Sustainable Computing: Informatics and Systems xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Hybrid resource provisioning for minimizing data center SLA violations and power consumption

Anshul Gandhi[a,*], Yuan Chen[b], Daniel Gmach[b], Martin Arlitt[b], Manish Marwah[b]

[a] Computer Science Department, Carnegie Mellon Univeristy, United States
[b] Sustainable Ecosystems Research Group, Hewlett-Packard Laboratories, Palo Alto, CA, United States

ARTICLE INFO

Article history:
Received 30 September 2011
Received in revised form 14 January 2012
Accepted 29 January 2012

Keywords:
Resource provisioning
Power management
Sustainable computing
Workload analysis

ABSTRACT

This paper presents a practical and systematic approach to correctly provision server resources in data centers, such that SLA violations and energy consumption are minimized. In particular, we describe a hybrid method for server provisioning. Our method first applies a novel discretization technique on historical workload traces to identify long-term workload demand patterns that establish a “base” load. It then employs two techniques to dynamically allocate capacity: predictive provisioning handles the predicted base load at coarse time scales (e.g., hours) and reactive provisioning handles any excess workload at finer time scales (e.g., minutes). The combination of predictive and reactive provisioning achieves a significant improvement in meeting SLAs, conserving energy and reducing provisioning cost.

We implement and evaluate our approach using traces from four production systems. The results show that our approach can provide up to 35% savings in power consumption and reduce SLA violations by as much as 21% compared to existing techniques, while avoiding frequent power cycling of servers.

1. Introduction

Data centers are very expensive to operate due to the power and cooling requirements of IT equipment like servers, storage and network switches. As demand for IT services increases, the energy required to operate data centers also increases. The EPA estimated that energy consumption in data centers exceeded 100 billion kWh in 2011, at a cost of $7.4 billion [10]. Rising energy costs, regulatory requirements and social concerns over green house gas emissions have made reducing power consumption critical to data center operators. However, energy efficiency is for naught if the data center cannot deliver IT services according to predefined SLA or QoS goals, as SLA violations result in lost business revenue. For example, Amazon found that every additional 100 ms of latency costs them a 1% loss in sales, and Google observed that an extra 500 ms in search page generation time reduced traffic by 20% [18]. Hence, another challenge data center operators face is provisioning IT resources such that SLA violations are minimized, to prevent business revenue loss. Today, SLA violations are often avoided by over-provisioning IT resources. This results in excessive energy consumption, and may also lead to increased expenditure due to capital overhead, maintenance costs, etc. Thus, an important question in data center resource management is how to *correctly provision* IT equipment, such that SLA violations are avoided and energy consumption is minimized.

The correct provisioning of resources is a difficult task due to variations in workload demands. Most data center workload demands are very bursty in nature and often vary significantly during the course of a single day. This makes it difficult to provision resources appropriately. A single size (static provisioning) cannot fit all, and will result in either over-provisioning or under-provisioning.

The solution we propose in this paper is based on three important observations. First, many workloads in data centers (e.g., Web servers) exhibit periodic patterns (i.e., daily, weekly and seasonal cycles). If we can identify these patterns in the workload, we can then adjust the resource allocation accordingly, and hence improve the accuracy of resource provisioning and reduce power consumption. Second, demand patterns are statistical in nature, and there will be deviations from historical patterns due to unforeseen factors such as flash crowds, service outages, and holidays. Though the volume of such fluctuations is small compared to the total demand, ignoring them completely can result in significant SLA violations. Third, provisioning is not free; there are various associated costs and risks. Frequent provisioning incurs both performance and energy penalties. For example, turning servers on can take a significant amount of time (up to several minutes) and consume a lot of power (close to peak power consumption) [12]. Frequent power cycling of servers causes “wear and tear”, which could result in server failure and service outage(s) [8].

* Corresponding author. Tel.: +1 4122684973.
E-mail addresses: anshulg@cs.cmu.edu (A. Gandhi), Yuan.Chen@hp.com (Y. Chen), Daniel.Gmach@hp.com (D. Gmach), Martin.Arlitt@hp.com (M. Arlitt), Manish.Marwah@hp.com (M. Marwah).

2210-5379/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.suscom.2012.01.005

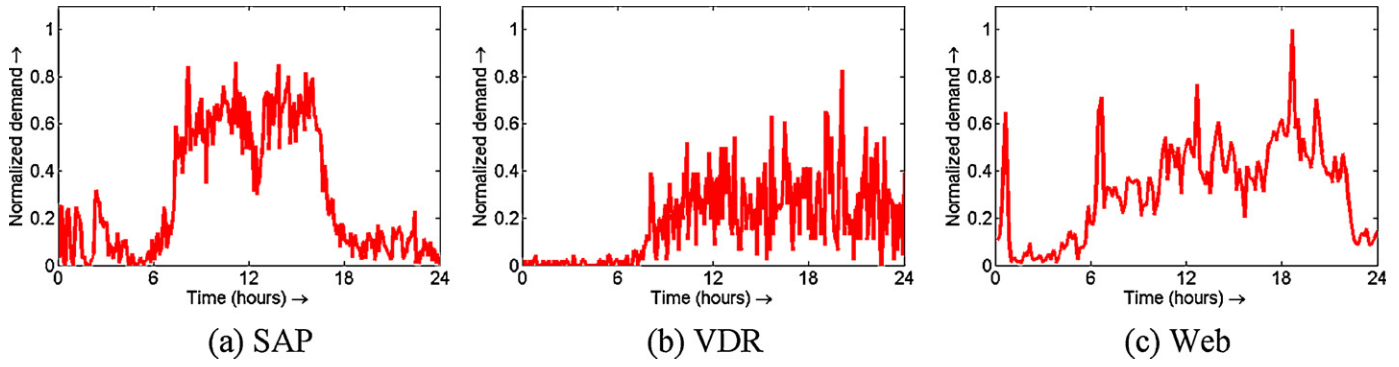


Fig. 1. Workload demand for a single day for (a) the SAP trace; (b) the VDR trace; and (c) the Web trace.

Based on the above observations, we propose a novel resource provisioning approach in this paper. Our main contributions are:

1. We present a detailed workload characterization study for three real applications and provide insight about their properties: variability and periodicity.
2. We propose a novel analysis technique—"workload discretization", to determine the "base" workload demand for a service. In particular, we propose a dynamic programming algorithm that can accurately capture the demand while minimizing the costs and risks from a provisioning perspective.
3. We develop a hybrid approach to provision IT resources: a predictive provisioning approach handles the base workload at coarse time scales (e.g., hours) and a reactive provisioning approach handles any excess demand at finer time scales (e.g., minutes). A coordinated management of these two approaches achieves a significant improvement in energy efficiency without sacrificing performance.
4. We implement our server provisioning system and evaluate it using empirical workload traces. The results reveal that our workload discretization algorithm better estimates the long-term workload demand from a resource provisioning perspective, and our hybrid provisioning system is superior to existing approaches in terms of meeting the system's SLA requirements while conserving power, reducing provisioning cost and avoiding frequent power cycling of servers.

The rest of the paper is organized as follows. Section 2 examines the demand distribution and properties of several workload traces from production systems. Section 3 describes our workload discretization technique and Section 4 presents our hybrid provisioning approach. Section 5 discusses our experimental evaluation. Section 6 reviews related work and finally Section 7 concludes the paper with a summary of our work and a description of future directions.

2. Workload characterization

To understand real data center workload demands, we begin with a detailed workload characterization of three types of applications.

2.1. Workload trace descriptions

1. SAP. We obtained a five-week-long workload demand trace of an SAP enterprise application. The trace was obtained from a HP data center that hosts enterprise applications such as customer relationship management applications for small and medium sized businesses. The trace captures average CPU and memory usage as recorded every 5 min.
2. VDR. VDR is a high-availability, multi-tier business-critical HP application serving both external customers and HP users on six continents. The VDR data set includes both transaction records (e.g., arrival rate information) and system resource utilization measurements collected at application server and database server tiers. The ten-day-long trace contains 5-min average data.
3. Web 2.0. We obtained data from a popular HP Web service application with more than 85 million registered users in 22 countries. The service caters to over 10 million users daily. The trace contains five days worth of machine-level statistics from about 200 servers located in multiple data centers.

2.2. Variability

Most data center workload demands are very bursty in nature and often vary significantly during the course of a single day. We now analyze the variability in workload demands of the above traces.

Fig. 1 shows 24-h workload demand (CPU usage) traces for the SAP, VDR and Web applications. The demand has been normalized by the maximum demand in each trace. We see that the workload demand varies a lot. For example, the SAP trace demand of that day varies from a minimum of almost 0 to a maximum of approximately 0.8 of the peak demand in the 5-week trace. Fig. 2 shows the maximum, 90%ile and mean demand for each workload trace. We see that the maximum demand is typically 4–5 times higher than the corresponding mean demand, suggesting huge variations across the trace time-series. For all traces, the 90%ile value is much higher than the mean value, suggesting that demand values are highly variable. This brings us to our first observation:

Observation 1: The workload demands typically have significant variability. This makes it difficult to provision resources appropriately, resulting in either over-provisioning or under-provisioning.

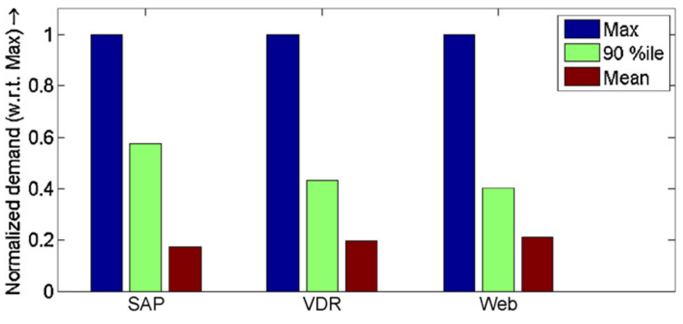


Fig. 2. Variability in the workload demands.

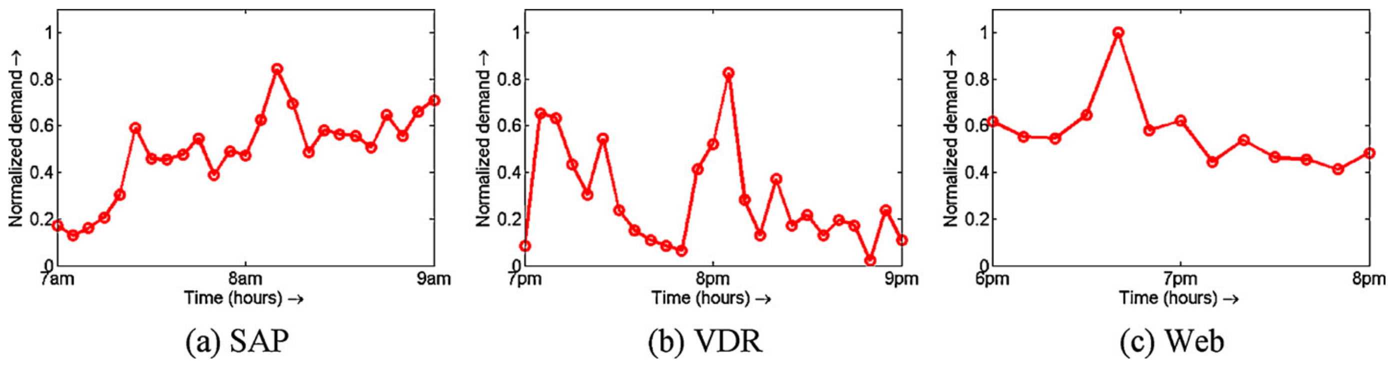


Fig. 3. Workload demand for 2 h for (a) the SAP trace; (b) the VDR trace; and (c) the Web trace.

The workload variability can be clearly seen in Fig. 3. Each graph shows periods where demand values exhibit large local variations. For example, the SAP trace (Fig. 3(a)) shows an initial demand of 0.2 at 7:15 am, which increases to 0.7 by 8:10 am, then decreases to 0.4 by 8:20 am. Even consecutive demand values vary by a factor as much as 2–3. The VDR and Web traces (Fig. 3(b) and (c)) show similar variability. This leads to our second observation:

Observation 2: *Workload demands can change abruptly during short intervals.* An important implication of this is that to handle such variations, a provisioning mechanism is required at short timescales.

2.3. Periodicity

Though there is large variability in workload demands, most workloads and in particular interactive workloads, exhibit clear short-term and long-term patterns that help in describing their behavior.

To capture these patterns, we perform a *periodicity analysis* of the workload demand traces to reveal the length of a pattern or a sequence of patterns that appear periodically. We use Fast Fourier Transform (FFT) to find the periodogram of the time-series data [5]. From this we derive periods of the most prominent patterns or sequences of patterns.

Fig. 4(a) plots the time-series and the periodogram for five days of the SAP trace. The peak at 24 h in the periodogram indicates that the SAP trace has a strong daily pattern (period of 24 h).

Likewise, Fig. 4(b) and (c) plots the time-series and the periodogram for five days of CPU demand collected from the VDR and Web application, respectively. The time-series plots and periodograms indicate that both the VDR trace and the Web trace also have a strong daily pattern.

Observation 3: *The workloads exhibit prominent daily patterns.* Based on this, we now consider how to identify and discretize such patterns.

3. Workload discretization

In Section 2, we showed that IT workloads often exhibit daily periodic patterns with a small percentage of noise (e.g., frequent and sporadic spikes). This section presents novel techniques to identify and discretize such patterns in historical workloads traces.

Our algorithms partition the workload demand into disjoint time intervals, such that within each time interval, the demand varies only slightly. A single representative demand value (e.g., mean or 90th percentile demand) is then chosen for each interval. We propose a dynamic programming algorithm to find a small number of time intervals and representative demand for each, such that the deviation from real demand is minimized. Keeping the number of intervals small is important, as more intervals imply more frequent provisioning changes and thus higher risks and costs. Compared to fixed-sized intervals used in existing solutions, variable-sized intervals describe the workload demands better. We also consider the provisioning cost, which is ignored by existing approaches.

3.1. Discretization

Our goal is to represent the daily pattern in workloads by discretizing their demands into consecutive, disjoint time-intervals with a single representative demand value in each interval. We now formally define discretization.

Workload discretization: *Given the demand time-series X on the domain $[s, t]$, a time-series Y on the same domain is a workload characterization of X if $[s, t]$ can be partitioned into n successive disjoint time intervals, $\{[s, t_1], [t_1, t_2], \ldots, [t_{n-1}, t]\}$, such that $X(j) = r_i$, some fixed demand, for all j in the i^{th} interval, $[t_{i-1}, t_i]$.*

Note that, by definition, any time-series, X, is a discretization of itself. For our purposes, we set $s = 0$ and t = period of workload. In the follow discussion, we assume a period of 24 h, based on our periodicity analysis.

The idea behind discretization is two-fold. First, we want to accurately capture the demand. To achieve this, the representative values, r_i, for each interval, $[t_{i-1}, t_i]$, should be as close as possible to the actual demand values of the time-series in the interval $[t_{i-1}, t_i]$. We determine the demand estimation error incurred by discretization, which we define as the percentage of the area between the original time-series and the discretized time-series with respect to the area under the original time-series.

Second, provisioning of IT resources is not free [8]. For this reason, we want to avoid having too many intervals and hence too many changes to the system, as this is not practical and can lead to many problems. For example, turning on additional servers to handle increasing demand takes time (up to 2 min in our experiments) and consumes substantial amounts of power during boot up [12]. As a result, frequent provisioning changes incur performance and energy penalties and lead to wear and tear of servers in the long run. Too many changes can also result in system instability.

In summary, the discretization should be as close as possible to the original time-series and should be represented by as few constant pieces as possible. In other words, we want to minimize the estimation error *and* the number of intervals in the discretization.

3.2. Fixed interval approach

The easiest way to discretize the daily demand pattern is to use a single interval, [0,86400 s] (24 h), and a single representative value, r, for that interval. The question then is what r should

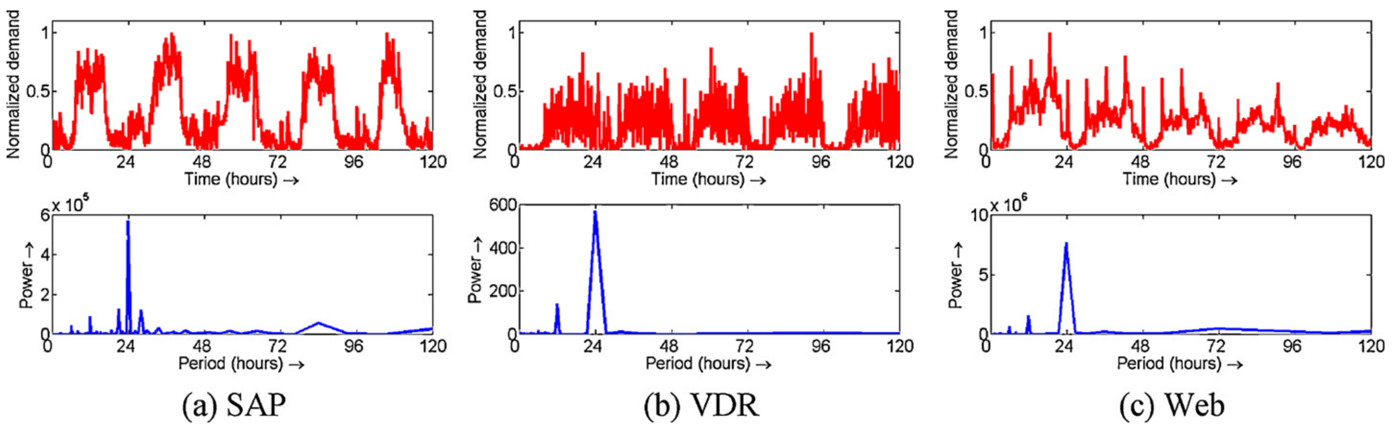


Fig. 4. Time-series and periodogram for (a) the SAP trace; (b) the VDR trace; and (c) the Web trace.

be. Fig. 5(a)–(c) plots the single interval discretization with r being the maximum (Max), 90th percentile (90%ile) and Mean, respectively, for the SAP trace. The graphs show that all three approaches perform poorly. Max and 90%ile significantly over-estimate the demand, whereas Mean incurs regions where the demand is over-estimated and regions where it is under-estimated. The errors of Max, 90%ile and Mean are 191%, 137% and 82%, respectively. This is expected, since the workload is bursty; thus a static size will not fit.

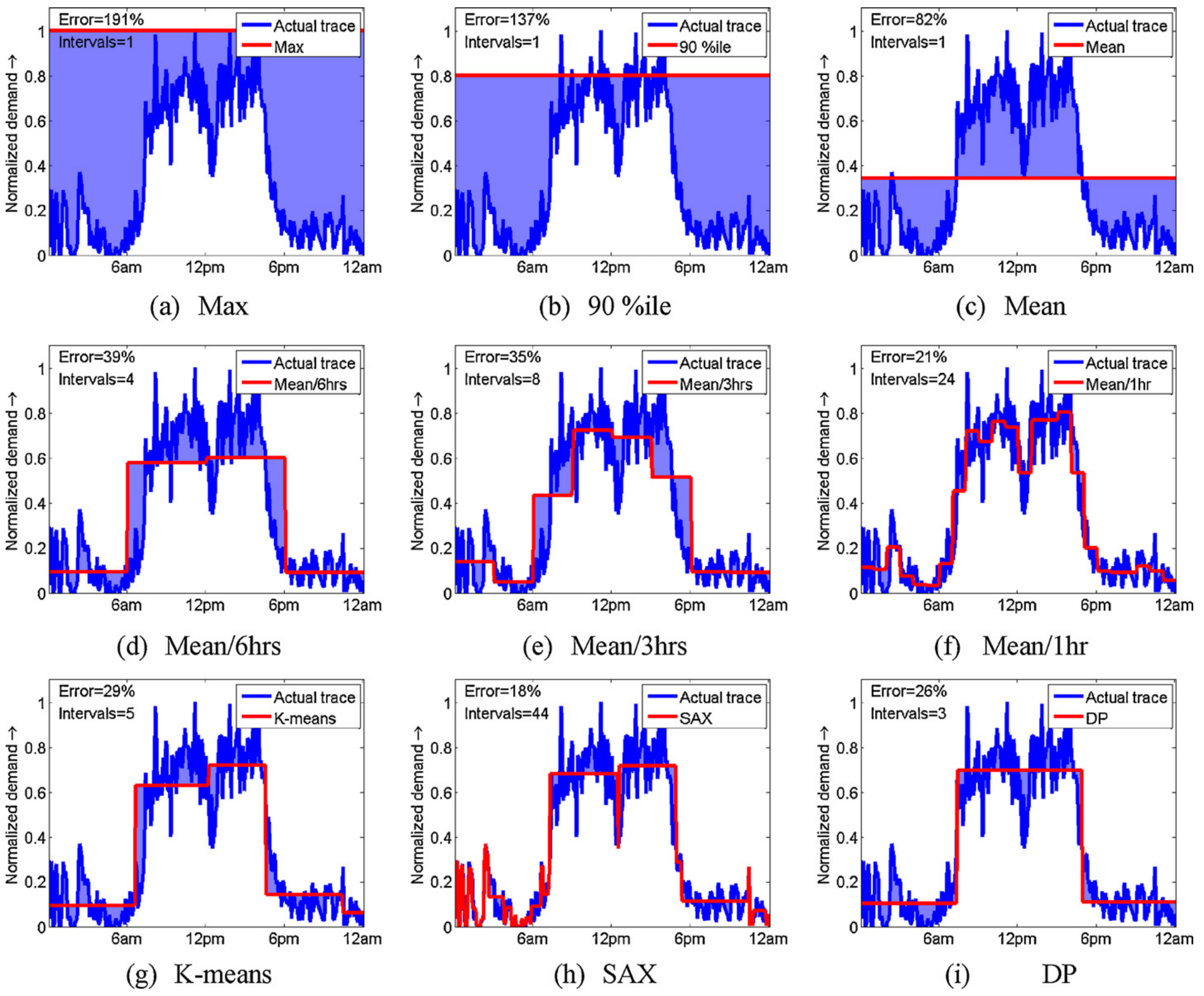


Fig. 5. Discretization for the SAP trace.

The obvious next step is to consider multiple interval approaches similar to piecewise aggregate approximation (PAA) [30]. Fig. 5(d)–(f) plots the discretization results for interval lengths of 6 h, 3 h and 1 h, respectively. We use Mean as the representative demand in each interval. As expected, the provisioning error goes down as the interval size decreases. However, decreasing the interval size increases the number of intervals, which could increase the number of provisioning changes. As previously discussed, this is undesirable. Thus, there is a clear trade-off between provisioning error and the number of intervals. We observed similar results for the VDR trace and the Web trace.

3.3. Variable interval approach

Owing to the limitations of the fixed interval approach, we now consider variable-sized intervals. The motivation behind variable sized intervals is that a change in demand can happen at any point of time, and not just at specific hours. Thus, by using variable-sized intervals, we can capture the arbitrary moments in time when the demand changes.

3.3.1. K-means

K-means is a clustering approach that discretizes the workload demand values into k clusters. For example, when $k = 3$, each workload demand reading is classified into high, medium, or low. *K*-means is widely used in data mining for cluster analysis. It works by partitions the given data into k clusters, such that the total sum, over all k clusters, of the difference between a data point and the mean of the cluster, is minimized. Mathematically, given a set of n data points, $X = \{x_1, x_2, \ldots, x_n\}$, *K*-means aims to partition X into k sets, $S_1, S_2, \ldots, S_k$, so as to minimize:

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2, \tag{1}$$

where μ_i is the mean of points that belong to set S_i.

In doing so, each data point is classified into the group it most closely resembles. In our case, by using *K*-means, we can classify our demand values into k levels, such that each data point can be associated with one of these levels.

While *K*-means is very useful for detecting clusters of similar values, it is not obvious how we can best use *K*-means to discretize the workload. In our experiments, we first use *K*-means to come up with k demand levels. We then walk through the demand trace, and partition it into intervals such that most of the points within an interval have the same load level. If a sequence of points with the same load level is followed by another sequence of points with a different load level, then we classify the respective sequences into different intervals. Once the entire workload trace is partitioned into intervals, we use Mean as the representative demand for each interval. This gives us the discretization for the workload trace using *K*-means.

Fig. 5(g) plots the *K*-means discretization for the SAP trace, with $k = 3$. We see that the *K*-means approach obtains a lower error than the "Mean/3 h" approach and uses fewer intervals. This clearly shows that allowing the interval length to vary helps in lowering the estimation error. We examined other values of k as well, but found that $k = 3$ provides the best tradeoff between estimation error and number of intervals.

3.3.2. SAX

SAX [17] is another discretization approach, specifically for time-series data. While SAX is widely used for compressing time-series data, it is also very popular for cluster analysis. In SAX (Symbolic Aggregate approXimation), the time-series domain (24 h in our case) is first partitioned into equal-sized intervals, similar to the fixed interval approach. For example, by choosing an interval size of 1 h, SAX partitions the time-series domain into 24 one hour intervals. Next, SAX computes the mean value for each of these intervals. Finally, these mean values are clustered, e.g., using *K*-means. Each of these clusters is then represented by a symbol, e.g., an alphabet, and the whole time-series is now approximated by a concatenation of these symbols.

Our primary goal is to partition the time-series into a few intervals, so as to minimize the estimation error. Thus, we can use SAX to determine the clusters for our workload trace. These clusters provide us with the demand levels for the trace. We now discretize the workload trace using these demand levels. Specifically, we start by converting each cluster into a single interval and then use Mean to represent the value of that interval. We now walk through the trace and if successive intervals have the same mean value, we group them into a single interval. We repeat the above process until we converge to a minimum number of intervals. At this point, the resulting partitioning of our trace into intervals forms our workload discretization using SAX.

Fig. 5(h) plots the SAX discretization for the SAP trace. We see that while SAX has relatively low estimation error, it results in 44 intervals. This is because SAX is not tailored to minimize the number of intervals. Thus, while SAX accurately estimates the demand, it is not practical for use in resource provisioning, as 44 intervals means 43 changes.

3.3.3. Dynamic programming

Thus far we have seen approaches that provide a clear trade-off between the provisioning error and the number of changes. This section presents our novel Dynamic Programming (DP) approach to discretize the time-series such that we reduce both.

$$\sum_{i=1}^{n} \sum_{t=t_{i-1}}^{t_i} (X(t) - r_i)^2 + f(n). \tag{2}$$

Eq. (2) is the objective function we want to minimize, where X is the time-series and $f(n)$ is a cost function of the number of changes or intervals, n. The goal of Eq. (2) is to simultaneously minimize the workload representation error and the number of changes. In some cases, one might prefer to minimize the square of the number of changes (or some other function of the number of changes). For this paper, we set, where c is a normalization constant, which we will discuss later. The objective function expresses the goal as minimizing the normalized number of changes and the representation error. The Mean-squared error is used to quantify the workload representation error. Note that in the most general case, both the number of changes and the representational error could be formulated as utility functions. We use dynamic programming, which is well suited to minimize the objective function given in Eq. (2). Minimizing the Mean-squared error for a given partition results in setting r_i to be the mean of the time-series values on that partition. That is:

$$\begin{aligned} &\min . \sum_{t=t_{i-1}}^{t_i} (X(t) - r_i)^2 \\ &\Rightarrow \frac{d}{dr_i} \sum_{t=t_{i-1}}^{t_i} (X(t) - r_i)^2 = 0 \cdot \\ &\Rightarrow r_i = \frac{\sum_{t=t_{i-1}}^{t_i} X(t)}{t_i - t_{i-1} + 1} \end{aligned} \tag{3}$$

Thus, we only need to partition the time-series domain, $[t_0 = 0, t_n = 24\,\text{h}]$, to optimize the objective function. Let us assume that we have the optimal partition of the domain $[t_0, t_n]$, and it is $\{[t_0, t_1], [t_1, t_2], \ldots, [t_{n-1}, t_n]\}$. Now, consider the optimal solution for the domain $[t_0, t_{n-1}]$. We claim that this is simply $\{[t_0, t_1], [t_1, t_2], \ldots, [t_{n-2}, t_{n-1}]\}$. This is because if the optimal solution for the domain $[t_0, t_{n-1}]$ was different, then we could just append the partition $[t_{n-1}, t_n]$ to it, and that should give us a solution for the domain $[t_0, t_n]$ with a lower objective function value than $\{[t_0, t_1], [t_1, t_2], \ldots, [t_{n-1}, t_n]\}$, which is a contradiction. Thus, the optimal solution for the domain $[t_0, t_n]$ contains the optimal solution for the domain $[t_0, t_{n-1}]$. Thus, this problem exhibits the optimal substructure property, and dynamic programming will result in an optimal solution. Note that even in the case of highly irregular traffic where no patterns can be found, the dynamic programming solution will still output a workload characterization. In particular, the workload characterization will simply be a single value, e.g., the mean or desired percentile of the observed demand, over the entire domain $[t_0, t_n]$. Thus, the dynamic programming solution is robust to irregularities in the demand pattern.

We now mention a rule of thumb to pick the normalization constant c. Note that the single interval approach with Mean as the representative value gives our objective function a value of:

$$Var = \sum_{t=t_0}^{t_n} [X(t) - \text{Mean}(X)]^2, \tag{4}$$

where Var is the variance of the time-series under consideration, X. By allowing a finite number of changes, say z, we want to ensure that we can reduce the representation error in our objective function by at least $Var/2$. Further, this new objective function must be smaller than Var. Thus, we want:

$$\frac{Var}{2} + c \cdot z < Var \Rightarrow c < \frac{Var}{2 \cdot z}, \tag{5}$$

Thus, for example, setting $z = 2$ results in $c < Var/4$. For simplicity, we set $c = Var/4$ in our dynamic programming solutions. In general, a larger value of z may result in a smaller representation error at the cost of an increase in the number of changes, whereas a smaller value of z may result in a larger representation error at the cost of a decrease in the number of changes. The choice of z in a real data center will depend on how often the system administrator is willing to power cycle the servers. We expect a power cycling frequency of once every 4–6 h to be acceptable. Thus, $z < 5$ should be an acceptable value for a 24 h demand trace.

Fig. 5(i) plots the discretization of the SAP trace using our DP approach. The results show that the DP approach achieves a lower error than K-means with fewer intervals. Compared to the SAX approach the error is slightly higher, but the number of intervals is much smaller (3 versus 44). We observe similar results for the VDR and Web traces. That is, the DP approach accurately estimates the demand using fewer intervals. This makes the DP approach a more practical technique for resource provisioning.

4. Hybrid provisioning

Sections 2 and 3 show that although workload demands are bursty, there are predictable patterns in them. Resource provisioning based on these patterns can improve accuracy significantly. However, there can still be some deviations from these patterns due to the bursty nature of data center workload demands. In particular, excess workload demand or a sudden spike in demand can cause performance problems. Also, there is a cost and risk associated with provisioning changes, and thus, we want to avoid frequent provisioning changes. Based on these observations, we propose a *Hybrid Provisioning solution*, which combines predictive provisioning with reactive provisioning.

The intuition behind our approach is that we capture periodic and sustained workload patterns from historical data, which we refer to as the base workload and then proactively (and predictively) provision for them. At run time, the deviation between the actual and predicted base workloads, which we refer to as the noise workload, is handled by reactive provisioning, which is invoked each time the request rate exceeds our prediction. Our solution also takes into account the costs and risks associated with provisioning.

4.1. Server farm provisioning

While our approach can be applied to general capacity planning and resource allocation problems, we consider a class of very important and popular applications in today's data centers: Web or enterprise applications which are hosted by a large pool of servers (e.g., tens of thousands of servers). Examples of such applications include Web search, online shopping, social networking services and business transactional services. Millions of users interact with such applications, by sending requests to the servers hosting the applications. Each application has an associated response time target that needs to be met. The actual performance of an application is determined by the resource capacity allocated to the application, i.e., the number of servers, and the workload intensity (or request rate). Given the response time target for an application, our goal is to determine the optimal number of servers for the application to meet its target without over-provisioning resources and thus wasting power.

4.2. Overview of hybrid provisioning

Fig. 6 depicts the conceptual architecture for how to combine predictive provisioning and reactive provisioning for server farms. Our solution comprises of four components.

1. A *base workload forecaster* analyzes historical workload traces (Fig. 6(a)) and identifies the patterns that form the base workload. We represent the patterns in a workload by discretizing it into consecutive, disjoint time intervals with a single representative demand value (e.g., mean or 90th percentile demand) in each interval (Fig. 6(b)).
2. A *predictive controller* proactively estimates and allocates the proper amount of capacity (e.g., the number of servers) needed to handle the base workload. For example, given the base demand (e.g., time varying request rate in Fig. 6(b)), it generates the capacity allocation (e.g., number of servers in Fig. 6(c)).
3. A *reactive controller* handles excess demand by adding additional capacity at short time scales (e.g., # of servers in Fig. 6(d)), in response to excess workload that is beyond the base workload, i.e., the difference between the actual workload and forecasted base workload.
4. A *coordinator* dispatches workload requests to servers and also communicates with the predictive and reactive controllers to provide information about incoming demand. When the actual workload (e.g., requests in Fig. 6(e)) arrives, the dispatcher assigns the portion of the workload not exceeding the base workload (e.g., requests in Fig. 6(f)) to resource pool 1 and any excess workload (e.g., requests in Fig. 6(g)) to resource pool.

The next few subsections describe the various components.

4.3. Base workload forecaster

The *base workload forecaster* first performs a periodicity analysis of the workload demand to reveal the length of a pattern or a sequence of patterns that appear periodically. We use Fast Fourier Transform (FFT) to find the periodogram of the time-series data [5] from historical data. From this, we derive the periods of the most prominent patterns. For example, the periodograms from Fig. 4

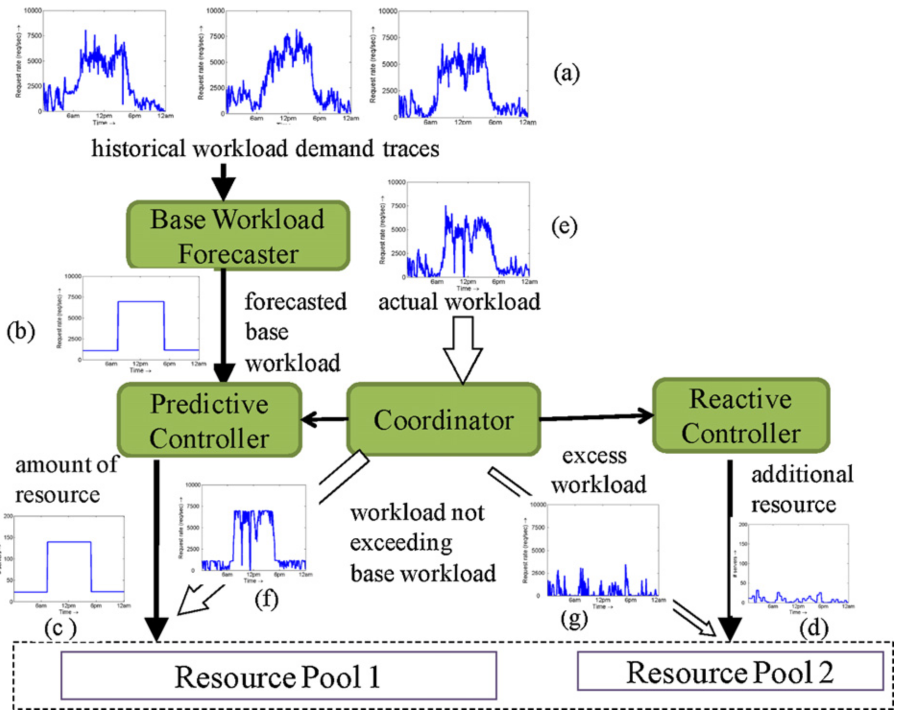


Fig. 6. Hybrid provisioning.

reveal that our traces have a strong daily pattern (period of 24 h). The base workload forecaster runs periodically (e.g., once a day). It divides the historical data into daily demands and forecasts the demand for the next day. To forecast the workload, we can just take the average of the mean historical daily demand. Advanced forecast techniques can be used, but are out of the scope of this paper [13]. Next, we identify and discretize patterns in the forecasted workload demand. Our goal is to represent the daily pattern in workloads by discretizing their demands into consecutive, disjoint time-intervals with a single representative demand value in each interval. As discussed in Section 3, we want to find a small number of time intervals and representative demand for each, such that the deviation from actual demand is minimized. Keeping the number of intervals small is important, as more intervals imply more frequent provisioning changes and thus higher risks and costs. Thus, we use our dynamic programming workload discretization approach from Section 3.3.3 on the mean historical daily demand to get the base workload.

4.4. Predictive controller

The *predictive controller* is responsible for handling the base workload. It receives the predicted base workload pattern (i.e., the output of discretization) from the *base workload forecaster* and then proactively estimates and allocates the proper amount of capacity required to handle the base workload. In particular, the controller uses a queueing performance model to determine how much capacity will be allocated to ensure that the SLA requirements are met for the forecasted demand without consuming excessive power.

We assume that a certain mean response time target (SLA requirement), t_0, in seconds, is given. Recall that our goal is to minimize power consumption while ensuring that we meet the mean response time target. We use an M/G/1/PS queuing model to estimate the number of servers needed. Assuming that demand follows a Poisson distribution with a mean request rate λ requests/s, we have:

$$\frac{1}{(1/s)-(\lambda/k)} < t_0, \tag{6}$$

where s is the mean job size in seconds, and k is the number of servers. From Eq. (6), we derive:

$$k = \left\lceil \frac{\lambda}{(1/s)-(1/t_0)} \right\rceil. \tag{7}$$

Eq. (7) estimates the number of servers needed to ensure that the mean response time target is met. While the assumptions behind Eq. (7) may not be ideal for real-world data center workloads, they provide a good approximation, as we will see in Section 5.2. Though important, performance and capacity modeling is not the focus of this paper. Note that the capacity allocated by the *predictive controller* is not affected by actual demand and will not change until a new base workload forecast arrives.

4.5. Reactive controller

Though many application workloads have predictable time varying demands, actual demands are statistical in nature and are likely to differ from the forecast. For example, the demand for the target day may vary from the historical demand due to unforeseen factors such as flash crowds, service outages, holidays, etc. Thus, we integrate a reactive controller to handle the noise in the demand. A reactive controller is invoked each time the actual workload demand is higher than the base workload, to provide additional resources for the excess workload demand. Since the reactive controller is not invoked when the actual demand is lower than the base workload forecast, the impact of over-provisioning is minimized if the predictive provisioning captures the base demand well. The results in Section 5 show that this hybrid approach works very well in practice.

We use a simple feedback approach with a fixed monitoring interval length to estimate the amount of noise in the workload. For example, if the monitoring interval length is 10 min, then we estimate the noise (workload demand above the predicted base demand) in the next 10 min interval to be the same as the noise in the current 10 min interval [6]. While more sophisticated approaches, such as ARMA or moving window [9], can be used for noise estimation, we find that a simple feedback based approach is very efficient and works well in practice for noise estimation. Note that errors in noise estimation will only affect the noise provisioning; the base workload demand will not be affected.

4.6. Coordinator

We logically divide a server pool into two partitions. One handles the base workload and is managed by the predictive controller. The other handles the noise (excess) workload and is managed by the reactive controller. These two server farms can (and probably will) physically co-exist. The coordinator forwards incoming requests to either the server farm dedicated to the base workload or the server farm dedicated to the noise (excess) workload, based on the predicted base demand and the actual demand. A simple scheme for dispatching requests is to simply load-balance the incoming requests among all servers irrespective of which server farm they belong to. Under this scheme, all jobs are treated equally. However, one can imagine a more sophisticated dispatching scheme which allows certain important jobs to receive preferential service over other jobs by dispatching the important jobs to the (more robust) base workload server farm. For example, for e-commerce sites such as Amazon or eBay, it might make more sense to prioritize shopping requests over browsing requests. In such cases, shopping requests can be dispatched to the server farm that handles the base workload. This server farm is less likely to incur provisioning changes and can thus provide uninterrupted service. The less important browsing requests can now be dispatched to either the base workload server farm, if there is available capacity, or to the noise workload server farm.

We now discuss the above dispatching schemes in more detail.

4.6.1. Load balancing dispatcher

For this scheduling scheme, the number of servers is provisioned based on the output of the predictive and reactive provisioning as described in Sections 4.4 and 4.5, respectively. Then, we simply load balance the incoming requests across the available servers. Thus, using Eq. (7), if the predictive provisioning sets n_1 servers for the base workload and the reactive provisioning sets n_2 servers for the noise workload, then we simply load balance the incoming jobs among the $(n_1 + n_2)$ servers.

4.6.2. Priority dispatcher

This dispatcher ensures that the base workload receives guaranteed performance by filtering the request rate into the base workload server farm and sending excess requests to the server farm dedicated for the noise. The priority scheduler forwards and load balances the job requests to the base provisioning servers (the n_1 servers in the previous example) as long as the request rate is below the forecasted base workload request rate. However, requests that exceed the forecasted request rate are forwarded to the reactive provisioning servers (the n_2 servers in the previous example). This can be achieved by monitoring the request rate every few seconds, and re-routing the incoming requests to the respective servers as required. For example, if the incoming request rate in the last monitoring interval was above the forecasted request rate by 10%, then we forward 10% of the incoming requests to the reactive provisioning servers. Compared to load balancing scheduling, this solution isolates the base workload from the noise workload, and provides stronger performance guarantees for the base workload.

5. Evaluation

This section evaluates our workload discretization and hybrid provisioning approaches via analytical trace analysis in Section 5.1 and via experimentation on a real test bed in Section 5.2.

We first present trace-based analysis results for different application traces and show that our hybrid server provisioning approach is superior to existing provisioning approaches. We then present our experimental results based on a real test bed implementation, which resemble our analysis results, and thus validate our trace-based analysis findings. We obtain workload demand traces from three real applications in HP data centers that were described in Section 2.1 and the WorldCup98 demand trace from the Internet Traffic Archives [31]. The first three traces exhibit prominent daily patterns but also vary significantly during the course of a single day. The WorldCup98 trace is less predictable, with peak demands influenced by factors such as the outcome of matches, which are difficult to predict.

5.1. Trace-based analysis

5.1.1. Methodology

This section evaluates our hybrid approach on different workload traces via an analytical approach. In particular, given a trace, we apply different workload discretization techniques from Section 3 on historical workload demand (e.g., days 1–7) to predict workload demand for a target day (e.g., day 8). Further, we consider different provisioning policies, predictive, reactive and hybrid, to provision resources for a single tier Web server farm. We assume each server has a fixed capacity and the number of servers required is the predicted demand divided by the server capacity. Lastly, we compare the different provisioning policies using the following metrics: percentage of SLA violations, total power consumption, and the number of provisioning changes.

If the actual demand for an interval exceeds the allocated capacity, then that interval is under-provisioned and is counted as an SLA violation. Note that interval here refers to the monitoring interval of the original workload traces. We use a linear model to estimate the power consumption of a server. The model expresses the power consumption, P, of a server that is x% utilized as:

$$P = P_{idle}\left(\frac{x}{100}\cdot(P_{max} - P_{idle})\right), \tag{8}$$

where P_{idle} and P_{max} are the power consumption of the server when it is idle and when it is 100% utilized, respectively. P_{idle} and P_{max} are derived experimentally (described in Section 5.2.1). As requests are load-balanced across servers, the utilization of each server is expected to be equal. Finally, we track the number of provisioning changes, i.e., number of times we change the server farm capacity, throughout the target day.

5.1.2. Provisioning policies

We now briefly describe the different provisioning policies. The *Predictive* policy provisions servers for the target day based on the 90%ile of the demand values for the past seven days. We consider four provisioning intervals for the predictive policy: 24 h, 6 h, 1 h and variable length intervals. For example, the *Predictive/1 h* approach considers the corresponding 1 h demand trace portions from each of the past 7 days and uses the 90%ile demand of these portions as the representative demand value for that hour of the target day. We note that *Predictive/24 h* represents static resource provisioning, i.e., no changes to the resource allocation are made during the day. The *Predictive/var* approach uses variable intervals from the workload discretization. The *Reactive* approach monitors the demand for 10 min and uses a feedback approach to provision servers for the next 10 min assuming the demand to remain

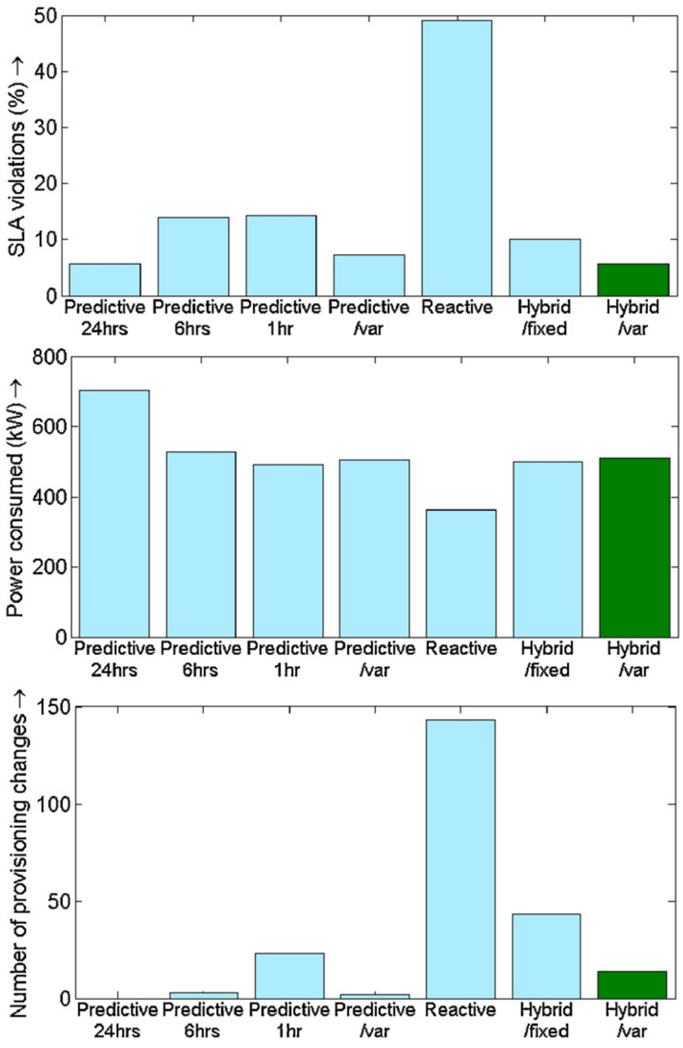


Fig. 7. Trace-based analysis results for the SAP trace showing the SLA violations, power consumption and number of provisioning changes.

the same as in the last interval. We study two hybrid approaches: the *Hybrid*/*fixed* policy does predictive provisioning every hour and invokes reactive provisioning each time the predictive provisioning underestimates demand; and the *Hybrid*/*var* approach that combines predictive provisioning with variable intervals and reactive provisioning as described in Section 3. We choose the 90%ile demand for all predictive approaches since the 90%ile provides a good tradeoff between resource consumption and quality. It leads to significant savings with relatively few SLA violations, as shown in Sections 2 and 3.

5.1.3. Results

Fig. 7 shows the SLA violations, power consumption and number of provisioning changes for the SAP trace. First, among the four predictive approaches, using variable intervals achieves the best balance of performance and power. The percentage of violations is comparable to the *Predictive*/24 *h* approach, but the power consumption is 30% less. The power consumption of *Predictive*/*var* is similar to that of *Predictive*/6 *h* and *Predictive*/1 *h*, but the performance is much better and the number of provisioning changes is smaller. We note that reducing the interval length, i.e., more frequent provisioning, helps to reduce power but increases SLA violations.

Second, our *Hybrid*/*var* approach out-performs all predictive approaches. Compared to *Predictive*/*var*, *Hybrid*/*var* reduces SLA violations from 7.3% to 5.6% while using a similar amount of power. The reduction in SLA violations results from the integration of the reactive provisioning to handle demand exceeding the predicted base demand.

Third, both hybrid approaches perform significantly better than the *Reactive* approach in terms of meeting SLA requirements. The *Reactive* approach has the highest percentage of violations. There are two reasons for this. First, the workload is highly variable and abrupt changes in short intervals are difficult to track. Second, it takes time to set up a server and thus the *Reactive* approach cannot react faster than the server setup time. Between the two hybrid approaches, our *Hybrid*/*var* approach outperforms *Hybrid*/*fixed*. In particular, by using variable intervals for predictive provisioning, *Hybrid*/*var* reduces SLA violations from 10.1% to 5.6% and the number of changes from 43 to 14. At the same time, power consumption stays the same. This further shows the effectiveness of our DP based workload discretization algorithm.

In summary, compared to purely predictive and reactive approaches, our *Hybrid*/*var* approach achieves better performance and conserves power. The *Hybrid*/*var* approach consumes 27% less power than the *Predictive*/24 *h* approach, which has similar performance. Compared to the *Reactive* approach, the *Hybrid*/*var* reduces SLA violations from 50% to 5.6%. *Hybrid*/*var* outperforms a hybrid approach with fixed provisioning intervals in terms of SLA violations and power consumption. Finally, *Hybrid* incurs 14 changes, less than one every hour, which is considered acceptable in practice.

To further understand how these different approaches allocate resources in response to workload demand changes, we show time series of the demand, the number of SLA violations per hour, the power consumption, and the number of active servers in Fig. 8. We see that the *Reactive* approach makes many provisioning changes and incurs many violations, since a change in provisioning is triggered by either an under-provisioning or an over-provisioning in the previous interval. The figure shows that our *Hybrid*/*var* approach captures most of the incoming workload demand. Thus, only a few changes are triggered by the reactive part of our approach. However, there are instances when the reactive controller is invoked, for example, around 2–3 am. This again justifies that a hybrid approach is required to handle real-world workload traces.

We conduct a similar analysis study for the Web application trace. The results for the Web trace are shown in Fig. 9. For the Web trace, shorter intervals for predictive provisioning reduce the SLA violations and power consumption, which is different from the results of the SAP trace. However, the *Predictive*/*var* approach is still among the best of all four predictive approaches. Also, importantly, our *Hybrid*/*var* approach provides the best tradeoff between SLA violations, power consumption and the number of provisioning changes among all policies.

In Section 2.3, we observed that the SAP, VDR and Web traces all had strong daily patterns. To explore the robustness of our hybrid provisioning technique, we now analyze traces of the WorldCup98 Web site. A characterization presented in [3] shows that this workload was less predictable, with peak demands influenced by factors such as the outcome of matches, which could not be predicted. In this workload, the peak demands exceeded the average demand per minute by a factor of up to 19. Our analysis results using the WorldCup98 trace are shown in Fig. 10. As with the SAP, VDR and Web traces, shorter intervals for predictive provisioning reduce the SLA violations and power consumption. The reactive approach suffers the most SLA violations and causes the most provisioning changes but consumes the least amount of power. The *Hybrid*/*var* approach incurs fewer SLA violations than all of the other approaches except *Hybrid*/*fixed*. However, *Hybrid*/*fixed* triggers about 5 times more provisioning changes than *Hybrid*/*var*. In summary, we conclude that the *Hybrid*/*var* method is reasonably robust to unpredicted

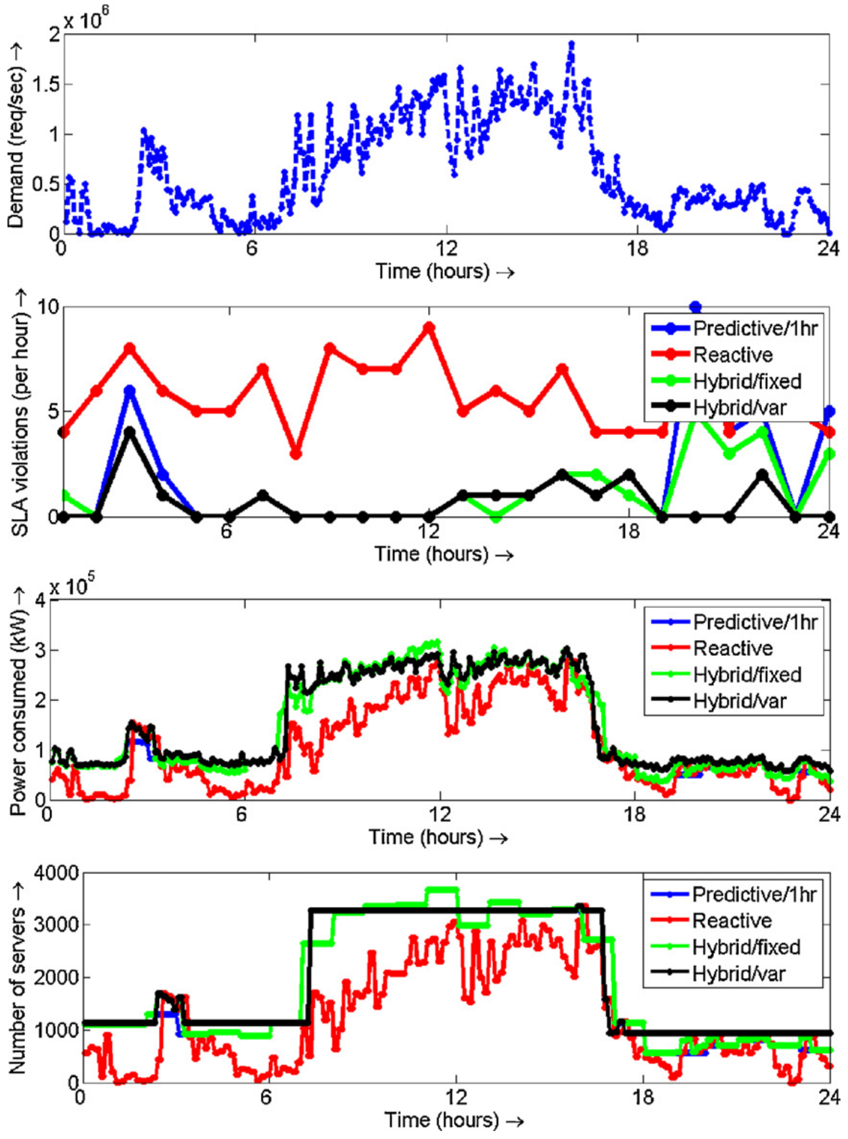


Fig. 8. Time-series for the demand, SLA violations, power consumption and the number of servers for the SAP trace.

fluctuations in the demand, and provides the best tradeoff between SLA violations, power consumption and number of provisioning changes.

5.2. Experimental results on a real testbed

The trace-based analytical results show the theoretical benefits of the *Hybrid* approach over existing server provisioning policies. To verify our claims and ensure that we have a practical and robust solution, we implement a provisioning system that dynamically allocates the number of active servers for a server farm that processes jobs or transaction requests. We validate our *Hybrid* approach, in particular the *Hybrid/var* policy, against other policies using the real-world workload demand traces described in Section 2.1.

5.2.1. Experimental setup

We set up a test bed of a single-tier server farm application. Our test bed includes 10 blade servers. Each server has two Intel Xeon E5535 quad-core processors running at 2.66 GHz, and 16 GB of memory. One server was used as a front-end load generator running httperf [20]. Another server was employed as a proxy/load-balancer, which distributes jobs and requests among the remaining eight Apache web servers.

We use httperf to replay the workload demand traces in the form of CGI requests. We first take the real trace data and scale the workload demand suitably to match our server farm capacity, since most of the traces were collected from production systems running thousands of servers. We then use httperf to replay the CGI requests according to the scaled request rate. Note that individual requests are generated according to the request rate defined in the trace file and the workload generation is independent of the web server setup and performance. In particular, the sequence and rate of requests generated via httperf is independent of the server provisioning policy employed. In the experiments, when different provisioning policies are compared, all the executions are driven by the same trace file at the same rates and hence the workload profiles for all executions are exactly the same. We use two trace files in our experiments: Web and VDR. Fig. 11 shows the time varying request rates for these traces for the test day.

Each request is directed, via the load-balancer, to one of the eight back-end machines. Business logic processing in enterprise and Web applications is often the bottleneck and the workloads are often processor intensive. The CGI script runs LINPACK [19], a

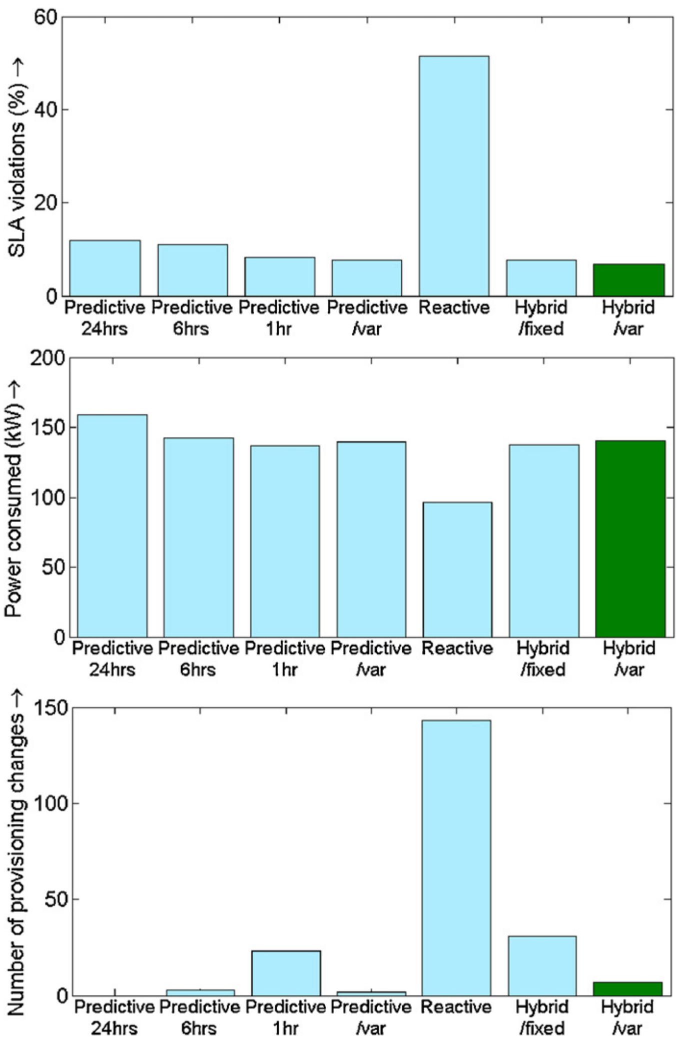


Fig. 9. Trace-based analysis results for the Web trace showing the SLA violations, power consumption and number of provisioning changes.

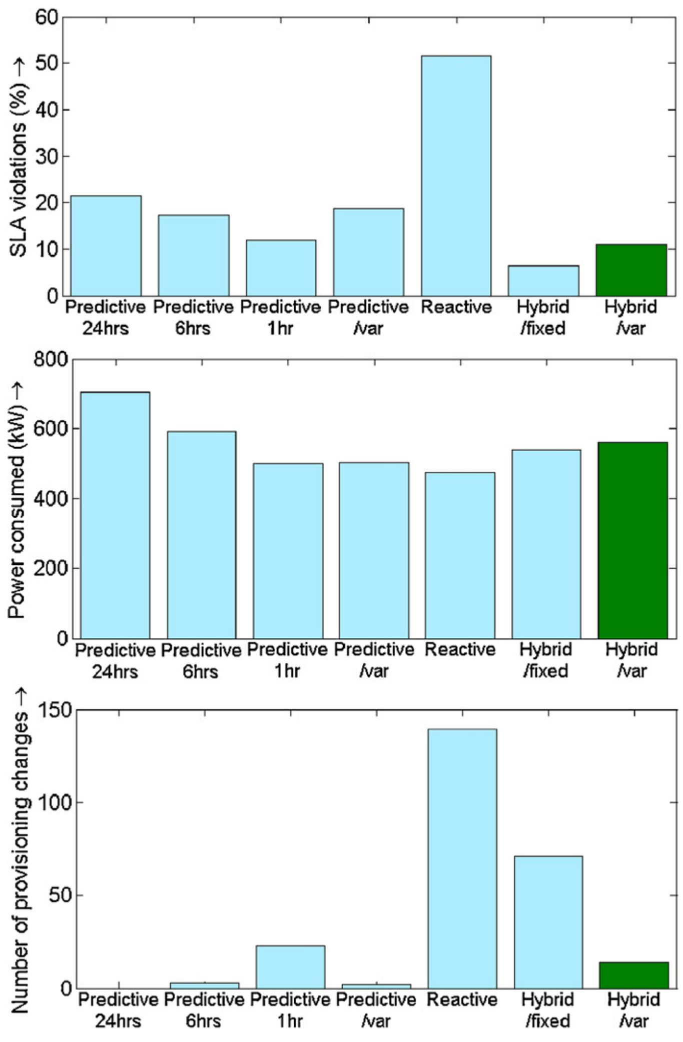


Fig. 10. Trace-based analysis results for the World Cup 98 trace showing the SLA violations, power consumption and number of provisioning changes.

multi-threaded CPU bound program, to simulate the request or job processing in real applications. By adjusting the LINPACK job sizes, we can easily model the variability of service times in real applications [11]. We use the blade servers' built-in management processor to turn servers on and off remotely and programmatically, and to obtain server power consumption readings. We collect response times from httperf logs.

5.2.2. Results

We evaluate four different polices: *Predictive Mean*/1 h, *Predictive 90%ile*/1 h, *Reactive* and our *Hybrid* approach (*Hybrid*/*var*). The predictive provisioning policies use the corresponding 1 h demand

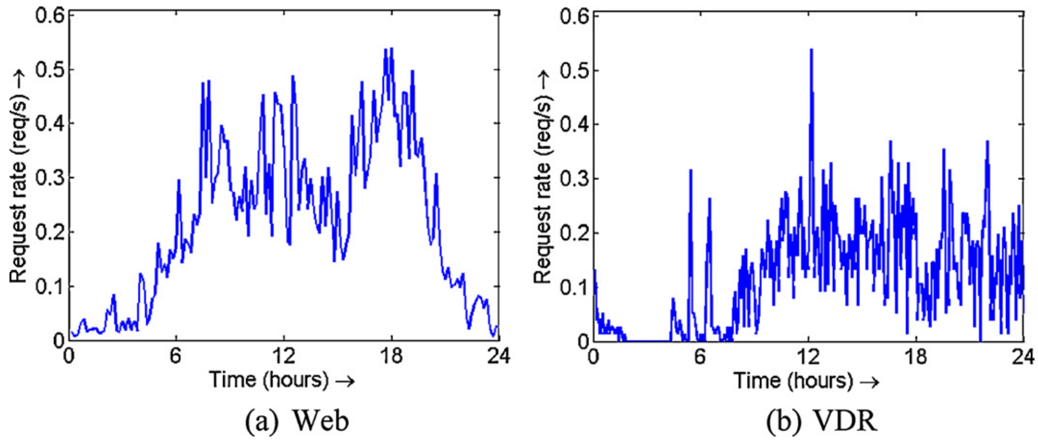


Fig. 11. Request rate traces for (a) Web and (b) VDR used in our experiments.

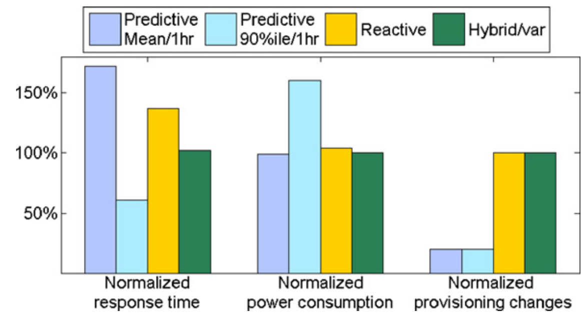


Fig. 12. Experimental results for the Web trace.

trace portions from the past days (Mean and 90%ile respectively) to predict the demand for the hour of the target day. Reactive provisioning invokes the reactive controller every 10 min and employs a simple feedback model that provisions servers for the next 10 min. For all policies and a given response time target, the number of active back-end servers is determined by the performance model described in Section 4.4.

This section shows experimental results based on the Web and VDR trace demands.

(a) Web trace: We conducted an experiment using the normalized workload demand for the Web trace. We scaled the demand trace to ensure that the 8 back-end servers can handle the peak demand without violating the response time SLA. Fig. 12 shows our experimental results of performance, power consumption and number of provision changes for the Web trace. The performance is the average response time normalized by the target response time, and the power consumption is normalized by the power consumption of our hybrid approach. We see that the *Predictive Mean*/1 *h* does not provision sufficient capacity for the workload demand while the *Predictive 90%ile*/1 *h* tends to over-provision capacity. The *Reactive* policy misses its response time target due to the lag in booting additional servers. By contrast, our *Hybrid*/*var* policy meets its response time target, while keeping power consumption low. In particular, our *Hybrid*/*var* policy provides a 35% savings in power consumption when compared to the *Predictive 90%ile*/1 *h* and lowers response time by as much as 41% when compared to the *Predictive Mean*/1 *hr*.

(b) VDR trace: We ran another experiment using the normalized workload demand for the VDR trace. Fig. 13 shows the normalized mean response time, power consumption, and number of provisioning changes for the different policies. Again, we see that the *Predictive Mean* and *Reactive policies* fail to meet the target mean response time and incur more SLA violations. Further, the *Reactive* policy has the highest number of provisioning changes. Both *Hybrid*/*var* and *Predictive 90%ile*/1 *h* succeed in meeting the response time target but our approach uses slightly lower power. Note that *Predictive 90%ile*/1 *h* significantly over-provisions resource for the Web trace (Fig. 12), which indicates that its performance really depends on the characteristics of the workload. By contrast, our approach works well for both traces.

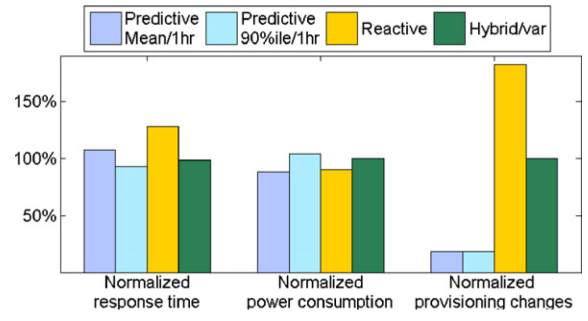


Fig. 13. Experimental results for the VDR trace.

In summary, we conclude that our hybrid approach outperforms other provisioning approaches such as the predictive and reactive policies, and combinations of both with fixed intervals, when considering SLA violations, power consumption, and the number of changes.

6. Related work

6.1. Workload analysis and characterization

Numerous studies have examined workload demand traces. Characterization studies of interactive workloads such as Web or media servers indicate that demands are highly variable, although daily and weekly patterns are common [2,7]. Rolia et al. find similar patterns in business application workloads [24].

A lot of research has been conducted to predict future workload demands. Vilalta et al. use classical time series analysis to distinguish between short-term and long-term patterns [28]. Dinda and O'Hallaron use linear models such as AR, moving average (MA), autoregressive moving average (ARMA), and autoregressive integrated moving average (ARIMA) models for the short-term prediction of application CPU demand [9]. Hoogenboom and Lepreau also use classical time series analysis to predict future workload demands [16]. Gmach et al. present a prediction algorithm using time series analysis to extract patterns from historical workload demand traces and predict future demands [13]. Hellerstein et al. use ANOVA (analysis of variance) to separate the non-stationary and stationary behavior of historical traces [15]. What distinguishes our work from others is our dynamic programming based algorithm for workload discretization and the incorporation of the provisioning cost in our workload analysis, which is important from a capacity planning and resource provisioning perspective.

6.2. Server provisioning

Existing server provisioning solutions can be broadly classified into predictive and reactive solutions.

Predictive provisioning assumes there is a predictable and stable pattern in demand and allocates capacity typically at the time-scale of hours or days based on the pattern. Castellanos et al. exploit the predictability in business applications' demand to improve the effectiveness of resource management [6]. With the adoption of virtualization technology, server consolidation has emerged as a promising technique to improve resource utilization and reduce power consumption [23,27]. Though these approaches can be effective to a certain extent, choosing the proper provisioning size is still a very difficult task. Our workload discretization technique can help with this. However, large, unpredictable demand surges could cause severe SLA violations.

Reactive provisioning on the other hand does resource allocation in short intervals (e.g., every few minutes) in response to workload changes. One approach is to use reactive control loops that are based on feedback control [1,22,26]. Others use reactive provisioning strategies for resource allocation and power management in virtualized environments. Typically, these approaches dynamically allocate CPU shares to virtual machines and/or migrate VMs across physical servers at runtime [21,29]. Purely reactive policies can react quickly to changes in workload demand, but issues such as unpredictability, instability and high provisioning costs limit their use in practice.

There are several approaches that *combine predictive and reactive control* [4,14,25]. While these approaches share some features with our hybrid approach, they differ in several aspects. First, our approach aims to optimize the performance, power consumption and provisioning cost simultaneously. The provisioning cost is particularly important to consider as it can significantly impact the performance and power consumption [12]. Second, we propose and apply a novel workload analysis technique, *workload discretization*, to determine provisioning intervals with variable lengths whereas predictive provisioning in other approaches uses simple fixed intervals. Third, our results show that our approach outperforms these other approaches.

We first presented the hybrid provisioning approach in [32]. In this work, we have added (1) a workload characterization of three real applications; (2) a detailed description, comparison and evaluation of several different workload discretization techniques; and (3) an evaluation of different provisioning policies for additional workload traces, including Web 2.0 and WorldCup98 traces.

7. Conclusion and future work

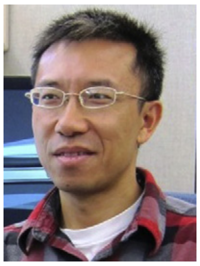
It is a challenging task to correctly provision IT resources in data centers to meet SLA requirements while minimizing power consumption. In this paper, we presented novel workload analysis and server provisioning techniques that enable data centers to meet their SLA requirements while conserving power and avoiding frequent power cycling of servers. We analyzed workload traces from production systems and developed a novel workload characterization technique to accurately capture predictable workload demand patterns. We presented a hybrid server provisioning approach that proactively provisions servers for the predictable demand pattern and leverages a reactive controller to provision for any unpredictable demand. We implemented and experimentally evaluated our work, and compared with existing and recent work on server provisioning. Our experimental results indicate that our hybrid approach successfully meets SLA requirements, is more power efficient than existing approaches and avoids frequent power cycling of servers.

In the future, we plan to incorporate dynamic CPU frequency scaling and power states into our approach. Further, we plan to investigate how we can extend our hybrid provisioning approach to multi-tier applications. While it should be easy to extend our hybrid approach to the application/logic tier, it is not clear how we can extend our approach to the data storage tier, which is usually the performance bottleneck. We are also interested in applying our workload discretization technique to other capacity planning and resource provisioning problems, including workload consolidation in virtualized environments and managing cooling resources in data centers.

References

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, B. Rochwerger, Océano-SLA based management of a computing utility, in: Proceedings of IEEE/IFIP International Symposium on Integrated Network Management, Seattle, WA, USA, 2001, pp. 855–868.
[2] M. Arlitt, C. Williamson, Web server workload characterization: the search for invariants, in: Proceedings of ACM SIGMETRICS, Philadelphia, PA, USA, 1996, pp. 126–137.
[3] M. Arlitt, T. Jin, Workload characterization of the 1998 world cup web site, IEEE Network 14 (May/June (3)) (2000) 30–37.
[4] M. Bennani, D. Menasce, Resource allocation for autonomic data centers using analytic performance models, in: Proceedings of International Conference on Automatic Computing, IEEE Computer Society, Washington, DC, USA, 2005, pp. 229–240.
[5] D. Brillinger, Time Series: Data Analysis and Theory, Holden-Day, San Francisco, CA, 1981.
[6] M. Castellanos, F. Casati, M. Shan, U. Dayal, iBOM: a platform for intelligent business operation management, in: Proceedings of International Conference on Data Engineering (ICDE), Tokyo, Japan, 2005, pp. 1084–1095.
[7] L. Cherkasova, M. Gupta, Characterizing locality, evolution, and life span of accesses in enterprise media server workloads, in: Proceedings of Workshop on Network and Operating Systems Support for Digital Audio and Video, New York, NY, USA, 2002, pp. 33–42.
[8] A. Coskun, R. Strong, D. Tullsen, S. Rosing, Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors, in: Proceedings of ACM SIGMETRICS, Seattle, WA, USA, 2009, pp. 169–180.
[9] P. Dinda, D. O'Hallaron, Host load prediction using linear models, Cluster Computing 3 (4) (2000) 265–280.
[10] Report to congress on server and data center energy efficiency, public law, U.S. Environmental Protection Agency, ENERGY STAR Program, 2007.
[11] A. Gandhi, M. Harchol-Balter, R. Das, C. Lefurgy, Optimal power allocation in server farms, in: SIGMETRICS 2009, Seattle, USA, 2009.
[12] A. Gandhi, M. Harchol-Balter, I. Adan, Server farms with setup costs, Performance Evaluation 67 (2010) 1123–1138.
[13] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Capacity management and demand prediction for next generation data centers, in: Proceedings of IEEE International Conference on Web Services, Salt Lake City, Utah, USA, 2007, pp. 43–50.
[14] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, A. Kemper, Adaptive quality of service management for enterprise services, ACM Transactions on the Web 2 (1) (2008).
[15] J. Hellerstein, F. Zhang, P. Shahabuddin, A statistical approach to predictive detection, Computer Networks 35 (1) (2001).
[16] P. Hoogenboom, J. Lepreau, Computer system performance problem detection using time series models, in: Proceedings of Summer USENIX Conference, 1993, pp. 15–32.
[17] J. Lin, E. Keogh, S. Lonardi, B. Chiu, A symbolic representation of time series, with implications for streaming algorithms, in: Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, CA, USA, 2003.
[18] G. Linden, Make data useful, http://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazoncom, 2006.
[19] Intel, Intel Math Kernel Library 10.0, LINPACK, http://software.intel.com/en-us/intel-mkl/.
[20] D. Mosberger, T. Jin, httperf-A tool for measuring web server performance, ACM Sigmetrics Performance Evaluation Review 26 (1998) 31–37.
[21] P. Padala, K. Hou, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Shin, Automated control of multiple virtualized resources, in: Proceedings of EuroSys, Nuremberg, Germany, 2009.
[22] S. Ranjan, J. Rolia, H. Fu, E. Knightly, QoS-Driven server migration for Internet data centers, in: Proceedings of IEEE International Workshop on Quality of Service, Miami Beach, FL, USA, 2002, pp. 3–12.
[23] J. Rolia, A. Andrzejak, M. Arlitt, Automating enterprise application placement in resource utilities, in: Proceedings IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Heidelberg, Germany, 2003, pp. 118–129.
[24] J. Rolia, X. Zhu, M. Arlitt, A. Andrzejak, Statistical service assurances for applications in utility grid environments, Performance Evaluation 58 (2+3) (2004) 319–339.
[25] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, Dynamic provisioning of multi-tier Internet applications, in: Proceedings of IEEE ICAC, June, 2005.
[26] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, A. Tantawi, An analytical model for multi-tier Internet services and its applications, ACM Transactions on the Web 1 (1) (2007).
[27] A. Verma, G. Dasgupta, T. Nayak, P. De, R. Kothari, Server workload analysis for power minimization using consolidation, in: Proceedings of USENIX Annual Technical Conference, 2009.
[28] R. Vilalta, C.V. Apte, J.L. Hellerstein, S. Ma, S.M. Weiss, Predictive algorithms in the management of computer systems, IBM Systems Journal 41 (3) (2002) 461–474.
[29] J. Xu, M. Zhao, J. Fortes, R. Carpenter, M. Yousif, Autonomic resource management in virtualized data centers using fuzzy logic-based approaches, Cluster Computing Journal 11 (3) (2008) 213–227.
[30] B.-K. Yi, C. Faloutsos, Fast time sequence indexing for arbitrary Lp norms, in: Proceedings of VLDB, Cairo, Egypt, 2000.
[31] The Internet traffic archives: WorldCup98, http://ita.ee.lbl.gov/html/contrib/WorldCup.html.
[32] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, M. Marwah, Minimizing data center SLA violations and power consumption via hybrid resource provisioning, in: Proceedings of the Second International Green Computing Conference (IGCC), July, 2011.

Anshul Gandhi is a PhD student in the Computer Science Department at Carnegie Mellon University, under the direction of Mor Harchol-Balter. He received his BTech in Computer Science and Engineering from the Indian Institute of Technology, Kanpur. His research interests are in designing and implementing power management policies for datacenters as well as general performance modeling of computer systems.



Yuan Chen is a senior research scientist in the Sustainable Ecosystem Research group (SERg) at HP Labs <http://www.hpl.hp.com/> in Palo Alto, CA. Yuan's research area is energy efficient computing with a focus on performance and power modeling of services and applications in data centers, and control and optimization of data center workload and resource management. His most recent work is developing an integrated management of the IT, cooling, and power supply subsystems to improve the energy efficiency and reduce the cost and environmental footprint of data centers. Yuan's past work includes automated IT monitoring and management, content-based publish-subscribe systems, high performance computing, and constraint programming. Yuan has published over 40 technical papers in peer-reviewed journals and conferences proceedings, including the Best Paper Award of International Green Computing Conference (IGCC 2011) and the Best Paper Award of IEEE/IFIP Network Operations and Management Symposium (NOMS 2008). He has one patent granted and over 20 patent applications pending. Yuan received a B.S. from University of Science and Technology of China <http://www.ustc.edu.cn/en/>, a MS from Chinese Academy of Sciences <http://www.ict.ac.cn/english/>, and a PhD from Georgia Institute of Technology <http://www.cc.gatech.edu>, all in Computer Science.

Dr. Daniel Gmach is a researcher at HP Labs Palo Alto. Before that he was a PhD student at the database group of the Technische Universität München where he graduated in 2009. He studied computer science at the University of Passau. His current research interests are in adaptive resource pool management of virtualized enterprise data centers, performance measurement and monitoring, hosting large-scale enterprise applications, database systems, and software engineering principles.

Martin Arlitt is a senior research scientist at Hewlett-Packard Laboratories (HP Labs) in Palo Alto, CA, where he has worked since 1997. His general research interests are workload characterization and performance evaluation of distributed computer systems. His 70+ research papers have been cited more than 5,000 times (according to Google Scholar). He has 15 granted patents and many more pending. He is an ACM Distinguished Scientist and a senior member of the IEEE. He has served on the program committee of numerous top tier conferences, such as ACM SIGMETRICS, the world's premiere venue for research publications on performance evaluation. He is co-program chair of ACM SIGMETRICS/Performance conference for 2012. He is the creator of the ACM SIGMETRICS GreenMetrics workshop. He is co-chair of the IEEE Computer Society's Special Technical Community on Sustainable Computing. Since 2001, Martin has lived in Calgary, AB, where he is also a senior research associate in the Department of Computer Science at the University of Calgary. In 2008, he was named a member of Calgary's Top 40 under 40.

Manish Marwah is a researcher in the Sustainable Ecosystem Research group (SERg) at HP Labs. He is conducting cross-disciplinary research on the applications of data mining techniques to enhance sustainability of cyber physical infrastructure, such as, data centers, buildings, etc. He received his PhD in Computer Science from University of Colorado at Boulder. His PhD thesis focused on building distributed systems with enhanced server fault-tolerance to provide seamless user experience. He has also worked for several years in the telecommunication industry, most recently as an architect with Avaya Labs. His research has led to more than 40 refereed papers in conferences and journals; and resulted in 8 issued patents (with over 35 pending). Manish received an MS in Computer Science and an MS in Mechanical Engineering from the University of Colorado at Boulder and a BTech from the Indian Institute of Technology, Delhi.