Towards Resource-Efficient Cloud Systems: Avoiding Over-Provisioning in Demand-Prediction Based Resource Provisioning

Liuhua Chen
Department of Electrical and Computer Engineering
Clemson University, Clemson, 29634
Email: liuhuac@clemson.edu

Haiying Shen
Department of Computer Science
University of Virginia, Charlottesville, 22904
Email: hs6ms@virginia.edu

***Abstract*—Demand-prediction based resource provisioning schemes help assure service level objectives (SLO) in cloud systems. We notice that if a provisioning scheme does not exclude bursts from historical resource demands in normal demand prediction or always uses a large padding to correct under-prediction, it will lead to resource over-provisioning and low resource utilization. To improve the previous schemes, in this paper, we present a Resource-efficient Predictive Resource Provisioning system in clouds (RPRP) that excludes bursts in demand prediction and has algorithms to specifically handle bursts to avoid resource over-provisioning. Rather than setting padding to a possibly high value, RPRP has a load-dependent padding algorithm that adaptively determines padding based on predicted demands. To handle bursts, RPRP embodies a responsive padding algorithm that adaptively adjusts padding to recover from both under-provisioning and over-provisioning. We implemented RPRP on top of Xen and conducted both trace-driven simulation and real-world testbed experiments. The experimental results show that RPRP achieves higher resource utilization, more accurate demand predictions, and fewer SLO violations than previous schemes.**

I. Introduction

In cloud systems, all hardware resources are pooled into a common infrastructure to be shared by applications. Commercial cloud providers such as Amazon's EC2 [1] abstract processor, memory, storage, and other resources in physical machines (PMs) into virtual machines (VMs) and sell them to tenants. After the service level agreements including service level objectives (SLOs) are enacted, tenants then deploy applications by accessing their own VMs and pay only for the resources they have acquired. In such a consolidated environment, the resources in physical machines are often under-utilized or over-utilized since application resource demands vary over time [2]. To ensure resource provisioning for guaranteeing SLOs, clouds can use demand-prediction based resource provisioning schemes [3–14] that allocate physical resources to VMs according to the dynamically estimated VM demands. Inaccurate demand estimation could lead to over-provisioning (hence resource under-utilization) or under-provisioning (hence SLO violations). Providing more resources achieves low SLO violations while leading to low resource utilization, and vice versa. Achieving the trade-off between the penalties associated with SLO violations

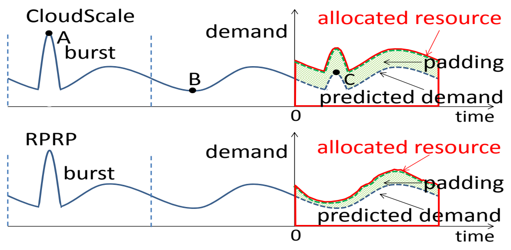


Figure 1: Prediction and padding.

and high resource utilization (hence revenue maximization) requires an accurate demand prediction methodology.

To improve the prediction accuracy of the previous demand-prediction based resource provisioning schemes, previous work [10] deal with demand mispredictions by adding a padding to a predicted demand. CloudScale [10] uses the average value of the demands at each time point in multiple periodical patterns from the historical record as the predicted demand at this time point in the next time period, and uses the maximum (or the 80th percentile value) of the high-frequency spectrum at each time point as the padding. This method does not exclude bursts either in demand prediction or in padding determination, and may use a padding larger than the actual needed amount. A burst is a sudden large increment of demand which occurs in a random short time and is difficult to predict. Note a large demand that occurs periodically is predictable and is not a burst. We notice that regarding bursts as normal demands in demand prediction may lead to over-predictions for normal demands but under-predictions for bursts. For example, as shown in Figure 1, suppose we predict the demand at a time period based on a historical record that covers two time periods. CloudScale uses the average values at points A and B to produce the predicted demand at point C, which is larger than the normal demand and smaller than the burst. Also, the padding can be very large by using the high-frequency spectrum or using the average of the latest ten prediction errors which could be for bursts. To handle underestimation, CloudScale also raises the resource allocation cap by ratio $\alpha>1$ until an underestimation error is corrected but it does not decrease the cap until the next prediction is performed even when it is higher than the real demand. Although it can lower the cap in the next prediction, the over-predictions problem can be significant especially when this happens frequently.

In this paper, we study three real traces and verify that i) bursts are common and occur irregularly in cloud systems, and ii) using the 80^{th} percentile value of the high-frequency spectrum as the padding can lead to high over-provisioning. Our trace measurement shows that the padding of a PM hosting 20 VMs can reach the capacity of the PM. To further improve the previous padding-based methods, we propose a Resource-efficient Predictive Resource Provisioning system in clouds (RPRP) that excludes bursts in demand prediction and specifically handles bursts to avoid resource over-provisioning. As a result, RPRP further reduces over-provisioning, hence improving resource utilization (Figure 1). Specifically, RPRP consists of the following algorithms.

(1) *A burst-exclusive prediction algorithm* that excludes bursts and noises to find the regular demand pattern, which improves resource utilization efficiency.

(2) *A load-dependent padding algorithm* that determines the padding for each predicted demand based on the historical predicted and actual demand record in order to minimize the expected total allocated resource while guaranteeing SLO.

(3) *A responsive padding algorithm* that scales the resource allocation cap up or down to meet the actual demand rather than simply multiplying resource allocation cap by a ratio (as in CloudScale) in order to improve resource utilization.

Algorithms (1) and (2) aim to exclude bursts in demand prediction and reduce padding to reduce over-provisioning while satisfying resource demands. Algorithms (3) aims to handle bursts or workload pattern deviation in a resource-efficient manner.

Our work is novel in that it significantly reduces allocated resource in predictive resource provisioning while guaranteeing SLO. Please note that we do not claim that our burst-exclusive prediction algorithm itself is novel. We present this algorithm only to show the advantage of excluding bursts in prediction. We conducted comprehensive trace-driven simulation and real-world testbed experiments to evaluate the performance of RPRP in comparison with previous methods. Our experimental results show that by only using Algorithm (2), RPRP reduces 58.4% of the padding value compared with CloudScale, and by only using algorithms (1) and (2), RPRP reduces 18% of the allocated resource while reducing 30% of the total number of SLO violations compared with CloudScale on average.

The rest of this paper is organized as follows. Section II briefly describes the related work. Section III presents the details of each component of RPRP. Section IV presents the performance evaluation of RPRP. Finally, section V summarizes the paper with remarks on our future work.

II. Related Work

There are plenty of researches on resource provisioning in clouds and datacenters. Some previous works implement static provisioning [15–22] that allocates physical resources to VMs only once based on the maximum static VM resource demands. However, static provisioning cannot fully utilize resources because of time-varying resource demands of VMs.

As an alternative of static provisioning, many works [3, 7, 9, 10, 23] dynamically allocate resource for applications by using workload forecasting techniques to predict expected workload parameters from measured system metrics. Bobroff *et al.* [3] used a time series forecasting algorithm [24] to predict demands. Chandra *et al.* [7] proposed a workload prediction algorithm using auto-regression and histogram based methods. Gong *et al.* [9] proposed an online resource demand prediction model, which uses a hybrid approach that employs signature-driven and state-driven prediction algorithms. The above methods differ from RPRP in that they only predict the trend of VM resource demand but do not handle mispredictions or the bursts, which may lead to insufficient resource allocation. Several works [10, 23] handle the mispredictions to improve prediction accuracy. CloudScale [10] employs online resource demand prediction and prediction error handling to achieve adaptive resource allocation. However, as explained previously, these methods tend to allocate more resource than required in burst prevailing situations and underestimation, which may lead to low resource utilization. Agile [23] builds a mapping function between the VM resource pressure (i.e., resource usage/allocated resource) and the SLO violation rate, and dynamically adjusts paddings based on monitored resource pressure to keep the SLO violation rate below a target. Agile determines paddings based on real resource usage rather than before resource allocation, while we focus on proactive padding determination before resource allocation in this paper.

Some live VM migration methods have been proposed [25–28] to handle overloads. Sandpiper [25] carries out dynamic monitoring and hotspot probing on the utility of system resources and migrates out VMs from hotspot PMs. Each VM has a volume-to-size ratio (VSR), where the size is its memory footprint and the volume is the product of CPU, network and memory loads (i.e., $1/((1-cpu)\times(1-net)\times(1-mem)))$. When selecting VMs to migrate from a hotpot PM, Sandpiper attempts to migrate the VM with maximum VSR to the PM with the least volume. VectorDot [26] utilizes vectors to represent the multi-dimensional resource requirements of VMs for migration. To reduce migration cost, RIAL [27] dynamically assigns different weights to different resources according to their usage intensity in the overloaded PM. Guo *et al.* [28] proposed Seagull to handle burst workload in local clusters by migrating applications that have the lowest cloud price to the cloud. Unlike these works that focus on VM migration, RPRP focuses on VM resource demand prediction and misprediction correction. These VM migration methods can use the demand prediction from RPRP in migration scheduling to improve performance. Our experimental results show the advantage of RPRP in VM migration.

III. Objective and System Design

A. Objective

The VM workload consists of multiple types of resource such as CPU, memory, I/O and network bandwidth. Like previous resource provisioning schemes [3, 7, 9, 10, 29, 30], we also predict the demand of each resource. In this paper, we use the CPU resource amount (%) as an example to present our work. We denote a VM's workload demand as a time series $D = \{d_{t_1}, ..., d_{t_i}, ..., d_{t_N}\}$ where t_i is a time. We denote *allocated resource* (i.e., provisioned resource) as $A = \{a_{t_1}, ..., a_{t_i}, ..., a_{t_N}\}$, where a_{t_i} indicates the amount at time interval from t_{i-1} to t_i. We use the term *utilized resource* to represent the actual resource consumption and denote it as $U = \{u_{t_1}, ..., u_{t_i}, ..., u_{t_N}\}$. We use *resource capacity* of a VM, denoted by C, to refer to the maximum amount of resource specified for a VM to use upon its creation. D, A and U are scaled to resource capacities C and fall into the range [0, 1]. We aim to more accurately predict demand $P = \{p_{t_{N+1}}, p_{t_{N+2}}, ..., p_{t_{N+T}}\}$ from the historical records of utilized resource U, based on which we dynamically allocate resource $A = \{a_{t_{N+1}}, a_{t_{N+2}}, ..., a_{t_{N+T}}\}$ to VMs. The goal of our VM resource provisioning strategy is to dynamically determine allocated resource A such that $d_{t_i} \leq a_{t_i} \leq C$ and meanwhile to minimize $a_{t_i} - d_{t_i}$.

B. Burst-exclusive Prediction

Trace analysis. In order to confirm whether resource utilization bursts resulted from a sudden increase of requests are common in cloud VMs, and whether CloudScale has relatively low resource utilization, we analyze three real traces: Google Cluster [31], PlanetLab VM [32] and WorldCup [33]. The Google Cluster trace records the CPU and memory resource usages on a cluster of about 11000 machines from May 2011 for 29 days. The PlanetLab trace contains the CPU utilization of each VM in PlanetLab every 5 minutes for 24 hours in 10 random days in March and April 2011. The WorldCup web server trace records per-minute workload intensity (in the form of the number of requests in each minute) and we used it to generate CPU resource usage every 5 minutes by multiplying the number with a scaling factor of 0.05. We picked one VM from each trace and measured the percentage of the CPU resource consumption of each VM in its capacity (called *CPU usage*). Figure 2 displays the resource usage of each VM over time (in hours) from the different traces. We can see that bursts are common and can occur at any time. Note that the pulses in the Google Cluster trace are not bursts since they are predictable pulse demands that occur periodically.

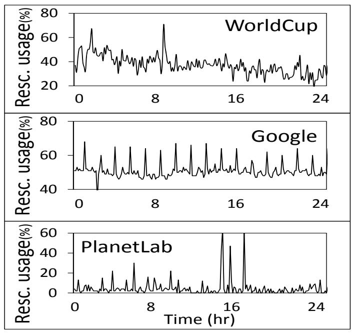
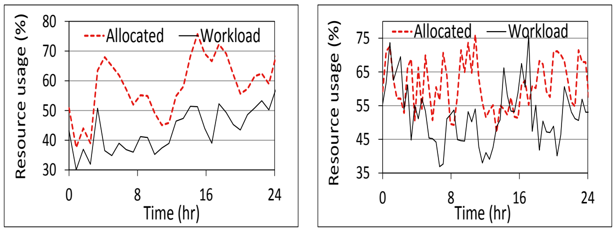


Figure 2: Resource usage bursts.

From the Google Cluster trace, we manually selected a low-burst-density VM and a high-burst-density VM that have fewer and more bursts in a unit time period than the average, respectively. We used their historical records of 48 hours to predict the demands and determine the paddings every 5 minutes in the next 24 hours using CloudScale. Figure 3(a) and Figure 3(b) show the resource demand and the allocated resource of the low-burst-density VM and the high-burst-density VM over 24 hours, respectively. These figures present the average value of every 10 time points (i.e., 50 mins). We find that CloudScale provides 33% and 17% more resources (i.e., $a_{t_i} - d_{t_i}$) in the low and high burst-density workloads, respectively.



(a) Low burst density. (b) High burst density.

Figure 3: Burst-based padding results of CloudScale.

Figure 4 shows the median, the 10th and 90th percentiles of the resource utilization efficiency (i.e., the ratio of the amount of utilized resource over the amount of allocated resource) at all the measured time points during the 24 hours. We see that the allocated resource is higher than the demanded resource most of the time in both workloads, and CloudScale cannot achieve a resource utilization efficiency close to 1 in either case. These results confirm that resource utilization bursts are common in cloud VMs, and CloudScale cannot achieve high resource utilization, which motivate us to explore a more accurate prediction scheme that reduces unnecessary resource provisioning hence increase resource utilization while guaranteeing SLO compliance.

Algorithm design. We then present the details of the burst-exclusive prediction algorithm. In order to exclude the bursts, RPRP relies on Fast Fourier Transform (FFT), which is a well-known algorithm for repeated pattern identification from waveforms. Since VM and server workloads from datacenters typically show a periodicity (in hours, days, weeks, and so forth) [34, 35], FFT is applicable for predicting workload demand in repeated periodic patterns $P=\{p_{t_{N+1}}, p_{t_{N+2}}, ..., p_{t_{N+T}}\}$ based on the historical utilization series (U=$\{u_{t_1}, ..., u_{t_N}\}$).

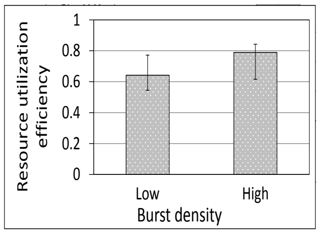


Figure 4: Statistics of resource utilization efficiency at all the measured time points during 24 hours.

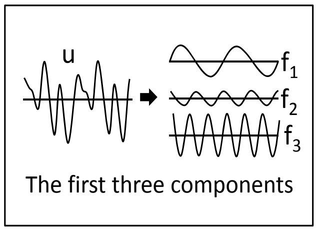


Figure 5: Fourier decomposition.

C. Load-dependent Padding

Setting padding to a possibly high value leads to over-provisioning though it can avoid under-provisioning. Actually, the variation of prediction errors is dependent on load levels in cloud [19]. Below, we first formulate the problem of padding determination to achieve both resource efficiency and SLO guarantee, and then introduce our load-dependent padding algorithm. To satisfy an SLO requirement that a certain percentage of demands must be satisfied [36], the expected probability that the padding can meet the resource demand must satisfy $\overline{Pr} \geq 1-\varepsilon$, where ε is a small value.

In the historical predicted and actual demand series during a time period $[t_1, t_N]$, we use $\mathscr{P}$=$\{\hat{p}_1, \hat{p}_2, ..., \hat{p}_M\}$ ($\hat{p}_1 < \hat{p}_2 < ... < \hat{p}_M$) to represent the M different predicted demand levels, and use $\mathscr{D}_{\hat{p}_i}$=$\{d_1, d_2, ..., d_{n_{\hat{p}_i}}\}$ ($d_1 \leq d_2 \leq ... \leq d_{n_{\hat{p}_i}}$, $n_{\hat{p}_i}$=$|\mathscr{D}_{\hat{p}_i}|$) to indicate the workload demands that were predicted to be $\hat{p}_i$ in descending order . Noted that N is the total number of workload demands in the demand series, and $n_{\hat{p}_i}$ is the number of workload demands in demand level $\hat{p}_i$. Then, $N=\sum_{j=1}^{M} n_{\hat{p}_j}$. We use $\delta(\hat{p}_i)$ to denote the padding value for $\hat{p}_i$. The probability that the allocated resource (a_{t_i}=$\hat{p}_i$+$\delta(\hat{p}_i)$) is sufficient to meet the demand is

$$Pr(\hat{p}_i) = \frac{|\{d_j \leq \hat{p}_i + \delta(\hat{p}_i) | d_j \in \mathscr{D}_{\hat{p}_i}\}|}{n_{\hat{p}_i}}. \tag{1}$$

The expected probability that the allocated resource is sufficient to meet the demand ($\overline{Pr}$) is

$$\overline{Pr} = \sum_{i=1}^{M} Pr(\hat{p}_i) \frac{n_{\hat{p}_i}}{N} = \sum_{i=1}^{M} \frac{|\{d_j \leq \hat{p}_i + \delta(\hat{p}_i) | d_j \in \mathscr{D}_{\hat{p}_i}\}|}{n_{\hat{p}_i}} \frac{n_{\hat{p}_i}}{N} \tag{2}$$

The expected allocated resource amount can be calculated by

$$\sum_{\hat{p}_i \in \mathscr{P}} [\hat{p}_i + \delta(\hat{p}_i)] \frac{n_{\hat{p}_i}}{N} \tag{3}$$

Problem: *given the probability distribution of the predicted demand levels ($\hat{p}_j$), the probability distribution of the actual demands for each $\hat{p}_j$ and ε, how can we determine the padding value $\delta(\hat{p}_i)$ for each $\hat{p}_i$ to achieve ($\overline{Pr} \geq 1-\varepsilon$) and meanwhile minimize the expected total allocated resource (Equ. (3))?*

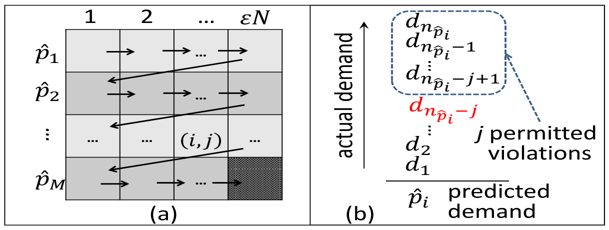


Figure 6: Dynamic programming algorithm. (a) an $M \times \varepsilon N$ dynamic programming matrix $\mathbb{M}$. (b) procedure to determine the allocated resource if we place j permitted violations on predicted demand level $\hat{p}_i$.

In the following, we present a solution of the above problem based on [19]. According to Equ. (2), to satisfy $\overline{Pr} \geq 1-\varepsilon$, the number of permitted violations $d_j > \hat{p}_i + \delta(\hat{p}_i)$ during the time period $[t_1, t_N]$ must be no larger than εN. Minimizing Equ. (3) can be transformed to minimizing $\sum_{\hat{p}_i \in \mathscr{P}} [\hat{p}_i + \delta(\hat{p}_i)] n_{\hat{p}_i}$. To solve the above problem, we can distribute εN permitted violations to different predicted demand levels to minimize $\sum_{\hat{p}_i \in \mathscr{P}} [\hat{p}_i + \delta(\hat{p}_i)] n_{\hat{p}_i}$. To this end, we rely on an $M \times \varepsilon N$ dynamic programming matrix $\mathbb{M}$ as shown in Figure 6(a).

In the matrix, $\mathbb{M}(i,j)$ represents the minimum total allocated resource when distributing j violations to the first i predicted demand levels ($\hat{p}_1, \hat{p}_2, ..., \hat{p}_i$). The arrows indicate the computation process of the dynamic programming in calculating each entry $\mathbb{M}(i,j)$, where j changes from 1 to εN and i changes from $\hat{p}_1$ to $\hat{p}_M$. It finally arrives at $\mathbb{M}(M, \varepsilon N)$, which is the minimum total allocated resource when distributing all εN violations to all M predicted demand levels, and the allocated resource a_{t_i}=$\hat{p}_i$+$\delta(\hat{p}_i)$ (hence the padding $\delta(\hat{p}_i)$) for each demand level $\hat{p}_i$ is also determined.

Figure 6(b) shows how to determine the allocated resource if we place j permitted violations on predicted demand level $\hat{p}_i$. There are $n_{\hat{p}_i}$ demand values that were predicted to be $\hat{p}_i$, $\mathscr{D}_{\hat{p}_i}$=$\{d_1, d_2, ..., d_{n_{\hat{p}_i}}\}$ ($d_1 \leq d_2 \leq ... \leq d_{n_{\hat{p}_i}}$. If j violations are permitted in the predicted demand level $\hat{p}_i$, then the allocated resource is $d_{n_{\hat{p}_i}-j}$, which is the value of ($n_{\hat{p}_i}$-j)th demand. Then, the total allocated resource for $\hat{p}_i$ is $d_{n_{\hat{p}_i}-j} \times n_{\hat{p}_i}$ and the padding is $\delta(\hat{p}_i) = d_{n_{\hat{p}_i}-j} - \hat{p}_i$.

The whole computation process consists of the initialization and the recurrence phases. The initialization phase handles the first row, i.e., $\mathbb{M}(1,j)$ ($j=1,...,\varepsilon N$). That is, it directly calculates the total allocated resource for placing j ($j=1,...,\varepsilon N$) violations on demand level $\hat{p}_1$ using the method introduced above. In the recurrence phase, the algorithm enumerates all the possible placements that i) place x ($x=0,1,...,j$) violations on $\hat{p}_i$ and ii) place $j-x$ violations on $\hat{p}_1$, $\hat{p}_2$, ..., $\hat{p}_{i-1}$. The minimum total allocated resource for part i) is calculated based on the method introduced above, which equals $d_{n_{\hat{p}_i}-x} \times n_{\hat{p}_i}$. The minimum total allocated resource for part ii) can be obtained by simply referring to $\mathbb{M}(i-1, j-x)$. We finally find out the placement that results in minimum $\mathbb{M}(i,j)$ for different arrangements:

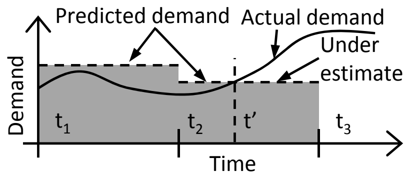


Figure 7: Underestimate correction. Demand prediction and resource allocation are performed at time t_1, t_2 and t_3. Responsive padding is performed at time t' where the allocated resource becomes insufficient for the demand before next prediction and allocation.

$\mathbb{M}(i,j)$=$\min_{0\leq x\leq j}\{\mathbb{M}(i-1,j-x)+d_{n_{\hat{p}_i}-x}\times n_{\hat{p}_i}\}$. Finally, we get $\mathbb{M}(M,\varepsilon N)$ and get the padding $\delta(\hat{p}_i)$) for each demand level from this placement schedule.

D. Responsive Padding

Usually VM demand prediction and resource allocation are carried out periodically (e.g., at time t_1, t_2 and t_3 in Figure 7). Since VM demand dynamically fluctuates within each predication period, the allocated resource sometimes becomes insufficient for the demand. For example, the allocated resource is detected as insufficient for the demand from time t', long before the next prediction and allocation time point t_3. The resource underestimation occurs from time t' to t_3. The same situation can happen to overestimation.

Then, we aim to keep the resource utilization efficiency within [T_l, T_u], where T_l and T_u are lower bound threshold and upper bound threshold, so that resource is fully utilized while SLO is satisfied. A resource utilization efficiency lower than T_l means overestimation and a resource utilization efficiency higher than T_u means underestimation. The black-box monitoring [25] can be used to detect underestimation (i.e., the utilization exceeds a high threshold for a sustained time) and overestimation.

Unlike CloudScale that only raises the resource allocation cap, we also decrease the cap adaptively to reduce over-provisioning. If underestimation is flagged (resource utilization efficiency $> T_u$) at time t', then

$$a_{t'+\Delta}=a_{t'}+\frac{1}{2}(u_{max}-a_{t'}) \tag{4}$$

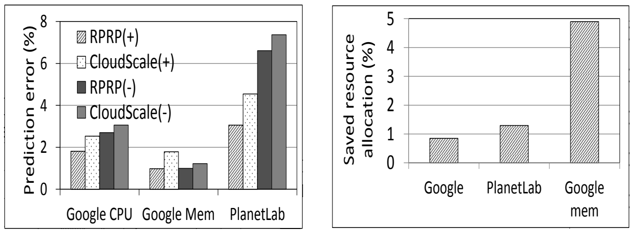
where u_{max} is the maximum recorded utilized resource of this VM, and Δ is a monitoring interval. If the resource utilization efficiency is lower than T_l after raising (t''), then

$$a_{t''+\Delta}=a_{t''}-\frac{1}{2}(a_{t''}-u_{t''}) \tag{5}$$

By respectively performing Equ. (4) and Equ. (5), we can quickly restrict the resource utilization efficiency within the range of $[T_l,T_u]$. Our responsive padding algorithm can achieve the goal of correcting underestimation while maintaining high resource utilization efficiency.

IV. Performance Evaluation

In this section, we present performance evaluation of RPRP in comparison with Sandpiper [25] and CloudScale [10], the details of which are explained in Sections I and II. We used the same parameters from [25] and [10] for Sandpiper and CloudScale, respectively, except the VM and PM capacities which were generated by the CloudSim simulator [32]. For RPRP, we set $\varepsilon = 5\%$ as the highest percentage of demands that are not satisfied in the SLO specification for load-dependent padding, and set T_u=0.9 and T_l=0.5 for responsive padding. We used three real traces: Google Cluster [31], PlanetLab VM [32] and WorldCup [33] in the experiments.



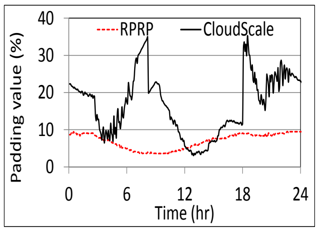
(a) Prediction error. (b) Saved resource allocation.

Figure 8: Performance of the burst-exclusive prediction algorithm.

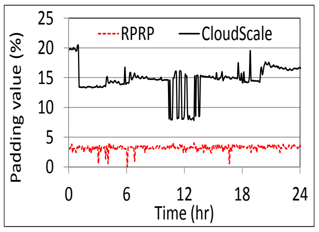
We first used the traces to analyze the effectiveness of each algorithm in RPRP in comparison with CloudScale (Section IV-A). We did not include Sandpiper in this section because it does not have demand prediction and padding. We then compared RPRP with both Sandpiper and CloudScale to measure the performance of resource utilization and SLO compliance through trace-driven experiments on the CloudSim simulator [32] (Section IV-B) and a prototype implemented on top of Xen in a real-world testbed (Section IV-C). We used 48-hour history utilization data from each trace to predict the resource demand at every 5 minutes [37] in the next 24 hours. We set the ratio of PM capacity to VM capacity to 3 (i.e., a PM can serve 3 VMs with 100% resource demands). The padding value is measured as the percentage over VM capacity.

A. Analytical Performance Evaluation

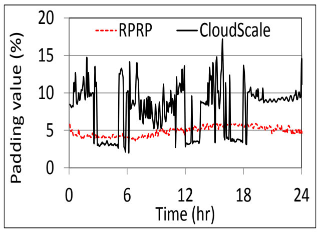
1) Performance of Burst-exclusive Prediction: We first study the performance of the prediction algorithms without padding. Figure 8(a) shows the average prediction error of CloudScale and RPRP calculated by $\frac{1}{n}\sum_{i=1}^{n}|\hat{p}_i-d_i|$ for the positive and negative prediction errors, respectively, using the Google Cluster and PlanetLab traces. We see RPRP has lower prediction errors than CloudScale. The results verify the higher prediction accuracy of our burst-exclusive prediction algorithm compared to the average-based algorithm in CloudScale since RPRP excludes the high-frequency components including bursts in predicting the normal demands while the average value does not exclude the bursts. As we mentioned previously, RPRP does not focus on improving CloudScale in the basic demand prediction approach. We present the experimental result only to show the advantage of excluding bursts in demand prediction. Compared to Google Cluster trace, both RPRP and CloudScale have



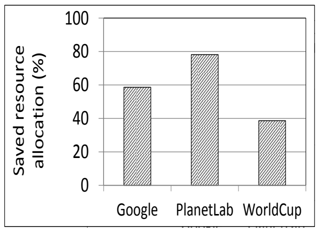
(a) Google Cluster trace



(b) PlanetLab trace



(c) WorldCup trace



(d) Saved resource allocation

Figure 9: Performance of the load-dependent padding algorithm.

larger prediction errors in the PlanetLab trace because its demands have weak periodic patterns. However, RPRP can still achieve an acceptable prediction accuracy and produces prediction errors less than CloudScale.

Figure 8(b) shows the saved resource allocation resulted from the burst-exclusive prediction algorithm of RPRP compared to the prediction algorithm of CloudScale (i.e., $\frac{CloudScale-RPRP}{CloudScale}$). We see that the result reaches around 1% for Google CPU trace and PlanetLab trace, and around 5% for Google memory trace. The reason is that RPRP filters out the bursts in prediction, while CloudScale does not exclude bursts in the average calculation. The result confirms that the burst-exclusive prediction algorithm in RPRP produces higher resource utilization compared to the average-based prediction algorithm in CloudScale. Note that these savings are resulted merely from excluding bursts, while the cumulative saving can still be substantial when multiple VMs are co-located. Excluding bursts can reduce 1036%, 1754% and 4592% absolute CPU consumption in a PM hosting 20 VMs [38] in 24 hours (i.e., 288 allocations) based on the Google CPU, PlanetLab and Google memory trace, respectively.

2) Performance of Load-dependent Padding: We compare the padding algorithms of CloudScale and RPRP. Figure 9(a)(b)(c) show only the padding values of three VMs picked from the three traces in RPRP and CloudScale. We see that CloudScale generates higher padding values than RPRP most of the time for all the three traces. This is because CloudScale uses the maximum or the 80^{th} percentile value of the high-frequency spectrum as the padding. RPRP determines the padding in order to achieve a certain probability of satisfying resource requests while minimizing the expected padding value. CloudScale results in a diverse padding values (i.e., 5%–35%, 10%–20% and 5%–15% for Google Cluster, PlanetLab and WorldCup traces, respectively) over time since high frequency components are often noise which has random magnitude. The minimum 5% padding means that the padding of a PM hosting 20 VMs can reach the capacity of the PM.

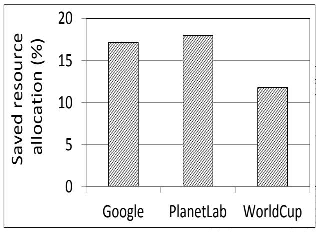
Figure 9(d) shows the saved resource allocation of the padding algorithm in RPRP compared to CloudScale for the previous three VMs. RPRP reduces 58.6%, 78.2% and 38.6% of the padding values compared with CloudScale for the Google Cluster, PlanetLab and WorldCup traces, respectively. RPRP reduces 14968%, 17468%, and 4665% absolute padding amounts of CloudScale on a PM hosting 20 VMs in 24 hours based on the three traces, respectively. These reduced padding amounts are substantial in improving the resource utilization of a PM that hosts multiple VMs.

3) Performance of Resource Provisioning (Prediction +Padding): We then evaluate the performance of resource provisioning that combines both the burst-exclusive prediction and load-dependent padding algorithms.

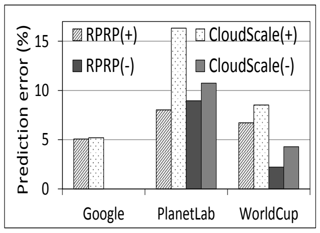
Figure 10(a) shows the saved resource allocation in resource provisioning consisting of both prediction and padding algorithms of RPRP compared to CloudScale. R-PRP saves 18%, 19% and 11% of the allocated resource in the traces of Google Cluster, PlanetLab and WorldCup, respectively. Then, by applying RPRP, a PM with 20 VMs [38] can save 7296%, 15744% and 7968% absolute CPU resource during 24 hours based on these traces. The reduction is relatively smaller than the reduction of the padding value in Figure 9(d) because padding constitutes a small percent of the predicted amount.

Figure 10(b) shows the average prediction error of CloudScale and RPRP for the positive and negative prediction errors, respectively. We can see that RPRP has lower prediction error than CloudScale in each trace, which indicates that RPRP achieves higher accurate prediction due to its burst-exclusive prediction algorithm and load-dependent padding algorithm, which determines the padding value adaptively based on the predicted demand level rather than using a possibly maximum or 80% percentile burst values as in CloudScale. The prediction error of PlanetLab trace is relatively higher than other traces because it does not exhibit as strong periodicity as other traces.

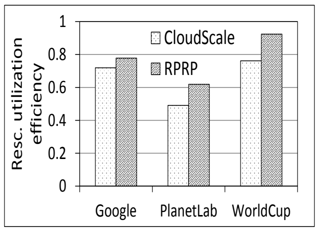
Figure 10(c) shows the resource utilization efficiency (i.e., the ratio of total utilized resources and the total allocated resources, $\sum_i u_i/\sum_i a_i$) of CloudScale and RPRP. RPRP can achieve higher resource efficiency due to its accurate burst-exclusive prediction algorithm and the resource-saving property of its load-dependent padding algorithm. CloudScale uses a possibly maximum or 80^{th} percentile values of high-frequency spectrum as padding for different demands, which wastes resources and leads to low utilization efficiency. RPRP's responsive padding further reduces allocated resources compared to CloudScale's responsive padding. To be comparable, we used the definition in [10] to measure the number of SLO violations. That is, an SLO violation



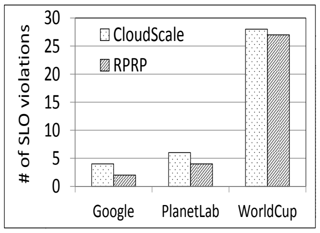
(a) Saved resource allocation



(b) Prediction error



(c) Resource utilization efficiency

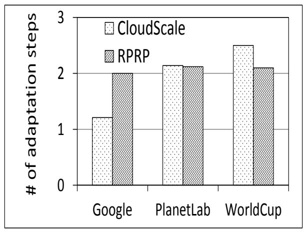


(d) The number of SLO violations

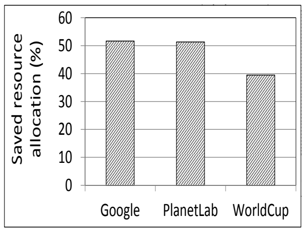
Figure 10: Performance of resource provisioning (prediction+padding).

is an under-prediction, i.e., the predicted demand is less than the actual demand. Figure 10(d) shows the number of SLO violations caused by under-predictions of RPRP and CloudScale from the three traces during the experiment time period. The number of under-predictions indicates the number of occurrences that the allocated resource (sum of predicted demand and padding) that failed to satisfy the actual demand during the experiment. We can see that RPRP generates a smaller number of under-predictions than CloudScale in all the three traces due to the same reasons indicated above.

4) Performance of Responsive Padding: Now, we analyze the responsive padding algorithms of CloudScale and RPRP without other algorithms. Figure 11(a) compares the number of adaptation steps for the responsive padding algorithms of CloudScale and RPRP. The minimum and maximum scale-up ratios in CloudScale are set to the default values in [10]. We can see that both of these two algorithms can quickly correct the mispredicted values in about two steps. The number of steps of RPRP is more stable than CloudScale under different traces because it does not require the tuning of parameters.



(a) The number of adaptation steps



(b) Saved resource allocation

Figure 11: Performance of the responsive padding algorithm.

Figure 11(b) shows the saved resource allocation from the responsive padding algorithms at the end of the experiment of RPRP compared to CloudScale. It shows that RPRP saves the allocated resource by 40%-50%. Using RPRP, a PM hosting 20 VMs can save up to 47348% absolute CPU resource in 24 hours. RPRP has a lower allocated resource than CloudScale in all the traces because its responsive padding algorithm can scale the resource cap up and down quickly to capture the actual resource demand, while the algorithm in CloudScale scales the resource cap to and stays at a high level.

5) Runtime overhead: In order to study the runtime overhead of the proposed algorithms, we tested the CPU time of each algorithm in RPRP compared with the corresponding algorithm in CloudScale with increasing problem sizes. Figure 12 presents the CPU time in seconds in logarithm scale for each algorithm. In this figure, *R* and *C* represents RPRP and CloudScale, respectively, and *-pre*, *-pad* and *-rp* represent the $\underline{\text{pre}}$diction algorithm, basic $\underline{\text{pad}}$ding algorithm and $\underline{\text{r}}$esponsive $\underline{\text{p}}$adding algorithm, respectively. The problem size for these algorithms are the number of predicted demands, the number of padding values to determine, and the number of adaptation steps. We see that *R-pre* is only slightly greater than *C-pre* (i.e., around 0.03–0.10 second) due to the reason that the FFT prediction (after FFT decomposition) in *R-pre* consumes more time than the average calculation (after using FFT to identify repeating patterns) in *C-pre*. *R-pre* remains in the same order of magnitude as *C-pre* as problem size increases. *C-pad* needs more time than *R-pad* and increases more rapidly than *R-pad*. This is because *C-pad* conducts FFT to determine high and low frequency components for the demand of each VM to determine padding value, while *R-pad* executes the dynamic program algorithm one time for determining all padding values in spite of its high time complexity. Note that *R-pad* runs periodically based on historical records to adapt to the changes. *R-rp* consumes less CPU time than *C-rp* because *R-rp* raises and decreases allocated resource more quickly than *C-rp* to reach the designed range. In total, CloudScale consumes more CPU time than RPRP, especially for the padding determination algorithm. From the figure, we can see that RPRP scales better than CloudScale.

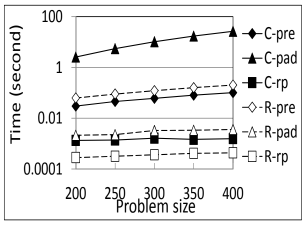
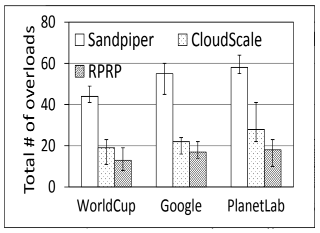


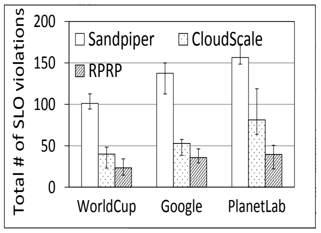
Figure 12: Runtime overhead.

B. Trace-driven Simulation

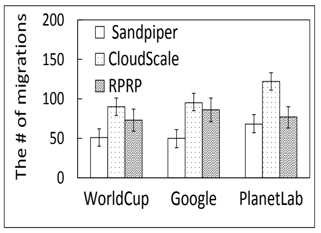
In this section, we evaluated the performance of RPRP in comparison with Sandpiper [25] and CloudScale [10] on the CloudSim simulator [32]. Sandpiper considers the multi-attribute feature of cloud resource and conducts VM migration from overloaded PMs based on the current VM



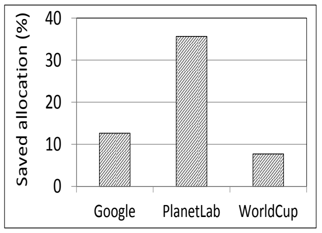
(a) Total number of overload PMs



(b) Total number of SLO violations



(c) The number of migrations



(d) Saved resource allocation

Figure 13: Trace-driven simulation results.

loads. Both RPRP and CloudScale employ online resource demand prediction and responsive padding (in RPRP) or underestimate handling (in CloudScale) algorithms to achieve prediction-based resource allocation, and use the VM migration method RIAL in [27] to conduct VM migration from PMs that are predicted to be overloaded and are actually overloaded. A PM's predicted load is the sum of the predicted demands of its VMs.

In the default setup, we set the number of PMs in the cloud to 1000, with capacities of 1.5GHz CPU, and 1536 MB memory, and set the number of VMs in the system to 2000, with capacities of 0.5GHz CPU and 512 MB memory. The threshold of resource utilization to check if a PM is overloaded was set to 0.9. We assume that the historical VM demand records are available at the beginning of simulation. At the beginning of the simulation, we randomly allocated the 2000 VMs to the 1000 PMs, which made a portion of the PMs overloaded (i.e., scaling conflict for VMs in the PM). After we started the simulation, every 5 minutes, Sandpiper uses the runtime utilization information to detect and eliminate overloaded PMs by VM migrations, and RPRP and CloudScale predict workload demands and migrate VMs out from predicted overloaded PMs and actually overload PMs. We measured the number of PM overloads and the number of VM migrations during one hour simulation time.

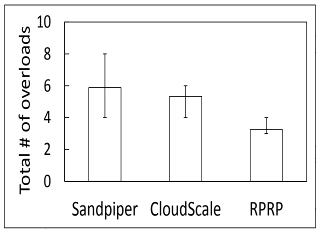
1) The Number of Overloads and SLO violations: Figure 13(a) shows the total number of overloads detected by PMs during one hour simulation. Figure 13(b) shows the total number of SLO violations during one hour simulation. The error bars indicate the 10^{th} and 90^{th} percentiles of the experimental results from ten repeated simulations. Both results in the two figures follows RPRP<CloudScale<Sandpiper. RPRP and CloudScale predict the VM resource demands and migrate out VMs from the PMs that are predicted to be overloaded. Thus, both of them reduce the number of overload occurrences and the number of SLO violations. We see that RPRP always generates fewer overloads and SLO violations than CloudScale in each trace, since it has more accurate forecasting and padding algorithms as we discussed in Section III. Compared to CloudScale, RPRP reduces 23%, 36% and 32% of the number of overloads in the Google Cluster, PlanetLab and WorldCup traces, respectively. It also reduces 32%, 51% and 41% of the number of SLO violations in the three traces, respectively. We further see that both the number of detected overloads and deviation (length of error bars) follow WorldCup<Google<PlanetLab due to their obviousness of demand patterns.

2) The Number of VM Migrations: Live VM migration would lead to a short service downtime, negatively affecting applications running on the migrated VMs. It also requires an extra amount of network bandwidth and cache warm-up at the destinations [39–41]. Thus, we should try to minimize the number of VM migrations. Figure 13(c) shows the number of migrations in the three methods with different workload traces. Sandpiper has the least VM migrations because it triggers migration only when detecting an overloaded PM, while CloudScale and RPRP conduct VM migration based on their predicted overloads, which may not be accurate. CloudScale produces more VM migrations than RPRP because its prediction accuracy is lower. RPRP is able to provide sufficient resource for the VMs in the cloud with a smaller number of VM migrations. The results indicate that the accurate resource demand prediction and padding algorithms of RPRP make it produce a comparable number of VM migrations to Sandpiper while producing fewer overload PMs than CloudScale.

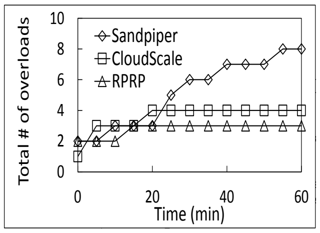
3) Total Allocated Resource: We calculated the cumulated allocated resources at the end of simulation. Figure 13(d) shows the saved resource allocation of RPRP compared to CloudScale. RPRP saves 12.6%, 35.6% and 7.7% of the allocated resource in the traces of Google Cluster, PlanetLab and WorldCup, respectively. RPRP can save 303%, 855% and 185% absolute CPU resource during one hour. This is because RPRP can more accurately predict resource demand and determine padding. The saved resource allocation follows WorldCup<Google<PlanetLab. This is because there are many bursts in the PlanetLab trace but fewer bursts in Google and WorldCup, leading to high estimation values in PlanetLab and more accurate estimation in Google and WorldCup. Compared to CloudScale, RPRP filters out bursts in prediction and avoids setting the padding to an amount as large as the burst in the past, and hence avoids resource waste.

C. Real-World Testbed Experiments

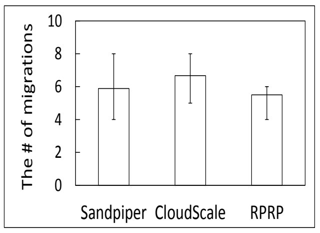
In this section, we evaluate the VM migration algorithm performance of RPRP, Sandpiper and CloudScale on a real-world testbed. The testbed consists of 5 PMs (2.00GHz Intel(R) Core(TM)2 CPU, 2GB memory, 60GB HDD) and a



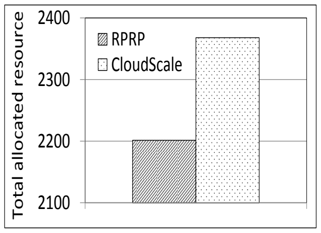
(a) The number of overloads



(b) Cumulated number of overloads



(c) The number of migrations



(d) Total allocated resource

Figure 14: Real-world testbed experimental results.

Network File System (NFS) server with a capacity of 80GB. We then implemented the three schemes in Java using the XenAPI library [42] running in a management PM (3.00GHz Intel(R) Core(TM)2 CPU, 4GB memory, running Ubuntu 11.04). The management PM gathered runtime resource utilization information every 5 minutes from the 5 PMs by calling XenAPI, and then sent out VM migration decisions to the 5 PMs asynchronously. All PMs were connected with a Cisco Catalyst 2950 switch. We created 11 VMs (1VCPU, 256MB memory, 8.0GB virtual disk, running Debian Squeeze 6.0) in the cluster and randomly assigned them to the PMs at the beginning of the experiment. We used the publicly available workload generator *lookbusy* [43] to generate CPU workloads of the VMs based on the Google Cluster trace. Similar to the trace-driven simulation, the VM migration algorithms were executed once every 5 minutes in the management PM. Since the workload deployed in the VMs keeps fluctuating, it is difficult to arrange exactly the same initial state for different strategies. To make the three methods comparable, we started testing different methods using the same initial VM-PM random mapping and ran each experiment 10 times. We assume that the historical VM demand records are available at the beginning of experiment. We used the historical records that cover 48 hours to predict the resource demands in the next 24 hours.

Figure 14(a) shows the median, 10^{th} and 90^{th} percentiles of the number of overloads detected during one hour experiment. Sandpiper has the highest median number since it triggers VM migrations only when an overloaded PM is detected. Also, it conducts VM migrations only based on current VM resource utilizations, thus failing to prevent overloading PMs in advance. RPRP outperforms CloudScale due to its more accurate prediction as discussed in Section III. The results are consistent with the simulation results in Figure 13(a) due to the same reasons. The reduction of the number of overloads here is not so obvious as in Figure 13(a) due to the smaller cluster size.

Figure 14(b) presents the cumulative number of overloads over time. Starting with the same VM-PM random mapping in the experiment, VM migration algorithms detect or predict overloaded PMs and trigger VM migrations. As shown in the figure, Sandpiper resolves currently overloaded PMs but cannot prevent future overloads, resulting in an increasing cumulative number of overloads over time. CloudScale and RPRP do not generate overloaded PMs after 20 minutes, which shows the advantage of demand prediction based resource provisioning. RPRP improves CloudScale by reducing the number of overloads due to more accurate demand prediction and padding determination. The figure also shows that most of the detected overloads by the two prediction methods are at the starting time of the experiments due to the initial random VM-PM mapping.

We notice that in the real-world testbed experiments, RPRP and CloudScale still cannot completely prevent the occurrence of overloaded PMs. This is caused by two reasons. First, due to the high variability and intensity of PM workloads in the clusters, the potentially overloaded PMs may not find PMs to migrate their VMs. Second, the VM migration algorithms consume computing and storage resource. When a PM is busy with VM migration, its resource utilization might reach a very high level, making the resource demand prediction more difficult.

Figure 14(c) presents the median, 10^{th} and 90^{th} percentiles of the number of VM migrations triggered by the three methods during one hour. Compared to CloudScale, RPRP has fewer migrations due to its accurate prediction, which helps avoid unnecessary migrations. These experimental results are consistent with the simulation results in Figure 13(c). Figure 14(d) shows the total allocated resource during the test. We can calcaulate that RPRP saves 7.0% of the allocated resource compared to CloudScale, due to the advantage of RPRP mentioned previously. The results confirm that RPRP significantly reduces the over-provisioning of CloudScale.

V. Conclusions

To reduce unnecessary resource provisioning while providing SLO guarantee, in this paper, we proposed RPRP, which consists of three algorithms: 1) a burst-exclusive prediction algorithm, 2) a load-dependent padding algorithm, and 3) a responsive padding algorithm. RPRP excludes bursts in deriving the dominant demand pattern and has specific algorithms to handle bursts in a resource-efficient manner. It adaptively determines padding based on demand amounts to minimize allocated resource while satisfying SLO. Its responsive padding algorithm adaptively adjusts co-located resources to more exactly meet the demands. We implemented RPRP on the top of Xen and conducted both trace-driven simulation and real-world testbed experiments.

The experimental results show the effectiveness of the proposed algorithms and that RPRP achieves higher resource utilization, more accurate demand prediction, and fewer SLO violations than previous schemes. Our experimental results show that algorithm 2) reduces 58.4% of the allocated resource of CloudScale. Also, by only using algorithms 1) and 2), RPRP reduces 18% of the allocated resource while reducing 30% of the total number of SLO violations compared with CloudScale on average. These reduced padding values are substantial in improving the resource utilization of a PM that hosts multiple VMs. In the future, we will extend RPRP to deal with resource provisioning for multiple co-located VMs with various priorities.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751. We would like to thank Dr. John Wilkes from Google for his valuable discussions and comments.

References

[1] "Amazon web service," http://aws.amazon.com/.
[2] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments." in *Proc. of EuroSys*, 2007.
[3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations." in *Proc. of IM*, 2007.
[4] J. Liu, H. Shen, and L. Chen, "Corp: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems," in *Proc. of MASS*, 2016.
[5] A. Sarker, C. Qiu, and H. Shen, "A decentralized network with fast and lightweight autonomous channel selection in vehicle platoons for collision avoidance," in *Proc. of MASS*, 2016.
[6] L. Yan and H. Shen, "TOP: vehicle trajectory based driving speed optimization strategy for travel time minimization and road congestion avoidance," in *Proc. of MASS*, 2016.
[7] A. Chandra, W. Gong, and P. J. Shenoy, "Dynamic resource allocation for shared data centers using online measurements." in *Proc. of SIGMETRICS*, 2003.
[8] Z. Li and H. Shen, "Designing a hybrid scale-up/out hadoop architecture based on performance measurements for high application performance," in *Proc. of ICPP*, 2015.
[9] Z. Gong, X. Gu, and J. Wilkes, "PRESS: Predictive elastic resource scaling for cloud systems." in *Proc. of CNSM*, 2010.
[10] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems." in *Proc. of SOCC*, 2011.
[11] L. Chen, H. Shen, and K. Sapra, "Distributed autonomous virtual resource management in datacenters using finite-markov decision process," in *Proc. of SOCC*, 2014.
[12] C. Qiu, H. Shen, and L. Chen, "Probabilistic demand allocation for cloud service brokerage," in *Proc. of INFOCOM*, 2016, pp. 1–9.
[13] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, and Y. Pan, "Stochastic load balancing for virtual resource management in datacenters," *IEEE Transactions on Cloud Computing*, 2016.
[14] C. Qiu, H. Shen, and L. Chen, "Towards green cloud computing: Demand allocation and pricing policies for cloud service brokerage," in *Proc. of Big Data*, 2015, pp. 203–212.
[15] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks." in *Proc. of SIGCOMM*, 2011.
[16] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing." in *Proc. of SIGCOMM*, 2012.
[17] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing." in *Proc. of HotPower*, 2008.
[18] B. Wu and H. Shen, "A time-efficient connected densest subgraph discovery algorithm for big data," in *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 305–314.
[19] Y. Hong, J. Xue, and M. Thottethodi, "Dynamic server provisioning to minimize cost in an IaaS cloud." in *Proc. of SIGMETRICS*, 2011.
[20] B. Wu and H. Shen, "Discovering the densest subgraph in mapreduce for assortative big natural graphs," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, 2015, pp. 1–6.
[21] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *Proc. of SOCC*, 2012.
[22] B. Kocoloski, J. Ouyang, and J. Lange, "A case for dual stack virtualization: consolidating hpc and commodity applications in the cloud," in *Proc. of SOCC*, 2012.
[23] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proc. of ICAC*, 2013.
[24] G. Jenkins, G. Reinsel, and G. Box, *Time Series Analysis: Forecasting and Control.* Prentice Hall, 1994.
[25] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration." in *Proc. of NSDI*, 2007.
[26] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers." in *Proc. of SC*, 2008.
[27] L. Chen, H. Shen, and K. Sapra, "RIAL: Resource intensity aware load balancing in clouds." in *Proc. of INFOCOM*, 2014.
[28] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. J. Shenoy, "Seagull: Intelligent cloud bursting for enterprise applications." in *Proc. of ATC*, 2012.
[29] Y. Lin, H. Shen, and L. Chen, "Ecoflow: An economical and deadline-driven inter-datacenter video flow scheduling system," in *Proc. of Multimedia*, 2015, pp. 1059–1062.
[30] L. Chen, H. Shen, and S. Platt, "Cache contention aware virtual machine placement and migration in cloud datacenters," in *Proc. of ICNP*, 2016.
[31] "Google Cluster Data," https://code.google.com/p/googleclusterdata/.
[32] R. Calheiros, R. Ranjan, A. Beloglazov, C. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *SPE*, 2011.
[33] "The IRCache Project," http://www.ircache.net/.
[34] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Capacity management and demand prediction for next generation data centers." in *Proc. of ICWS*, 2007.
[35] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters." in *Proc. of INFOCOM*, 2014.
[36] "Service Level Agreements," http://azure.microsoft.com/en-us/support/legal/sla/.
[37] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "VMware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, 2012.
[38] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. of SOCC*, 2012.
[39] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines." in *Proc. of NSDI*, 2005.
[40] A. Nagarajan, F. Mueller, C. Engelmann, and S. Scott, "Proactive fault tolerance for HPC with Xen virtualization." in *Proc. of ICS*, 2007.
[41] L. Chen, S. Patel, H. Shen, and Z. Zhou, "Profiling and understanding virtualization overhead in cloud," in *Proc. of ICPP*, 2015.
[42] "Xenapi," http://www.xenproject.org/developers/teams/xapi.html.
[43] "lookbusy," http://devin.com/lookbusy/.