Virtualizing Power Distribution in Datacenters

Di Wang, Chuangang Ren, Anand Sivasubramaniam
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA
{diw5108,cyr5126,anand}@cse.psu.edu

ABSTRACT

Power infrastructure contributes to a significant portion of datacenter expenditures. Overbooking this infrastructure for a high percentile of the needs is becoming more attractive than for occasional peaks. There exist several computing knobs to cap the power draw within such under-provisioned capacity. Recently, batteries and other energy storage devices have been proposed to provide a complementary alternative to these knobs, which when decentralized (or hierarchically placed), can temporarily take the load to suppress power peaks propagating up the hierarchy. With aggressive under-provisioning, the power hierarchy becomes as central a datacenter resource as other computing resources, making it imperative to carefully allocate, isolate and manage this resource (including batteries), across applications. Towards this goal, we present *vPower*, a software system to virtualize power distribution. vPower includes mechanisms and policies to provide a *virtual power hierarchy* for each application. It leverages traditional computing knobs as well as batteries, to apportion and manage the infrastructure between co-existing applications in the hierarchy. vPower allows applications to specify their power needs, performs admission control and placement, dynamically monitors power usage, and enforces allocations for fairness and system efficiency. Using several datacenter applications, and a 2-level power hierarchy prototype containing batteries at both levels, we demonstrate the effectiveness of vPower when working in an under-provisioned power infrastructure, using the right computing knobs and the right batteries at the right time. Results show over 50% improved system utilization and scale-out for vPower's over-booking, and between 12-28% better application performance than traditional power-capping control knobs. It also ensures isolation between applications competing for power.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General

Keywords

Datacenters, Power Management, Batteries

1. INTRODUCTION

Power consumption of datacenters has tremendous implications on both operating (op-ex) and capital (cap-ex) expenditures. While energy consumption has been the target of many prior studies to reduce op-ex, there is growing interest (e.g. [17, 21, 7, 30, 11]) in reducing cap-ex of power provisioning. Power infrastructure/distribution costs can contribute to over a third of a large datacenter's amortized monthly expenditures, with each provisioned watt costing between $10-20 [11, 18]. Consequently, it has become increasingly attractive to under-provision this resource. With aggressive under-provisioning, the power hierarchy - from incoming utility lines, switch gear and UPS units, going down to Power Distribution Units (PDUs) and even server power supplies - becomes a precious resource that needs to be carefully allocated and managed across applications to ensure safe operation. As in any other hardware resource, virtualizing this infrastructure can hide the under-provisioning (scarcity) from applications, providing each with the illusion of a dedicated/insulated hierarchy, while extracting the maximum value from each provisioned watt. This paper presents *vPower*, a software system to create virtual power hierarchies on this valuable physical resource.

We draw an analogy with physical memory, a valuable and under-provisioned resource. Even single applications need capacities larger than the available physical memory. Further, co-existing applications that are time and/or space multiplexed on that system, need to share this limited capacity. Virtual memory was introduced to overbook its capacity while insulating applications from each other. It provides applications with the illusion of a large address space without their having to deal with physical memory management explicitly. Dynamic memory management (using malloc()/free()) is at best employed at the virtual memory interface by the application. The underlying system/OS treats these calls more as hints in managing physical memory. We believe that as power infrastructure is aggressively under-provisioned, it becomes as central as other computing resources, mandating a similar management strategy. However, virtualizing power infrastructure poses unique challenges and opportunities:

- Power distribution uses a hierarchical network spanning several servers, and the levels in the hierarchy that are under-provisioned play a key role in how the resource has to be managed. A power over-draw in one part of the hierarchy can imply a reduction in power capacity for another part. Further, with many datacenter applications spanning multiple servers, we need the ability to create, manage and police virtual sub-hierarchies (unlike virtualizing physical memory of a single machine).

- Power demand management is system controlled by modulating other resources - mainly the CPU through scheduling, migration, consolidation and power state (DVFS) modulation. Application participation/hints, such as a `palloc()`, to guide such power management, has not been explored in greater depth, particularly for applications spanning several servers. In many conventional applications, since memory is an explicit abstraction, their direct involvement to give such hints seems more natural. On the other hand, conventional applications are oblivious of power as a resource, making the ability to give such hints less apparent. However, we believe that with many datacenter applications going through extensive profiling, debugging and tuning before they go into production [23], there may be adequate opportunities to gauge their power needs. For instance, [23] details the pre-production profiling of Google datacenter applications, on diverse server hardware before they are deployed on production systems, and one could envision collecting their power demands during such pre-production profiling phases. These can subsequently be used as hints in production mode. More importantly, since these are hints, over or under stating these needs or even stating these at a coarse temporal resolution, may still be better than not specifying at all.
- Demand modulation using computing knobs - scheduling, DVFS, migration, etc. - is no longer the only (indirect) means for controlling this resource. Supply side knobs that leverage batteries and other energy storage devices placed in one or more layers of the hierarchy, have recently been proposed [17, 21, 39] to facilitate aggressive power provisioning. A battery in a given level can temporarily boost the power capacity of the sub-hierarchy under it, even if the capacity of the line above this sub-hierarchy cannot sustain this power draw. With potentially several hierarchically distributed batteries, the choice of which to use to temporarily boost a sub-hierarchy's capacity, and a fair allocation of this capacity across applications, poses interesting trade-offs.

Provisioning virtual power hierarchies requires (i) understanding an application's power needs, (ii) admitting and co-locating the right set of applications, (iii) accounting for their power usage, and (iv) policing their execution to ensure fairness and effectiveness of power usage. This paper presents the design, implementation and evaluation of *vPower*, incorporating mechanisms and policies for these functionalities. It provides an optional and flexible interface (`palloc()`) for applications to give hints of their power needs, at different resolutions and accuracies. vPower has three main components - admission control/placement manager, accounting manager and enforcer. The admission control/placement manager regulates admission of applications, places them on appropriate servers and allocates power hierarchies. The accounting manager continuously tracks the power draw of different applications, that can span several servers, with respect to their allocations. It is the enforcer's job to (i) ensure no emergencies/violations during runtime (i.e., power on a line does not exceed provisioned capacity), (ii) maximize resource utilization, (iii) insulate the usage across applications, and (iv) ensure fairness. It uses a combination of computing knobs, such as DVFS and migration, and battery-based power boosts, for these purposes. Simultaneously it also incorporates intelligence to source power from the right batteries in the hierarchy at the right time.

We prototype vPower on a 2-level hierarchy, with batteries in both layers, containing 8 servers. Using representative datacenter benchmarks, we study the impact of different policies on system consolidation, performance and scale-out, fairness, and effectiveness in suppressing power emergencies. vPower provides 12-28% better performance than conventional approaches that use only computing knobs, on a rack that is 50% under-provisioned. It improves utilization by over 50% compared to an approach which conservatively schedules based on peak load requirements. It also effectively isolates mis-behaving applications ensuring other applications are not penalized. To our knowledge, this is the first paper that attempts to virtualize the power hierarchy of a datacenter spanning several servers, incorporating a combination of computing knobs and multi-level batteries.

2. BACKGROUND AND RELATED WORK

Power Hierarchy: Figure 1 shows a typical 4-level datacenter power hierarchy. At the highest layer, switchgear scales down the voltage of utility power, which then passes through centralized UPS units. The UPS units get another feed from backup diesel generators. Power then goes through Power Distribution Units (PDUs) to different racks, which then gets distributed through power strips to individual servers which all have their own power supplies.

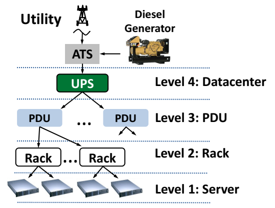


Figure 1: Power Hierarchy

Power Capping: When under-provisioning a layer, we need to ensure the resulting power limits (caps) are obeyed to ensure safety. Demand Response for power capping [2] has primarily leveraged three broad computing knobs: (i) Spatial Allocation/Consolidation [5, 31, 36, 24, 9] of workloads when they arrive, exploiting statistical multiplexing of the power profiles of different applications (enabling more aggressive power provisioning than would have been possible if we consider the peaks of all in unison); (ii) Migration [2, 34] of these workloads spatially to a different part of the datacenter (with more headroom), or even to a different datacenter, and (iii) Temporally Shaping the demand by power state modulation (throttling) of hardware devices [32, 40, 13, 3, 25, 22] and/or workload scheduling [28, 6].

Leveraging Energy Storage/Batteries: Some datacenters use distributed UPS units - server level in Google [14], rack level in Microsoft [27] and Facebook [10], though their motivation is to avoid double-conversion losses (and not power capping). There are recent proposals [17, 15, 21, 39] to provision extra battery capacities in different layers of the hierarchy. Whenever the power needs in the sub-hierarchy hit safety limits (termed emergencies in this paper), the batteries can step in to temporarily source/supplement the power, so that the layers higher up do not notice the emergency. These studies have demonstrated that power availability is not compromised by keeping a reserve capacity that is sufficient to fail-over to diesel generators upon power outages. Computing knobs are still needed to handle long emergency durations that are beyond the capacities of the provisioned storage. These studies have shown that provisioning additional battery capacities, whether placed centrally [15], or distributed at server/rack levels [17, 21], is economically attractive, with the consequent cap-ex savings of under-provisioning the power infrastructure more than offsetting additional battery capacity costs. Further, a recent work [39] has also shown that placing energy storage devices (not just batteries) in multiple layers of the hierarchy, rather than single-level placement, can be even more economically attractive. Battery capacity (over-)provisioning/placement is orthogonal to this paper, and we leverage all this prior work, considering these energy storage devices, at different layers, as part of the power infrastructure, towards creating virtual power sub-hierarchies. In fact, lower capacities at one or more levels make it even more important to manage these devices intelligently, further motivating this work. Trade-offs between sharing versus interference in using these batteries across applications is an important consideration for building virtual hierarchies.

Power Virtualization: Unlike other resources, virtualization of power has not been studied extensively. Relevant work in this area is on accounting and control of power/energy usage of workloads in a co-hosted environment (e.g. [1, 35]) or on a multicore server [33], and coordinating (possibly conflicting) power management decisions of individual virtual machines co-hosted on a physical server [29]. However, our work looks at (i) virtualizing entire power hierarchies and not just a single server, and (ii) looks to leverage multi-level energy storage (batteries) to achieve this goal (not just computing knobs). As we will see, sharing and isolation of battery usage across applications, especially in a hierarchical setting, introduces several new considerations in the power hierarchy management. Virtualizing and managing batteries across applications has been mainly studied (e.g. [4, 41]) in the context of embedded and mobile devices, where the primary goal is to prolong battery runtime, rather than power capping.

3. VPOWER SYSTEM ARCHITECTURE

Goals and Problem Statement: We work with an abstraction of the power hierarchy which has an imposed power cap P_i^{cap} at one or more levels L_i, where $L_i = L_1$ (server level) to L_L (datacenter level). Our focus here is only on mechanisms and policies to adhere to these imposed P_i^{cap} values, and a detailed evaluation of the trade-offs with different P_i^{cap}s (examined in prior work [15, 21, 17, 39]) is orthogonal to this paper. Specifically, the implementation and evaluation platform in this paper uses a 2-level hierarchy, L_1 (server) and L_2 (rack) with associated power caps P_1^{cap} and P_2^{cap}, but the discussions and results can generalize to more levels. At each level, we assume the presence of energy storage - our experimental platform uses lead-acid batteries, which are currently the most common in datacenter UPS units. The goal is to ensure that the power draw never exceeds P_i^{cap} at all corresponding L_is in the hierarchy, using a combination of (i) demand-response computing knobs - workload placement, consolidation, migration, scheduling, power state modulation (DVFS states), etc., and (ii) batteries with maximum power P_i^{bmax} and energy capacity E_i^{bmax} available at each level L_i in the hierarchy which can temporarily step in to provide the extra power needed to handle the needs beyond P_i^{cap}. Table 1 summarizes these notations. This paper explores application interfaces, software mechanisms and policies to attain this goal when hosting multiple applications that share this power infrastructure. In the process, we need to adhere to application SLAs, meet these SLAs with minimum amount of resources (i.e., maximize consolidation and utilization), and ensure fairness in how this power is allocated and shared between applications.

Notation	Description
L_i	Power sub-hierarchy i
P_i^{cap}	Power cap of L_i
P_i^t	Power draw of L_i at time t
P_t^{req}	Requested power from an application for time t
P_i^{bmax}	Power capacity of a battery at L_i
E_i^{bmax}	Energy capacity of a battery at L_i

Table 1: Some Common Notations

Overview of System Architecture: Towards this goal, we implement a software-based virtual power hierarchy (vPower) (from the datacenter at the root, to the servers) for each application, within which it can safely execute, insulated from any power-related emergencies arising from co-existing applications, even when their aggregated peak demand can exceed the power cap in any level. The power hierarchy that is being virtualized includes the capacities of all the power equipment (switchgear, transformers, UPS units, PDUs, and individual power outlets) from the root, all the way down to individual servers running this application. In addition, it also virtualizes the battery capacities of each layer.

While one could conservatively book/reserve for the peak power demands of applications, and admit only those whose collective peak demand fits within the power caps P_i^{cap}, overbooking can improve system utilization, since simultaneous and sustained peak demands from all applications may be less common. vPower has to: admit the right set of applications (*admission control* described in section 4.2.1); place/co-locate them with the right/synergistic mix of applications in the sub-hierarchy that would lead to fewer power emergencies (*placement* described in section 4.2.1); and suppress any power violations, if and when they occur, from propagating higher up the hierarchy using both computing knobs and right batteries (*enforcer* described in section 4.2.3). An *accounting manager* needs to track the dynamic power consumption of different applications, as described in section 4.2.2, to provide up-to-date information for the enforcer to ensure fairness. As with any other shared resource management, vPower can leverage application-level power related information that can be explicitly provided using a `palloc()` interface described in section 4.1.

The interfaces to the "power infrastructure hardware" that can be exploited by our system are shown in Table 2. Apart from interfaces to monitor the instantaneous power draw on any line in the hierarchy (`power()`), there are also interfaces to batteries in each level to monitor their state-of-charge (`soc()`), and control their charge and discharge rates. There are programmable power electronics circuitry to control the latter, but since our experimental setup does not provide this functionality, our evaluations account for such control capability in the management decisions. In addition, the system also uses hardware/kernel/VM interfaces to change DVFS states, modulate scheduling and migration decisions, etc.

Our system architecture is pictorially depicted in Figure 2 for a 2-level hierarchy with server and rack level batteries, and power caps to be enforced at each of these two levels. There are two main software components: (i) one running as a driver within each server to initiate server level control knobs (DVFS, scheduling, etc.); and (ii) another run-

Interface	Description
`power(...)`	returns instantaneous power draw of given line
`soc(...)`	return state of charge of a battery
`charge(...)`	charge battery at specified rate
`discharge(...)`	discharge battery at specified rate

Table 2: Hardware Interfaces to Power Infrastructure

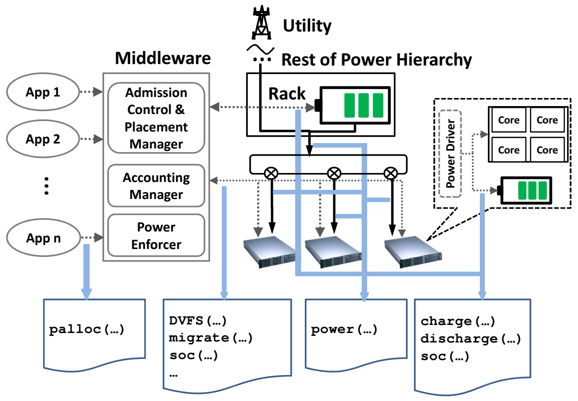


Figure 2: vPower Architecture

ning as middleware on a dedicated server, which implements the management/control policies and appropriately interfaces with the server drivers. The latter also interfaces with the power distribution network to monitor (measure power draw, battery SoC, etc.) and control their operation (sourcing from batteries, switching off outlets, etc.). The next section gives details on the design choices and implementation details for vPower.

4. VPOWER MECHANISMS AND POLICIES

4.1 The palloc() Interface

As with any other resource, providing detailed and accurate information of an application's power profile can help manage it better, especially in aggressively under-provisioned settings. There are two dimensions to providing this information - how much? and over what duration? - and our application interface is accordingly specified as

```
palloc(<power,interval>,<power,interval>,...)
```

where the arguments are tuples specifying anticipated application power needs during different time intervals over its execution. Ideally one would like to accurate power needs at fine temporal resolution as is depicted in the power profile graphs of Figure 3. While many datacenter applications are long running services (e.g., web searching, memcached, etc.) and/or periodic/repetitive workloads (e.g., web crawling), with many of them going through profiling and fine-tuning phases before going into production mode [23], one may be able to obtain detailed and accurate power needs through profiling, barring execution and data dependency vagaries. Further, prior work on load predictability (time-of-day behavior in web services, flash crowd behavior for media services, etc.) may be useful for these specifications, as is prior work on phase characterization [19] to make such requests ahead of their need. However, getting accurate and fine-grained information (depicted as "Exact") may not always be feasible, and we accommodate several loose specifications (Table 3) both in requesting power, as well as in the temporal durations (possibly not even needing to specify durations) as below.

Along the temporal dimension, one need not even give

	No Time	Entire Execn.	Every 60 secs.	Every Instant
Max. Power	PMax	PMax-entire	PMax-60	Exact
90th Per. Power	P90	P90-entire	P90-60	-
Avg. Power	PAvg	-	-	-
Min. Power	PMin	-	-	-

Table 3: Example palloc() Specifications considered in evaluations.

any time information (as in PMax, PMin, PAvg, P90) if jobs are long running without much variance. It is also possible to estimate broadly defined execution phases (as in the MapReduce profile shown in Figure 3 where the power draws are quite different between map and reduce phases) and accordingly specify the intervals. In the interest of a uniform comparison across workloads, we introduce a range of pre-determined interval sizes between the "Every Instant" and "No Time" extremes, and consider representative intervals of entire application duration (PMax-entire, P90-entire) and 60 seconds (PMax-60, P90-60) in our evaluations.

Along the power dimension, one could consider a wide range of values starting from the peak (PMax) (which can be conservative depending on the time interval it is specified for), a 90th percentile (P90) of this peak (less conservative), an average over the interval (PAvg) or even as low as the minimum power (PMin). Note that our interface does not preclude an application from specifying any power value, over any particular duration. However, our system ensures that regardless of what the application specifies (which is treated more as a hint), it will insulate applications from each other. By giving bad hints - either intentionally or unintentionally - an application can at best hurt itself. For instance, when an application specifies values higher than PMax - this would come at the cost of our system possibly not admitting this application when power budgets are tight. At the other end, when an application requests very low power needs, and then (misbehaves) starts consuming higher power than what it initially specified, our enforcer will step in and penalize it.

4.2 The Middleware

The middleware is the core software component for allocating and controlling power across applications. It consists of an admission control and placement manager, an accounting manager and a power enforcer. We next explain their goals, design tradeoffs and implementation details.

4.2.1 Admission Control and Placement Manager

Goal: Based on an application's palloc() hints, this manager evaluates whether it can accommodate these needs without violating existing allocations, and if so where in the hierarchy it should be placed (i.e., co-location with others). The shared power infrastructure (even if the computing load is spread across different physical servers), and shared energy storage (batteries) at multiple layers in the hierarchy, introduces additional considerations to this problem compared to approaches that only consider computing resources.

Design Tradeoffs and Discussions: Note that, the net power draw of a hierarchy can be temporarily boosted beyond its provisioned capacity by as much as the *sum* of the power draws that can be sustained by all the batteries (distributed and hierarchical) within that hierarchy. Our admission control policies are therefore based on our reliance on these batteries to achieve power capping:

- *Conservative Policy:* In this policy, batteries are not taken into consideration for admitting applications. Batteries may still step in during execution to suppress peaks if they arise, which may be rare because of the conservative admission control. Hence, it may be better to use less stringent power needs in palloc() when using this policy.
- *Moderate Policy:* This allows a certain percentage (e.g., 20%) of battery power (P_i^{battAD}) and energy (E_i^{battAD}) capacity to be available in addition to normal line capacity when admitting applications. It can admit more, with possible subsequent emergencies when batteries run out.
- *Aggressive Policy:* This is the same as Moderate, with 100% (still leaves reserve capacity to ensure power availability mandates upon outages [16, 15]) of the capacity used for admission control. It is better to specify more stringent needs in palloc() (e.g., PMax or P90) in conjunction with this policy, to lessen emergency occurrences.

Once admitted, we need to decide where to locate/co-locate this application. With distributed batteries across the hierarchy, the degree of balancing/unbalancing their usage serves as the main criteria when exploring this question. This can be captured by the correlations - High, Low and Anti - of the power profile of this application with those in the sub-hierarchy/server where it is being considered. Co-location effectiveness is also a function of the admission control policy. Co-location of anti-correlated workloads may be a better option when conservative admission control is used, since the latter is less reliant on batteries, and the anti-correlation would lessen the possibility of power emergencies. At the other end, co-location of correlated workloads may be better for an aggressive admission control policy (which may have higher emergencies) since it aggregates (batches together) the discharging of batteries, giving longer time windows for charging, similar to how unbalancing of load creates more opportunities for server shutdown in [31]. The design choices along these two dimensions are qualitatively summarized in Table 4 indicating the priority order of correlation degrees for placement under a given admission control policy. We will show experimental results to corroborate these choices.
Implementation: When an application specifies its power needs via palloc(P_t^{req},Δt), the admission control algorithm checks two aspects of this requirement - power (P_t^{req}) and energy ($P_t^{req} \times \Delta t$) needs - and is admitted only if both can be met. For applications which only specify power (e.g., PMax, P90, PAvg, PMin) due to the lack of time information, we deal with energy constraint on a best effort basis and simply assume the energy constraint is met for admission. These checks are recursively made starting from the root. For each sub-hierarchy L_i, a check is made to ensure: (i) for every time interval t, total allocated power ($P_{i,t}^{alloc}$) for existing applications plus the new power needs (P_t^{req}) from the requesting application should be less than or equal to the sum of provisioned power cap of this sub-hierarchy (P_i^{cap}) plus the amount of power from batteries (P_i^{battAD}) dedicated for admission control (as per the above 3 policies); (ii) total allocated energy for existing applications (E_i^{alloc}) plus the new application's energy needs ($\sum(P_t^{req} \times \Delta t)$) should be less than or equal to the sum of maximum energy from the outlet (E_i^{outlet}) and energy from batteries (E_i^{battAD}) committed to admission control. Different amounts of power/energy from batteries represent the aggressiveness of admission control policies as described above. Once admitted to a sub-hierarchy, we place an application based on the choices from Table 4.

	Conservative $P_i^{battAD} = 0$ $E_i^{battAD} = 0$	Moderate $P_i^{battAD} = 0.2P_i^{bmax}$ $E_i^{battAD} = 0.2E_i^{bmax}$	Aggressive $P_i^{battAD} = P_i^{bmax}$ $E_i^{battAD} = E_i^{bmax}$
High Corr.	(3)	(3)	(1)
Low Corr.	(2)	(2)	(2)
Anti-Corr.	(1)	(1)	(3)

Table 4: Placement choice for a given admission control policy. Priority order is (1), (2) and then (3).

4.2.2 Accounting Manager

Goal: It dynamically tracks an application's power with respect to its allocation across its servers.
Design Tradeoffs and Discussions: We adopt a simple credit based accounting mechanism [4] to track the difference between power reservation and its actual consumption for each application. An application is given positive credits if it consumes (P_t) below its reservation (P_t^{req}) while credits are subtracted if its power goes above reservation. However, to prevent misbehaving applications from continuously banking credits without using them, a bound is enforced on the amount of credits that it can bank (e.g., the bound can be the battery capacity allocated to an application). Similarly, to prevent misbehaving applications from continuously discharging battery, a bound is enforced on the amount of credits that can be in debt.
Implementation: For each application, we maintain a data structure called PCB (Power Control Block), which contains its power needs, accumulated credits, assigned server(s), etc., and the credits are updated periodically. Outlet power is sampled for each server at a second granularity, which suffices if only one application is running on that server. Within a server, one could use apportioning techniques between co-existing applications using system metrics for power accounting as in [20, 35, 33], though outlet level server metering suffices in our evaluations which places applications on distinct servers.

4.2.3 Power Enforcer

Goal: Despite admission control, there may be power emergencies during the runtime due to under-estimates of application needs and/or aggressive over-booking as explained earlier. The enforcer uses different computing demand-response knobs (DVFS/clock throttling and migration) and batteries in different layers of the hierarchy to handle these emergencies. Specifically, issues related to which knob(s) to use for different applications, which batteries to employ in a hierarchical setting, are some considerations in the design of the enforcer. The goal is to meet application performance SLAs, as well as maximize system utilization, while ensuring fairness between applications in the choice of knobs (performance detrimental computing knobs versus use of batteries).
Design Tradeoffs and Discussions: Consider any two hierarchy levels L_i and $L_{i-1,j}$, with the latter having $j = 1..n$ components (and hence its 2-dimensional representation) directly connected to L_i. We use the following notations: the provisioned peak power (power cap) of the outlet that level L_i can draw from is P_i^{cap}; the required aggregate power draw by level L_i is $P_i^t = \sum_{j=1}^{n} P_{i-1,j}^t$ at time t; each component

$L_{i-1,j}$ can draw power from its own battery with energy capacity $E_{i-1,j}^{bmax}$ and maximum discharge/charge power $P_{i-1,j}^{bmax}$, and some or all of the parent battery at L_i depending on the capacity of the provisioned line between $L_{i-1,j}$ and L_i. One or more of the $n+1$ (n at L_{i-1} and 1 at L_i) batteries can be used to shave all or part of the power violation at L_i.

Lemma. *No matter which strategy is taken to discharge batteries (i.e., which battery to use), a given peak P_i^t exceeding P_i^{cap} at L_i can be shaved by batteries in the hierarchy if the following three conditions are satisfied: (A) $(P_i^t - P_i^{cap}) \leq min(P_{i-1,j}^t, P_{i-1,j}^{bmax}), \forall j$, (B) a battery is discharged only when $P_i^t > P_i^{cap}$, and (C) discharging decisions of all batteries are coordinated (i.e., the aggregate power drawn from all $n+1$ batteries at instant t is at most $(P_i^t - P_i^{cap})$).*

This lemma implies that all batteries in the hierarchy can be treated as one large battery placed at L_i provided the three conditions are obeyed. Condition (A) says that the violation at L_i can be shaved by removing the contribution from any child component $L_{i-1,j}$ by sourcing that component from its local battery. Hence, any one of the batteries at $i-1$ can be used to suppress the violation. Condition (B) says that the battery at $L_{i-1,j}$ is used only to address the violation at the parent L_i, and not to suppress a peak violation that happens at $L_{i-1,j}$ itself (i.e., only the parent is under-provisioned and not each child). These, together with perfect coordinated control of all child batteries to shave this peak (Condition C), gives the simplistic illusion of a centralized larger battery of energy capacity $E_i^{bmax} = \sum_{j=1}^{n} E_{i-1,j}^{bmax} + E_i^{bmax}$ and power capacity $P_i^{bmax} = \sum_{j=1}^{n} P_{i-1,j}^{bmax} + P_i^{bmax}$ at L_i. The proof is given in our technical report [38]. However, in practice, these conditions may not hold. We need to, thus, carefully choose the right battery to handle the emergencies as described below to lessen the probability of violating these conditions. For clarity, we discuss these issues assuming L_i (parent) is a rack, and each child $L_{i-1,j}$ is an individual server.

Parent's violation is larger than power consumption of m children (i.e., $(P_i^t - P_i^{cap}) > \sum_{j=1}^{m} P_{i-1,j}^t$): If some policy schedules an imbalanced lower workload on m servers than the other $(n-m)$ servers, and has drained out the batteries of these $n-m$ other servers and the rack battery because of prior usage, then the power violation at the parent can not be handled by batteries alone. Even if these m servers completely draw power from their local batteries, the reduction will not suffice to address the emergency at the parent. The problem gets accentuated when m decreases, especially as we move towards the root. To address this concern, it is better for policies to (i) balance the load across servers to increase $P_{i-1,j}$ for under-utilized servers, (ii) use up the batteries at the servers which have less power demanding applications first before going to others in cases of load imbalance.

Parent's violation is larger than power suppliable by m children batteries (i.e., $(P_i^t - P_i^{cap}) > \sum_{j=1}^{m} P_{i-1,j}^{bmax}$): Consider a case where a policy leaves residual capacities in only these m batteries after prior usage, and has already drained out the other $n-m+1$ batteries (including the rack battery). In this case, the residual capacities of these m server batteries are not sufficient to handle the rack violation (again the problem accentuates when m decreases as we move to the root). To address this concern, it is better to (i) use batteries evenly at the servers, and (ii) save rack battery for such power violations since they are typically provisioned with larger power/energy capacity to handle a potentially larger load.

Child is itself under-provisioned (i.e., $P_{i-1,j}^t > P_{i-1,j}^{cap}$ for some j): With aggressive under-provisioning deeper in the hierarchy, a child battery may have been used to shave its own peak, rather than the peak of a parent, thereby violating Condition B. Subsequently, when called upon to suppress the peak of a parent, there may not be residual capacity. Since a battery can be useful to suppress peaks from propagating higher in the hierarchy and not in the reverse direction, we use the following general guidelines in our policies: (i) reserve some capacity for suppressing emergencies at its own level, and (ii) use higher level batteries for the higher level violations as much as possible (provided lower levels stay within their budget).

Implementation: Note that palloc() from applications, in combination with our accounting manager, helps track the positive/negative credits accumulated by each application. Our runtime enforcer takes these credits into consideration when employing the computing and battery knobs in apportioning the emergency suppression mechanisms across applications. Such proportional power allocation (whether it be in the computing knobs or in the power from batteries), is similar to proportional fairness studied in other resources [37]. The enforcer prioritizes the order of employing knobs based on the impact on performance (e.g., migration can be relatively costly as studied in [17]), as well as the duration and stringency of the emergency.

When the estimated duration of violations exceeds a threshold, the power enforcer will start to migrate applications with least accumulated credits to destination hierarchies with sufficient slack. For smaller emergency durations, it uses local DVFS and battery knobs. Based on the guidelines of hierarchical battery usage discussed above, we use the hints coming from palloc() to implement the enforcer as follows: (i) reserve certain local battery capacity at each server if any local power violations are anticipated; (ii) use local batteries for smaller power violations, saving the shared higher level batteries for larger ones, and supplement it with DVFS if batteries alone cannot handle the need; and (iii) as far as possible, source from batteries of servers with low anticipated future demand.

When there is slack in power usage, batteries are re-charged in the following order: (i) batteries without sufficient charge on servers estimated to have local power violations are given first priority; (ii) batteries with lower state of charge are given higher priority; (iii) batteries on servers estimated to have low power demand (hence unlikely to incur power violations) are given the least priority.

5. EVALUATION

5.1 Experimental Setup

We evaluate vPower on a scaled-down prototype using a cluster (rack) of $n = 8$ servers. The face-plate rating of these servers is 450W, idle power is around 120W and the peak power that we can push them to across our workloads is 300W. The dynamic power consumption can be modulated with several DVFS states and clock throttling states. The "Power Driver" at each server changes power states using the MSR registers. Each server is equipped with a UPS unit which serves as the server level battery. A larger capacity UPS is used as the rack-level battery. We consider a 4-minute battery per server, of which we leave a residual capacity of 3 minutes for availability guarantees. The UPS is able to report its load and remaining battery runtime. By switching off the incoming power to a UPS unit over the network, we create the illusion of a power outage to the UPS to source power from its batteries. This is a conservative way compared to a more elaborate approach which has power electronics circuitry to instantaneously source only the additional power needs (excess over the cap) from batteries with the remaining coming from outlet power. Since our experimental setup does not provide this functionality, our evaluations account for such control capability in the management decisions. vPower can leverage the server battery for enforcing server level caps, and can use one or more (and perhaps taking turns to extend the duration) server batteries and/or the rack battery for enforcing rack level caps. We use another cluster as the destination for migrating workloads. All our applications are hosted as VMs, with Red-Hat Linux 5.5, under Xen. A separate machine runs our "Middleware" which implements the algorithms, and sends appropriate throttling and VM migration commands to the server power driver.

5.2 Workloads

We consider a suite of 8 representative datacenter applications (Table 5), that are both user-facing (interactive) and batch workloads. Interactive applications include the Yahoo! Cloud Serving Benchmark (YCSB) [8], an in-memory key-value store Memcached benchmark [26], two other applications (MediaServer, WebSearch) from Cloudsuite [12]. Batch applications include a WebCrawling benchmark from Cloudsuite, a Hadoop MapReduce application (word count used in several analytics applications), a GPU application in CUDA implementing the Black-Scholes financial model using a NVIDIA card, and a virus scanner (VirusScan). The last 2 are not amenable to migration from the server where they are executing. Figure 3 shows the power profiles on a representative server running these applications, and their runtime is given in Table 5.

Workload	Type	Runtime (Secs)
YCSB	user-facing	587
MediaServer	user-facing	370
Memcached	user-facing	1020
WebSearch	user-facing	629
MapReduce	batch	365
WebCrawling	batch	370
GPU	batch	1000
VirusScan	batch	888

Table 5: Workloads

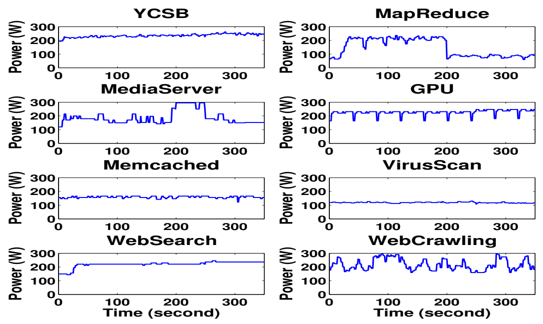


Figure 3: Application power profiles. For clarity, only first 350 seconds of execution is shown.

5.3 Evaluation Metrics

We consider the following metrics from the system and application perspectives:

- *Power violations:* the number of times that a power cap is violated at any of the two levels. Note that in our complete system, the enforcer will ensure no violations. We use this metric to compare the pros and cons of isolating the admission control/placement policies by removing the power enforcer in those experiments. In practice, the extent (magnitude and duration) of the violations should be tracked, but in the interest of clarity when presenting results we find that the number of power violations is a reasonable proxy for comparisons.
- *Degree of consolidation:* the number of applications that can be co-located on the same server/rack without exceeding the specified power caps. Consolidation can extract more value from existing compute and power infrastructure, and is also attractive from the viewpoint of lowering energy consumption/costs.
- *Scale-out:* the number of instances that an application can be replicated, to improve its throughput. This metric is more useful when the goal is to accelerate the performance of a single application (e.g., Memcached) as opposed to consolidation of disparate applications.
- *Performance Degradation:* the percentage degradation (of response time, throughput, completion time, etc. depending on the application) of running an application under a power cap with respect to the same experiment running without a power cap.
- *Fairness:* a measure of how the system treats co-existing applications from the performance viewpoint when meeting the power budgets. Rather than a single numerical metric, our results will clearly depict the differences between policies in penalizing applications.

5.4 Impact of Admission Control Policies

We begin by evaluating the three admission control policies - Conservative (II), Moderate (III) and Aggressive (IV) - and compare them with two baseline schemes - "Baseline-power" (I) which admits applications as long as adding their peak power consumption ($PMax$) does not violate the caps (i.e., assumes no statistical multiplexing or dynamic modulation knobs and is thus extremely conservative), and "Baseline-utilization" (V) which admits applications considering only the average utilization of the applications (which is somewhat representative of how consolidation is currently conducted, without regard to any power caps in the hierarchy, and is consequently very aggressive). As explained in section 4.2.1, the effectiveness of these policies depends on the specifications/hints that an application can give, and we consider the design space for the specifications given earlier in Table 3. In the following experiments, we set a rack level power cap of 1200W (approximately half of the maximum power to which we can push our rack). To isolate the impact of admission control, we remove the power enforcer in these experiments (which can lead to power violations depending on the aggressiveness of admission control), and study the resulting consolidation and scale-out effectiveness.

Consolidation:. Admission control restricts the consolidation degree (the number of applications in a rack) with the possible benefit of fewer power emergencies. To study these trade-offs, we conduct a stress-test where we try to place as many of the eight applications (as allowed by the admission control policy) in the rack, and examine the resulting violations at the rack level. In reality, the choice for co-locating applications on a rack or a server would depend on additional issues such as performance interference, storage locality, security, etc., over and beyond power caps. However, we ignore such considerations to mainly isolate the impact of the power cap at the rack level, and allow up to 8 applications to co-exist in the same rack, but place each application on a separate server, i.e., there is no server level interference, but there could be rack level interferences.

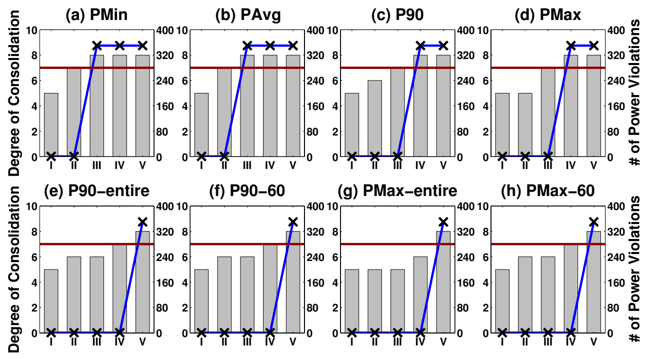


Figure 4: Consolidation Results for different Admission Control policies and Power Need specifications. Bars depict consolidation degree (left y-axis) and points on line depict number of power violations (right y-axis) of rack power cap set at 1200W. I:Baseline-power, II:Conservative, III:Moderate, IV:Aggressive, V:Baseline-utilization. Horizontal line represents consolidation degree (= 7) possible with "Exact" power specification without any power violations.

Figure 4 shows the trade-offs between consolidation degree and its consequences on power violations for each of the three admission control policies, comparing them with the two baseline extremes. Results are shown for the application power specifications outlined earlier in Table 3. In these graphs, we also show the maximum possible consolidation that can be attained (which is 7) without having any violations (and not requiring either batteries or computing knobs in the runtime), as a horizontal line. From these results, we make the following observations:

- A conservative admission control policy (II) which does not consider batteries, is comparable to "Baseline-power" (I) that provisions for the peak, and degenerates to the latter when the applications use the PMax specification (Fig 4 (d)). Unless applications grossly under-estimate their power (PMin in Figure 4 (a)), this policy is not preferable.
- At the other end, the aggressive policy (IV) achieves the same consolidation degree as "Baseline-utilization" (in the cases where a time interval is not specified).
- As long as the consolidation degree is less than or equal to 7, there are no violations, even when there is no dynamic enforcement i.e., statistical multiplexing of workload profiles suffices to keep the power draw within the cap. This is the case for all 3 policies in all the specifications which have a duration component (either "entire" or "60 seconds" in Figure 4 (e-h)).
- It is only when durations (even the execution time of the application) are not specified, that the moderate and aggressive policies start overbooking the power infrastructure, which can potentially lead to power violations - these would be suppressed at runtime by the enforcer with either battery or computing knobs. While this may appear counter-intuitive (i.e., no time component should lead to less flexibility in multiplexing needs and thereby lower consolidation), recall that there are 2 criteria to admission control - power and energy. Even though power multiplexing may be better with temporal information, note that when time durations are not specified, the energy criteria is always assumed to be met (section 4.2.1) from batteries, allowing more applications to be co-located.

Scale-Out:. Instead of co-locating disparate applications within a given infrastructure capacity, a datacenter may only be interested in accelerating the performance of a single application with additional instances for scale-out. We conduct similar experiments by creating more instances of the Memcached server workload within the rack as allowed by the admission control policies. Figure 5 (a) shows a representative result for the P90 specification, which reiterates the observations made in the consolidation studies. With more aggressive admission control, we can add more instances, with the number of memcached servers supported by the rack going to 8 for the Aggressive (IV) policy. Violations also increase, but we will shortly show that these can be effectively managed without significant performance degradation when we introduce the enforcer. Without any runtime enforcement, one can achieve a scale-out to at most 5 servers (horizontal line) if we are constrained by the power cap.

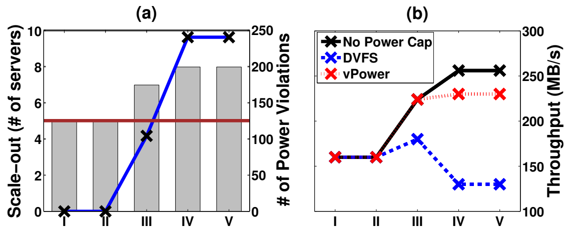


Figure 5: Memcached Scaleout with P90. I:Baseline-power, II:Conservative, III:Moderate, IV:Aggressive, V:Baseline-utilization. In (a), the horizontal line represents scale-out degree (= 5) using "Exact" power specification without any power violations, bars depict scale-out degree, and points on line represent number of power violations.

5.5 Impact of Placement Policies

We conduct experiments to study the impact of the policies when placing two applications - WebCrawling and MediaServer - in the same rack, with each application placed on its own server. These two are representative of more sinusoidal behavior in the power profile, helping us to study the influence of cross-correlations. To capture different cross-correlations, we simply vary the starting time of these applications, helping us capture a wide range of multiplexing possibilities (correlations). The "P90" interface is used to specify the power needs of these applications, and the enforcer is in place to avoid violations at the potential cost of performance degradation. As explained in section 4.2.1, placement choices go hand-in-hand with admission control, and we consider the correlation factor interactions with the aggressiveness of admission control as is depicted in Table 6. Note that the power caps need to be changed in order to ensure the two applications are admitted in all the admission control schemes, i.e., power cap is increased by the amount of battery capacity as you move from aggressive (right column where the power cap is 300W) to conservative (left column where the power cap is 500W) in Table 6. Consequently, one should not compare the performance degradations across columns. Rather, the trend within each column, and how that trend changes when we move from aggressive to conservative is what is important. As we can see, the columns to the left prefer an anti-correlated co-location of workloads, while the aggressive admission control prefers a more correlated placement. These observations validate our choice of placement priorities based on the admission control policies as discussed earlier in section 4.2.1.

	Conservative	Moderate	Aggressive
Rack Cap	500W	460W	300W
High corr.	(1,1)	(8,6)	(15,14)
Low corr.	(0,0)	(0,0)	(20,18)
Anti-corr.	(0,0)	(0,0)	(25,20)

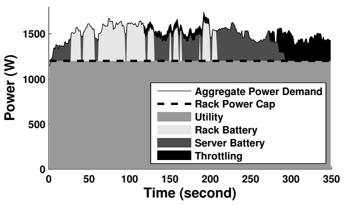
Table 6: Performance degradation of placing WebCrawling + MediaServer in the same rack with different cross-correlations and admission control policies. Tuples (WebCrawling, MediaServer) show % degradation w.r.t. running the same experiment without a power cap.

5.6 Effectiveness of Enforcer

Performance. Results in Figures 4 and 5 (a), showed the impact of the admission control policies without the enforcer in place, which can lead to power violations in some of the cases. We now reinstate the enforcer to suppress these violations, and show the resulting application degradation for those two sets of experiments with the "P90" specification in Figure 6 (a) and Figure 5 (b) respectively.

Application	DVFS	vPower
YCSB	49	20
MediaServer	24	9
Memcached	49	25
WebSearch	107	48
MapReduce	72	41
GPU	60	25
WebCrawling	39	10
VirusScan	0	0

(a) Perf. Degradation (%)



(b) Power Profile (Sourcing and Capping)

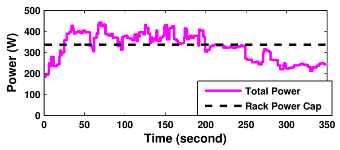
Figure 6: P90 specification in the Figure 4 experiment with Enforcer

As can be seen, while the degradation is non-zero in most applications for the consolidation experiment (Figure 6 (a)), it is still significantly better than a DVFS-only approach (an Oracle-based best DVFS states are chosen to adhere to the power caps). Degradation with vPower is between 15% to 69% lower than in a DVFS-only approach (12%-28% performance improvement) for the more power hungry applications (VirusScan draws relatively low power). The effectiveness of the enforcer can be explained with Figure 6 (b), which shows how it meets the aggregate power demand of all 8 applications - through normal power (utility), one or more batteries, and DVFS throttling when necessary. With vPower, DVFS is employed mainly when batteries do not suffice (from 270 seconds onwards). There are periods in-between (e.g. 120-200 seconds), when some applications reach their credit limits, and DVFS is used to enforce their debt, as seen in a small top portion of the curve in this region. Server batteries are the first choice for small amplitude violations while the rack battery is used for most large amplitude violations. The choice of which battery to use, and the fairness to differently meet application demands (through batteries or throttling) is explained in more detail later in this section. In the shown zoomed-in 350 second time window, there is little opportunity for re-charging the batteries, though such re-charging does happen with power slack from 370 seconds (not shown in Figure 6 (b)).

Similarly, the scale-out experiment for Memcached in Figure 5 (b) shows vPower giving throughput close to the uncapped case, and a value that is 25%-75% higher than the throughput of a DVFS-only option. Note that for aggressive policies (IV and V), the throughput of DVFS-only enforcement actually drops below the throughput of less aggressive policies (I, II and III); while the throughput with vPower actually increases despite the power cap. This shows that vPower can help applications such as Memcached achieve better scaleout capabilities when hosted in aggressively underprovisioned power infrastructure.

Battery Management. We now show two examples depicting the importance of picking the right batteries at the right time towards maintaining the power caps, and show how the enforcer in vPower chooses the better option than always opting to first use the rack battery or server batteries.

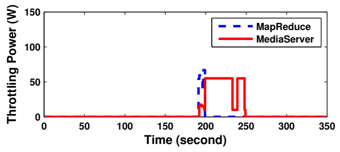
Rack (Shared) Battery First being the better option: We run an experiment with a rack of two servers running MapReduce and MediaServer respectively, and the aggregate rack level power draw is shown in Figure 7 (a). Both applications specify their power needs using "P90-60", and we set both a server-level power cap of 240W and a rack level power cap of 340W (we proportionally reduce the amount of power that can be drawn from the rack level battery, which was originally provisioned for 8 servers). As per heuristics described earlier in section 4.2.3, when the enforcer anticipates local power violations, it first discharges rack (shared) battery for rack power violations and uses server level (local) batteries for such violations only when the shared battery reaches its lower threshold. This leaves more charge in local batteries for later use, and Figures 7 (e) and (f) show the amount of power that is capped for each application by throttling and the state of charge (SoC) of batteries (at the two servers and at the rack), respectively with vPower. On the other hand, a scheme (local battery first) that discharges local batteries greedily and goes to the rack battery only when the former runs out of charge, will not be able to handle the higher load that may come later to exceed the server level power cap. This can be seen in the throttling power and SoC graphs in Figures 7 (c) and (d) respectively. Using up the MediaServer server's battery in the first 200 seconds to handle rack level violations, leads to its inability to handle its own power violations later on, resulting in a 9% performance penalty (Figure 7 (b)). Note that we intentionally leave a residual capacity of 20% of the usable battery capacity (which already excludes residual capacity for availability guarantees) due to battery lifetime issues [39]. Interestingly, MapReduce is throttled by vPower at around time 200s, despite having capacity in its local battery. This is because it is in debt, and its local battery is being conserved for possible subsequent rack level violations, in the interest of fairness that is covered later in this section.



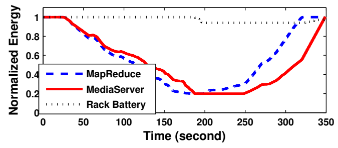
(a) Aggregate Rack Power

	Local-First	vPower
MediaServer	9	0
MapReduce	1	1

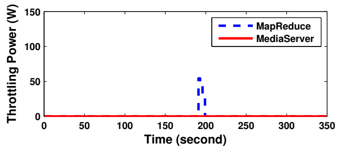
(b) Performance Degradation (%)



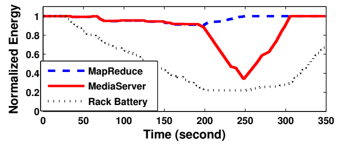
(c) Throttling (Local First)



(d) SoC (Local First)

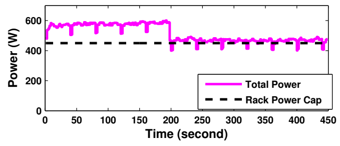


(e) Throttling (vPower)



(f) SoC (vPower)

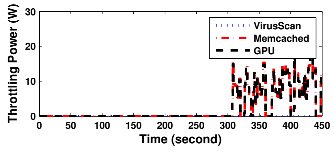
Figure 7: Shared Battery First is Better Option (MediaServer+MapReduce)



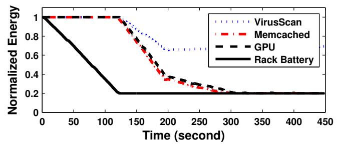
(a) Aggregate Rack Power

	Shared-First	vPower
VirusScan	0	0
GPU	8	0
Memcached	15	0

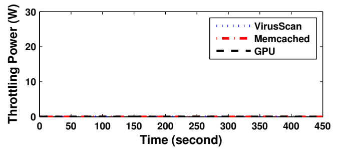
(b) Performance Degradation (%)



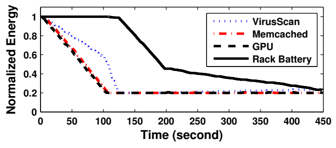
(c) Throttling (Shared First)



(d) SoC (Shared First)



(e) Throttling (vPower)

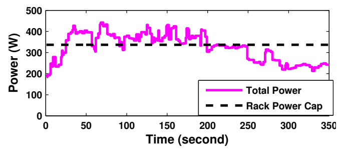


(f) SoC (vPower)

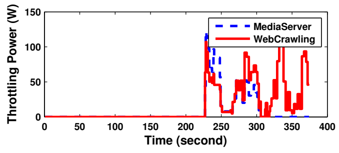
Figure 8: Local Battery First is better Option (Memcached+GPU+VirusScan)

Server (Local) Battery First being the better option: On the other hand, Figure 8 (a) shows the aggregate power of three applications (Memcached, GPU and VirusScan) running on a server each in the rack, specifying their power needs using "P90-60". We intentionally reduce the runtime of VirusScan to 200s and show results for a duration of 450s. Only a rack level power cap is set at 450W and we proportionally reduce the amount of power that can be drawn from the rack battery as in the previous experiment. vPower anticipates no local power violations, and also anticipates an earlier completion of VirusScan with the P90-60 specification. Based on its heuristics, it uses local batteries first to handle rack level power violations in this case, and in fact prioritizes the draw from VirusScan server's battery before it finishes (Figure 8 (f)), removing any necessity for throttling (Figure 8 (e)). However, obliviously using a shared battery first approach in this case mandates subsequently throttling Memcached and GPU, causing performance degradation of 15% and 8% in these applications, respectively.

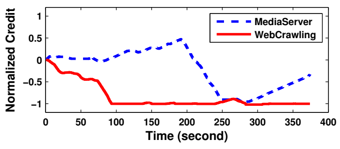
Fairness. We finally evaluate vPower's ability to enforce isolation between power draws of different applications for fairness. As noted earlier, there are pros and cons in the stringency of power demands made by an application. Asking for very stringent power may lessen their chances of being admitted, making them wait longer for power allocation. At the other end, even though grossly under-specifying the power may get them admitted, vPower will ensure that such applications do not mis-behave/mis-appropriate much more power in the runtime at the expense of others. The accounting mechanism in vPower, enforces a bound on the amount of credits that can be banked, as well as a bound on the credits that can be in debt. To illustrate these issues, we conduct an experiment with WebCrawling and MediaServer, each running on its own server. WebCrawling uses PMin=180W for its power specification (a significant under-statement), while MediaServer uses PAvg=180W. Rack power cap is set to 325W, with no individual server power caps. We show the benefits of vPower's accounting mechanism by comparing its execution with that for a scheme which uses DVFS and batteries as vPower, but without the accounting mechanisms as shown in Figure 9. Without accounting in place, power needs of WebCrawling are continuing to be met (there is no throttling for it for the first 225 seconds) through batteries, and both applications are being penalized subsequently (12-13% performance degradation). MediaServer is being penalized in this case for WebCrawling's fault, making it unfair. On the other hand, with our accounting mechanism in place, WebCrawling's credits deplete rapidly, while MediaServer saves/accumulates its credits. Consequently the penalization for WebCrawling (which is mis-behaving) steps in a lot sooner, and MediaServer is not affected at all.



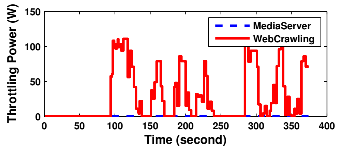
(a) Aggregate Power



(b) Throttling (vPower w/o Account.)



(c) vPower Account.



(d) Throttling (vPower w/ Account.)

Application Name	w/ Account.	w/o Account.
MediaServer	0	12
WebCrawling	28	13

(e) Performance Degradation (%)

Figure 9: Insulation from Power-hungry Applications (MediaServer+WebCrawling)

6. CONCLUDING REMARKS AND FUTURE WORK

With aggressive under-provisioning of the power infrastructure, this resource becomes as valuable as any other computing resource that needs to be appropriately rationed and managed between competing applications. Virtualizing this infrastructure gives the illusion of a potentially larger and insulated infrastructure for each application, voiding the need to expose its physical capacity and management to the applications. Until now, apportioning power has at best used modulation of other computing resources (demand-side knobs) - CPUs in particular through DVFS, scheduling, migration, etc. However, power distribution can also benefit tremendously from supply-side resources such as energy storage devices (batteries), at potentially multiple layers in the hierarchy. Hence, explicit virtualization of power hierarchies is essential, than just controlling power draws through demand-side knobs as has been the case until now. Towards this goal, this paper has presented the design and implementation of vPower, to create, allocate and manage virtual power hierarchies for co-existing datacenter applications.

vPower allows applications (not essential) to explicitly request their power needs at different resolutions using a palloc() interface. We have shown that such information can considerably help system performance, while still shielding individual applications from explicitly managing this resource, similar to the malloc() analogy. Specifying needs even as a high percentile (e.g., P90,) and not necessarily the exact maximum, and fairly coarse time resolutions (a minute or coarser at application phases such as Map/Reduce) seems to offer a good trade-off point. vPower uses these specifications in determining whether to admit them, and if so, where to place them. We have shown that ignoring the battery capacities, and conservatively allocating for the potential peaks to avoid any power emergencies, results in over 30% reduction in system utilization. This also highly limits the scale-out capabilities of applications such as memcached. Consequently, an aggressive admission control policy that places correlated workloads together - to offer more opportunities for batteries to recharge - is a better policy.

Despite an aggressive admission control policy, vPower's enforcer is able to effectively manage the power violations. We have shown that in our experimental rack of 8 servers, vPower shows 12-28% better performance than for a scheme that uses purely demand-side computing knobs, on a 50% under-provisioned power infrastructure. It also provides at least 50% higher throughput than the latter, for a memcached scale-out workload in this aggressively under-provisioned system. We have also demonstrated that the choice of batteries to draw upon during the runtime in a hierarchical 2-layer setting is very important. Based on a theoretical framework that identifies the conditions when this decision making becomes important, we have incorporated heuristics into vPower for battery sourcing. We have shown that vPower does much better than greedily using up local batteries or shared batteries first. Finally, we have demonstrated the effectiveness of vPower in fairly treating applications, by insulating the misbehavior of one application from the power needs of another. Our contributions are applicable regardless of where batteries are placed, the choice of energy storage technologies, and battery capacities. In fact, more stringent battery capacities make it even more important to manage hierarchies intelligently.

There are several interesting directions for future work. We have only been able to prototype a small 2-level hierarchy on our available experimental testbed, and we would like to conduct similar evaluations on a larger and deeper hierarchy, and examine a more hierarchical/distributed implementation of vPower for its scalability. Despite flexibility in our software to modulate the draw/charge rates of batteries, and possibly using those to source only a portion of the power exceeding the cap, our current hardware platform does not have those facilities. We are looking to build such capabilities in the future. Having shown the utility of a palloc() interface, even if it is at coarse resolutions, in managing the under-provisioned infrastructure, there is considerable scope for leveraging this interface. First, as described in [23], many current datacenter applications undergo extensive pre-production profiling on different hardware platforms, making them well-suited for exploiting this interface (as even a command line argument). Second, long-running services (e.g. mail, search, etc.) that may have predictability in load patterns (e.g. time-of-day behavior) could also use prior history for these purposes. Finally, a challenging area for future work is targeting cloud datacenters with sporadic jobs dynamically arriving, where pre-profiling may be difficult. While the infrastructure could assume average/worst case scenarios for such applications, one could envision a new set of runtime abstractions for power. Analogous to data structures that are explicitly (by the program) or implicitly (by the runtime system) allocated/deleted in (virtual) memory, we propose to investigate such "power" abstractions for this aggressively under-provisioned resource, to make palloc() a more natural feature, just as malloc() is in today's programs/runtime-systems.

7. ACKNOWLEDGMENTS

This work was supported, in part, by NSF grants 0811670, 1152479, 1205618, 1213052, 1147388, CAREER award 0953541, and a research award from Google.

8. REFERENCES

[1] F. Bellosa, A. Weibel, M. Waitz, and S. Kellner. Event-Driven Energy Accounting for Dynamic Thermal Management. In *Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2003.

[2] R. Bianchini and R. Rajamony. Power and Energy Management for Server Systems. *IEEE Computer*, 37(11), 2004.

[3] O. Bilgir, M. Martonosi, and Q. Wu. Exploring the Potential of CMP Core Count Management on Data Center Energy Savings. In *Workshop on Energy Efficient Design*, 2011.

[4] Q. Cao, D. Kassa, N. Pham, Y. Sarwar, and T. Abdelzaher. Virtual Battery: An Energy Reservation Abstraction for Embedded Sensor Networks. In *Proceedings of RTSS*, 2008.

[5] J. Chase, D. Anderson, P. Thakur, and A. Vahdat. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of SOSP*, 2001.

[6] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware Server Provisioning and Load Dispatching for Connection-intensive Internet Services. In *Proceedings of NSDI*, 2008.

[7] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam. Profiling, Prediction, and Capping of Power Consumption in Consolidated Environments. In *Proceedings of MASCOTS*, 2008.

[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud

Serving Systems with YCSB. In *Proceedings of SoCC*, 2010.

[9] G. Dhiman, G. Marchetti, and T. Rosing. vGreen: A System for Energy-Efficient Management of Virtual Machines. *ACM TODAES*, 16(1):6:1–6:27, 2010.

[10] Facebook Rack-level UPS for Improved Efficiency. `http://www.datacenterknowledge.com/archives/2011/04/07/`.

[11] X. Fan, W.-D. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-Sized Computer. In *Proceedings of ISCA*, 2007.

[12] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of ASPLOS*, 2012.

[13] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal Power Allocation in Server Farms. In *Proceedings of SIGMETRICS*, 2009.

[14] Google Server-level UPS for Improved Efficiency. `http://news.cnet.com/8301-1001_3-10209580-92.html`.

[15] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and Limitations of Tapping into Stored Energy For Datacenters. In *Proceedings of ISCA*, 2011.

[16] S. Govindan, D. Wang, L. Y. Chen, A. Sivasubramaniam, and B.Urgaonkar. Towards Realizing a Low Cost and Highly Available Datacenter Power Infrastructure. In *Workshop on HotPower*, 2011.

[17] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar. Leveraging Stored Energy for Handling Power Emergencies in Aggressively Provisioned Datacenters. In *Proceedings of ASPLOS*, 2012.

[18] J. Hamilton. Internet-scale Service Infrastructure Efficiency, ISCA Keynote, 2009.

[19] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of MICRO*, 2006.

[20] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. Bhattacharya. Virtual Machine Power Metering and Provisioning. In *Proceedings of SOCC*, 2010.

[21] V. Kontorinis, L. Zhang, B. Aksanli, J. Sampson, H.Homayoun, E. Pettis, T. Rosing, and D. Tullsen. Managing Distributed UPS Energy for Effective Power Capping in Data Centers. In *Proceedings of ISCA*, 2012.

[22] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power Management of Datacenter Workloads Using Per-Core Power Gating. *IEEE Computer Architecture Letters*, 8(2):48–51, 2009.

[23] J. Mars, L. Tang, and R. Hundt. Heterogeneity in Homogeneous Warehouse-Scale Computers: A Performance Opportunity. *IEEE Computer Architecture Letters*, 10(2):29–32, 2011.

[24] M. R. Marty and M. D. Hill. Virtual Hierarchies to Support Server Consolidation. In *Proceedings of ISCA*, 2007.

[25] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of ASPLOS*, 2009.

[26] Memslap: Load Testing and Benchmarking a Server, 2012. `http://docs.libmemcached.org/memslap.html/`.

[27] Microsoft Reveals its Specialty Servers, Racks, Apr. 2011. `http://www.datacenterknowledge.com/archives/`.

[28] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making Scheduling Cool: Temperature-Aware Workload Placement in Data Centers. In *Proceedings of USENIX*, 2005.

[29] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *Proceedings of SOSP*, 2007.

[30] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood. Power routing: Dynamic power provisioning in the data center. In *Proceedings of ASPLOS*, 2010.

[31] E. Pinheiro, R. Bianchini, E.Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Workshop on COLP*, 2001.

[32] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No Power Struggles: Coordinated Multi-level Power Management for the Data Center. In *Proceedings of ASPLOS*, 2008.

[33] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. Power Containers: An OS Facility for Fine-Grained Power and Energy Management on Multicore Servers. In *Proceedings of ASPLOS*, 2013.

[34] J. Stoess, C. Klee, S. Domthera, and F. Bellosa. Transparent, power-aware migration in virtualized systems. In *GI/ITG Fachgruppentreffen Betriebssysteme*, number 2007-23, pages 3–8, 2007.

[35] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In *Proceedings of USENIX*, 2007.

[36] A. Verma, G. Dasgupta, T. Kumar, N. Pradipta, and R. Kothari. Server Workload Analysis for Power Minimization Using Consolidation. In *Proceedings of USENIX*, 2009.

[37] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible Proportional-share Resource Management. In *Proceedings of OSDI*, 1994.

[38] D. Wang, C. Ren, and A. Sivasubramaniam. Virtualizing Power Distribution in the Datacenter. Technical Report CSE-13-004, The Pennsylvania State University, 2013.

[39] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy. Energy Storage in Datacenters: What, Where, and How Much? In *Proceedings of SIGMETRICS*, 2012.

[40] X. Wang and M. Chen. Cluster-level Feedback Power Control for Performance Optimization. In *Proceedings of HPCA*, 2008.

[41] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proceedings of ASPLOS*, 2002.