Pocolo: Power Optimized Colocation in Power Constrained Environments

Iyswarya Narayanan, Adithya Kumar, Anand Sivasubramaniam
The Pennsylvania State University
{iun106,azk68,axs53}@psu.edu

***Abstract*—There is a considerable amount of prior effort on co-locating applications on datacenter servers for boosting resource utilization. However, we note that it is equally important to take power into consideration from the co-location viewpoint. Applications can still interfere on power in stringent power constrained infrastructures, despite no direct resource contention between the coexisting applications. This becomes particularly important with dynamic load variations, where even if the power capacity is tuned for the peak load of an application, co-locating another application with it during its off-period can lead to overshooting of the power capacity. Therefore, to extract maximum returns on datacenter infrastructure investments one needs to jointly handle power and server resources. We explore this problem in the context of a private-cloud cluster which is provisioned for a primary latency-critical application, but also admits secondary best-effort applications to improve utilization during off-peak periods. Our solution, Pocolo, draws on principles from economics to reason about resource demands in power constrained environments and provides answers to the when/where/what questions pertaining to co-location. We implement Pocolo on a Linux cluster to demonstrate its performance and cost benefits over a number of latency-sensitive and best-effort datacenter workloads.**

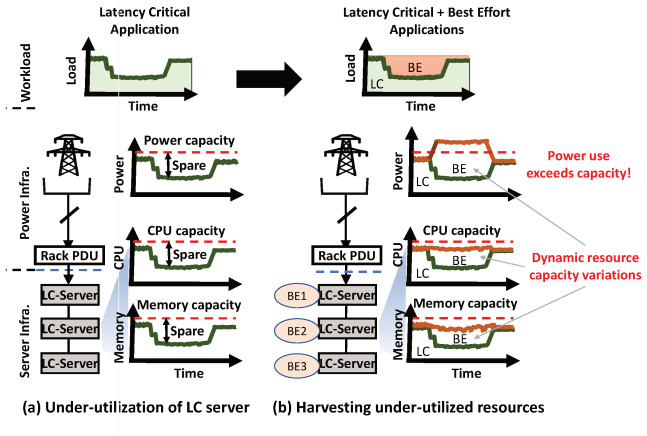


Fig. 1: Harvesting spare resources in power constrained clusters: (i) Naively admitting an application to utilize spare resources can push the server power draw to exceed provisioned power capacity. (ii) Spare resource capacity (including power) varies dynamically.

I. Introduction

Infrastructure costs are a significant expenditure in building and operating large datacenters, necessitating the need to extract the maximum value out of each provisioned resource. The whole rationale behind cloud computing, whether public or private, is to achieve this goal by sharing resources. This helps to boost utilization, providing us the opportunity to use resources on demand rather than dedicate them for individual users/applications who may not need the entire provisioned capacity all the time. Additionally, cloud operators co-locate applications even on a single server, based on their resource needs and Service-Level Objectives (SLOs), to push the utilization even further. There has been a tremendous amount of progress towards this goal in trying to build system to identify and co-locate applications, minimize their resource interference, and isolate their resource usage [1–6]. This becomes more important as applications vary dynamically in their resource needs, necessitating the marriage/co-location of only those applications that do not interfere in their resource needs, not just for the present but also in the future since dynamically moving applications across servers incurs high overheads. To date, much of the work in this space, has focused mainly on identifying what we term, "primary resources" such as CPUs, caches, memory, interconnects, and storage resources as points of contention from the co-location viewpoint [1–5, 7, 8]. On the other hand, another important resource namely "power" which is a "secondary resource" consumed as a consequence of consuming the primary resources, has not received as much attention from the contention viewpoint. As noted in several studies [9–12], the provisioned power infrastructure is as important as the compute/IT resources, and we need to equally extract the most from each provisioned watt. This paper shows that even if we avoid capacity contention for primary resources when co-locating applications, we can still run into power contention making power an important consideration for the where, what and when co-location questions.

We illustrate this problem in Fig. 1. Consider a user-facing latency critical application such as web-search running on a cluster of servers that is right-sized to meet its SLO at its peak load. Co-hosting other applications during this peak load period would lead to violations in SLO, and hence there is no co-location on any of the cluster servers during such periods. However, many of these latency critical applications go through dynamic variations, such as diurnal load behavior as is shown in Fig. 1(a). In order to better utilize the IT infrastructure, we co-locate some other background applications on the servers of this cluster during such off-peak periods. One could apply results from prior research studies such as [1, 3] to co-locate other applications with the web-search

application. For instance, we have two candidate applications RNN and Graph analytics, which be run as a background application along web-search without affecting the SLOs of web-search. As can be seen in Fig. 1, the aggregate server resource utilization (CPU and memory) of due to such co-location during this off-period is still at most the utilization during the peak period, validating the merits of prior studies proposing co-location choices.

However, while the utilization of all other resources is within the capacity of the infrastructure that is meant to handle the peak load, the power consumption of the co-located system overshoots the power of the peak load in the original web-search application as is shown in Fig. 1(b). This is because total server power (secondary resource) consumption is additive over the consumption of power by all primary resources. As is to be expected, co-location algorithms, which are usually based on performance-SLO or resource reservations, select applications that do not contend in their primary resource - CPU, caches, memory, storage, etc. - needs. With the co-located applications simultaneously using very different resources, the power consumption adds up, possibly overshooting what the original application was consuming during its peak period. As numerous prior studies have pointed out, the power infrastructure is itself very expensive, running into $10 to $20 per watt [10, 11, 13], and aggressive power provisioning to extract the maximum value from each watt is critical. If the power infrastructure in our example is tightly provisioned to only handle the peak load of the web-search which the cluster was originally intended for, then the co-location during the off-peak period will over-shoot this power budget despite adhering to the performance criteria that normal co-locations strive for. In such situations, power becomes the dominant resource for contention, prohibiting the system from fully utilizing other primary resources. This paper specifically focuses on this problem - when there is spare resource capacity, *which applications should we co-locate, not just to minimize interference due to primary resources, but also to minimize the power interference and provide the best performance in a power constrained environment?*

To answer this question, we need to jointly understand the resource and power needs of the potential co-runners in order to reason about application placement and resource management. There are several challenges that we have to address. The resource needs of the applications spans multiple resources – several primary (direct) resources and a secondary resource power (indirect). Further, we want to understand the resource needs not only for the present (in a myopic fashion), but also for future load fluctuations as well. While there are prior works that tackle these challenges, these (i) focus on a single operating condition and not the entire load range [1, 3] and rely on systems mechanisms (e.g. migration to a spare server) to cope with the outcomes of their myopic decisions; (ii) are not designed for platforms that are aggressively under-sized in their power capacity where the explicit goal is to maximize performance per watt as opposed to achieving it as a by-product of improving server utilization [6]; (iii) help to reason about fairness in allocating direct resources [7, 8] as opposed to reasoning about overall performance of co-located applications in the presence of a power constraint (an indirect resource).

Recognizing the unique needs of the power constrained platforms in improving their infrastructure efficiency, we undertake the efforts to systematically reason about resource preferences of an application in the presence of an indirect resource. Towards that, we make the following contributions.

- We borrow principles from the field of economics where *cost* is an indirect resource (similar to power in our case) that people use to buy one or more *resources* to maximize their *satisfaction*. There are well-known methods and metrics in economics to understand resource preferences in a cost budget constrained scenario to maximize user satisfaction. Drawing on these principles, we present a framework to reason about resource demands in power constrained servers that maximizes performance.
- The key insight from our analysis is that applications can be co-located with the primary latency sensitive application if they derive higher performance per watt from a different set of resources (other than the ones needed by the primary application). We use the notion of indirect utility function to formally capture the resource preferences of the applications under power constraints.
- Based on these insights, we present a framework, **Pocolo**. It uses indirect utility metrics to identify power-optimized co-locations and manage server resources power efficiently.
- We have implemented Pocolo on a Ubuntu 16.04 server using 2 direct resources (CPU and LLC ways) and the indirect power resource, and studied the impact of co-locating different best-effort applications (typically found in the datacenter) with several latency-critical foreground applications (websearch, image-dnn, sphinx and TPCC). Further, we have a cluster manager that uses these insights to assign the right application to run alongside the latency sensitive applications.
- Our results show that: (i) Even if we naively assign random applications to the same server running the latency sensitive application, our resource allocation strategy within that server provides 8% performance improvement with a power and energy reduction of 7% and 16% respectively, while adhering to the latency SLO for the latency critical application; (ii) Further, if we have a smarter cluster level scheduler, the performance is boosted even further to 18% with a power and energy reduction of 8% and 27% respectively. This translates to 16% reduction in overall datacenter Total Cost of Ownership (TCO) as we are able to extract better performance per watt from both capital server/power infrastructure as well as by reducing operating energy expenditure.

II. Background and motivation

A. Right-sizing infrastructure for a primary application

Datacenters incur significant capital expenditure to provision both servers as well as power infrastructure [10]. Therefore, minimizing their capital costs is of utmost importance to

the datacenters operators. Towards this goal, datacenters right-size their infrastructure based on the needs of the primary application in the cluster. By doing so, the datacenter operators achieve cost benefits by incorporating their knowledge of application characteristics, estimated resource needs, and demand projections into long-term capacity planning specific to the respective primary applications [9, 14–17]. In this work, we consider a datacenter comprising of multiple such clusters, where each cluster is carefully planned to host a primary latency-critical application (e.g. Search, Storage).

B. Harvesting spare resources for secondary application

Many of the latency-critical applications that run on the datacenter infrastructure are user-facing, and are prone to diurnal demand variations [6, 18–20]. Therefore, the infrastructure is under-utilized when the load is low. One way to address under-utilization due to diurnal variation is to consolidate the number of active servers during low loads. While this solution is beneficial for some clusters in reducing their energy use (operating expenditure), it is inefficient as it strands the provisioned server and power capacity (capital expenditure). Hence, to fully utilize the provisioned capital investment, datacenters increasingly admit multiple applications during periods of low loads [6, 21]. These applications are executed on a best-effort basis using spare resources available in the cluster, *without impacting the performance of the primary latency-critical application*.

The goal of such systems is to fully utilize the cluster resources. Note that, the goal is is not just to improve the capacity utilized, but also improve their utility (i.e. performance per unit cost). Therefore, even when the infrastructure is provisioned for only a single primary application, the cluster utility is now the aggregate throughput of all its applications which includes both latency critical primary application and best-effort (BE) secondary applications. Thus, it is important for us to revisit the resource allocation and management problem in this context of running both primary and best-effort applications in a cluster whose resources are provisioned only by planning for the primary application. In this context, there are two capacity constraints that one should take into consideration. They are:

- the right-sized power infrastructure capacity which constraints the maximum power draw of the server; and
- the dynamic variations in spare resource and power capacity in the server, driven by dynamic load variations in the primary application.

These capacity constraints directly impact the overall performance of the system, which we study next.

C. Challenges unique to this platform

Impact of right-sized power infrastructure: Private clusters achieve significant cost benefit by right-sizing their power capacity based on the needs of the primary application [9, 16]. This imposes a static limit on the server power capacity because the provisioned power capacity can be different for different applications. In this work, we consider four primary applications with server power provisioning needs ranging from 132 W to 180 W (more details in Section- V).

This right-sized power capacity has implications on the utility of the cluster when we admit a new application to utilize spare resources. Let us consider a server provisioned for xapian [22] (a leaf node/server in a distributed web search application) as its primary application. At its peak load, xapian consumes all the server resources (12 cores at 2.2 GHz frequency, all 20 LLC ways in a Xeon-2650 server) and a power of 132 watts to meet its performance SLO. However, at a low load of 10% of its peak, xapian requires only 1 core at a frequency of 2.2 GHz, 2 cache ways and a power of 64 W to meet its performance SLO. This results in the availability of spare resources namely, 11 cores, 18 cache ways, and a power budget of 70 watts to admit other best-effort applications. Fig. 2 studies the power draw of running different best-effort applications alongside this xapian application at 10% load. It shows that none of the applications keep the server within its provisioned power capacity as the power draw of the server now ranges between 138 watts to 155 watts, a 5% to 17% increase compared to the provisioned server power capacity of 132 W required for xapian at its peak load.

To stay within the power capacity limit, the server reduces power draw of the best-effort application. Fig. 3 shows the performance (y-axis) of the the different best-effort applications (x-axis) in the presence and absence of power capacity constraints. As we can see, all these applications have similar throughput in the absence of any power capacity constraints. However, when they are imposed with the 70 W power budget, their performance drops from 3% (LSTM and RNN) to 20% (Graph), even if they use same server resources (cores/LLC). Clearly, it is beneficial for the overall throughput of the server if we schedule either LSTM or RNN alongside a 10% load xapian workload when compared to scheduling either of Pbzip or Graph as the best-effort workload. This illustrates that when power becomes the constrained resource, it limits full utilization of all other resources in a server. Therefore, *in the presence of limited power capacity, the cluster also needs to consider power as a resource in placing applications.*

Impact of dynamic capacity limits: While the primary latency-critical application has absolute priority for any resource it needs, the best-effort application can only work with the spare/unused resources of the server. Since the load of the primary application also varies dynamically, the spare capacity also varies along with the load. Hence, from the perspective of the best-effort application, the available capacity of all the resources (including power) also vary dynamically. For example, if the load of the primary application increases, the server reclaims resources from the best-effort application as much as necessary to maintain its latency within the SLO. In our experimental platform, an increase in load from 50% to 80% for xapian can reduce the spare resources by 4 cores, 5 cache ways, and 10 W of power headroom. Therefore, the best-effort application should be able to benefit from the entire range of operating points of the primary application as opposed to a single operating point. For example, Fig. 4 presents the

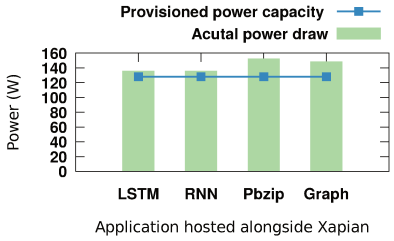


Fig. 2: Power draw of server exceeds its provisioned capacity on running other applications alongside xapian.

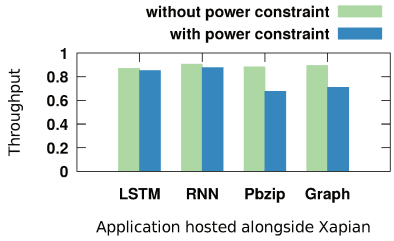


Fig. 3: Performance of the applications are the same without power constraint and different in its presence.

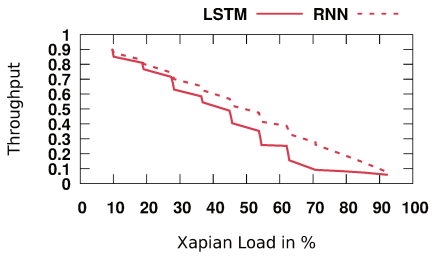


Fig. 4: RNN is a better candidate to run alongside Xapian if we consider its entire load spectrum.

throughput of the two best-effort applications (LSTM and RNN) under 10% to 90% of the peak load for xapian. It shows that RNN is able to derive better performance at all loads when compared to LSTM, whereas both these applications were suitable candidates when we looked at a fixed operating point of 10% load as illustrated previously in Fig. 3. Therefore, *the system should also account for dynamic capacity variations when placing best-effort applications alongside any primary application in addition to their power needs*.

In summary, to improve the utility of cluster infrastructure in resource harvesting platforms: (i) we need to account for both direct resources and power needs of the applications; (ii) we should consider not only the instantaneous performance benefits but also a range of operating conditions. We next show how to systematically navigate these challenges.

III. How to maximize the utility of a cluster under a power constraint?

Our high level goal is to match a set of best-effort applications to servers provisioned for latency-critical applications in such a way that effectively utilizes the spare server resources. To arrive at such placement decision we need to understand the performance of a best-effort application when matched with a latency-critical application.

Understanding intrinsic resource needs: Our intention is to characterize the intrinsic resource needs of the primary application as it directly determines the spare resource capacity available for the best-effort application. This is a complex problem because of the following reasons:

- The applications have resource needs in several dimensions: compute, memory, storage, and power. Therefore we need to explore a rich design space for resource allocations.
- Among all the resources, power is an indirect resource – as consumption of any other resource also results in power consumption. Therefore, we cannot independently study resource and power allocations.
- We want to capture the resource usage characteristics of the application not just for an instantaneous load, but for a dynamic operating range (e.g. diurnal load range).
- We want to reason about the resource needs not only for the primary application, but also for its co-runner.

While there are prior works on characterizing resource needs, they (i) ignore power needs of the direct resources [6]; (ii) focus for a single operating condition and not the entire load range [3]; (iii) use random search to explore the design space. While random search provides a feasible solution for server resource management, we also want to reason about placement decisions at cluster level.

To address this unique scenario, we borrow well-known techniques from the field of economics [23, 24] that can help us to *jointly* address all these aspects. We discuss them in detail below.

Understanding resource needs of the primary application using indifference curves: Indifference curves are used to understand the resource needs of a single agent (an application in our setting) to sustain a given load.

Fig. 5 presents the indifference curves for sphinx (a speech recognition) application. Here, the x-axis and the y-axis capture the allocation of two types of direct resources. i.e. cores and LLC cache ways respectively. The solid curves represent iso-load points ranging from 20% to 80% of the peak load of sphinx. The given allocation of cores (in x-axis) and cache ways (y-axis) can sustain the given load without violating its SLO. It shows that different combinations of cores and LLC cache ways can meet the target latency at a given load. i.e. the application can substitute cache ways with more cores or vice-versa. The application is indifferent to any of the allocations in the iso-load line as they all meet the given load while adhering to the latency SLO.

However, power is an indirect resource. It is consumed in all the direct resources. Therefore, the application expends power in adding cores and caches capacity. Among all the direct resource allocations, we are interested in the allocations that consume least power while meeting the performance SLO at a given load as it would correspond to higher spare capacity in terms of power for any other co-runner. We use the the dotted line to capture this. For example, if load of the latency-critical application increases from 20% to 40%, the least power hungry transition is to change the allocation from allocation-A to allocation-B. Therefore, in a power constrained scenario as ours, the server can reclaim more power headroom (in addition to spare cores and cache ways) by transitioning through these allocations as and when the load changes.

Implications on the spare capacity for the secondary

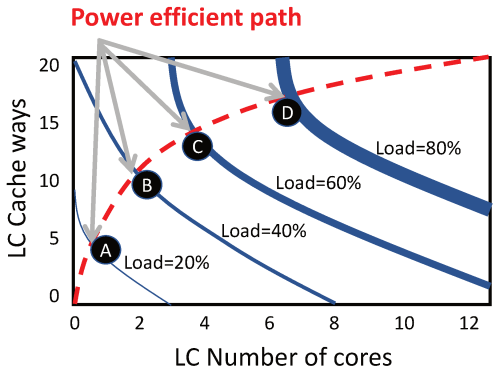


Fig. 5: An application is indifferent to any allocation of cores and caches that provides same performance (solid curves representing iso-load). In the presence of power constraints, it prefers the allocations that results in least power use (dotted curve that intersects with the solid curves).

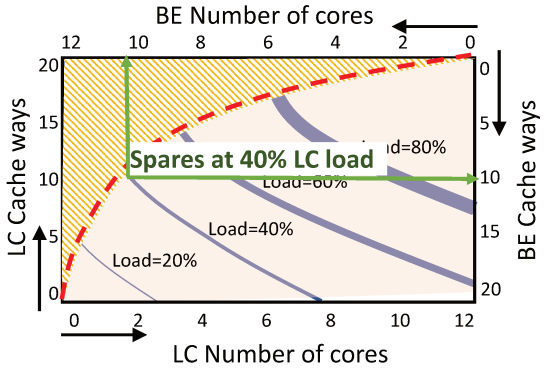


Fig. 6: The bottom-left and the top-right serve as the origin for the primary and the secondary application respectively. It shows the power-efficient allocation of two direct resources for the primary application and the corresponding spare resources which is used to run the secondary application. The dotted curve represents the power-efficient configurations for the latency-critical application. The striped region represents the feasible allocations for the best-effort application under a power-efficient operation of the latency-critical application.

application: The consequence of power-efficient allocation on the spare resources available for the secondary application is shown in Fig. 6. We use the Edgeworth-box [24] to understand the allocations of two types of resources between two applications. Here, the axes with the origin at the lower left corner represents the resource allocated to the primary application and the axes with its origin at the upper right corner of the box represent the spare resources allocated to the best-effort application. The primary and secondary axes are complements of each other. For example, at 20% load, primary application uses 1 cores and 5 cache ways. It corresponds to secondary application receiving of 11 cores (12 cores in the server - 1 cores used by the primary) and 15 cache ways (20 cache ways in the server - 5 cache ways used by the primary).

As the load increases for the primary application, it needs more cache ways than cores. Therefore, the secondary application can get up to 10 cores and 10 cache ways as the load of the primary drops to 20%. It implies that a best-effort application that is able to derive more performance per watt using compute cores would be able to make better use of all the available spare resources – plentiful cores, limited power, and limited cache ways in this example. Therefore, *to effectively utilize this server we have to place a best-effort application that derives higher performance per watt from a different set of resources (other than the ones needed by the primary application to derive higher performance per watt) (striped region in Fig. 6).* We use this key insight to improve cluster utility.

While the Edgeworth-box helps us to characterize this for two-types of resources, we can represent this more generally for more than two types of resources, and analytically reason about the demand for these resources.

Formalizing resource usage characteristics using indirect utility functions: Towards that, we use the concept of indirect utility functions to generalize this idea when using k-types of direct resources (cores, LLC cache ways, memory bandwidth, etc.) and a single indirect resource (power). Indirect utility functions are used in economics to study the demand of individual consumers [23] for different items (e.g. food) that maximizes their satisfaction within their cost budget. In this setting, we want to understand the relative demand of different direct resources in a server that maximizes the application performance (throughput under target latency) under power constraints. In a general form, performance of an application i is given by,

$$Performance_i = Utility_i(r_{i1}, .., r_{ik}, Power_i)$$

where, $(r_{i1}, ..., r_{ik})$ and $Power_i$ are the respective allocations of the direct resources and the power to an application i. The utility function can take many forms. In this work, we consider Cobb-Douglas indirect utility function. It takes the following form in the presence of an indirect resource:

$$Utility(r_{i1}, .., r_{ik}, Power_i) = \alpha_{i0} \prod_{j=1}^{k} r_{ij}^{\alpha_{ij}} \quad (1)$$

$$s.t. p_{static} + \sum_{j=1}^{k} r_{ij} p_{ij} \leq Power_i \quad (2)$$

Here, the parameters α_{ij} capture the relative impact of allocation of the direct resources on the performance of the application, and the parameters p_{ij} capture the relative impact of allocation of the direct resource on the power draw of the application. α_{i0} and p_{static} are constants. Prior works [8] have shown that Cobb-Douglas direct utility function is well-suited to capture application performance that require more than one type of direct resource for its execution (e.g. cores and caches). It also captures the resource indifference effect shown previously in figure 5 where several combinations of cores and cache allocations resulted in the same performance. As we will show later in evaluations, the Cobb-Douglas indirect utility function is well-suited to jointly capture the relative

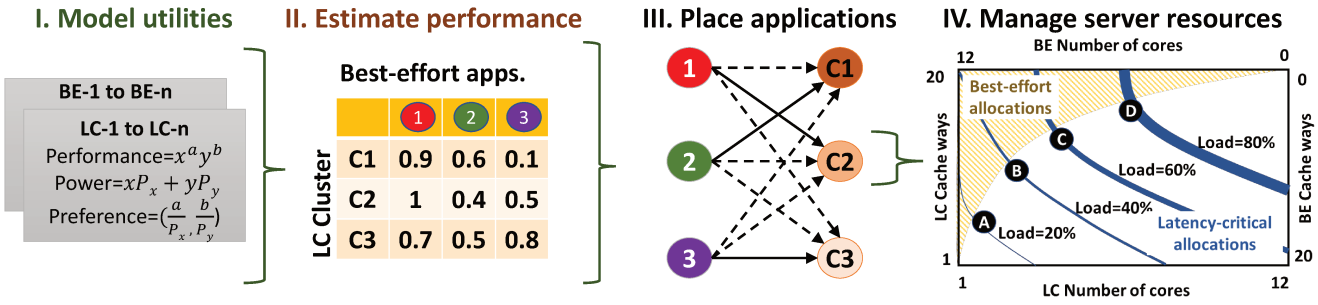


Fig. 7: I. Fit indirect utility model on profiled data. II. Use model parameters to estimate relative performance when placed alongside the latency-sensitive applications. III. Solve the matrix to identify a suitable a placement. IV. Use indirect utility model to manage server resources for the primary application in a power efficient manner.

impact of direct resources on an application's power draw and performance.

Given this indirect utility function, one can analytically derive the demand for the direct resources under any power budget that maximizes the application performance [23]. The demand for r_{ij} that maximizes the utility under a budget of $Power_i$ is given as $\frac{Power_i - P_{static}}{p_{ij}} \frac{\alpha_{ij}}{\sum_j \alpha_{ij}}$. Here, the factor $\frac{\alpha_{ij}}{p_{ij} \sum_j \alpha_{ij}}$ provides the relative preference of an application for the direct resource irrespective of its load or power budget. Higher value of this metric represents higher performance-per-watt from the resource, and therefore more need for the resource. This metric provides the relative demand for direct resources that operates the application in the most power-efficient way irrespective of the load, and help us quickly reason about the placement decisions for the entire load region of the application. For example, the $\frac{\alpha_{cores}}{p_{cores}}$ and $\frac{\alpha_{caches}}{p_{caches}}$ for the primary application of sphinx is 0.28 and 0.72 respectively (these metrics are scaled to add to 1). Similarly, they are 0.13 and 0.87 respectively for the secondary application of LSTM; 0.8 and 0.2 respectively for the secondary application of Graph. It indicates that Graph would derive better performance-per-watt using cores than caches and it is different from what sphinx requires for power efficient operation. Therefore, is better suited to run alongside sphinx across all loads compared to LSTM. We also observe this in our system evaluations in Section V. In summary, once we have the resource preferences for each best-effort and latency-critical application, we can co-locate applications that do not require the same set of resources for power-efficient operation.

IV. System Implementation

We next discuss the salient details of our prototype implementation shown in Fig. 7 which comprises of (i) modelling utilities, (ii) estimating performance using utility functions, (iii) cluster-level placement decisions and (iv) server-level management decisions.

A. Modelling utilities and capture resource preferences

We describe the process of deriving the utility model parameters for each application. This entails profiling the performance and power draw of the applications under different allocations of the direct resources and fitting the utility model to determine the resource preferences.

Profiling: We use samples of application performance and power under different settings of the allocation of the direct resources using fine grained resource allocation knobs in today's servers. We use the metric of maximum achievable application load (requests per second) within the target latency for the latency-critical application, and throughput as the performance metric for the best-effort applications. These performance metrics are commonly available on-line through application instrumentation (in private cloud setting as ours) and the power metrics are available on-line through server/socket power meters. Telemetry systems in today's datacenters periodically collected these metrics for each application at fine temporal granularity in private datacenters [25, 26].

Model fitting: We estimate the preference vector by fitting the power and performance profiles to the Cobb-Douglas indirect utility model. As an initial guard against model inaccuracies, we use samples where the tail latency of the primary application has at least 10% slack with respect to its SLO latency. The utility function is $performance = \alpha_0 \prod_{j=1}^{k} r_j^{\alpha_j}$ such that $P_{static} + \sum_j r_j p_j = Power$. We estimate α_j and p_j using linear regression. We transform the performance model in to linear form using log transformation. $log(performance) = log(\alpha_0) + \sum_{j=1}^{k} \alpha_j log(r_j)$ After which, we estimate the performance parameters (α_j) using least square method. Similarly, we estimate the power parameters also using least square method. When using the power measurements, we use application-level power meter [27] to apportion static/leakage power of the CPU and LLC ways.

We capture the resource preferences of application i for each resource (j) using α_{ij} and p_{ij}. The applications either provide their fitted parameters using historical knowledge or they are sampled online during execution.

B. Place applications using their resource preferences

The goal of the cluster manager is to identify a suitable latency-critical server for each of the best-effort applications that would benefit the application performance at all load conditions of the latency-critical application. The cluster manager populates a performance matrix to make placement decisions as shown in figure 7 (II). The performance matrix captures

the relative throughput of the best-effort application when placed alongside a latency-critical application. To populate this matrix, the cluster manager uses the Cobb-Douglas utility function using the derived co-efficient of α_j and p_j. It first estimates the spare resource capacity in a server hosting a latency-critical application using the Cobb-Douglas utility model solution that minimizes for power usage (e.g. allocation-A in Fig. 5) for the dynamic range of the LC application. Then, it translate the spare resource capacity to performance of the BE application using the Cobb-Douglas utility function using the co-efficient of the BE applications. The cluster manager's goal is to identify an assignment that maximizes the overall performance. There are standard methods to solve this [28–30] (e.g. Linear Programming, Hungarian method, randomization). We use a LP solver to identify an assignment that maximizes the overall cluster performance (similar to LP solvers used in [31, 32]).

C. Manage server resources in a power-efficient manner

The server manager is responsible for allocation and isolation of resources between the primary and secondary application. In our prototype, we consider allocation and isolation of two important direct resources (cores along with their private L1/L2 caches and LLC cache ways) and an indirect resource of power. This can easily be extended to other resources.

Primary application: The server manager monitors the load and the slack in 99th percentile tail latency of the primary application over a time window of every second and makes allocation decision. Upon a significant increase or decrease in latency slack, it quickly changes the allocation configuration to the power-efficient configuration for the current load. This is done trivially using the analytical solution for the Cobb-Douglas utility function [23]. This is a constant time operation (less than a millisecond). It provides the number of cores and caches that meets the given load at least power consumption. Note that real systems are prone to load uncertainties and model inaccuracies. We borrow control techniques from prior works by using the tail latency of the application as feedback to fine tune the allocations (including core frequency) to maintain a latency slack of at least 10% with respect to the latency SLO.

Secondary application: The spare resources that are not allocated/reserved for the latency-critical applications are allocated to the best-effort application. The server manager periodically measures the power draw of the server using every 100 ms, and throttles the power draw of the secondary application to stay within the provisioned power capacity. Towards that, it first uses the fine-grained knob of per-core frequency to reduce power draw, and then limits the CPU execution time to further reduce power draw if needed.

V. Evaluation

A. Experimental Setup

Server: We evaluate all our experiments on an Intel Xeon E5-2650 platform. The configuration of this server is presented in Table I. This server is equipped with Intel's socket and DRAM power meter. It allows core assignment using Linux's taskset command, and LLC cache ways allocation using Intel's Cache Allocation Technology, and per-core frequency scaling configurable using Linux's cpupowerutils. We disable the deep-sleep state for all the cores-allocated to the primary application. We run all the experiments with turbo-boost disabled. All the applications are run on Ubuntu 16.04.

Property	Configuration
Processor	Intel Xeon E5-2650
Cores	12 cores
Frequency	1.2 GHz to 2.2 GHz
LLC capacity	30M, 20 ways
Memory	256GB DDR4
Storage	480GB SSD
Power	Idle:50 W, Active:135 W

TABLE I: Server configuration

Primary Applications: Our cluster comprises of 4 servers, each running a latency-critical application. They are,

- **img-dnn [22]** is image inference service that uses deep neural network. It is a representative of image recognition applications that are commonly used in online image search, optical character recognition, etc. It operates on MNIST data.
- **sphinx [22]** represents speech recognition applications commonly used in voice assistant services such as Siri, Cortanta, etc. It performs speaker independent continuous-speech recognition using hidden markov models. It operates on CMU's AN4 speech data.
- **xapian [22]** is a web search engine. It represents a leaf node in a distributed web search service like Google search or Bing. It operates on index built from English Wikipedia.
- **TPC-C [33]** is an on-line transaction processing application. It is a representative of applications with persistent storage needs. We use MySQL database as the backend.

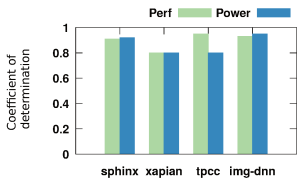
Table II presents the peak load, tail latency, and peak power needs of these applications on a Xeon-2650 server platform (see Table I). e.g. sphinx server requires 182 W of provisioned power whereas img-dnn server requires only 133 W.

Best-effort applications: We consider four best-effort applications from three important domains.

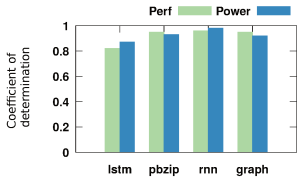
- **Deep learning training [34]:** Many services use deep learning models. The training phase of these models are throughput oriented, and can be done on a best-effort basis. We consider the following representative applications from Keras: a Long Short-Term Memory (LSTM) model for IMDB sentiment classification and a Recurrent Neural Network(RNN) model that learns to perform addition.
- **Graph analytics [35]:** Graphs are an important class of data analytics applications in large private clusters (e.g. Web Search, Social Networking). We consider PageRank on Twitter data set as a representative application.
- **Compression [36]:** Datacenters spend a significant time in compressing data for a wide variety of applications [37]. We use pbzip2 as a representative application.

Application	img-dnn	Sphinx	Xapian	TPC-C
Domain	Image search	Speech recognition	Web search	Persistent database
95th percentile latency	10 ms	1.8 s	2.588 ms	51 ms
99th percentile latency	20 ms	3.03 s	4.020 ms	707 ms
Peak server load	3500 requests/s	10 requests/s	4000 requests/s	8000 requests/s
Peak server power	133 W	182 W	154 W	133 W

TABLE II: Latency critical applications: Server-level characteristics.

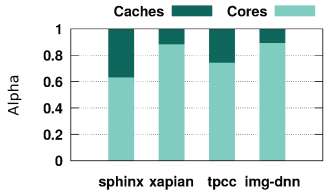


(a) Latency critical applications

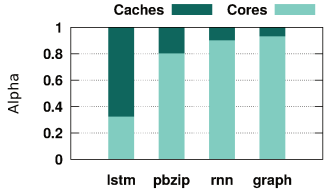


(b) Best effort applications

Fig. 8: Goodness of fit.

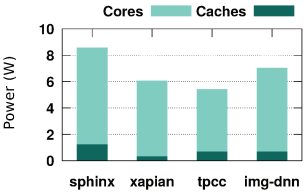


(a) Latency critical applications

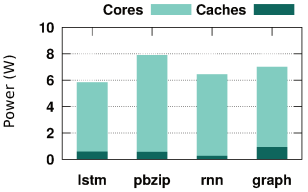


(b) Best effort applications

Fig. 9: Direct utility.

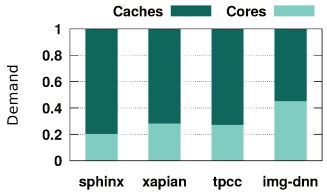


(a) Latency critical applications

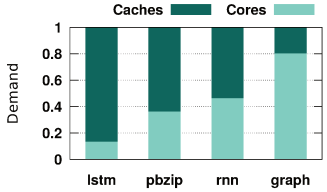


(b) Best effort applications

Fig. 10: Power needs.



(a) Latency critical applications



(b) Best effort applications

Fig. 11: Indirect utilities.

B. Evaluation organization

We first show the resource and power needs of the primary LC application and the secondary BE applications using the fitted model parameters. Then, we evaluate the overall power and performance benefits of jointly tackling server and power needs of the applications. We study the co-location decision under POColo compared to an exhaustive placement search. Finally, we present the total cost of ownership analysis.

C. Application characterization

We next present application power and performance characteristics using the Cobb-Douglas indirect utility model.

First, we study the goodness-of-fit for the parameters related to direct resource (α_j) and power (p_j) under different allocations of direct resources (r_j) (where j is cores and LLC cache ways) computed using linear-regression. We use the coefficient-of-determination (R-squared) to capture this, where the value closer to 1 represents an ideal fit. Fig. 8a and 8b shows the metric for both latency-critical (under 20% latency slack) and best-effort applications respectively. All applications have R-squared between 0.8 to 0.95 for performance and 0.8 to 0.98 for power, indicating a good fit.

Using the relative preferences derived based on the Cobb-Douglas utility functions, we illustrate the differences in placement decisions depending on whether power needs are taken into account. Let us consider Sphinx, which is a LC application. If we do not account for power, it prefers more cores over cache ways in the ratio of $\alpha_{cores} : \alpha_{caches}$ as 0.6:0.4 in Fig.-9a. Correspondingly, we could place LSTM from the BE candidates as it has complementary preferences compared to Sphinx ($\alpha_{cores} : \alpha_{caches}$ as 0.32:0.68 in Fig.-10b). However, if we include power needs too, both Sphinx and LSTM prefer more cache ways over cores ($\frac{\alpha_{cores}}{p_{cores}} : \frac{\alpha_{caches}}{p_{caches}}$ as 0.2:0.8) as show in Figs-11a and 11b. Under power constraints, Graph is a better candidate to run alongside Sphinx as it has complementary resource preferences ($\frac{\alpha_{cores}}{p_{cores}} : \frac{\alpha_{caches}}{p_{caches}}$ as 0.80:20).

D. Impact on server performance

We next study the benefits of two components of our solution that incorporate power awareness (server management and placement) compared to the baseline policy.

- **Random co-location (Random):** The cluster manager randomly assigns the best-effort application to any available latency-critical server. The server uses state-of-the-art feedback-based controller (Heracles [6]) to allocate and isolate server resources to stay within the target latency for the primary application. It allocates cores and caches from any one of the feasible allocations in the indifference curve. i.e. resources are not differentiated by their power use. Best-effort application use the rest of the un-allocated spare resources. Its power use is throttled to stay within the server's provisioned power capacity. This serves as our baseline.
- **Power Optimized Management (POM):** This is the first component of our proposal where the feedback-based server manager operates the primary application within the target latency using Cobb-Douglas indirect utility model guided search for power efficient allocations. The secondary application receives any resource not used by the primary. The placement decisions are power unaware.
- **Power Optimized Co-location and server management (POColo):** It incorporates both placement and management components of our proposal. The cluster manager places

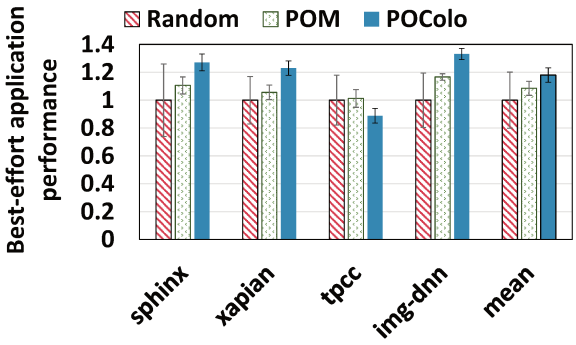


Fig. 12: Throughput of the best-effort applications placed alongside the latency-critical application shown in the x-axis. Higher is better. Pocolo provides higher overall throughput.

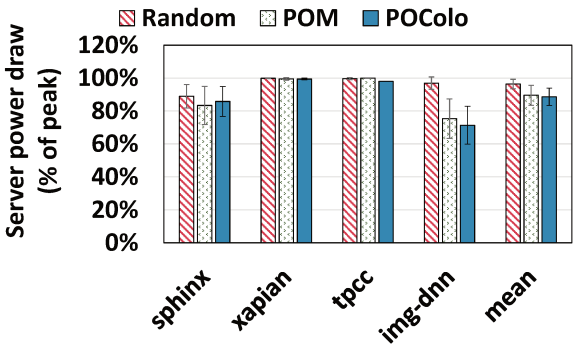


Fig. 13: Power draw of the latency-critical server normalized to its provisioned peak power capacity. Lower is better.

application by estimating the performance of the applications using their preference for the direct resources and power ($\frac{\alpha_j}{p_j \sum_j \alpha_j}$). The server manager operates the primary application within its target latency. It uses the utility model and latency-feedback to guide resource allocation.

Fig. 12 present the throughput of the secondary best-effort application (y-axis) while maintaining a latency slack of at least 10% for the primary latency-critical application. Here, the x-axis presents the primary application, and the y-axis presents the throughput for its co-runner averaged across the primary load (under a uniform load distribution from 10% to 90% in steps of 10%). The bars represent different policies for application placement and management. It shows that even when we naively admit an application, managing resources using POM to efficiently utilize power for the primary automatically increases average throughput by 8%. Further, by jointly incorporating resource and power needs of the application at the placement time, Pocolo achieves an 18% improvement in average throughput for best-effort applications. Note that POColo optimizes for overall cluster throughput (i.e. the sum of aggregate throughput of all servers), and is not designed to consider fairness. Therefore, in order to maximize overall performance of the cluster, it allows poorer performance for some co-locations (e.g. co-runner of TPCC) while most effectively matching other co-locations.

We next present the resulting power utilization of the servers. Fig. 13 presents the primary application in x-axis, and the server's average power utilization under co-location in y-axis. As we can, the power utilization under the random policy is almost always high with an average of 96% (of the peak capacity) across the four latency-critical servers in our cluster. This results in frequent power capping. In contrast, average power utilization for both POM and PoColo is only around 88%, an 8% reduction. It shows that the POM and Pocolo achieves 8% and 18% higher throughput respectively, while simultaneously reducing the cluster's power draw by 8%. It reduces the need to throttle server power draw by design. This clearly shows the benefits of incorporating indirect resource utility to reduce power use.

E. Co-location decision and resource preferences

To understand the co-location choices of POColo, we compare its decision against an exhaustive placement search that considers across all 4x4 combinations in our experimental platform. In Figs. 14, the x-axis presents the % of the peak load of the latency-critical application. The y-axis presents the average throughput of the server (total throughput of latency-critical and best-effort application). Pocolo chooses to assign Graph to sphinx server. As we can see, Graph is able to utilize the server resources at all loads of the Sphinx. Also note that, the resource preference of graph is 0.8 and 0.2 for cores and caches, whereas it is 0.2 and 0.8 for sphinx. Therefore, these applications do not require the same resources to operate in a power efficient manner and are able to derive better performance. Similarly, LSTM is matched to img-dnn, whereas RNN/Pbzib are matched to Xapian or TPCC with equal probability. These placement decisions can be similarly be explained for other applications.

F. Impact on datacenter TCO

We next present the cost benefits using a publicly available total cost of ownership model [13]. The details of the cluster are as follows: 100000 servers where each server costs $1450, provisioning power infrastructure costs $9/W, energy usage costs 7 cents per KWhr and power usage efficiency (PUE) of 1.1. We present the TCO of the different resource management strategies studied previously in Section-V-D to provide a constant amount of throughput. In addition to these policies, we also consider the TCO needs of the baseline (Random policy) in the absence of power under-provisioning, referred to as Random(NoCap). Here, each server is provisioned with 185 W (max power needs of all primary application).

We present the amortized monthly costs of the datacenter infrastructure in Fig. 15. It shows Pocolo results in 12%, 16% and 8% lower TCO compared to Random(NoCap), Random and POM respectively. Random(NoCap) does not aggressively under-provision its power infrastructure. Therefore, it incurs the highest power infrastructure costs. In contrast, Random is aggressively under-provisioned in its power capacity and poorly uses available power capacity. Therefore, it incurs the higher server provisioning costs to compensate for that. Pocolo incurs lowest server, power and energy costs. Thus, it adheres

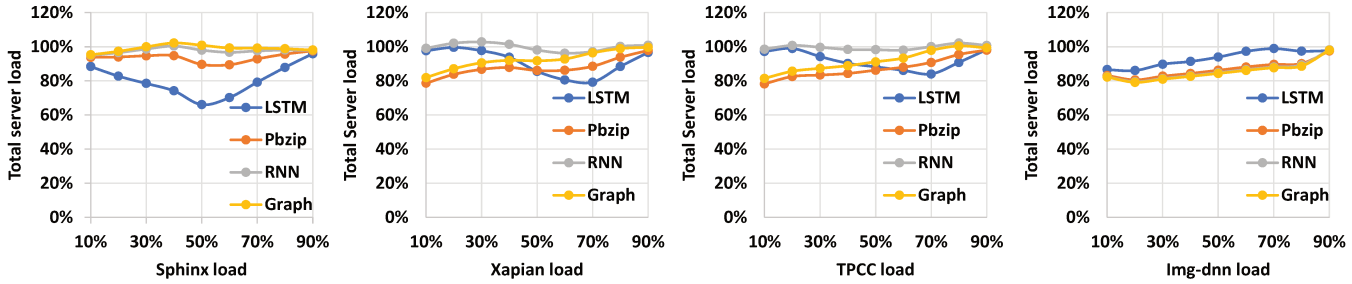


Fig. 14: Total server load under Pocolo for all placement combinations. Pocolo assigns Graph to Sphinx, LSTM to img-dnn, and RNN or Pbzip alongside either Xapian or TPCC as these placements improve overall throughput.

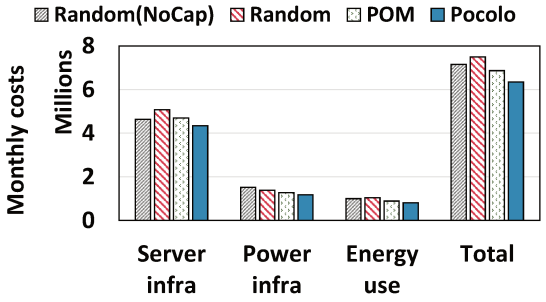


Fig. 15: Amortized monthly TCO.

to the goal of improving the cluster utility (performance per cost) by optimizing both capital expenses of server and power infrastructure as well as the operating expense of energy use.

G. Scope and limitations

Relevance to resource types: Our solution can be applied for resources that can be substituted within an application as shown using indifference curves (e.g. memory bandwidth, network bandwidth and storage read bandwidth). Moreover, this solution expects the resources preferences of the applications to be convex. Otherwise, the allocations will be inefficient.

Relevance to admitting multiple best-effort applications: We analyze only one best-effort application that fully utilizes spare server resources. If there are more than one best-effort application, they can be scheduled to time-share the server (e.g.first-come first-served, shortest job first). Spatial sharing would entail further partitioning of direct resources and power, which we intend to explore as future work.

VI. Related Work

Datacenter infrastructure: Improving datacenter infrastructure efficiency is an important area of research. Prior efforts in this area spans capacity planning for server [14, 18] and power infrastructure [9, 16, 38] as well as operational support to manage server [1–3, 6, 21] and power usage [12, 25, 39–41]. These works do not consider an important and emerging trend where a power constrained cluster hosting latency-critical applications tries to improve its resource efficiency by re-purposing its spare resources. This work undertakes this challenge by building upon advances in cluster management and server management.

Admittance control and job placement: Cluster-level management entails admittance and placement of applications towards a system objective such as meeting the performance SLO, resource reservations, or even to balance power utilization. These efforts include placing applications to efficiently meet their resource reservations [42–44] or avoid contention for shared direct resources [1, 3]. Similarly, there are efforts on cluster management to cope with variations in power availability [12, 45], utilization [16, 46, 47] and to improve power efficiency [48]. These are well-suited to cope with applications' resource requirements in terms of server resources (CPU, LLC) or power, and not both. The scenario tackled in this work requires the cluster manager to jointly consider the resource requirements of the applications in terms of both server resources as well as power.

Server resource management: Resource managers in the server are responsible for partitioning and isolating server resources between co-hosted applications. Towards that, prior works have proposed mechanism to isolate shared resources [49, 50], and policies [6–8, 27, 51–53] to apportion these resources. Of these, resource elasticity fairness [8] is closely related to our work. It aims to achieve fair partitioning of two direct resources (caches and memory bandwidth) in the public cloud. In this work, we focus on partitioning two direct resources (CPU cores and caches) and an indirect resource (power) in a private cloud in order to maximize performance per watt (as opposed to fairness).

VII. Conclusion and Future Work

We show that when power becomes a scarce resource, it prohibits effective utilization of the capacity in other resources (e.g. CPU, caches). We conduct an exhaustive treatment of this subject in the context of a private cluster, which is provisioned for a latency-critical application, but also admits best-effort applications during off-peak periods to improve resource utilization. Our analysis shows that the power optimized co-location can reduce both capital and operating expenses of the datacenters by effectively utilizing both server resources and power. These benefits stem from application level differences in the performance and power draw of the direct resources in the server. While we have addressed this problem specific to a private cluster designed for primary latency-critical application, this is an equally important problem for all kinds of datacenters. We intend take the insights from this work to guide other settings of datacenter operations.

VIII. Acknowledgements

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

References

[1] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 77–88.

[2] ——, "Quasar: Resource-efficient and qos-aware cluster management," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[3] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 607–618.

[4] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the International Symposium on Cloud Computing*, 2011.

[5] Q. Llull, S. Fan, S. M. Zahedi, and B. C. Lee, "Cooper: Task colocation with cooperative games," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 421–432.

[6] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 450–462.

[7] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the Conference on Networked Systems Design and Implementation (NSDI)*, 2011.

[8] S. M. Zahedi and B. C. Lee, "Ref: Resource elasticity fairness with sharing incentives for multiprocessors," in *ACM SIGPLAN Notices*, vol. 49, no. 4. ACM, 2014, pp. 145–160.

[9] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical profiling-based techniques for effective power provisioning in data centers," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2009.

[10] L. A. Barroso, U. Hölzle, and P. Ranganathan, "The datacenter as a computer: Designing warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 13, no. 3, pp. i–189, 2018.

[11] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 75–86, 2012.

[12] Í. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: Managing datacenters powered by renewable energy," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1. ACM, 2013, pp. 51–64.

[13] J. Hamilton, "Overall data center costs," https://perspectives.mvdirona.com/2010/09/overall-data-center-costs/.

[14] I. Goiri, K. Le, J. Guitart, J. Torres, and R. Bianchini, "Intelligent placement of datacenters for internet services," in *2011 31st International Conference on Distributed Computing Systems*. IEEE, 2011, pp. 131–142.

[15] I. Narayanan, A. Kansal, A. Sivasubramaniam, B. Urgaonkar, and S. Govindan, "Towards a leaner geo-distributed cloud infrastructure," in *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2014.

[16] D. Wang, C. Ren, and A. Sivasubramaniam, "Virtualizing power distribution in datacenters," in *Proceedings of the Annual International Symposium on Computer Architecture*, 2013, pp. 595–606.

[17] I. Narayanan, A. Kansal, and A. Sivasubramaniam, "Right-sizing geo-distributed data centers for availability and latency," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[18] 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 the Conference on Networked Systems Design and Implementation (NSDI)*, 2008.

[19] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the International Symposium on Cloud Computing*, 2012.

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

[21] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, Í. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[22] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2016.

[23] A. Mas-Colell, M. D. Whinston, J. R. Green *et al.*, *Microeconomic theory*. Oxford university press New York, 1995, vol. 1.

[24] J. M. Levy, *Essential microeconomics for public policy analysis*. ABC-CLIO, 1995.

[25] 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 *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[26] J. Lee, C. Kim, K. Lin, L. Cheng, R. Govindaraju, and

J. Kim, "Wsmeter: A performance evaluation methodology for google's production warehouse-scale computers," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 549–563.

[27] 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 the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[28] J. Matousek and B. Gärtner, *Understanding and using linear programming*. Springer Science & Business Media, 2007.

[29] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, "Matching is as easy as matrix inversion," *Combinatorica*, vol. 7, no. 1, pp. 105–113, 1987.

[30] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[31] L. Suresh, J. Loff, N. Narodytska, L. Ryzhyk, M. Sagiv, and B. Oki, "Synthesizing cluster management code for distributed systems," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019.

[32] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *Proceedings of the International Symposium on Cloud Computing*, 2011.

[33] T. Council, "TPC-C benchmark, revision 5.11," 2010.

[34] A. Gulli and S. Pal, *Deep Learning with Keras*. Packt Publishing Ltd, 2017.

[35] 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 the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 37–48.

[36] Jeff Gilchrist, "pbzip2: parallel bzip2 file compression," https://linux.die.net/man/1/pbzip2.

[37] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 158–169, 2016.

[38] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2007.

[39] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *ACM SIGOPS operating systems review*, vol. 35, no. 5, pp. 103–116, 2001.

[40] 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 the International Symposium on Computer Architecture (ISCA)*, 2008, pp. 48–59.

[41] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *Proceedings of the Annual Technical Conference (USENIX ATC)*, 2011.

[42] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 455–466.

[43] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[44] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[45] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*. IEEE, 2007, pp. 5–5.

[46] C.-H. Hsu, Q. Deng, J. Mars, and L. Tang, "Smoothoperator: reducing power fragmentation and improving power utilization in large-scale datacenters," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 535–548.

[47] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, "Power routing: dynamic power provisioning in the data center," in *ACM Sigplan Notices*, vol. 45, no. 3. ACM, 2010, pp. 231–242.

[48] D. Wong, "Peak efficiency aware scheduling for highly energy proportional servers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[49] 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.

[50] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multicore cache management," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2009, pp. 89–102.

[51] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2006, pp. 423–432.

[52] 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 *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011.

[53] I. Narayanan and A. Sivasubramaniam, "Mediating power struggles on a shared server," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.