Unleash Stranded Power in Data Centers with Rack Packing

Abstract

Data center infrastructures are highly underutilized on average. Typically, a data center manager computes the number of servers his facility can host by dividing the total power capacity of each rack by an assigned "peak power rating" for each server. However, this scheme suffers from the weakness of all static provisioning schemes – it does not account for the variability of load on the servers. We propose an algorithm that studies the power consumption behavior of the servers over time and suggests optimal ways to combine them in racks to maximize rack power utilization. The server placement problem is a version of vector bin packing [2], and our solution – RackPacker – approximates a near-optimal solution efficiently using a number of domain-specific optimizations. One of the central insights we use is that the different servers hosting a single application typically show strongly correlated, but often somewhat time-shifted, power consumption behavior. Hence, we find servers that show anti-correlated, or strongly time-shifted behavior and pack them together to maximize rack utilization. Our initial experiments with RackPacker show substantially superior results than static packing.

1 Introduction

The explosion of on-line services we are witnessing today is the result of a paradigm shift – a move to the Internet as a computing platform. Most commonly used applications – email, document editors, collaboration and organization tools, media, and games – are now offered as services. On-line service providers maintain data centers with tens – or even hundreds – of thousands of servers in order to host these applications. Modern data center infrastructure, excluding the IT equipment they host, can cost hundreds of millions of dollars. A majority of this cost can be attributed to the electrical and mechanical infrastructure, which distributes power and cooling to servers, storage, and network devices. Designing data centers to maximally utilize their capacities is therefore a crucial architectural concern for the growth of the so-called "Cloud Computing" paradigm.

The capacity of a data center is defined in many dimensions: power, cooling, space, water, network bandwidth, etc. Running out of resources in any of these dimensions means that the service provider needs to build or rent another data center to facilitate business growth. Among these resources, power is usually the first to be exhausted because of the load limitation on the power grid and the increasing power density of computing[1] . However, recent studies [9, 6] have found that the average data center's power resources are highly underutilized.

In this paper, we look at ways to optimize the power utilization in data centers by addressing the following question: *How many servers can a facility with a given power capacity host?* In common practice, this number is arrived at by dividing the provisioned power capacity by the power rating of each server. This rating might either be the nameplate rating of the server (which is usually a substantial over-estimate), or – which is slightly better – the server's experimentally measured peak power consumption. However, both these schemes suffer from the weakness of all static provisioning solutions – they do not account for the variability of load on the servers and the resulting dynamics of their power consumption.

We propose an algorithm that studies the power consumption behavior of the servers over time, and suggests optimal ways to combine them in racks to maximize power utilization. At the heart of such a dynamic provisioning scheme is the following intuition: the actual power consumption of each server is not always (and often very rarely) equal to its peak; hence, by intelligent *over-subscription* of the provisioned power, we can unleash the stranded power to host more servers. In other words, if we employed this scheme to populate our facility, we would exceed its power capacity if all of the servers were running at peak load; however, since the probability of such an event is vanishingly small, we are (with very high probability) fine.

[1]defined as the amount of power consumed by a rack of servers occupying a unit space (e.g. square foot)

Our solution takes advantage of two technology trends in data center computing: 1) *virtualization*: the use of virtual machines (VM) to consolidate services and ease software migration; and 2) *power capping*: the ability to adjust the power state of a server to prevent it from exceeding a given power cap. With VMs, it is easy to move services among physical servers, so that "matching" servers can be placed together to reduce the probability of exceeding a power budget. With power capping, the rare events of exceeding power limits can be mitigated by reduction in performance. Although we still aim at minimizing the power capping probability, reaching or temporarily exceeding power capacity will not cause catastrophic failures.

With these assumptions, our algorithm – RackPacker – solves what we term the *server placement* problem: *Given actual power consumption profiles over a period of time for a set of servers, what is the least number of racks that they can be packed into without exceeding any rack's power cap?* A brute force optimization formulation can reduce this problem to *vector bin packing* [2], where d time instances of interest are d dimensions of an object and the bin size in each of the d-dimension is the rack power cap. However, in this formulation, d could be several thousands if the provisioning cycle is a day and power samples are collected every 30 seconds. Since this vector bin packing formulation leads to an NP-hard problem with prohibitively large dimensions, we use a number of domain-specific optimizations to arrive at a near-optimal solution efficiently. One of the central insights we use is that some, but not all, servers' power consumption can be strongly correlated due to their application dependencies or load balancing designs. Hence, it is desirable to find servers that show anti-correlated, or strongly time-shifted behavior and pack them together to minimize power capping probability. Our experiments with RackPacker show from 15-30% improved efficiency in packing servers in racks. Note, however, that RackPacker provides a probabilistic solution – should server power consumption diverge significantly from the norm, rack capacity can be exceeded.

In Section 2, we describe the background and common practice on rack power provisioning and show the opportunity for unleashing stranded power. We then describe our algorithm – RackPacker – in Section 3. We discuss the evaluation of RackPacker in Section 4 and present related work in Section 5. Finally, Section 6 presents some key discussion points and concludes.

2 Stranded Power

To understand the rack packing challenges and opportunities, we first describe the power distribution and provisioning architecture in a typical data center. Power consumed by a data center is usually divided into *critical power*, which is UPS backed up and used by IT equipment, and *non-critical power*, which is used by cooling and other parts of the facility that do not require UPS backup. In this paper, we only consider critical power utilization.

Critical power in a data center is delivered to remote power panels (RPP) in each server room (usually called server co-locations or colos), split into many circuits there, and then distributed to server racks in that colo. Every circuit has a defined capacity, and is regulated by a circuit breaker, which is the physical defense for catastrophic power failures. For redundancy purposes, a rack usually has multiple circuits, each in the form of a power strip. Servers, typically dual corded, spread their power load across the power strips they plug into. Figure 1 shows the power provisioning chart for a rack with 3 circuits, with power load evenly distributed over the circuits, (i.e. each server is plugged into two of the three power strips). There are two overheads that limit the amount of power usable by the servers: *spike protection* and *failover protection*. Assume each circuit is rated at single phased 30Amps and 208V, then the total available power at each circuit is 6.24KW[2]. However, 10% to 20% of the total power is reserved to handle spikes in the power grid or load (10% is shown in this plot). Furthermore, in order to support failover – in the sense that when one of the three power strips fails, all servers can safely use the remaining two power strips– another 30% of the total power has to be set aside. Thus, the *usable power* to the servers is only up to 60% of the total power — 3.74 KW per circuit, or 11.2KW for the entire rack. In fact, this rather conservative power provisioning baseline encourages probabilistic over-subscription, since temporarily exceeding the power cap is likely to be safe.

The common practice of power provisioning, however, does not even fully utilize the 60% usable power. Server

[2]Technically, it is 6.34KVA. For ease of discussion, we ignore the power factor and treat W and VA interchangeably.

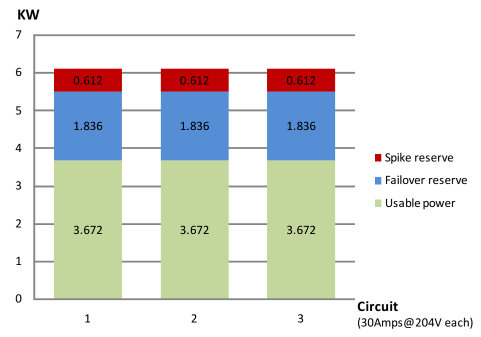


Figure 1. An illustration of power provisioning at the rack level. About 40% of available power is reserved for handling spikes and failover.

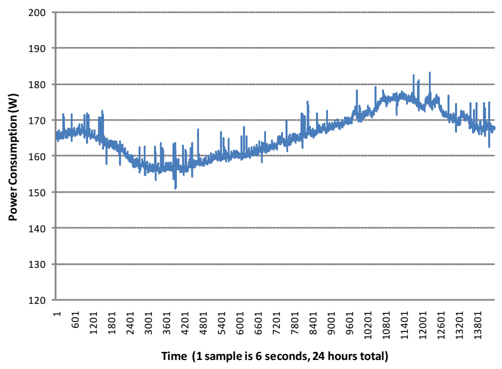


Figure 2. A real power consumption trace of a production server.

vendors usually give an estimated *nameplate* power consumption indicating the maximum possible power consumption of the server. For example, the power calculator tool from HP [5] rates 395W for a ProLiant DL360 G5 server, with two Xeon E5410 2.33GHz quad-core CPUs, four 2GB DIMM, and two 146.8GB SAS HDD. In other words, a 11.2KW rack can host at most 28 such servers, even though each server only occupies one unit in a typical 44 unit rack.

In reality, the nameplate power allocated to a server is never fully used. Using server profiling, one can arrive at a discounted power rating, which is lower than the nameplate power rating. For example, if the DL 360 server has never consumed more than 300W, using the discounted power rating, a rack can host 37 such servers.

Static power provisioning, even with discounted power rating, can still leave a large amount of power stranded. Figure 2 shows a power consumption trace over a day of a production server accessed by millions of users. We have two observations. First, server power consumptions change due to the load fluctuation. Slow and quasi-periodic load fluctuation has been observed in a lot of web traffic, including web sites [1] and instant messaging [3]. This fluctuation can become even more significant as idle power consumption is decreasing for newer servers. Secondly, in addition to the slow fluctuation, there are spikes, caused by short term load variation such as scheduled processing intensive tasks or flash crowd visitors. The discounted power rating – being a worst case estimate – must include both the peak of the fluctuation and the highest spikes; thus it can be overly conservative.

Power over-subscription can take advantage of two dynamic properties of actual server power traces:

- Not all servers fluctuate to the peak at the same time. The usage patterns of on-line services can be diverse. For example, websites for financial news and services may reach their peak around late morning when both east and west coast customers are on-line and the stock market is open. However, home entertainment sites may reach their peak in the evening. If we can bundle services that are maximally out of phase, then the peak of the sum is less than the sum of the peaks.

- Servers that are managed by the same load balancer or have close dependencies can have strong correlations among their spikes. Statistically, placing services that are *anti-correlated* will lead to smaller probability of their seeing simultaneous spikes.

These observations motivate us to design RackPacker, which statistically guarantees that over-subscribed sets of servers do not exceed rack level power caps.

3 The RackPacker Approach

3.1 A Running Example

Throughout the rest of the paper, we use 831 servers from a popular on-line service as a running example for our discussion. Functionality-wise, these servers largely belong to three categories, which we call Types 1, 2, and 3. They are divided into several clusters, where each cluster is managed by a load balancer. Server workloads

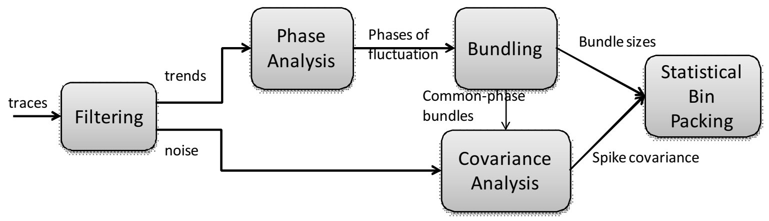


Figure 3. The flow of the RackPacker algorithm.

show strong correlations, because of both functionality dependencies and load balancing effects. For example, when there is a flash crowd, servers behind the same load balancer experience a power spike at the same time, while servers across load balancers are less correlated. Due to the nature of the application, we also observe that about 2 hours after servers of type 1 reach their peak workload, servers of type 3 reach their peak. In addition to the tight coupling among server tiers, the relatively high CPU utilization, reaching over 75% at peak load, make this a challenging set of servers for rack packing.

These servers have a nameplate power rating of 350W; based on this number, a 11.2KW rack can host 32 servers. In other words, we need 26 racks to host these servers in the most conservative situation.

3.2 Rackpacker Overview

RackPacker takes a data-driven approach that uses collected power consumption traces to support server placement decisions. We assume that services are hosted by virtual machines, even though there may be only one VM per physical server. VMs enable fast re-positioning of services without moving the servers physically. This allows the server placement decisions to be made frequently – at a weekly or even daily basis– and aggressively. The RackPacker algorithm, thus, only needs to predict short term traffic growth. In the rest of the paper, we use the terms server and service interchangeably. That is, a server of type 1 refers to a VM hosting service type 1 running on a physical server. We only consider homogeneous server hardware.

Figure 3 shows the key components in the RackPacker algorithm.

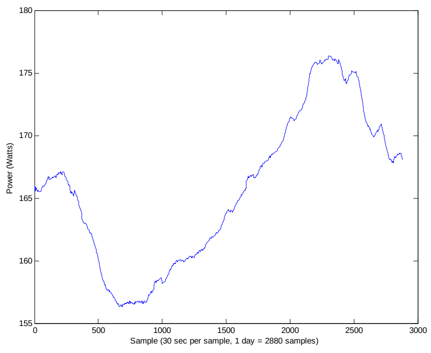
By profiling or monitoring a server operation, we model the server power consumption with a time series (rather than a single number). The time series is first filtered to obtain the low frequency power consumption baseline, and the high-frequency noise that captures spikes. The noise signal has zero mean. Its variance represents how "spiky" the transient power consumption can be. The goal of obtaining the low-frequency components is to identify the baseline fluctuations reflecting workload trends, specifically their phase. Using this phase information, we can sift through the servers and bundle those that are most out of phase. The bundles are then treated as the unit for rack packing. The high-frequency noise goes through a covariance analysis that measures the likelihood that two bundles may have spikes at the same time. This statistical measure, together with the baseline of the bundles is used in a statistical bin packing algorithm to find a (sub-)optimal server placement solution.

Thus, RackPacker has three major steps: filtering, bundling, and packing. In the rest of this section, we describe each of these steps in detail.

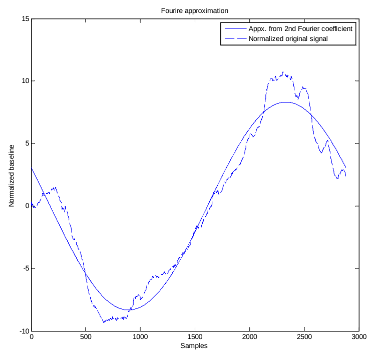
3.3 Filtering and Classification

The goal of filtering is to separate workload trends from noisy transients. A typical approach is to compute a moving average with a sliding window on the power traces, which is equivalent to low-pass filtering. Let S be the set of servers of interest, P_s be the power profile time series of server $s \in S$ with M samples, and T be the sliding window size to compute the moving average. Then, the baseline B_s is computed as $B_s(i) = \frac{1}{T}\sum_{j=(i-T+1)}^{i} P_s(j), i = \{1...M\}$ (with patching zeros when $i \leq T$), and noise $N_s = P_s - B_s$. Figure 4 presents the results of filtering the time series shown in Figure 2. Figure 4(a) is the baseline signal obtained by a 30 minutes moving average. The residual noise signal and its histogram are shown in Figure 4(c) and Figure 4(d). We use σ_s to represent the standard deviation of the noise.

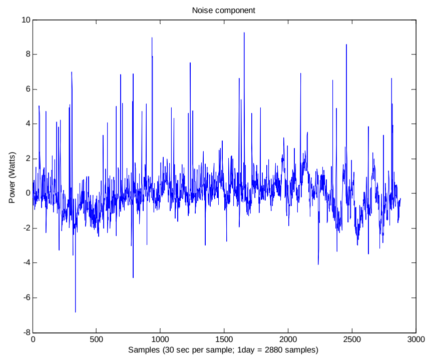
To obtain and compare the relative times at which different servers peak, we perform discrete Fourier transform (FFT) on the baseline signal. In particular, since the most significant fluctuation has the period of a day, we expect that the second FFT coefficient has the largest magnitude. Indeed, for the power profile in Figure 2, the normalized magnitude of the first 10 FFT coefficients are $[0, 4.2790, 0.2240, 0.7166, 0.4953, 0.1057, 0.1303, 0.0738, 0.0393, 0.0609]$.



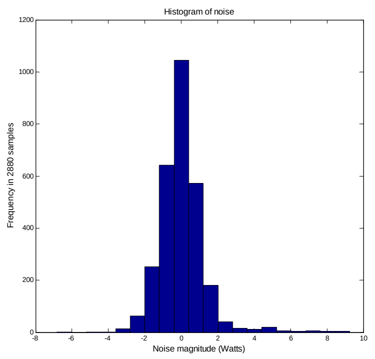
(a) The baseline trace.



(b) Fourier approximation using 2nd coefficient



(c) The noise trace.



(d) Noise histogram.

Figure 4. Fitering and approximation of the power consumption time series.

It is clear that the second component is at least an order of magnitude greater than other components, indicating that it is a good approximation of the overall shape of the power profile.

We denote the second FFT coefficient of the baseline power profile by f_s. Note that f_s is a complex number that represents a sine wave that can be written as $|f_s|Sin(\omega t + \phi_s)$, where $|f_s|$ is the magnitude and ϕ_s is the phase. In a slight abuse of terminology, we call ϕ_s the *primary phase* of the service. For example, Figure 4(b) compares the signal reconstructed by the second Fourier coefficient with the original signal. We clearly see that the second coefficient captures well the overall shape of the original power profile.

Based on the relative magnitudes of the noise level and the fluctuation $|f_s|$, the servers can be classified as *flat* or *fluctuating*. Intuitively, a fluctuating server shows substantial load variation above and beyond its noise. In our example, we consider servers whose power profile has $|f_s| < 3\sigma_s$ to be flat. By this definition, 830 out of the 831 servers fluctuate. Fluctuating servers that show significant phase difference will potentially pack well together, and deserve special attention. This brings us to the bundling step.

3.4 Bundling

The goal of bundling is to find small sets of servers whose primary phases "match". Ideally, if the average of f_s across all servers is 0, then the fluctuations cancel each other out . However, in real data centers, this may not be possible. Therefore, the total power load fluctuates at the data center level. Let $\bar{\phi}$ be the average phase of all f_s. Then the best packing approach should spread the data center peak load evenly to all racks. Hence, the target for the bundling process is to make the average phase of each bundle as close to $\bar{\phi}$ as possible.

Another benefit of a common phase for all bundles is dimension reduction. As stated earlier, given a set of power profile time series, we need to verify that at each time instance the total power consumption at each rack does not exceed the power cap with high probability. When server power profiles show distinct phases, we need to perform this verification at the peak time of every power profile. By bundling servers into common phase groups, we only need to verify the time instance when the common phase sine wave reaches the peak.

The bundling process can be explained using complex vectors. The complex coefficient f_s of server s can be viewed as a vector in the complex coordinates, as can the average vector $\bar{f}$ with phase $\bar{\phi}$. Then each vector can be decomposed by projecting it to the direction of $\bar{f}$ and to the direction that is orthogonal to $\bar{f}$. Figure 5(a) illustrates this projection. Let f_1 be the 2^{nd} FFT coefficient of server 1, and $\bar{f}$ be the average vector across all servers. Then we project f_1 on $\bar{f}_s$ to obtain $\bar{f}_1$, and then $\tilde{f}_1 = f_1 - \bar{f}_1$. If there exists f_2, whose projection $\tilde{f}_2$ on the direction that is orthogonal to f_s satisfies, $\tilde{f}_2 + \tilde{f}_1 = 0$, then bundling server 1 and server 2 together achieves the common phase. Once common phase bundles are created, further bundling can be performed along the $\bar{f}$ direction so that positive and negative magnitudes cancel each other out .

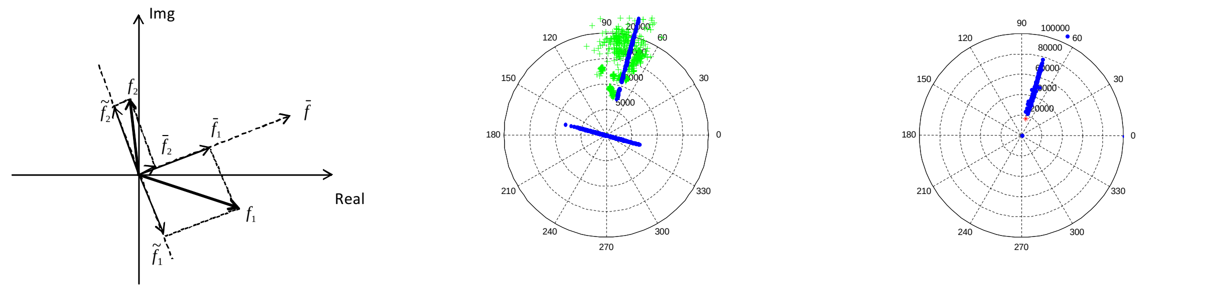
Algorithm 1 shows the pseudocode for this bundling step. There are two parameters that affect bundling per-

Algorithm 1 Pseudocode for Bundling phase of RackPacker

```
RackPacker: Bundling
 1: Compute the mean $\bar{f}$ of $\{f_s\}$ for all fluctuating servers. Compute the angle $\bar{\phi}$ of $\bar{f}$.
 2: For each vector $f_s$ with magnitude $|f_s|$ and angle $\phi_s$, project $f_s$ to the direction of $\bar{\phi}$ and $\bar{\phi} + \pi/2$:
 3:         $\bar{f_s} = |f_s| cos(\bar{\phi} - \phi_s)$
 4:         $\tilde{f_s} = -|f_s| sin(\bar{\phi} - \phi_s)$
 5: Sort $\tilde{f_s}$ in a descent order.
 6: Select the unbundled server $s$ with the largest $|\tilde{f_s}|$, and place it in a bundle $b$.
 7: Compute the size of $|b|$ and $\tilde{b}$, the length of $b$ along the $\bar{\phi} + \pi/2$ direction.
 8: **if** $|\tilde{b}| < \epsilon_B$ **then**
 9:   Finish with current bundle and repeat 6.
10: **else**
11:   **if** There is no unbundled server **then**
12:     Finish.
13:   **else**
14:     Select unbundled server $s\prime$ such that $|\tilde{f_{s\prime}} + \tilde{f_b}|$ is minimized.
15:     **if** the size of $b + s\prime > BundleCap$ **then**
16:       Finish current bundle without putting $s\prime$ in $b$
17:     **else**
18:       Add $s\prime$ in $b$, and repeat 7.
19:     **end if**
20:   **end if**
21: **end if**
22: Treat each bundle as flat. For every bundle $b$, compute its baseline $B_b = \sum_{s \in b} B_s + \max_{t \in T} |f_b|$, and its
    variance $\sigma_b$ from the variance and covariance of the noise vectors of the servers in the bundle.
```

formance: the max bundle size $BundleCap$ and the cancellation threshold ϵ_B. Intuitively, the smaller we make ϵ_B, the closer the bundled vectors get to the $\bar{f_s}$ direction. However, one cannot bundle too many servers together since they could then exceed the power cap. As we will discuss later, the packing performance is also affected by the correlation of the noise factors. Since noise is not considered in the bundling process, we want to limit the bundling size to give flexibility to the packing step.

Figure 5 shows the results of bundling the 830 fluctuating servers in our running example. Figure 5(b) shows the original vectors with '+' markers, and their decomposition to the mean and its orthogonal directions with '.' markers. The vectors in the orthogonal directions are canceled out by the bundling process, and Figure 5(c) shows the vectors after bundling. The maximum bundle size is 3, when we set the bundle power cap to be one-tenth of the rack power cap.



(a) Bundle two servers toward the common phase. (b) The decomposition of the vectors for 831 servers. (c) The bundling results of 831 servers based on the decomposition.

Figure 5. Bundling based on the decomposition of the 2^{nd} FFT coefficient vectors.

3.5 Packing

Once bundles are created with the same phase, the packing process uses a modified bin packing algorithm for the final placement. A particular challenge that the packing step addresses is the correlations among the spikes.

The goal of the packing phase is to assign bundles to racks in such a manner as to minimize the probability of exceeding the rack power cap. In order to minimize this probability, the packing phase packs together bundles that show minimal correlation in their spikes (noise). Correlated bundles spike in lockstep; this can result in a heightened likelihood of exceeding the rack cap in the event of load spikes such as flash crowds.

In order to compute sets of bundles that show minimal noise correlation, the packing phase proceeds as follows. First, the bundles are ordered in descending order of size. Bundle size for a bundle b is computed as $\sum_{s \in b} B_s + CF * \sigma_b$, where σ_b is the standard deviation of the bundle noise, and CF stands for confidence factor, a configuration parameter (3, here).

We then iterate through this ordered list of bundles and assign them to racks one by one. A bundle b is deemed to fit into a rack r if $\sum_{b' \in r} B_{b'} + B_b + CF * \sigma_{r,b} < C_r$, where $\sigma_{r,b}$ is the standard deviation of the rack noise ($=$ sum of the noise of each bundle in that rack) combined with the noise of bundle b, and C_r is the rack cap. Given a non-empty rack r, to arrive at the next bundle that we'll attempt to pack into r, we order the unassigned bundles in ascending order of their covariance with the current contents of r. We then try to find a bundle from this ordered list that will fit into r. If no such bundle is found, we create a new rack and repeat the process. Algorithm 2 presents the pseudocode for this phase.

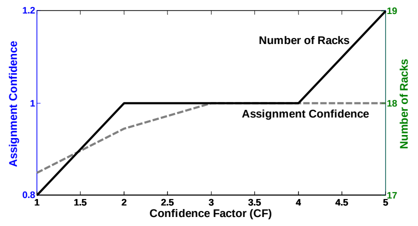
Algorithm 2 Pseudocode for Packing phase of RackPacker

```
RackPacker: Packing
1: Sort the bundles in descending order by $B_b + CF * \sigma_b$, where CF = confidence factor, a configuration parameter. Call this list $L$.
2: Pick a bundle $b$ from the top of the list $L$ and assign it to rack R.
3: For all bundles in $R$, compute $B_R = \sum_{b \in R} B_b$, and $\sigma_R = \sqrt{\sum_{b \in R} \sigma_b^2 + 2 \sum_{b_1, b_2 \in R} covariance(b_1, b_2)}$.
4: **while** list $L$ non-empty **do**
5:     Pick a bundle $b'$ from $L$ that is most uncorrelated with all the bundles in $R$, and add it to $R$.
6:     For all bundles in $R$, compute $B_R$ and $\sigma_R$ as above. If $B_R + CF * \sigma_R > C_R$, remove the last bundle from $R$.
7: **end while**
8: Repeat from 2 with a new rack.
```

4 Evaluation

Parameter	Value
Rack Cap	11200 W
Bundle Cap	1120 W
ϵ_B	20
Confidence Factor (CF)	3

(a) RackPacker Configuration Parameters



(b) Choice of Confidence Factor

Figure 6. Simulation parameter choices.

We have implemented RackPacker in MATLAB. Figure 6 shows our choice of parameters for the implementation. The choice of the parameter "Confidence Factor (CF)" is illustrated in figure 6. Here assignment confidence is computed as the percentage of racks that fail to stay within the rack cap over a week's trace of data. We see that the choice of the CF value results in a tradeoff between assignment confidence and packing efficiency.

In evaluating RackPacker, we wish to answer the following questions:

1. *How does RackPacker compare with the prevalent server assignment algorithms?* We wish to see if there is a strong argument for using RackPacker in place of existing solutions.

2. *What kinds of workloads is RackPacker best suited for? Conversely, are there workloads for which RackPacker is not suitable?* We wish to know what kinds of applications benefit the most from RackPacker.

We tackle each of these questions in order in this section.

4.1 RackPacker: Comparative Performance

To compare the efficacy of RackPacker against current solutions, we use the following metrics:

- **Stranded Power:** This is the difference between provisioned power and actual power consumed per rack. Minimizing stranded resources is the goal of a good provisioning scheme. Hence, the less the stranded power per rack, the better the server assignment algorithm.

- **Packing Efficiency:** This is the number of racks needed to host the given set of servers. We wish to minimize this number in order to improve the utilization of the data center.

Algorithm 3 Psuedocode for Static Assignment. Note that power(s) can be the nameplate rating of s, or the peak measured power for s.

```
Static Assignment Pseudocode
1: Order the servers randomly. Call this list serverlist.
2: Remove the first server s from serverlist and assign it to the first rack. Compute this rack's power consumption
   as: rackpower(1) = power(s)
3: while serverlist is not empty do
4:     Remove server s (of type t, say)from top of serverlist
5:     if Fit Criterion: rackpower(curr_rack)+power(s) < rack power cap then
6:         Assign server s to current rack and update its rackpower
7:     else
8:         Create a new rack, and assign s to it.
9:     end if
10: end while
```

We compare RackPacker with two flavors of static assignment: (1) Nameplate Rating-Based assignment, and (2) Peak Power-Based assignment. Both these schemes employ striping, where each type of server is distributed uniformly across all the racks. This results in each rack containing approximately the same relative proportion of each type of server. The *nameplate rating-based scheme* uses the power rating on the server as a measure of

Number of server types		3
Number of servers	Type 1	329
	Type 2	283
	Type 3	219
	Total	831
Average power consumed	Type 1	199.4 W
	Type 2	194.7 W
	Type 3	210.1 W
Peak power consumed	Type 1	268.8 W
	Type 2	262.6 W
	Type 3	270 W
Data timespan		1 week

Table 1. Description of data against which RackPacker and other solutions are evaluated

its power consumption. Since this number is usually a substantial over-estimate, we also provide a comparison point called the *peak power-based scheme*, which uses the measured peak power consumption of the server in place of the nameplate rating. This is the most aggressive static power provisioning approach, which assumes that the peak in the future does not exceed the peak in the past. Algorithm 3 presents the pseudocode for both these static assignment schemes. In this section we present analytical results for the nameplate rating-based scheme, and simulated results for the peak power-based scheme, and the RackPacker algorithm. In the graphs that we present, the algorithm labelled "Static" refers to the peak power-based scheme.

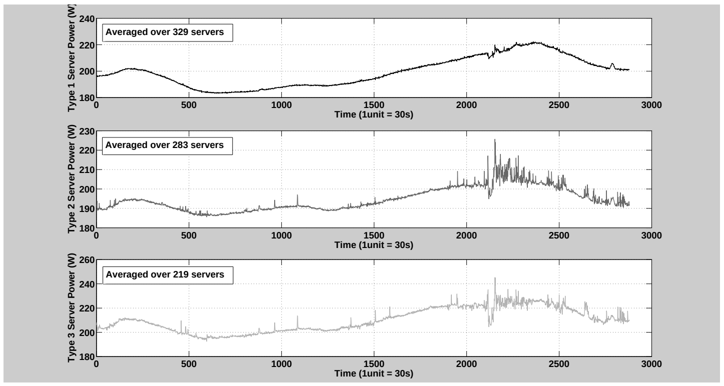
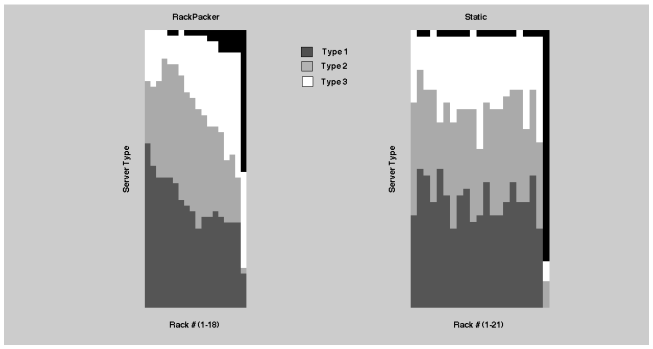
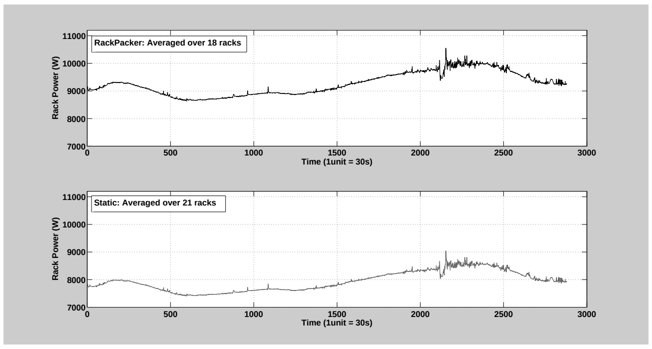


Figure 7. Average Power Consumption Behavior For The Different Server Types

We evaluate each of these three server assignment algorithms on real power consumption data obtained from a production data center. The data spans 831 servers for a production application. These servers belong to one of three types, corresponding to different tiers of the application. Table 1, and figure 7 describe the data. The data spans a week, but we train the various algorithms on one day's data, and validate the computed assignment against the remaining days.



(a) Server Assignments Computed by RackPacker and Peak Power-Based Static Assignment Algorithms.



(b) Server Assignments Computed by RackPacker and Peak Power-Based Static Assignment Algorithms.

Figure 8. Server assignment results from a realistic workload trace.

Figure 8(a) is a pictorial representation of the server assignments computed by RackPacker, and the peak power-based scheme. We find that RackPacker results in 14% more efficient assignment, using only 18 racks against 21 for the peak power-based static assignment. Further, figure 8(b) shows the power consumed per rack, averaged over all racks for each of these assignments. The rack cap was assumed to be 11,200 W. We see that RackPacker results in much less stranded power. RackPacker does much better when compared with the nameplate rating-based scheme. Recall that using nameplate numbers, we need 26 racks to host these servers. Thus here we see a 30% improvement in packing efficiency.

4.2 RackPacker: Workload Exploration

In the previous section we showed that RackPacker can improve utilization substantially for a real data center scenario. Now we will explore what kinds of workloads RackPacker is best suited to.

The workload presented in figure 7 represents a single-application hosting center. The three types of servers represent three tiers of the application; we see that these tiers operate essentially in lockstep, with load variation being consistent across the tiers. Here we will explore two other data center scenarios. The data for these scenarios is generated through controlled modification of the real data from table 1.

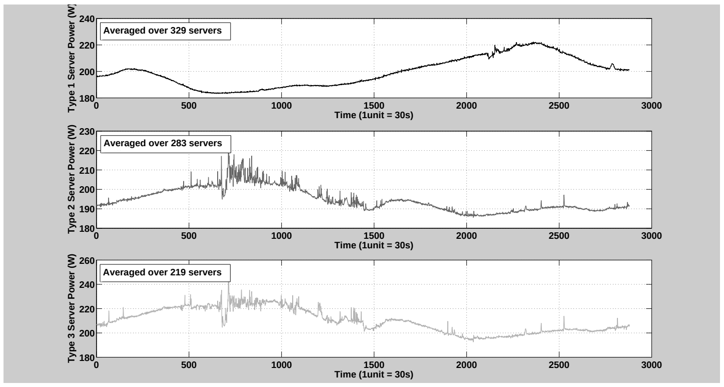
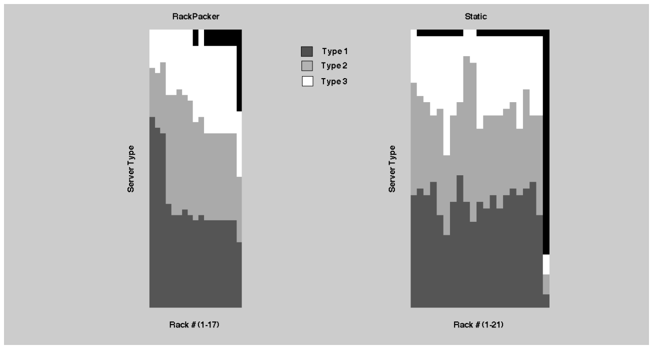
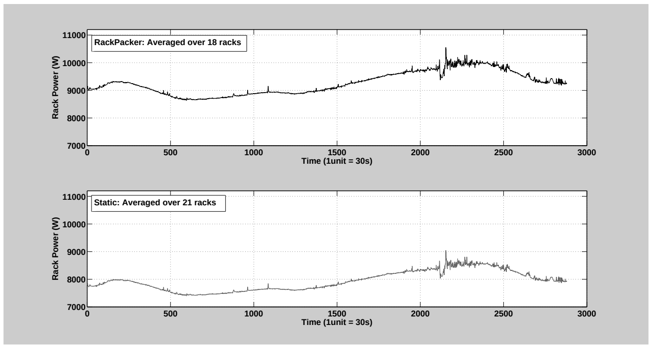


Figure 9. Average Power Consumption Behavior For The Different Server Types



(a) Server Assignments Computed by RackPacker and Peak Power-Based Static Assignment Algorithms.



(b) Server Assignments Computed by RackPacker and Peak Power-Based Static Assignment Algorithms.

Figure 10. Server assignment results from a workload trace with shifted phases.

Dedicated Multi-Application Hosting Center: Here we consider data centers that host a small number of applications (more than one). Figure 9 shows the data we generated to represent this scenario. Again, there are three types of servers, but Types 2 and 3 belong to a different application than Type 1 – they are thus phase shifted. Figure 10(a) shows the server assignment computed by RackPacker and the peak power-based static scheme.

Again, we find that RackPacker achieves 19% better packing efficiency, using 17 racks against 21 for the static scheme. Figure 10(b) shows the corresponding reduction in stranded power. The nameplate rating-based scheme needs 26 racks (as computed above); RackPacker is now 34% more efficient. In general, we expect that phase shifted servers will benefit more from RackPacker.

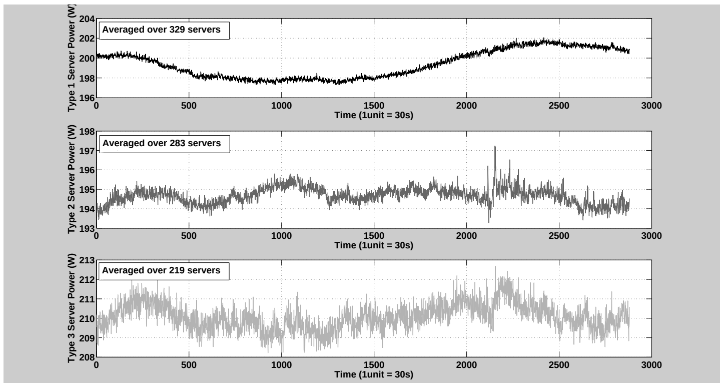
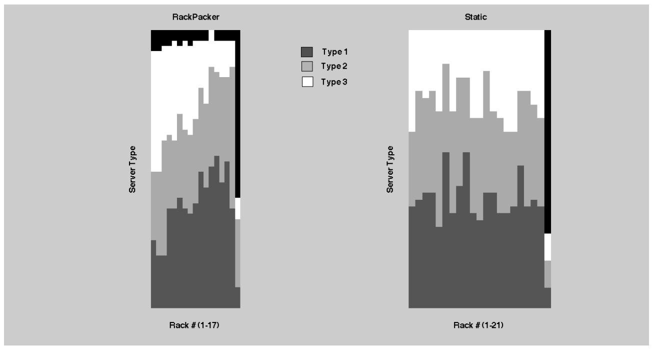
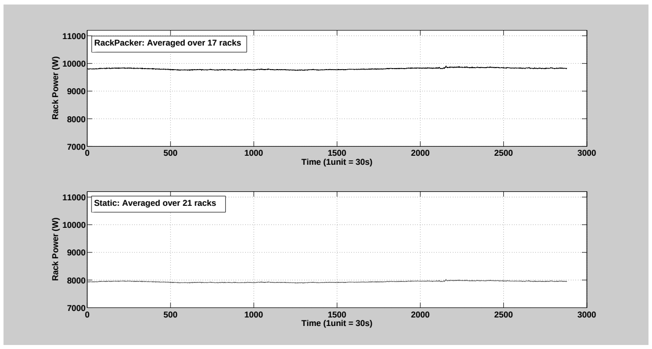


Figure 11. Average Power Consumption Behavior For The Different Server Types



(a) Server Assignments Computed by RackPacker and Peak Power-Based Static Assignment Algorithms.



(b) Server Assignments Computed by RackPacker and Peak Power-Based Static Assignment Algorithms.

Figure 12. Server assignment results from a workload trace with randomized phases.

Mixed Hosting Center: Here we consider data centers that host a very large number of applications; this represents the cloud computing scenario, where the servers are leased out to various companies that host different applications on them. Figure 11 shows the data we generated to represent this scenario. Here we see that there are numerous types of servers, and their correlations are less obvious. Figure 12(a) shows the server assignment computed by RackPacker and the peak power-based static scheme. Figure 12(b) shows the average rack power utilization for each of these assignments. Again, we find that RackPacker outperforms the static schemes substantially.

5 Related Work

In this paper, we present a scheme for intelligent oversubscription of data center power. The idea of power oversubscription is not new, and has been explored in the literature in numerous ways. The common theme in prior work, however, is that power tracking/capping are the means used to achieve this oversubscription. To the best of our knowledge, server placement – which sets of servers are placed in which racks – has not been studied as a means of improving data center utilization. Thus, RackPacker is intended to supplement prior work by intelligent server placement that reduces the need for rack-level power capping.

Fan et al [6] study the aggregate power usage characteristics of large collections of servers for different classes of applications over a period of six months and conclude that cluster-level power capping is a feasible and practical means of improving data center utilization. Their conclusion is based on the intuition that even if power utilization is high at server and rack levels, it is unlikely to be too high at cluster level (since a large number of servers would need to be simultaneously heavily loaded, for this to happen). However, they offer no other insights to implementing power capping.

Muse [1] is a game-theoretic, distributed power management architecture. The goal is to reduce the power consumption of hosted applications by allocating only as many servers as are needed to serve the arriving requests. Muse uses a load prediction model called “flop-flip” which combines two exponentially weighted moving averages of observed load to achieve stable and reasonably agile load estimations. Game theory is used to translate these load estimates to the number of active servers needed per application. Idle servers are shut down to save power.

Chen et al [4] use two control knobs to restrict application power usage: the number of active servers, as well as their performance states. They use queueing theory to compute request arrival rate over some epoch, and a feedback control loop to correct the predictions over a sub-epoch. Their controller then solves the following optimization problem: given the predicted throughput, what is the optimal number of servers to allocate for each epoch, and what is the frequency they should each be run at, for each sub-epoch.

Lefurgy et al [8] use CPU throttle states to implement power capping. CPU throttling reduces the clock speed, with power consumption dropping proportionally. The solution employs a control feedback loop running at each server. The server's power consumption is monitored periodically, and its CPU speed is set to target this load for the next epoch. The authors show how to make this model stable, with bounded settling time.

Heath et al [7] add a degree of sophistication to their controller by taking into account the heterogeity of the servers in the data center. Given the bandwidths of all the different resources, the controller's optimization problem is to find the request distribution from clients to servers, and among servers, in such a way that the demand for each resource is not higher than its bandwidth, and we minimize the ratio of cluster-wide power consumption over throughput.

Finally, our idea of translating the server placement problem into a form of multi-dimensional bin packing is inspired by Chekuri et al [2]. They present an approximate algorithm to pack d-dimensional vectors (servers) into d-dimensional bins (racks) to minimize the maximum load on any dimension. This algorithm, which represents the theoretical best solution for this problem, does not scale well in practice since it requires d to be much less than the average number of servers per rack.

6 Discussion and Conclusion

Efficient use of data center infrastructure is a pressing issue for the scalability of the IT industry. Due to conservative and static estimation of server power consumption, traditional approaches for power provisioning leave large amounts of provisioned power stranded. RackPacker is a data driven approach for power provisioning. By analyzing real power traces from servers, we obtain the baseline, fluctuation phase, and noise levels for each server. Leveraging this information, we can find sets of anti-correlated servers, in term of both fluctuation phase and noise covariance, that are best candidates for sharing the same rack. Our simulation results from real workload traces show that even with tightly coupled and high utilization services, we can achieve over 30% better packing performance compared to the nameplate rating-based provisioning mechanism. We can save 14% space in comparison to even the most aggressive static assignment approach.

RackPacker works the best when there are significant fluctuations on workload and power consumption. There are two reasons that strong fluctuations are increasingly common in server workloads. On-line services are getting more and more geo-focused. That is, many services are designed for users from particular countries or geo-locations. As a result, the workload on these servers reflects usage patterns and the peak load is concentrated in a small time span. Another trend is that the server hardware and software are becoming increasingly power aware. Server idle power is decreasing, while the peak power consumption stays relatively flat. This implies that the power consumption of servers, under variable workload will show fluctuating patterns.

There are several practical concerns when applying RackPacker to real data center operations. We did not consider the rack height constraints when evaluating RackPacker. It is easy to apply rack packing to reduce the power capping if rack height is a constraint. In this case, a data center can add more racks with smaller total power per rack. Sometimes, administrative advantages and security regulations can limit the flexibility of moving services within or across data centers. In addition, current data center networking architecture is hierarchical. Servers are divided into subnets and those in the same rack can only be in the same subnet (VLAN). However, many data centers are dominated by a relatively small number of services each employing a huge number of servers on the same VLAN. Solving the power provisioning problem for these services brings immediate benefits. We did not explicitly address in this paper how to proportionally provision cooling with server assignment. Cooling should not be a big concern in this context, since data centers' cooling capacities are designed to match their peak power consumptions.

As a data driven approach for resource management, RackPacker algorithm can be applied to other scenarios, in particular service consolidation via virtualization. Similar to the problem of finding "matching" servers for a rack, one would like to find matching services that can share the same physical server. The difference is that power is an additive resource, ignoring the power factor, but other resources in a physical server may not be additive. For example, depending on cache misses, the time delays of retrieving data from storage can differ significantly when multiple services share the same hardware. Modeling multi-modality resources and optimizing their utilization is challenging future work.

References

[1] CHASE, J. S., ANDERSON, D. C., THAKAR, P. N., VAHDAT, A. M., AND DOYLE, R. P. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2001), ACM, pp. 103–116.

[2] CHEKURI, AND KHANNA. On multi-dimensional packing problems. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)* (1999).

[3] CHEN, G., HE, W., LIU, J., NATH, S., RIGAS, L., XIAO, L., AND ZHAO, F. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), USENIX Association, pp. 337–350.

[4] CHEN, Y., DAS, A., QIN, W., SIVASUBRAMANIAM, A., WANG, Q., AND GAUTAM, N. Managing server energy and operational costs in hosting centers. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2005), ACM, pp. 303–314.

[5] COMPANY, H.-P. D. Hp power calculator utility: a tool for estimating power requirements for hp proliant rack-mounted systems, 2007.

[6] FAN, X., WEBER, W.-D., AND BARROSO, L. A. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture* (New York, NY, USA, 2007), ACM, pp. 13–23.

[7] HEATH, T., DINIZ, B., CARRERA, E. V., JR., W. M., AND BIANCHINI, R. Energy conservation in heterogeneous server clusters. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming* (New York, NY, USA, 2005), ACM, pp. 186–195.

[8] LEFURGY, C., WANG, X., AND WARE, M. Power capping: a prelude to power shifting. *Cluster Computing 11*, 2 (2008), 183–195.

[9] LOHR, S. Data centers are becoming big polluters, study finds. In *The New York Times, Technology* (Oct 16, 2008).