Mediating Power Struggles on a Shared Server

Iyswarya Narayanan, Anand Sivasubramaniam
The Pennsylvania State University
{iun106,anand}@cse.psu.edu

Abstract—**Most of today's servers, with numerous CPU cores and other plentiful direct resources, host co-located workloads using mechanisms to reduce hardware resource contention. However, power is an equally important indirect resource in a server that is shared between the co-located applications, for which they can contend, especially when power budgets are tight. We refer to this as a "power struggle". While there is a considerable amount of prior effort on server power capping, they are largely oblivious to power as an indirectly shared resource. This indirect resource exhibits a unique set of properties – spatial non-multiplexing, non-convex, fluidic, time-shifting and dynamic capacity – which have not been explicitly tackled so far. We propose policies that explicitly consider these properties to mediate power struggles, implement these on real hardware, and provide experimental results to show the performance benefits of our solution.**

I. Introduction

The power draw of a datacenter that houses thousands of servers has severe consequences on its cost, scalability and environmental footprint. Capping the power of a datacenter over different time scales is very important for its demand response capabilities [1–3]. Such capping can be used to sustain a higher compute capacity within a provisioned power infrastructure, re-shape the demand curve based on power availability, and help a datacenter participate in grid-scale demand response. The cap would be set by constraints specified by these goals, with the possibility of the cap itself changing dynamically.

A datacenter needs to maximize the utilization of every watt within the cap to extract maximum returns on its substantial capital investments - server, power and cooling infrastructure. There have been several efforts to tackle power struggles at datacenter scale [2–7], many of which [2–6] require recursively translating datacenter level power caps into caps at lower levels of the power distribution hierarchy - clusters, racks and individual servers. Equally, there have been proposals and implemented mechanisms to cap the power for each server, and/or each application. However, a server is increasingly hosting multiple workloads, with each application simultaneously exercising its different hardware components. Even if the applications do not contend for these direct hardware resources, they contend for the power allocated to this server, an indirect resource consumed by the direct resources. Hence, power is an equally, if not more, important shared resource (as CPU cores, memory, etc.) on a server that needs to be carefully managed amongst applications.

In this context, in contrast to (i) works such as [2–6, 8] which deal with contention for power across servers, this work addresses the contention within a shared server; (ii) works such as [2, 9] which only consider aggregate control, this work treats power as a shared resource and explicitly apportions power to colocated applications; (iii) works such as [10–12] which treat power as a direct resource, this work deals with power as an indirect resource and explicitly partitions an application's power budget to each of the direct resources.

Consolidation creates Resource Contention: In this work, we consider that a datacenter level power constraint is translated into a cap/budget for a server, $P_{cap}(t)$ for each instant t. The server itself hosts several co-located workloads, a growing trend in all kinds of datacenters. With increasing number of cores in each server, deeper degrees of server consolidation offers attractive cost benefits to the datacenter operator. However, co-locating workloads on a single server, even with bountiful cores, can create contention for hardware resources - caches, memory, interconnects, and storage. This has led to a considerable amount of work over the years on (i) identifying applications that can effectively use these resources without contention [13, 14], (ii) hardware and software mechanisms to isolate the resource usage of each application to insulate them from each other [10, 11, 15–19], and (iii) policy frameworks to allocate resources for each application [20–24]. However, as power becomes increasingly critical and scarce, it vital to explicitly manage power across co-located applications.

Power is also a Resource (but different!): The importance of power requires treating it on par with other critical resources in overall management. While there has been numerous research studies on reducing and managing power use, treating power as an indirectly shared resource for explicit allocation and isolation amongst coexisting applications in a server has been little explored. This is the gap that this work intends to fill. At best, a few prior studies have dealt with this problem at the server scale. For instance, Heracles [10] mitigates contention for direct resources, treating power as yet another direct resource of static capacity (thermal limit) managed using the frequency knob. However, as a resource, power presents unique challenges and opportunities making its management different from that for other resources:

- *Non-multiplexing in Space:* Consider two applications requiring very disparate resources, e.g. CPU and memory. They can co-exist on the same server without much direct resource interference - which we call multiplexing in space. On the other hand, two applications can consume completely disjoint "direct" resources (not contending for them), but each of these resources in turn consumes power, leading to a *power struggle* depending on the server level power budgets. Even two CPU intensive applications that are allocated their

respective CPU cores (without direct resource contention) can contend for power based on the server level power budgets. Such a struggle for the shared power capacity makes it even more important to manage it explicitly, and not as a consequence of managing other resources.

- *Indirect Resource and Non-convexity:* Unlike other hardware resources that are directly consumed, power is "indirectly" consumed by the resources consumed by the application. This mandates a coordinated control across different direct components consuming this common indirect resource. Further, the relationship between usage of other resources and power can be non-convex - the incremental usage of the server cores does not translate linearly to the corresponding incremental power [12, 25]. As the relative contribution of "non-core" hardware - DRAM, caches, accelerators, etc. - grows, this trend is accentuated [26–29].
- *Space Shifting/Fluidic:* Being an indirect resource can sometimes become a benefit. Direct resources are compartmentalized i.e. CPU capacity may not be shifted to increase memory capacity, even if under-utilized. In contrast, power is fluidic, allowing its consumption in one component to be seamlessly shifted to another based on application needs.
- *Time Shifting:* Energy storage devices (ESDs) are finding interesting usage in datacenter servers [30–32] for temporarily boosting power when supply is constrained, and banking energy when there is adequate power slack. With such devices, power availability/demand can be time-shifted, which is not possible with most other direct resources.

Contributions: Recognizing these unique opportunities and challenges, this work demonstrates the importance of explicit power management in shared servers.

- We show that traditional server power management strategies [9, 10, 12, 33] are not well-equipped to manage power as a shared resource as they ignore the unique nature of power.
- Using examples, we identify requirements for power management in shared servers: (i) capping not just the aggregate power draw, but partitioning power across individual applications, (ii) when partitioning an indirect resource, we also need to partition it further across direct resources, (iii) coordinating power draw in space and time across applications, and (iv) collectively leveraging energy storage when available to compensate for non-proportionality of power consumption.
- We build a framework to implement these requirements on an actual Linux platform. Using mixes of representative datacenter applications, we run several experiments on a private cloud setting with varying server level power budgets. Results show that treating power as an indirectly shared resource provides a 20% improvement in overall server throughput even for a relatively loose power cap, compared to state-of-the-art power allocation in today's servers [33]. When we move to a more stringent power cap, we get much more substantial benefits of 70% respectively. A space and time coordinated use of a Lead-Acid battery on the server gives a throughput boost of nearly 2×.
- Our cluster scale evaluations show that our approach successfully mediates power struggles during peak shaving events. It improves cluster power efficiency by 4% and 12% compared to server consolidation and RAPL [2] respectively.

II. Power Struggles

A server typically hosts multiple applications, which share its physical resources (cores, caches, etc.), with performance degradation arising from any contention in these resources. There has been a plethora of work [10, 16, 18, 19] proposing mechanisms to spatially and/or temporally partition these physical resources to mitigate the performance degradation. Despite the alleviation of direct resource contention, the very critical indirect resource - namely *power* - is also shared across co-located applications, which can lead to a "power struggle" and thereby performance degradation for applications.

A. Definition and Example

We define power struggle as the performance degradation of an application when run in conjunction with one or more applications on a server that has sufficient physical resources for each of them (to potentially reach the performance of running in isolation), but the overall power allocated to the server is not sufficient enough.

Regardless, of whether an application is running or not, a server expends a certain amount of idle power P_{idle} constituted by the leakage current in LLCs, DRAM self-refresh, fans, spinning disks, etc. For instance, on our Dell PowerEdge platform, P_{idle} is around 50 W. In addition to the 50 W of idle power, turning on a core to run an application also turns on certain "uncore" components (LLC, on-chip network, memory controller, QPI, etc.) which we refer to as chip-maintenance power P_{cm}, that is around 20 W on our server. If an application (whether A or B) is run in isolation on this server, the overall server power shoots up to 90 W which is $P_{idle} + P_{cm} + P_{dynamic}$. The remaining (90-50-20 = 20 W) is the actual dynamic power $P_{dynamic}$ expended in executing the application. If we run both A and B simultaneously, with each getting all the resources it had when it was run in isolation (i.e. separate cores, caches, etc.) to ensure minimal performance interference, the overall server power amounts to 50 (P_{idle}) + 20 (P_{cm}) + 20 ($P_{dynamic\_A}$) + 20 ($P_{dynamic\_B}$) = 110 W.

Now, let us consider a power-constrained datacenter where we want to cap this server's power by 10%, i.e., restrict it to operate under 99 W. To implement this cap, let us say we fairly apportioned this cap to each of A and B respectively, i.e., each is capped at 49.5 W. One possible way of attaining this reduction of 5.5 W (i.e., 110/2 - 49.5) for each of A and B, is by dropping the DVFS state of each core assigned to each of these applications (i.e., 2 GHz to 1.5 GHz). Note that, even in this power capped operation, A and B are not interfering in the physical resources. However, since each has to operate at a lower frequency to cooperatively live within the power cap, they each suffer a performance deterioration of 5% and 20% respectively. We refer to this as a power struggle.

B. Problem Formulation

In this work, we focus on mitigating power struggles in a power constrained server which has sufficient direct resources to simultaneously host more than one application. Toward this, we use a server architecture as depicted in Fig 1. There are two sockets with several cores each, their private L1 and L2 caches, a shared LLC for each, one memory controller on each socket which is each connected to DIMMs. The applications spatially multiplex the direct resources in a server typical of existing systems[10, 34, 35]. The server may also have energy storage devices for aiding power management [30, 36] - lithium-ion or lead-acid batteries. Apart from the power going to the two applications (dynamic power), the total power draw at any time goes for P_{idle}, P_{cm} and charging of ESD. All of this has to be sustained within P_{cap} at all times.

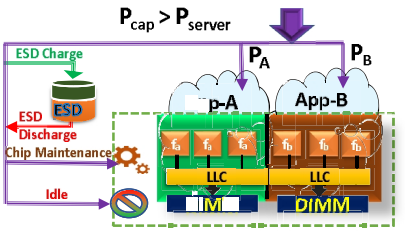


Fig. 1: Applications spatially multiplex the direct resources in the server. Server power draw goes towards shared hardware (P_{idle}, P_{cm}), applications (P_A, P_B), and the ESD.

Our goal is to mitigate power struggles between co-located applications on a server, i.e. make each application's performance close to its uncapped execution. We pose it as a single maximization objective with all applications weighed evenly:

$$\text{Maximize} \sum_{X \in \mathcal{A}} \frac{Perf_X(P_{cap}, \mathcal{A})}{Perf_X\_nocap} \quad (1)$$

where $Perf_X\_nocap$ is the performance of X on the consolidated server in the absence of power caps; $Perf_X(P_{cap}, \mathcal{A})$ is the performance of X when run in combination with other co-located applications in set $\mathcal{A} - X$ under P_{cap}. i.e. The total power draw of the server should stay within P_{cap} at all times:

$$P_{idle} + P_{cm} + \sum_{X \in \mathcal{A}} P_X + ESD_{charge} - ESD_{discharge} <= P_{cap} \quad (2)$$

where P_X is the power draw of each application in $\mathcal{A}$; ESD_{charge} and $ESD_{discharge}$ are the charge and discharge power of the ESD respectively.

Solving this problem is non-trivial because the search space to achieve the above objective (1) is very large:

First, there exists multiple applications on each server ($\mathcal{A}$). For each application X in $\mathcal{A}$, there exist several fine-grain power allocation knobs to manage power draw, such as:

- Per-core frequency scaling - Power can be shifted to/from individual cores of an application using per-core DVFS. f_X is the frequency setting of the cores of application: $f_X \in \{f_{min}(1.2\ GHz), \ldots, f_{max}(2\ GHz)\}$ of the hardware, and can be tuned in steps of 100 MHz.
- Core consolidation - An application can also modulate power use within its cores by power gating some of its cores and their private caches, with the rest of its cores taking on the additional work. This can be tuned depending on the maximum cores allocated to an application: $n_X \in \{n_{min}(1), .., n_{max}(6)\}$.
- Memory power allocation - Power can be allocated individually to each of the two DIMMs in our system (each connected to its own memory controller) using Intel's DRAM RAPL interface using an explicit power specification, i.e. $m_X \in \{m_{min}(3\ W), ..., m_{max}(10\ W)\}$, in units of 1 watt.

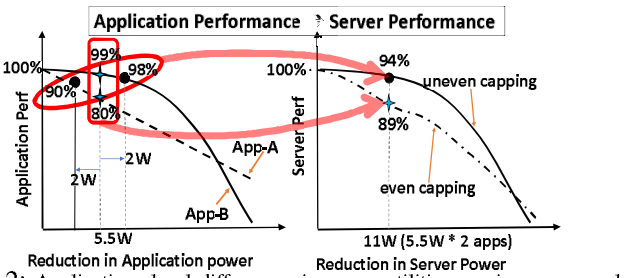


Fig. 2: Applications level differences in power utilities require an unequal apportioning of power.

Second, energy storage device [30, 36] is an additional knob for control. ESD has to be charged or discharged depending on its usefulness to the application, and it requires coordination of its charge power and discharge power to stay within the available power budget (P_{cap}).

Therefore, we propose to exploit the unique properties of power to develop heuristics to mediate power struggles.

C. How do we mediate Power Struggles?

Towards addressing power struggles, we use the example in Section II-A and analyze different scenarios.

Requirement R1 *As power is a shared resource, we not only cap overall server power draw, but also cap power for each application.*

Fig. 2 plots the loss in performance (y-axis) for both A and B normalized to the performance of uncapped operation, as a function of the cap on the x-axis. Note that the slopes/shapes of the two curves are different, with the additional complication of the slope itself varying for different power reductions. Let us impose a server cap of 10%, i.e., 99 watts, with the cap for each of A and B being 49.5 watts (for fairness). This results in a reduction of 5.5 watts from A and B. Note that this results in a much more serious performance penalty for A, compared to that for B (20% for A vs. 1% for B.) showing that even though we have been fair in power allocation, the consequence is unfair in terms of performance. However, the same overall 10% power cap (of 99 W) could have been achieved by reducing 3.5 watts from A, and 7.5 watts from B, where the consequent performance degradation for A and B would have been 90% and 98% respectively, which would have been a better choice from the system's perspective.

This implies that just because a server's power is capped at a certain limit, it should not necessarily translate into an even apportioning of this cap across different applications. Instead, we should take the relative utility of each watt at each power budget for each application (Fig. 2) in apportioning the cap.

Requirement R2 *Power is an indirect resource. Therefore, we need to spatially shift power across direct resources of an application* based on the needs of the application.

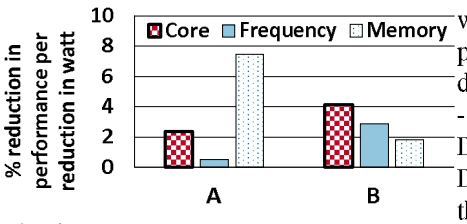


Fig. 3: Resource level differences in power utilities imply that partitioning an indirect resource requires partitioning it across the direct resources.

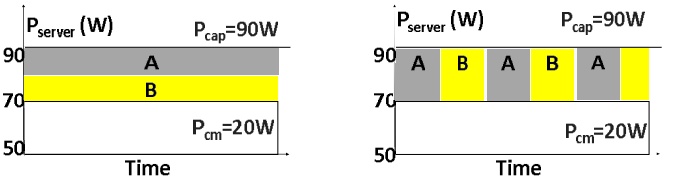
Consider Fig. 3 where the utility per watt for each direct resource - adding a core, DVFS of a core, DRAM - (rather than as a whole) for each application. A is more memory intensive than B, and consequently benefits substantially more in having additional watts devoted to memory. In fact this is even more beneficial than adding an extra core or techniques such as turbo-boosting. Even after deciding the watts to allocate for each application as per Requirement 1, we still have to decide how best to apportion these watts of that application amongst the different physical resources. This requirement is unique to power, due to its nature of being indirectly consumed by several physical resources.

Requirement R3 *Power cannot be multiplexed spatially. Therefore, we need to coordinate power draw across applications and their direct resources.*

Let us consider co-located execution of A and B which consumes 50 (P_{idle}) + 20 (P_{cm}) + 20 ($P_{dynamic\_A}$) + 20 ($P_{dynamic\_B}$) = 110 W. If the power capacity is dropped to 90 W, we only have flexibility with controlling the magnitude of $P_{dynamic}$ and when it is expended for each application, since we cannot play with P_{idle} and P_{cm}.

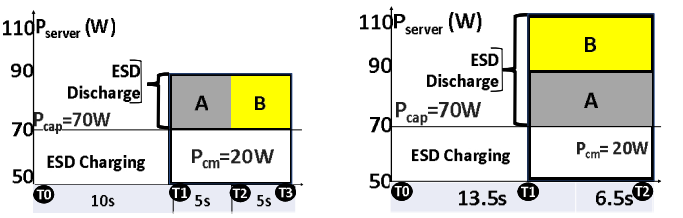
We could frequency scale both of them at the same time, so that each consumes 10 W of dynamic power to result in an overall server consumption of 50 + 20 + 10 + 10 = 90 W at any time as in Fig. 4a. We call this as *coordination in space*. Or, as in Fig. 4b, where with alternate duty cycling (i.e. one coming on when another is off and vice-versa), we can ensure that at any time only 50 + 20 + 20 = 90 W is consumed to stay within the cap. We call this as *coordination in time*. Note that there may be different performance consequences for these two choices. If the power cap becomes even more stringent, say 80 W, and let us say that even with the slowest DVFS state we can at best get to a minimum of 10 W of dynamic power for an application, the alternate duty cycling mechanism is the only alternative for adhering to this cap. This implies that we need to be aware of co-located applications in temporally multiplexing the power consumption in physical resources across different application in order to adhere to its allocation (in one case, it is always at 10 W and in the other case it alternates between 0 W and 20 W). This is despite adequate available of direct resources for the applications.

Requirement R4 *Power draw is non-convex due to P_{cm}. To address this non-convexity, applications can collectively leverage energy storage when available to time shift their power use to achieve better performance.*



(a) Coordination in space. (b) Coordination in time.

Fig. 4: Coordinating power use between applications.



(a) Alternate duty cycling. (b) Consolidated duty cycling.

Fig. 5: Addressing non-convexity of P_{cm} using ESD.

Realizing the importance of energy storage as an important power management knob, large datacenter operators are deploying ESDs locally within each server [30, 36]. Prior works [37–39] have shown the benefits of energy storage to cap and/or temporally shift the power needs. However, all of them have used this knob either for a single application, or for the server as a whole oblivious to individual applications. We note that co-located applications require an additional consideration when using ESD for power capping. Say the overall server power budget is very stringent at 70 watts. As observed in R3, this budget is insufficient to run even 1 application at a time (i.e., a minimum of 80 watts would be needed to do the alternate duty cycling). However, with energy storage, we can do the following as illustrated in Fig. 5. From T0 to T1, we do not run any application and keep the server idle. This gives us a P_{cap} (70 W) - P_{idle} (50 W) headroom, which we can use to charge the energy storage device to a capacity of 200 Joules.

Once we have a reasonable capacity in the storage device, we can decide to do one of the following: (a) As in Fig. 5a, we can run just one of A and B, alternating with each other, both at its full dynamic power needs of 20 watts each. Note that, even though the cap is still at 70 watts, the additional 20 watts is supplied by the charged energy storage device for an overall period of T3 - T1 = 10 s, such that individual applications run for 5 seconds each (where A runs from T1 to T2 and B runs from T2 to T3). (b) However, there is an even better alternative as shown in Fig. 5b, where rather than alternating between A and B, it is better to simultaneously run both since the overall useful work that can be sustained by the charged energy storage is even higher. Since P_{cm} is incurred only once, even when multiple applications are concurrently running, its energy gets amortized between the two applications when they run concurrently as opposed to alternating. As a result, the (b) alternative illustrated in Fig. 5b would allow the charged storage device to sustain 6.5 seconds of execution of both A and B while (a) illustrated in Fig. 5a would allow only 5 seconds of execution of both, i.e. a

30% increase in effectiveness because of consolidation-aware exploitation of energy storage.

In summary, to mediate power struggles on a shared server, we need to jointly addressing the above four requirements[1].

III. Implementing the Requirements

There are several challenges associated with implementing the requirements. First, the system has to apportion the available power budget across applications and the direct resources available to the application (R1 and R2), despite lack of a-priori knowledge of power utilities. Second, the system has to explore several design choices (R3a, R3b and R4) in order to coordinate application power draws to stay within the server's power budget. Third, the server and the applications are prone to dynamic changes, and the system has to adapt to these dynamic variations. We next discuss our approach to these problems.

A. What to allocate?

Recall our discussions in R1 (Sec. II-C), where we pointed out that the power utilities (the slopes in Fig. 2) are different for different applications, and are also different at different power levels. Hence, in order to achieve our objective, we need to systematically explore the power consumed by the application (P_X) and the corresponding performance ($Perf_X$), for different setting of the power allocation knobs f, n and m. However, the knowledge of P_X and $Perf_X$ may not be available a-priori, and it may not be possible to exhaustively measure all the settings of (f, n, m) for every application. So, we use an online sparse sampling strategy that measures power utilities only for minimal settings of (f, n, m), and estimate the rest by collaboratively learning from other applications.

Collaborative filtering is typically used in recommender systems to predict the preference of a user by learning from the preferences for many other users. Here, we would like to estimate the power utility of an application by using the utilities from previously seen applications. Colloborative filtering uses a matrix to capture power and performance of previously seen applications for different settings of the power allocation knobs. In this matrix, each row corresponds to an application, and each column corresponds to the power allocation knob setting, each value represents power and performance. To estimate power and performance of a new application, the system measures power and performance online for a few samples of (f, n, m) and estimates the rest by minimizing the estimation errors for the measured values using the matrix. We use the measured and inferred estimates to maximize the objective function 1. The output of this optimization provides power allocated to each application (P_X) and the knob setting to allocate power within the direct resources(f_X, n_X, m_X).

Implementation: App Utilities in Figure 6 shows our implementation. We populate the power matrix by measuring

[1]We consider private cloud infrastructure in this work. Nevertheless, all requirements are still applicable for public clouds. All requirements are applicable for latency-critical applications including R4 which requires cluster-level coordination such as using Blink [39].

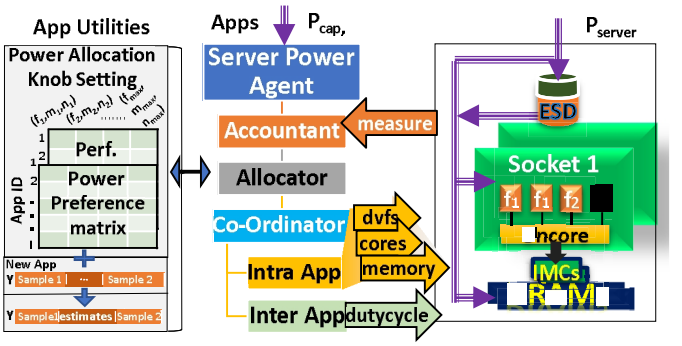


Fig. 6: System architecture: Applications' power utilities are learnt dynamically and is used to allocate available server power budget across applications. Coordinator spatially shifts power within each application and temporally coordinates across applications and ESD.

socket and DRAM power draw of an application using Intel RAPL interface [33, 40]. We populate the performance matrix by measuring the application performance using the open-source `heartbeats` [41] interface. This enables us to sample the application performance under different dynamic settings for the power allocation knobs. The collaborative learning framework (implemented in R) uses the sparse samples and the preference matrix, and estimates the power and performance for unknown settings. These serve as an input to the the `PowerAllocator` which partitions available power budget across each application and its direct resources.

B. How to coordinate?

After deciding how much power for each application (P_X) and how much power for each of its resources (f_X, n_X, m_X), we need to coordinate between these applications (R3 and R4). Towards that, we employ the following heuristics.

- *Coordinate in space to reduce power draw simultaneously (R3a) as shown in Fig. 4a:* This is our first option, since states of applications are preserved in their respective private caches (L1/L2).
- *Coordinate in time to alternate/duty-cycle (R3b):* If the power cap is too stringent, disallowing R3a, we resort to multiplexing in time across applications as in Fig. 4b. Here, each of P_A, P_B, ..., etc. will become 0 watts during its idle periods. However, the drawback is that some of the application's state in private caches would get flushed during those idle periods. This is the only other option if R3a cannot be employed, and energy storage is not available.
- *Coordination in space and time by leveraging energy storage (R4):* The energy storage knob can address the deficiency of R3b, and would be preferable whenever available. Further, it can also address the power non-proportionality of the P_{cm} as was discussed in the previous section. To utilize energy storage, power is coordinated in both time and space. During the charging period, power draw of the server becomes:

$$P_{idle} + P_{cm} + P_{dynamic} + ESD_{charge} \leq P_{cap} \quad (3)$$

with $P_{cm} + P_{dynamic} = 0$ as the applications coordinate to put the server to deep sleep state (PC6 state in Intel proces-

sors). During discharge period, the applications coordinate in space to keep the aggregate power draw of the server within the P_{cap} by utilizing the discharge power from ESD such:

$$P_{idle} + P_{cm} + P_{dynamic} - ESD_{discharge} \leq P_{cap} \quad (4)$$

Implementation: `Coordinator` refines the output from the `PowerAllocator`. It coordinates power use in space when all applications receive non-zero power budget. Otherwise, in the absence of ESD, only those with non-zero power can be run at that time. The rest of the applications sleep until it is their turn. We use alternate duty-cycling for fairness. However, when server has an energy storage device and the power budget disallows R3a, the coordinator computes the duty cycle period for the system (off: δ_1 to δ_2) and (on: δ_2 to δ_3) as:

$$\frac{(\delta_2 - \delta_1)}{(\delta_3 - \delta_2)} = \frac{(P_{idle} + P_{cm} + \sum P_X - P_{cap})}{\eta(P_{cap} - P_{idle})} \quad (5)$$

where, P_X is the power draw of the applications, and η is the ESD efficiency. The applications collectively charge the ESD during $\delta 1$ to $\delta 2$ and uses the stored energy to exceed P_{cap} during $\delta 2$ to $\delta 3$ respectively.

We enforce allocations using the following knobs in Linux-3.10. We use `taskset` to consolidate application cores (n) [42], and `cpupower` to set their frequencies (f), DRAM RAPL to set memory power (m), and task suspend/continue commands to coordinate the applications in time.

C. When to re-allocate/re-calibrate?

Until now, we have considered a steady-state condition for the server and the applications. But, the system is prone to dynamic changes. In order to adapt to these changes, the system reallocates power, and also re-calibrates of the power utility models (if necessary), upon the following events.

E1. Change in server power budget: Server P_{cap} can change dynamically depending on the datacenter level power budget. This triggers power re-allocation.

E2. Arrival of an application: When a new application is scheduled to run on the server, it triggers calibration of utility curves for this new application, as well as reallocation of power budget for all applications.

E3. Departure of an application: As an application finish execution and exits the server, it triggers power reallocation to apportion available power across remaining applications.

E4. Dynamic changes within application: The power needs of the application can change dynamically due to load variations or phase changes within an application. This leads to re-calibration of power utility curves for this application, and reallocation of power across applications.

Implementation: We use a `Accountant` for this purpose. It keeps track of the server power cap, scheduled applications, and the status of each application. We implement events E1 and E2 as explicit message to the `Accountant` specifying the change in P_{cap} or the arrival of a new application. The accountant periodically (in the order of microseconds) polls the status of the application and the server power draw. It triggers E3, if an application has finished execution. It triggers E4, if the power draw of an application changes significantly from its allocated power budget.

IV. Evaluation

Hardware platform: Our experimental setup consists of a dual socket Intel Xeon-2620 server shown in Table I. It allows independent control of frequency and sleep states at the core level, and coordinated control of socket sleep states. It exposes Intel RAPL interface for socket and DRAM power allocation. These knobs help us to spatially shift power. The server is equipped with Lead-Acid UPS for energy storage allowing temporal power shifting.

Processor	Xeon-2620
Cores	12
Freq.	1.2-2GHz
Freq. steps	9
LLC	15MB
Memory	8GB DDR3
NUMA	2 nodes
P_{idle}, P_{cm}	50W, 20W
$P_{dynamic}$	60W

TABLE I: Server Configurations.

Applications: We target datacenters requiring flexibility in modulating power draw based on supply dynamics (e.g. renewable power variations) and applications that are amenable to such modulations. So, we study applications from the following workloads, together with their dynamic arrivals and departures: data analytics (kmeans, APR [43]), graph analytics (BFS, triangle counting, connected components, shortest path [44]), search indexing (page rank [44]), memory streaming [45], media processing (X264, facesim, and ferret [46]). These benchmarks represent both compute and data intensive class of applications, with diverse requirements on the direct resources and their corresponding power draw. All these applications can be co-located on the server without exceeding its rated power. We randomly choose 15 pairs from the above applications as shown in Table II.

Mix	App1 (Type)	App2 (Type)
	Non-latency-critical co-locations	
1	STREAM (memory)	kmeans (analytics)
2	Connected (graph)	kmeans (analytics)
3	STREAM (memory)	BFS (graph)
4	facesim (media)	BFS (graph)
5	ferret (media)	Betweenness (graph)
6	ferret (media)	PageRank (search)
7	facesim (media)	Betweenness (graph)
8	X264 (media)	Triangle Count (graph)
9	APR (analytics)	Connected(graph)
10	PageRank (search)	kmeans (analytics)
11	ferret (media)	SSSP (graph)
12	facesim (media)	X264 (media)
13	APR (analytics)	kmeans (analytics)
14	X264 (media)	SSSP (graph)
15	APR (analytics)	X264 (media)

TABLE II: Application mixes.

Utility curves: We use the online collaborative filtering to dynamically build the utility curves. To find out how well this online approach works, in Fig. 7, we plot the consequence of doing this online at a specified sampling fraction (x-axis) by measuring the consequent power and performance (y-axis) with respect to that of an optimal strategy which exhaustively samples all settings. We use 5-fold cross validation (80% of the applications are used to estimate the metrics for 20%) to estimate the fraction of configurations to sample. This has been averaged across all the application mixes used in our evaluations. At low sampling rates, the error in power

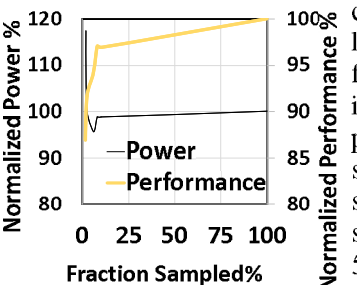


Fig. 7: Calibration of online sampling fraction.

estimation results in power over-shoot at the server, not adhering to the imposed cap. However, increasing the sampled fraction reduces error in power estimation, and consequently the server power draw stays within limit. We see similar trend in performance as well. Based on this, we fix the online sampling rate at 10% for subsequent evaluations. All the results include these sampling and re-allocation overheads.

Experiments: We next study the impact of power struggles under the following dynamic settings: (i) a server level power budget of 100 W which allows the applications to run simultaneously but requires spatial coordination, (ii) a dynamic drop in the power cap to 80 W which necessitates applications to coordinate in time even when they use different physical resources, (iii) dynamic application arrivals and departures, (iv) cluster scale impact of server power management. We will compare the benefits of each of the steps of our approach (given in Section II-C) with an utility unaware power capping.

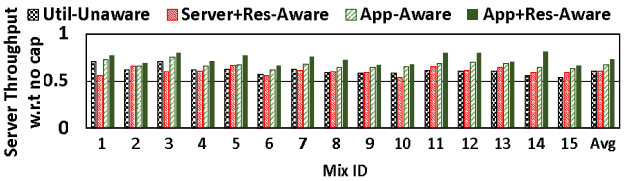
A. Requiring Spatial Coordination

Schemes: Under a $P_{cap} = 100\ W$, the two applications can run simultaneously on their respective hardware, but their power draw need to be coordinated spatially to live within the cap. For this case, we study the following options:

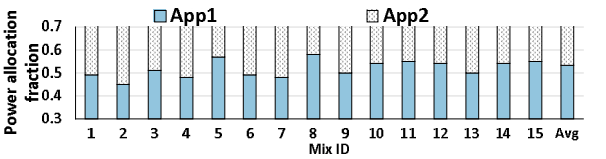
- **Util-Unaware (baseline-1)** is a fair power allocation policy. It is unaware of the power utilities and equally allocates the available power budget to all co-existing applications. We use RAPL [33] hardware knob to allocate power.
- **Server+Res-Aware (baseline-2)** is aware of power utilities of direct resources in a server, but is unaware of application-level differences. It uses the resource-level power utilities averaged across all applications.
- **App-Aware** is aware of the application level difference in power utility. It uses overall application power utilities to make its allocation, and does not tune it any further based on the direct resource utilities of individual applications.
- **App+Res-Aware** is aware of the resource level power utility for an application, as well as the benefits of apportioning the given power budget to its individual resources. So, it partitions power allocated to each application and recursively down to each of its physical resources.

Results: Fig. 8a presents the results for the workload mixes in terms of their throughput normalized with respect to uncapped execution for all policies. We also present the power allocated to the two individual applications and their respective speedups, for the App+Res-Aware in Figs. 8b and 8c.

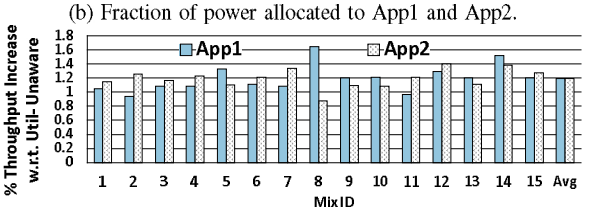
From these results, we observe the following: (i) the last two bars, in all cases, improve the overall server throughput by recognizing individual application needs. Even the App-Aware case, on an average, boosts the overall server throughput by 10% compared to both Util-Unaware and Server+Res-Aware. As opposed to a 50-50 split, the differences in power utilities across applications, warrants a 46%-54% split, on the average, across these applications. This clearly points out the deficiencies of much of the current state-of-the-art, which simply monitor and cap server level power consumption without



(a) Server performance for various application mixes.



(b) Fraction of power allocated to App1 and App2.

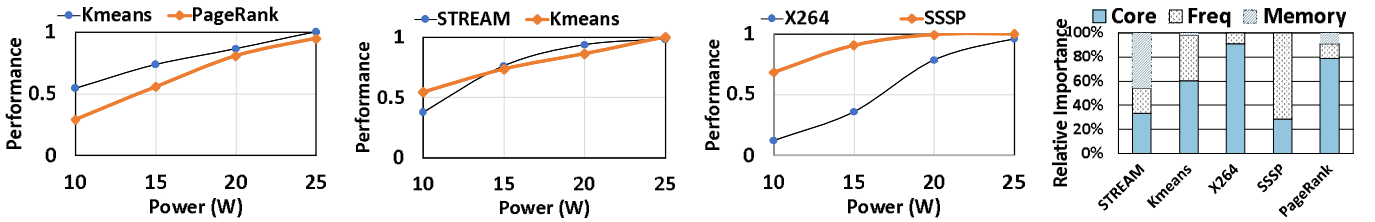


(c) Performance of App1 and App2.

Fig. 8: Power management at $P_{cap} = 100W$

looking into the details of co-located workloads. (ii) When we go from App-Aware to App+Res-Aware, the average server throughput increases by another 10%. This has very important consequences, i.e. it does not suffice to partition power based on utility (as done for many of the direct resources [10, 16]). It is even more important to apportion power amongst different direct resources. These results can be better understood by examining the performance of specific workload mixes:

- Mix-10 runs compute bound `PageRank` and `kmeans`. In this case, re-apportioning an application's power across the different hardware resources, does not make much sense - it is better allocated for CPU cores. However, these applications differ in their marginal benefits per watt, as is shown in Fig 9a. As a result, App-Aware shows better performance than the application unaware strategies, since it allocates 16.5 watts of the 100 watt cap (reduced by $P_{idle} + P_{cm}$) to `PageRank` and only 13 watts going to `kmeans` (see 55%-45% split for mix-10 in Fig 8b). Such a 55%-45% allocation of power between the two (compared to a 50%-50% split) results in a corresponding server throughput increase of 10% compared to the best baseline. This illustrates the need to carefully apportion the power between individual applications rather than just cap the overall server power draw.
- Mix-1 runs memory intensive `STREAM` and compute intensive `kmeans` applications. Fig. 9b shows the inter-app power utility for this mix. These applications do not differ much in their utilities when their power budgets are 15 watts each, making App-Aware not very different from the Util-Unaware baseline. Note that in Fig. 8b, the power allocations for the two applications in mix-1 are not very different. However, the Fig. 9d shows that they differ substantially in their resource level power utility. As a result, App+Res-Aware gives close to 5% and 15% performance gains respectively for `STREAM` and `kmeans`. It illustrates that while partitioning an indirect



(a) Mix-10. (b) Mix-1. (c) Mix-14. (d) Intra-app Resource.

Fig. 9: Power utility differences across applications and their hardware resources.

shared resource, it is important to partition it further across the direct resources of an applications to maximize its utility.

- Mix-14 is a co-location of `X264` and `SSSP`. Fig. 9c shows the inter-application power utility and Fig. 9d gives the intra-application resource level power utility. It shows that the applications differ in both application level and resource level utilities. App-Aware, which only cares about the former, performs a 55%-45% power split between the two applications (compared to the 50-50 split) to provide 10% performance gains at the server level. Further, apportioning power spatially to each hardware component based on its utility to each app using App+Res-Aware gives an even bigger boost.

B. Requiring Temporal Coordination

We now show the effectiveness of our approach with a more stringent server power cap of 80W. The power available for $P_{dynamic}$ reduces to 10W (=80W(P_{cap})-50W(P_{idle})-20W(P_{cm})). This power budget cannot simultaneously sustain both applications, since each needs a minimum of 10 W to run. To remain within the power cap the system has to control power allocation in time, i.e. duty cycle the applications. We consider the following schemes: (i) Util-Unaware: It is a fair power allocation policy which duty-cycles amongst the co-located applications in a fair manner (i.e. all get the same ON-OFF periods). (ii) Server+Res-Aware: It uses server level resource utility to allocate power, and duty-cycles in a fair manner. (iii) App+Res-Aware: Here we allocate power budgets to each application as in the previous subsection. To enforce these allocations, we appropriately set the ON-OFF periods for each application and allocate budgets for each hardware resource during the ON-periods. (iv) App+Res+ESD-Aware: The other schemes take turns on who should run at any time, ensuring that someone is always running. Here, either all applications run at the same time (amortizing P_{cm}), or none of them do (incurring no P_{cm}) - they all have simultaneous OFF-ON periods. However, since their peak draw during the ON-period would exceed the cap, this scheme uses the ESD to supplement the draw during the ON-period, which is banked during the previous OFF-period. These are tuned based on the storage characteristics (power/energy capacity, efficiency, etc.). We use Lead-Acid battery which gives us a 60-40 OFF-ON duty cycle to remain within the 80 W cap.

Results: Fig. 10 shows the overall server performance normalized to uncapped execution for all policies. Based on these results, we observe the following: (i) in all cases, the last

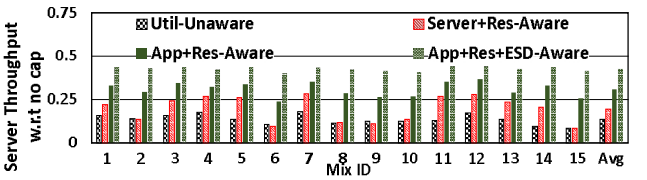


Fig. 10: Power management at $P_{cap} = 80W$.

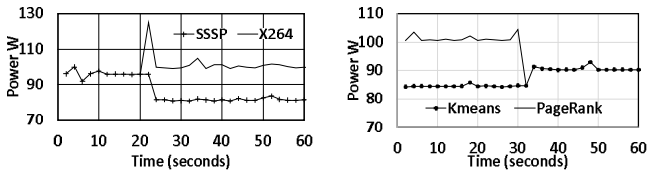
two bars which represent the consolidation aware strategies improve the overall server throughput. (ii) compared to Fig. 8a, the relative performance boosts are much higher under this stringent power cap. In other words, the more stringent the cap, the more important it is to do co-location aware power management. (iii) The new and important observation to note is that there is substantial rewards to be had by completely not scheduling any application during some periods compared to the other three where applications take turns running. Note that the server itself is NOT switched off, and it is only the sockets which are put into deep sleep state so that the wake up times are in 100s of microseconds [47].

C. Adapting to Dynamic Arrivals/Departures

Next, we also study the effectiveness of our mechanisms with dynamic application arrivals and departures.

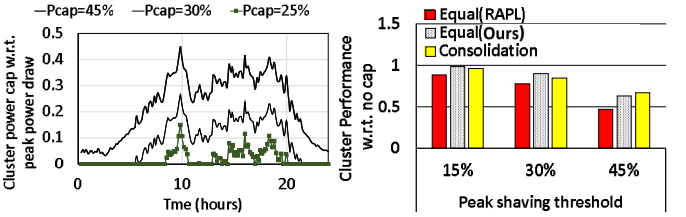
On an application's arrival: The applications can run without any power cap as long as the server power draw is within P_{cap}. We consider a setup where the server's P_{cap} remains the same and the arrival of a new application pushes the server above its power cap. We illustrate this using Mix-14: `SSSP` and `X264` for a P_{cap} = 100W in Fig 11a. Here, `SSSP` is the only application running for the first 20 seconds. At that point `X264` arrives, triggering power reallocation as shown in Fig. 11a. `SSSP`'s power is reduced from 25 W to 12 W, and `X264` is allocated 18 W. This reallocation also triggers resource level power apportioning for these applications (not shown in this figure). This results in `SSSP` retaining its current frequency (2 GHz). However, it consolidates its power use in cores (from 6 to 3). On the other hand, `X264` shifts its power use in frequency (2 GHz to 1.4 GHz) to other direct resources to achieve this power allocation. All of this is achieved within a span of 800 ms on our server.

On an application's departure: We next illustrate power reallocation as an application departs the system using Mix-10 which runs `kmeans` and `PageRank`. We consider the case when the applications are running under a power budget of



(a) Arrival (b) Departure

Fig. 11: Impact of application arrival/departure.



(a) Dynamic power caps. (b) Aggregate performance.

Fig. 12: Cluster level peak shaving

P_{cap}=100 W. Here, as explained earlier, power is allocated in the ratio of 45% and 55% to `kmeans` and `PageRank` respectively (see Fig. 11b). However, `PageRank` finishes execution and departs the system. Once the `Accountant` detects this, it triggers power reallocation. The `PowerAllocator` identifies that there is enough headroom for `kmeans` to increase its power draw, and it removes the cap on `kmeans`. This triggers resource level power reallocation enabling `kmeans` to activate more cores, and scale up all their frequencies.

D. Cluster Power Management

We next discuss cluster scale benefits of our approach. We consider the scenario when the cluster performs peak shaving [48]. Figure 12a shows the dynamic power caps of a cluster to shave 15%, 30% and 45% of peak power draw from a publicly available cluster power trace [49]. We replay these power caps on our prototype using 10 servers, and study the following cluster power management strategies:

- **Equal(RAPL):** The cluster manager equally apportions available power across all servers. Each server uses RAPL to stay within its power cap. This is the state-of-the-art in today's datacenters [2].
- **Equal(Ours):** The cluster manager evenly apportions available power across all servers. The servers use our proposed App+ Res+ESD-Aware power policy to manage power in each server. Note that the servers use the ESD only during periods of very stringent power cap.
- **Consolidation+Migration(no cap):** The cluster manager powers only as many servers as possible as allowed by the cluster level power budget. Hence, a power cap is not imposed on any active server. The cluster manager migrates applications to these servers considering direct resource interference. It is more efficient as it incurs less $P_{idle} + P_{cm}$. However, it may be not be feasible in the presence of large application states or network bottlenecks [1, 10].

Results: Figure 12b presents the aggregate performance of the cluster (y-axis) under different power management policies at different levels of cluster level power caps (x-axis). As can be seen, under all scenarios, enforcing cluster level power budget using RAPL results in only 47%-89% of the uncapped cluster performance. Instead, mediating power struggles using our proposed strategy provides 63%-99% performance. Note that this performance is equivalent or even better than the performance of server consolidation (by 3-5%) which powers only as many servers as allowed by the power budget without power capping any of them. Our proposal results in better energy proportionality, as it is able to extract higher performance per available watt by mediating server level power struggles. It further eases the complexity of cluster manager to adapt to dynamic power variations by capping individual servers as opposed to migrating jobs across servers. Note that under our proposed policy, the ESD is used only under very stringent power budget. It has negligible impact on the battery lifetime as this operating region is primarily impacted by the shelf-life of the Lead-Acid battery more than its cycle life [31].

V. Related Work

Many prior works have studied the challenges of resource management at various levels in the datacenter hierarchy.

Datacenter level: Prior works on datacenter resource management [50–57] have focused on server provisioning, application admittance, scheduling, etc. to manage variations in application demand and resource requirements. Orthogonal to direct resources, power capacity of datacenter may be limited due to under-provisioned infrastructure [58] or due to participation in power markets [59–61], on-site power generation [7, 48], or even due to power outages. While these works focus at the datacenter scale, their benefits depend closely on the effectiveness of server level power management.

Server level: Prior works have emphasized the importance of resource allocation and isolation in shared servers, and have developed mechanisms and policies [10, 15, 16, 18, 19, 62] to achieve this for direct resources. While there are some efforts on power allocation [5, 10, 12, 63], these ignore application and resource level differences in power utilities. Moreover, power is isolated primarily by limiting CPU time or frequency, while more recent hardware advances support finer grain of control (e.g. deep sleep of cores/sockets, per-core frequency scaling, DRAM power allocation, local energy storage). This enables us to actively manage power as an indirect shared resource using spatial and temporal shifting knobs.

VI. Concluding Remarks

This paper has studied the issue of power struggles in shared servers, and developed power management heuristics on shared servers. We show that the importance of rationing out power to individual applications, and to each of its physical resources, grows with the stringency of the power cap, and the difference in marginal utilities between applications, with energy storage playing a very important role in the effective management. This paper has opened doors to further research into this topic of indirect resource contention: (i) integration with cluster/datacenter level scheduling and job allocation mechanisms to individual servers; (ii) hardware mechanisms for fine-grained power isolation in these shared servers.

VII. Acknowledgements

This research was supported by National Science Foundation grants NSF-1714389, 1909004, 1629915, 1629129, 1526750, 1763681, 1912495 and a DARPA/SRC JUMP award.

References

[1] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," *Sustainable Computing: Informatics and Systems*, vol. 3, no. 3, pp. 183–193, 2013.

[2] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, "Dynamo: facebook's data center-wide power management system," in *ISCA*, pp. 469–480, 2016.

[3] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: Coordinated multi-level power management for the data center," in *ISCA*, pp. 48–59, 2008.

[4] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, "Power routing: dynamic power provisioning in the data center," in *ASPLOS*, pp. 231–242, 2010.

[5] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *USENIX ATC*, 2011.

[6] D. Wang, C. Ren, and A. Sivasubramaniam, "Virtualizing power distribution in datacenters," in *ISCA*, pp. 595–606, 2013.

[7] Í. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: managing datacenters powered by renewable energy," in *ASPLOS*, pp. 51–64, 2013.

[8] M. A. Islam, X. Ren, S. Ren, A. Wierman, and X. Wang, "A market approach for handling power emergencies in multi-tenant data center," in *HPCA*, pp. 432–443, 2016.

[9] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," in *ASPLOS*, 2016.

[10] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *ISCA*, 2015.

[11] H. Zhu and M. Erez, "Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems," in *ASPLOS*, pp. 33–47, 2016.

[12] 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 *ASPLOS*, 2013.

[13] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *MICRO*, pp. 248–259, 2011.

[14] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *ASPLOS*, pp. 77–88, 2013.

[15] Intel®, "Improving Real-Time Performance by Utilizing Cache Allocation Technology." http://www.intel.com/content/www/us/en/communications/cache-allocation-technology-white-paper.html, 2015.

[16] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, pp. 423–432, 2006.

[17] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multicore cache management," in *EuroSys*, pp. 89–102, 2009.

[18] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *MICRO*, pp. 374–385, 2011.

[19] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *MICRO*, pp. 62–75, 2015.

[20] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation in clouds," in *IEEE CLOUD*, pp. 418–425, 2010.

[21] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *NSDI*, 2011.

[22] S. M. Zahedi and B. C. Lee, "REF: Resource elasticity fairness with sharing incentives for multiprocessors," in *ASPLOS*, pp. 145–160, 2014.

[23] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *The journal of supercomputing*, vol. 54, no. 2, pp. 252–269, 2010.

[24] H. Wang and P. Varman, "Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation," in *FAST*, pp. 229–242, 2014.

[25] M. Zhu and K. Shen, "Energy discounted computing on multicore smartphones," in *USENIX ATC*, 2016.

[26] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," in *ASPLOS*, pp. 225–238, ACM, 2011.

[27] H.-Y. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson, and M. J. Irwin, "Core vs. uncore: The heart of darkness," in *DAC*, pp. 1–6, 2015.

[28] M. Arora, S. Manne, I. Paul, N. Jayasena, and D. M. Tullsen, "Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on CPU-GPU Integrated systems," in *HPCA*, pp. 366–377, 2015.

[29] J. Zhan, J. Ouyang, F. Ge, J. Zhao, and Y. Xie, "Hybrid drowsy SRAM and STT-RAM buffer designs for dark-silicon-aware NoC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016.

[30] Microsoft Server and Cloud Platform Team , "Microsoft reinvents datacenter power backup with new Open Compute Project specification." https://blogs.technet.microsoft.com/hybridcloud/2015/

03/10/microsoft-reinvents-datacenter-power-backup-with-new-open-compute-project-specification/, 2015.

[31] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy, "Energy storage in datacenters: what, where, and how much?," in *SIGMETRICS*, 2012.

[32] I. Narayanan, D. Wang, A.-A. Mamun, A. Sivasubramaniam, and H. K. Fathy, "Should we dual-purpose energy storage in datacenters for power backup and demand response?," in *HotPower*, 2014.

[33] Intel, "Intel® 64 and IA-32 Architectures Software Developers Manual Volume 3B: System Programming Guide, Part 2." http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf, 2016.

[34] "Xen 4.3 NUMA Aware Scheduling." https://wiki.xen.org/wiki/Xen_4.3_NUMA_Aware_Scheduling, 2015.

[35] "VMware", "The cpu scheduler in vmware vsphere 5.1." https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-vsphere-cpu-sched-performance-white-paper.pdf, 2013.

[36] CNET, "Google uncloaks once-secret server." https://www.cnet.com/news/google-uncloaks-once-secret-server-10209580/, 2009.

[37] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *ISCA*, pp. 341–351, 2011.

[38] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing, "Managing distributed UPS energy for effective power capping in data centers," in *ISCA*, pp. 488–499, 2012.

[39] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: Managing server clusters on intermittent power," in *ASPLOS*, 2011.

[40] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: memory power estimation and capping," in *ISLPED*, pp. 189–194, 2010.

[41] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats for software performance and health," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 347–348, 2010.

[42] R. M. Love, "taskset." https://linux.die.net/man/1/taskset, 2004.

[43] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A benchmark suite for data mining workloads," in *IISWC*, pp. 182–188, 2006.

[44] S. Beamer, K. Asanović, and D. Patterson, "The GAP benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[45] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[46] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, pp. 72–81, ACM, 2008.

[47] R. Schöne, D. Molka, and M. Werner, "Wake-up latencies for processor idle states on current x86 processors," *Computer Science-Research and Development*, vol. 30, no. 2, pp. 219–227, 2015.

[48] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad, "Opportunities and challenges for data center demand response," in *IGCC*, pp. 1–10, IEEE, 2014.

[49] 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 *NSDI*, 2008.

[50] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *SOSP*, pp. 103–116, 2001.

[51] L. A. Barroso, J. Clidaras, and U. Hölzle, *The datacenter as a computer: An Introduction to the design of warehouse-scale machines*, vol. 8. Morgan & Claypool Publishers, 2013.

[52] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *EuroSys*, p. 18, 2015.

[53] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *USENIX ATC*, pp. 499–510, 2015.

[54] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical scheduling for diverse datacenter workloads," in *SOCC*, p. 4, 2013.

[55] A. Kumar, I. Narayanan, T. Zhu, and A. Sivasubramaniam, "The fast and the frugal: Tail latency aware provisioning for coping with load variations," in *WWW*, 2020.

[56] I. Narayanan, A. Kansal, A. Sivasubramaniam, B. Urgaonkar, and S. Govindan, "Towards a leaner geo-distributed cloud infrastructure," in *HotCloud*, 2014.

[57] I. Narayanan, A. Kansal, and A. Sivasubramaniam, "Right-sizing geo-distributed data centers for availability and latency," in *ICDCS*, pp. 230–240, IEEE, 2017.

[58] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers," in *EuroSys*, pp. 317–330, 2009.

[59] H. Chen, M. C. Caramanis, and A. K. Coskun, "The data center as a grid load stabilizer," in *ASP-DAC*, pp. 105–112, 2014.

[60] H. Wang, J. Huang, X. Lin, and H. Mohsenian-Rad, "Exploring smart grid and data center interactions for electric power load balancing," *Performance Evaluation Review*, vol. 41, no. 3, pp. 89–94, 2014.

[61] I. Narayanan, D. Wang, A. Sivasubramaniam, H. K. Fathy, and S. James, "Evaluating energy storage for a multitude of uses in the datacenter," in *IISWC*, 2017.

[62] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *OSDI*, pp. 45–58, 1999.

[63] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *SOSP*, pp. 265–278, 2007.