using Equation (1) and C_H is the capacity reserved by the hypervisor.

The implementation of Powercap Allocation entailed updating corresponding DRS methods to understand that a host's effective capacity available for constraint correction could be increased using the unreserved power budget, and adding a powercap redivvy step optionally run at the end of the constraint correction step. Powercap Balancing, which leverages elements of the powercap redivvying code, involved creating a new method to be called before the DRS balancing method. Powercap Redistribution changed DPM functions to consider whether to turn on/off hosts based not only on utilization but also on the available power budget.

5 Evaluation

In this paper we evaluate CloudPowerCap in the DRS simulator under three interesting scenarios. The first experiment evaluates CloudPowerCap's capability to rebalance normalized entitlement among hosts while avoiding the overhead of VM migration. The second experiment shows how CloudPowerCap allows CPU and memory capacity trade-offs to be made at runtime. This experiment includes a relatively large host inventory to show the capacity trade-offs at scale. The third experiment shows how CloudPowerCap reallocates the power budget of a powered-off host to allow hosts to handle demand bursts.

In these experiments, we compare CloudPowerCap against two baseline approaches of power cap management: *StaticHigh* and *Static*. Both approaches assign equal power cap to each host in the cluster at the beginning and maintain those power caps throughout the experiment. *StaticHigh* sets power cap of the host to its peak power, maximizing throughput of CPU intensive applications. However for applications in which memory or storage become constrained resources, it can be beneficial to support more servers to provision more memory and storage. Hence in *Static*, the power cap of a host is intentionally set lower than the peak power of the host. Compared with *StaticHigh*, more servers may be placed with *Static* to enhance the throughput of applications with memory or storage as constrained resources. However both approaches lack the capability of flexible power cap allocation to respond to workload spikes and demand variation.

5.1 DRS Simulator

The DRS simulator [8] is used in developing and testing all DRS algorithm features. It is a high-fidelity simulator and provides a realistic execution environment, while allowing much more flexibility and precision in specifying VM demand workloads and obtaining repeatable behavior close to the real hardware.

The DRS simulator simulates a cluster of hosts and VMs. A host can be defined using parameters including number of physical cores, CPU capacity per core, total memory size, and power consumption at idle and peak. A VM can be defined in terms of number of configured virtual CPUs (vCPUs) and memory size. Each VM's workload can be described by an arbitrary function over time, with the simulator generating CPU and memory demand for that VM based on the specification.

Given the input characteristics of hosts and the VMs' resource demands and specifications, the simulator mimics CPU and memory schedulers, allocating resources to the VMs in a manner consistent with the behavior of hosts in a real DRS cluster. The simulator calculates VMs' migration cost in accordance with several realistic factors, for example, VMs' read/write memory access and the available I/O and network bandwidth. The simulator also models the hypervisor CPU and memory overheads.

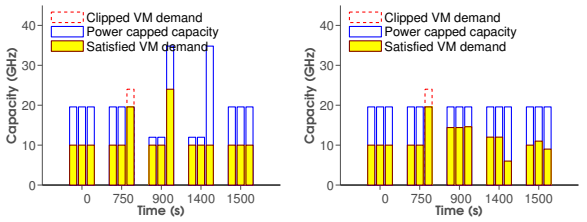
The simulator is able to estimate the power consumption of the hosts based on the power model given in Equation (1) in Section 2.2. For this work, the simulator was updated to respect the CPU capacity impact associated with a host's power cap.

5.2 Headroom Rebalancing

CloudPowerCap can reassign power caps to balance headroom for bursts, providing a quick response to workload imbalance due to VM demand changes. Such reassignment of power caps can improve robustness of the cluster and reduce or avoid the overhead of VM migration for load balancing. To evaluate impact of CloudPowerCap on headroom balancing, we perform an experiment in which 30 VMs, each with 1vCPU and 8GB memory, run on 3 hosts with the configuration shown in Table 1. Figures 6a and 6b plot the simulation results under CloudPowerCap and Static with a static power cap allocation of 250W per host, respectively. Initially, at time 0 seconds, the VMs are each executing similar workloads of 1 GHz CPU and 2 GB memory demand, and are evenly distributed across the hosts. At time 750 seconds, the VMs on one host spike to 2.4 GHz demand, thereby increasing the demand on that host above its power-capped capacity. When DRS is next invoked at time 900 seconds (running every 300 seconds by default), its goal is to rebalance the hosts' normalized entitlements. Under the static power cap, DRS migrates the VMs to balance the normalized entitlements. In contrast, CloudPowerCap reassigns the hosts' power caps to reduce the caps on the light-loaded hosts (to 215W) and increase them on the heavy-loaded host (to 320W). This addresses the host overutilization and imbalance without latency and overhead associated with VM migration, which is particularly important in this case, since the overhead further impacts the workloads running on the overutilized host. At time 1400 seconds, the 2.4 GHz VM demand spike ceases, and those VMs resume running at their original 1 GHz demand until the experiment ends at time 2100 seconds. Again, CloudPowerCap avoids the need for migrations by reassigning the host power caps to their original values. In contrast, Static performs two DRS entitlement balancing phases and migrates several VMs at time 900 seconds and 1500 seconds.

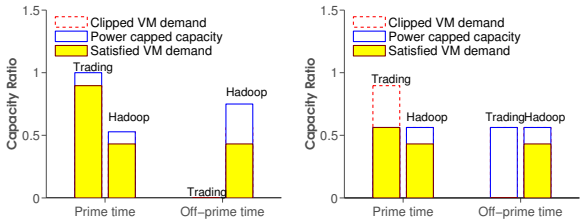
	CPU Payload Ratio	Migration
CPC	0.99	0
Static	0.89	7
StaticHigh	1.00	0

Table 3: CloudPowerCap rebalancing without migration overhead



(a) CloudPowerCap (b) Static

Figure 6: Headroom balancing on a group of 3 hosts. Hosts are grouped at each event time.



(a) CloudPowerCap (b) Static

Figure 7: Trade-offs between dynamic resource capacities. *Trading* indicates a group of servers running production trading VMs while *Hadoop* represents servers run production Hapdoop compute VMs.

Table 3 compares the CPU payload ratio, which is the ratio between actual CPU capacity and allocated CPU capacity to the VMs, under CloudPowerCap, Static using 250W static host power caps, as well as StaticHigh using the power caps equivalent to the peak capacity of the host. For Static, the CPU overhead caused by VMs migration has a significant overall impact on the CPU payload delivered to the VMs because the cycles needed for VMs migration directly impact VMs' performance. For CloudPowerCap, there is a relatively small impact to performance after the burst and before DRS can run CloudPowerCap to reallocate the host power caps. The power cap setting can be executed by the host within 1 millisecond and introduces minor payload overhead.

5.3 Flexible Resource Capacity

CloudPowerCap supports flexible use of power to allow trade-offs between resource capacities to be made dynamically. To illustrate such a trade-off at scale, we consider a cluster of hosts as described in Section 2.1. We model the situation in which the cluster is used to run both production trading VMs and production hadoop compute VMs. The trading VMs are configured with 2 vCPUs and 8 GB and they are idle half the day (off-prime time), and they run heavy workloads of 2x2.6 GHz and 7 GB demand the other half of the day (prime time). They access high-performance shared storage and hence are constrained to run on hosts with access to that storage, which is only mounted on 8 hosts in the cluster. The hadoop compute VMs are configured with 2 vCPUs and 16 GB and each runs a steady workload of 2x1.25 GHz and 14 GB demand. They access local storage and hence are constrained to run on their current hosts and cannot be migrated. During prime time, the 8 servers running the trading VMs do not receive tasks for the hadoop VMs running on those servers; this is accomplished via an elastic scheduling response to the reduced available resources [29]. Figure 7 shows the simulation results of the cluster under CloudPowerCap and the Static configuration of power caps.

Table 4 compares the CPU and memory payload delivered for three scenarios, and shows the impact on the trading VMs. The staticHigh scenario involves deploying 25 servers with power caps of 320 W, which immediately and fully supports the trading VMs prime time demand but limits the overall available memory and local disks in the cluster associated with the 25 servers. The Static scenario instead involves deploying 32 servers with each host power cap statically set to 250 Watts. This scenario allows more memory and local disks to be accessed, increasing the overall CPU and memory payload delivered because more hadoop work can be accomplished, but limits the peak CPU capacity of each host, meaning that the trading VMs run at only 62 percent of their prime time demand. With CloudPowerCap, the benefits to the hadoop workload of the static scenario are retained, but the power caps of the hosts running the trading VMs can be dynamically increased, allowing those VMs' full prime time demand to be satisfied.

	CPU Ratio	Mem Ratio	Trading Ratio
CPC	1.24	1.28	1.00
Static	1.21	1.28	0.62
StaticHigh	1.00	1.00	1.00

Table 4: CloudPowerCap enabling flexible resource capacity. Trading ratio indicates the ratio that production trading VMs demands in prime time are satisfied.

5.4 Standby Host Power Reallocation

CloudPowerCap can reallocate standby hosts' power cap to increase the capacity of powered-on hosts and thereby their efficiency and ability to handle bursts. To demonstrate this, we consider the same initial setup in terms of hosts and VMs as in the previous experiment. In this case, all VMs are running a similar workload of 1.2 GHz and 2 GB memory demand. At time 750 seconds, each VM's demand reduces to 400 MHz, and when DRS is next invoked at time 900 seconds, DPM recommends that the VMs be consolidated onto two hosts and that another host is powered-off. After the host has been evacuated and

powered-off at time 1200 seconds, CloudPowerCap reassigns its power cap to 0 and reallocates the rack power budget to the two remaining hosts, setting their power caps to 320W each. At time 1400 seconds, there is an unexpected spike. In the case of statically-assigned power caps, the host that was powered-off is powered back on to handle the spike, but in the CloudPowerCap case, the additional CPU capacity available on the 2 remaining hosts given their 320 W power caps is sufficient to handle this spike and the powered-off host is not needed.

	CPU Payload Ratio	Migration	Power Ratio
CPC	1.00	10	1.00
Static	0.98	19	1.36
StaticHigh	1.00	10	1.00

Table 5: CloudPowerCap reallocating standby host power

Table 5 compares the CPU payload in cycles delivered to the VMs for CloudPowerCap, Static, and StaticHigh. In this case, a number of additional vMotions are needed for Static, but the overhead of these vMotions does not significantly impact the CPU payload, because there is plenty of headroom to accomodate this overhead. However, Static consumes much more power than the other 2 cases, since powering the additional host back on and repopulating it consumes significant power. In contrast, CloudPowerCap is able to match the power efficiency of the baseline, by being able to use peak capacity of the powered-on hosts.

6 Related Work

Several research projects have considered power cap management for virtualized infrastructure [21, 18, 16, 28, 19, 14]. Among them, the research most related to our work is [19], in which authors proposed VPM tokens, an abstraction of changeable weights, to support power budgeting in virtualized environment. Like our work, VPM tokens enables shifting *power budget slack* which corresponds to *headroom* in this paper, between hosts. However the power cap management system based on VPM tokens are independent of resource management systems and may generate conflicting actions without coordination mechanisms.

In contrast, interoperating with a cloud resource management system like DRS also allows CloudPowerCap to support interesting additional features: 1) CloudPowerCap accommodates consolidation of physical servers caused by dynamic power management while previous work assumed a fixed working server set, 2) CloudPowerCap is able to handle and facilitate VM migration caused by correcting constraints imposed on physical servers and VMs, 3) CloudPowerCap can also deal with and enhance power cap management in the presence of load balancing. In most previous work, only part of these features are provided.

The authors of [21] describe managing performance and power management goals at server, enclosure, and data center level and propose handling the power cap hierarchically across multiple levels. Optimization and feedback control algorithms are employed to coordinate the power management and performance indices for entire clusters. In [28], the authors build a framework to coordinate power and performance via Model Predictive Control through DVFS (Dynamic Voltage and Frequency Scaling). To provide power cap management through the VMs management layer, [18] proposed throttling VM CPU usage to respect the power cap. In their approach, feedback control is also used to enforce the power cap while maintaining system performance. Similarly, the authors in [16] also discussed data center level power cap management by throttling VM resource allocation. Like [21], they also adopted a hierarchical approach to coordinate power cap and performance goals. In [14], authors proposed a relatively accurate model to estimate power consumption of the VM based on not only CPU utilization but also memory and disk usage. However this work has no discussion of dynamical power budget provisioning across a virtualized cluster.

While all of these techniques attempt to manage both power and performance goals, their resource models for the performance goals are incomplete in various ways. For example, none of the techniques support guaranteed SLAs (reservations) and fair share scheduling (shares). Some build a feedback model needing application-level performance metrics acquired from cooperative clients, which is rare especially in public clouds [1].

Although power management in virtualized cluster is extensively studied previously [20, 22, 15], which focus on reducing power consumption while maintaining preformance and is different to the goal of CloudPowerCap.

7 Conclusion

Many modern data centers have underutilized racks. Server vendors have recently introduced support for per-host power caps, which provide a server-enforced limit on the amount of power that the server can draw, improving rack utilization. However, this approach is tedious and inflexible because it needs involvement of human operators and does not adapt in accordance with workload variation. This paper presents CloudPowerCap to manage a cluster power budget for a virtualized infrastructure. In coordination with resource management, CloudPowerCap provides holistic and adaptive power budget management framework to support service level agreements, fairness in spare power allocation, entitlement balancing and constraint enforcement.

References

[1] BEN-YEHUDA, O. A., BEN-YEHUDA, M., SCHUSTER, A., AND TSAFRIR, D. The resource-as-a-service (RaaS) cloud. In *HotCloud'12* (2012).

[2] BERTSEKAS, D., GALLAGER, R., AND HUMBLET, P. *Data networks*. Prentice-Hall, 1992.

[3] DAVIS, J., RIVOIRE, S., AND GOLDSZMIDT, M. Star-Cap: Cluster Power Management Using Software-Only Models. Tech. rep., MSR-TR-2012-107, Microsoft Research, 2012.

[4] DELL INC. Dell Energy Smart Management.

[5] FAN, X., WEBER, W.-D., AND BARROSO, L. A. Power provisioning for a warehouse-sized computer. *SIGARCH Comput. Archit. News 35*, 2 (June 2007), 13–23.

[6] FU, Y., HOLLER, A., AND LU, C. CloudPowerCap: Integrating Power Budget and Resource Management across a Virtualized Server Cluster. `http://arxiv.org`.

[7] GARTNER RESEARCH. Shrinking Data Centers: Your Next Data Center Will Be Smaller Than You Think.

[8] GULATI, A., HOLLER, A., JI, M., SHANMUGANATHAN, G., WALDSPURGER, C., AND ZHU, X. VMware Distributed Resource Management: Design, Implementation, and Lessons Learned. *VMware Technical Journal* (Mar 2012).

[9] GULATI, A., SHANMUGANATHAN, G., HOLLER, A., AND AHMAD, I. Cloud-scale resource management: challenges and techniques. In *HotCloud* (2011).

[10] HAMILTON, J. Overall data center costs. `http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx`.

[11] HEWLETT-PACKARD, INTEL, MICROSOFT, PHOENIX, AND TOSHIBA. Advanced Configuration and Power Interface Specification. Tech. rep., 2011.

[12] HP INC. HP Power Capping and HP Dynamic Power Capping for Proliant Servers.

[13] IBM INC. IBM Active Energy Manager.

[14] KANSAL, A., ZHAO, F., LIU, J., KOTHARI, N., AND BHATTACHARYA, A. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 39–50.

[15] KUSIC, D., KEPHART, J., HANSON, J., KANDASAMY, N., AND JIANG, G. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing 12*, 1 (2009), 1–15.

[16] LIM, H., KANSAL, A., AND LIU, J. Power Budgeting for Virtualized Data Centers. In *USENIX ATC* (2011).

[17] MINAS, L., AND ELISON, B. The problem of power consumption in servers, 2009. `http://software.intel.com`.

[18] NATHUJI, R., ENGLAND, P., SHARMA, P., AND SINGH, A. Feedback driven QoS-aware power budgeting for virtualized servers. In *FeBID* (2009).

[19] NATHUJI, R., SCHWAN, K., SOMANI, A., AND JOSHI, Y. VPM tokens: virtual machine-aware power budgeting in datacenters. *Cluster computing* (2009).

[20] PINHEIRO, E., BIANCHINI, R., AND HEATH, T. *Dynamic Cluster Reconguration for Power and Performance*. Kluwer Academic, 2003.

[21] RAGHAVENDRA, R., RANGANATHAN, P., TALWAR, V., WANG, Z., AND ZHU, X. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News* (2008), vol. 36, ACM, pp. 48–59.

[22] RANGANATHAN, P., LEECH, P., IRWIN, D. E., AND CHASE, J. S. Ensemble-level Power Management for Dense Blade Servers. In *ISCA* (2006), pp. 66–77.

[23] STRUNK, A. Costs of Virtual Machine Live Migration: A Survey. In *IEEE Eighth World Congress on Services* (2012).

[24] VMWARE, INC. Resource Management with VMware DRS, 2006.

[25] VMWARE, INC. VMware Distributed Power Management Concepts and Use, 2010.

[26] VMWARE, INC. VMware vSphere vMotion: Architecture, Performance, and Best Practices in VMware vSphere 5, 2011.

[27] WALDSPURGER, C. A., AND WEIHL, W. E. Lottery scheduling: flexible proportional-share resource management. In *OSDI* (1994), USENIX Association.

[28] WANG, X., AND WANG, Y. Coordinating Power Control and Performance Management for Virtualized Server Clusters. *IEEE Transactions on Parallel and Distributed Systems 22*, 2 (Feb. 2011), 245–259.

[29] WHITE, T. *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.