Efficient Resource Provisioning in Compute Clouds via VM Multiplexing

Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet,
Dimitrios Pendarakis
IBM T. J. Watson Research Center
Hawthorne, NY 10532
{xmeng, canturk, kephart, zhangli, ericbou, dimitris}@us.ibm.com

ABSTRACT

Resource provisioning in compute clouds often requires an estimate of the capacity needs of Virtual Machines (VMs). The estimated VM size is the basis for allocating resources commensurate with demand. In contrast to the traditional practice of estimating the size of VMs individually, we propose a joint-VM provisioning approach in which multiple VMs are consolidated and provisioned together, based on an estimate of their aggregate capacity needs. This new approach exploits statistical multiplexing among the workload patterns of multiple VMs, i.e., the peaks and valleys in one workload pattern do not necessarily coincide with the others. Thus, the unused resources of a low utilized VM can be borrowed by the other co-located VMs with high utilization. Compared to individual-VM based provisioning, joint-VM provisioning could lead to much higher resource utilization. This paper presents three design modules to enable such a concept in practice. Specifically, a performance constraint describing the capacity need of a VM for achieving a certain level of application performance; an algorithm for estimating the aggregate size of multiplexed VMs; a VM selection algorithm that seeks to find those VM combinations with complementary workload patterns. We showcase that the proposed three modules can be seamlessly plugged into applications such as resource provisioning, and providing resource guarantees for VMs. The proposed method and applications are evaluated by performance data collected from about 16 thousand VMs in commercial data centers. The results demonstrate more than 45% improvements in terms of the overall resource utilization.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Modeling techniques

General Terms: Algorithms, Management, Measurement

Keywords: Cloud computing, Provisioning, Virtualization

1. INTRODUCTION

In modern virtualization based compute clouds, applications share the underlying hardware by running in isolated Virtual Machines (VMs). Each VM, during its initial creation, is configured with a certain amount of computing resources (such as CPU, memory and I/O). A key factor for achieving economies of scale in a compute cloud is resource provisioning, which refers to allocating resources to VMs to match their workload. Typically, efficient provisioning is achieved by two operations: (1) static resource provisioning. VMs are created with specified size and then consolidated onto a set of physical servers. The VM capacity does not change; and (2) dynamic resource provisioning [20, 18, 15]. VM capacity is dynamically adjusted to match workload fluctuations. Static provisioning often applies to the initial stage of capacity planning. It is usually conducted in offline and occurs on monthly or seasonal timescales [7, 32]. Such provisioning functionality has been included in many commercial cloud management softwares [27, 28, 11, 19, 16].

In both static and dynamic provisioning, VM sizing is perhaps the most vital step. VM sizing refers to the estimation of the amount of resources that should be allocated to a VM. The objective of VM sizing is to ensure that VM capacity is commensurate with the workload. While over-provisioning wastes costly resources, under-provisioning degrades application performance and may lose customers. Traditionally, VM sizing is done on a VM-by-VM basis, i.e., each VM has an estimated size based on its workload pattern. In a significant departure from such an individual-VM based approach, we advocate a *joint-VM provisioning* approach in which multiple VMs are consolidated and provisioned based on an estimate of their aggregate capacity needs. Conceptually, joint-VM provisioning exploits statistical multiplexing among the dynamic VM demand characteristics, i.e., the peaks and valleys in one VM's demand do not necessarily coincide with the other VMs. The unused resources of a low utilized VM, can then be directed to the other co-located VMs at their peak utilization. Thus, VM multiplexing potentially leads to significant capacity saving compared to individual-VM based provisioning. The savings achieved by multiplexing are realized by packing VMs more densely into hardware resources without sacrificing application performance commitment. While this increases the overall consolidation ratio, the additional virtualization overheads associated with scheduling somewhat higher number of VMs is generally minimal as long as the VM footprints fit in the provisioned capacity [31]. The savings with our joint-sizing approach are up to 40% according to our analysis on the utilization data from a production data center.

The above seemingly simple concept poses several research challenges. For example, given a set of VMs to be consolidated and provisioned, how to estimate their total capacity needs that neither break application performance commitments nor waste resources? Since any combination of VMs can be potentially provisioned together, how to find combinations that saves the most capacity? What are the potential scenarios for applying this technique in compute clouds? In this work, we address these questions in detail. Specifically, the primary contributions of this work are:

- We introduce a Service-level-agreement (SLA) model that map application performance requirements to resource demand requirement. We propose a systematic method to estimate the total amount of capacity for provisioning multiplexed VMs. The estimated aggregate capacity ensures that the SLAs for individual VMs are still preserved.
- We present a VM selection algorithm that seeks to find those VMs with the most compatible demand patterns. The identified VM combinations lead to high capacity savings if they are multiplexed and provisioned together.
- We illustrate effective and feasible applications of the proposed technique for capacity planning and for providing resource guarantees via VM reservations. Both applications can be easily employed in existing cloud and virtualization management infrastructures with minimal intrusion and substantial benefits in return.

We conduct simulations to evaluate the proposed methods by using a massive dataset collected from commercial data centers. The dataset includes 159-thousand VMs and spans three months. In the capacity planning application, joint provisioning uses 45% less physical machines for hosting the same set of VMs. In the VM reservation application, joint provisioning improves the ratio of admitted VMs by 16% on average, and up to 75% with more stringent SLA requirements. These results demonstrate the significant potential by leveraging VM multiplexing.

The rest of the paper is organized as follows. Section 2, surveys prior work related to resource provisioning and VM multiplexing. Section 3, motivates the use of VM multiplexing to improve resource utilization efficiency. It uses data from commercial global data centers to demonstrate the potential capacity gains with joint-VM based provisioning. Section 4, provides an overview of our methodology. Section 5, describes the underlying SLA model for resource provisioning. Section 6 describes the algorithm for joint-VM sizing. Section 7 presents a method for selecting compatible VMs for multiplexing. Section 8 discusses the use cases for VM multiplexing and provides the experimental evaluations. Last, Section 9 offers our conclusions.

2. RELATED WORK

As resource provisioning is a common management task in modern virtualization-based compute clouds, it has become an important ingredient of commercial cloud management products including VMware Capacity Planner [27] and CapacityIQ [28], IBM WebSphere CloudBurst [11], Novell PlateSpin Recon [19] and Lanamark Suite [16]. Meanwhile, the topic has been widely studied in the research community. Some existing work apply application/VM profiling and statistical modeling for long-term resource provisioning [1, 7, 8, 18]. Other work focus on short-term, dynamic provisioning techniques [20, 18, 15]. While all these products and research work provide valid solutions, they stand on a VM-by-VM basis, that is, they consider each VM's resource need separately. Such an approach provides a reasonably effective solution to the provisioning problem, yet it generally leads to low resource utilization [13, 22, 25]. In contrast, our work exploit the workload multiplexing among multiple VMs. We demonstrate that how multiple VMs' capacity requirement can be satisfied collectively with a much lower total resource consumption.

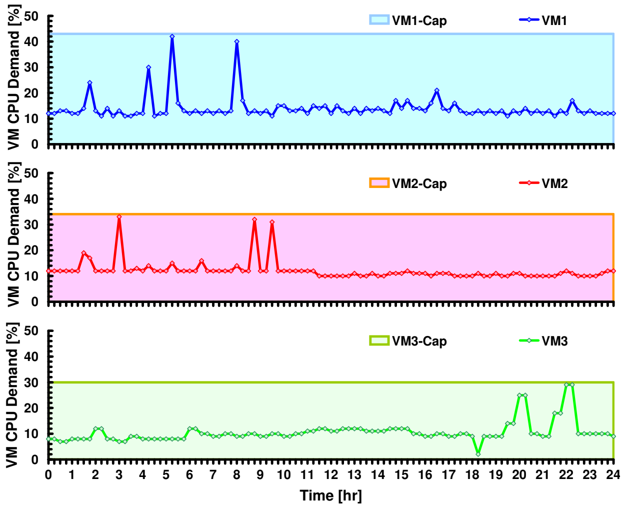


Figure 1: Individual VM capacity requirements.

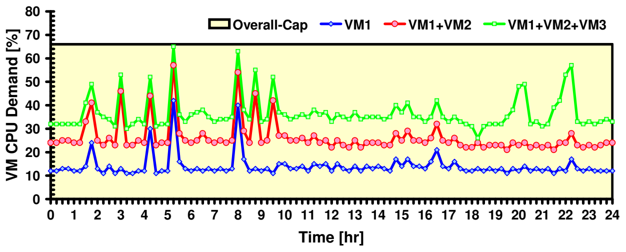


Figure 2: Aggregate capacity requirement for multiplexed VMs.

A few prior arts consider concepts similar to VM multiplexing for improving resource utilization. Specifically, Sonnek and Chandra [24] identify VMs that are most suitable for being consolidated on a single host. They propose to multiplex VMs based on their CPU and I/O boundedness, and to co-locate VMs with higher potential of memory sharing. Wood et al. [33] present a method for co-locating VMs with similar memory content on the same hosts for higher memory sharing. Gupta et al. [10] further progress this memory sharing method by limiting memory sharing within page boundaries. Govindan et al. [9] propose to consolidate VMs based on their communication patterns. Govindan et al. [5, 8] use statistical multiplexing of applications to identify applications that fit into given power budgets. In comparison to all these studies, our work also apply VM multiplexing to consolidate VMs more densely on hosts. Nevertheless, our work provide a general performance model and enabling techniques that ensure the application performance is not degraded by VM multiplexing.

3. MOTIVATION

Our proposed *joint-VM provisioning* approach stems from

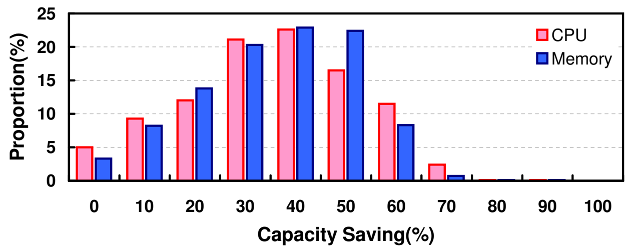


Figure 3: Potential capacity savings for VM provisioning using VM multiplexing.

an observation on the VM resource demand in actual data centers. It is well known that the applications enclosed by VMs - and thus the VMs themselves - exhibit time-varying resource demand patterns with bursts of high demand periods, intermixed with low-utilization regions [3, 23, 26]. Furthermore, our measurement on a large set of VMs shows that many VMs, even in the same data center, exhibit demand patterns with different, unaligned distributions of these *peaks* and *valleys*. Therefore, while a capacity planner that operates over singleton VMs is bound by the peaks of *each* individual VM, a joint-VM approach can potentially exploit the *multiplexing* among the demand patterns of multiple VMs to reach an aggregated capacity measure that is only bound by the aggregate peak behavior.

Figures 1 and 2 illustrate an example with three VMs from a production data center. Figure 1 depicts the monitored CPU demand of each VM over a 24-hour period. Each VM exhibits a time-varying demand pattern with interspersed peaks. However, the peaks for each VM occur at different time. The figure also depicts a simple capacity bound required by each VM that is based on conservatively satisfying each instantaneous peak, i.e., the capacity required for each VM is set as the maximum demand ever observed in the history time period. Based on this capacity model, the total capacity required for the three VMs is 104%.

In contrast, Figure 2 depicts the *multiplexed* behavior of the three VMs when they are jointly provisioned. The figure shows the aggregated CPU demand. Here we see the increased number of peaks in the aggregated demand. The total capacity required to satisfy the demand of all the three VMs is only 67%, a dramatic improvement compared to the separate provisioning scenario.

To assess the potential capacity savings with multiplexing in VM capacity planning at the enterprise scale, we extend the above comparison to a massive dataset collected from a set of commercial data centers. The dataset involves 15,897 VMs that reside on 1325 physical hosts, managed by tens of regional hosting operators and used by hundreds of enterprise customers. The dataset includes configurations of each host, and the CPU and memory utilization ratio of each VM for up to three months. All the evaluation in the rest of this work is based on this dataset.

In this assessment, we first identify that 94% of the 1325 hosts contains more than one VM. For each of such hosts, similar to the previous example, we compare the sum of VM capacity needs between using the separate and the joint provisioning. Both the CPU and memory resources are considered in this comparison. In either provisioning method, the peak demand is still used to bound the capacity. Figure 3 plots the histogram of the capacity savings achieved by joint provisioning for all those hosts with more than one VM. The histogram shows significant savings by the joint provisioning. Overall, the average CPU (or memory) capacity saving over individual VM provisioning is around 40%, which clearly demonstrates the compelling potential benefits of VM multiplexing.

4. METHODOLOGY OVERVIEW

Our VM multiplexing and joint-VM provisioning approach is composed of three function modules, which collectively capture the necessary steps for defining the multiplexed VMs, and their individual and joint capacity requirements. These three modules include: (1) a general SLA imposed on VM capacity; (2) a joint-VM sizing algorithm that calculates the total capacity needs for multiplexed VMs; and (3) a VM selection algorithm that identifies *compatible* VM combinations for being consolidated and provisioned jointly. Below, we describe how these three modules cooperate within a general resource provisioning framework.

Given a set of VMs, the VM selection algorithm identifies VM combinations that achieve high capacity savings if provisioned together. The selection criterion is how complementary the VM demand patterns are. Highly complementary VMs will be grouped together by the selection algorithm. Eventually the selection algorithm partitions VMs into multiple sets. Those VMs in the same set will be consolidated onto the same physical server and thus can be considered as a *super VM*. To provision such a super VM, we first need to calculate its aggregate capacity need. To this end, we introduce a SLA model and a joint-VM sizing algorithm. The SLA model defines a relation between the VM capacity and the performance level of the applications running on the VM. Moreover, the SLA model makes it convenient to derive a constraint on the super VM capacity simply based on specified SLA for individual VM. Based on the derived constraint and the aggregate workload of the super VM, the joint-VM sizing algorithm proceeds to calculate the super VM's capacity need, which is the minimum amount of resources that should be allocated to the super VM without degrading individual VM's SLA.

We apply such a VM multiplexing approach and demonstrate its benefits in two cloud management operations. The first application is VM consolidation. We identify compatible VMs, provision them jointly in a compute cloud and significantly reduce hardware requirement. Second, we define a *joint reservation* model to provide VM-level resource guarantees in a virtualized environment. By identifying compatible VMs and their SLA, we are able to derive a joint reservation level based on individual VM reservations. We group compatible VMs in resource pools, and enforce joint reservations at the resource pool level. All of these enable dramatically improved VM consolidation ratio in the cloud.

5. VM SLA MODEL

In this section, we describe a SLA model that is used as the basis for determining VM capacity. We consider a discrete-time scenario in which time is slotted into intervals with equal length. We first define the SLA model for a single VM.

DEFINITION 1 (SINGLE-VM SLA MODEL). *Suppose VM i is allocated a fixed capacity within a time frame $[1, L]$. Let $x_i(t)$ denote VM i's workload volume in time slot t. A con-*

straint on the capacity of VM i is expressed as

$$\frac{1}{k_i}\sum_{s=0}^{k_i-1}\mathcal{I}(\frac{\sum_{t=sk_i+1}^{(s+1)k_i}x_i(t)}{T_i}\ \textit{exceed capacity})\le\beta_i \quad (1)$$

where $T_i\in\{1,2,4,\ldots\}, k_i=\frac{L}{T_i}$ *is an integer*, $0\le\beta_i\le 1$

In the above inequality, $\mathcal{I}$ is defined as

$$\mathcal{I}(y)=\begin{cases}1, & \text{if } y \text{ is true}\\ 0, & \text{if } y \text{ is false}\end{cases}$$

In Inequality (1), $\frac{\sum_{t=sk_i+1}^{(s+1)k_i}x_i(t)}{T_i}$ is the average workload in a time interval of length T_i. The above SLA constraint is interpreted as following: if the entire time frame is divided into multiple intervals with equal length T_i, the proportion of the intervals in which the cumulative workload exceeds the VM capacity must be no more than a threshold β_i. The two parameters T_i and β_i are specified according to the performance requirement of the running applications. T_i reflects how much latency on average a request from the applications is expected to tolerate. The value of T_i should be close to the normal response time experienced by the applications. For a reason explained later, T_i is only allowed to take a value as a power of two. β_i is a threshold limiting the chance of capacity violation. The above SLA model is quite general and can be applied to match various application performance requirements. For example, time-sensitive applications such as multimedia often require an immediate fulfillment of their workload demand. A SLA model with smaller T_i and β_i may fit well. On the other hand, time-elastic, long-running applications such as disk backup are usually tolerable of infrequent short-term capacity insufficiency. Accordingly a SLA model with large T_i and β_i is more suitable. It is worth mentioning that when both T_i and β_i are close to zero, this SLA model is equivalent to the peak-load based sizing strategy mentioned in Section 3.

The proposed SLA model is used to derive the capacity needed for provisioning VMs. Obviously, once T_i and β_i are given, the single-VM sizing problem is equivalent to finding the minimum capacity that satisfies Inequality (1). Now we further extend the SLA model to the joint-VM case, in which multiple VMs are consolidated and provisioned together.

Definition 2 (Joint-VM SLA model). *Suppose n VMs are provisioned jointly and allocated a fixed capacity within a time frame $[1, L]$. Given n parameter tuples (T_i, β_i) ($T_i\in\{1,2,4,\ldots\}$, $0\le\beta_i\le 1$), a constraint imposed on the capacity of the n VMs is following*

$$\frac{1}{k}\sum_{s=0}^{k-1}\mathcal{I}(\frac{\sum_{t=sk+1}^{(s+1)k}\sum_{i=1}^{n}x_i(t)}{T}\ \textit{exceed capacity})\le\beta \quad (2)$$

where $T=\min_i T_i, k=\frac{L}{T}, k$ *is an integer*, $\beta=\min_i T_i\min_i\frac{\beta_i}{T_i}$

Compared to Inequality (1), the only difference in (2) is to replace individual VMs' workload by the sum of all VMs' workload. We can prove the following theorem

Theorem 1. *If n VMs satisfy the joint-VM SLA constraint in Inequality (2), each VM must also satisfy the single-VM SLA constraint in Inequality (1) with parameters T_i and β_i.*

Proof: We only need to show that if Inequality (2) holds, then Inequality (1) holds for any VM i. Because every T_i is a power of 2 and T is the minimum among all T_i, a time interval $[1, T_i]$ can be divided into $\frac{T_i}{T}$ shorter intervals. If the average of $\sum_i x_i(t)$ in the period $[1, T_i]$ exceeds capacity, there must be at least one shorter interval in which the average exceeds capacity as well. This fact leads to

$$\mathcal{I}(\frac{\sum_{t=1}^{T_i}\sum_i x_i(t)}{T_i}\ \textit{e.c.})\le\sum_{s=0}^{\frac{T_i}{T}-1}\mathcal{I}(\frac{\sum_{t=sT+1}^{(s+1)T}\sum_i x_i(t)}{T}\ \textit{e.c.})$$

where *e.c.* is an abbreviation of "exceed capacity".

The above inequality is only for the interval $[1, T_i]$. Similar inequalities hold for the other intervals with length T_i, e.g., $[T_i+1, 2T_i], [2T_i+1, 3T_i], \ldots$. By summing up all such inequalities on their left and right sides respectively, we have

$$\sum_{s=0}^{\frac{L}{T_i}-1}\mathcal{I}(\frac{\sum_{t=sT_i+1}^{(s+1)T_i}\sum_i x_i(t)}{T_i}\ \textit{e.c.})\le\sum_{s=0}^{\frac{L}{T}-1}\mathcal{I}(\frac{\sum_{t=sT+1}^{(s+1)T}\sum_i x_i(t)}{T}\ \textit{e.c.})$$

Because Inequality (2) holds, the right side of the above inequality is no more than $\beta\frac{L}{T}=\beta\frac{L}{\min_i T_i}$, so

$$\sum_{s=0}^{\frac{L}{T_i}-1}\mathcal{I}(\frac{\sum_{t=sT_i+1}^{(s+1)T_i}\sum_i x_i(t)}{T_i}\ \textit{e.c.})\le\beta\frac{L}{\min_i T_i}$$

Considering $\beta=\min_i T_i\min_i\frac{\beta_i}{T_i}$, we further have

$$\sum_{s=0}^{\frac{L}{T_i}-1}\mathcal{I}(\frac{\sum_{t=sT_i+1}^{(s+1)T_i}\sum_i x_i(t)}{T_i}\ \textit{e.c.})\le L\min_i\frac{\beta_i}{T_i}\le L\frac{\beta_i}{T_i}$$

which becomes Inequality (1) and proves that VM i complies with the single-VM SLA constraint with parameters T_i and β_i. □

The insight behind the above proof is that the SLA constraint in the joint case is taking the most stringent one from all individual VM's constraint. Thus, when a joint-VM size ensures that the joint-VM constraint is satisfied, it is guaranteed that individual VM's SLA must be satisfied. This important property becomes the foundation for the joint-VM sizing algorithm presented in the next section. Also, it is worth mentioning that when all VMs have identical T_i and identical β_i, there will be $T=T_i$ and $\beta=\beta_i$ in Definition 2, i.e., since every VM has the same SLA , the SLA of the joint-VM will take the same as well.

6. JOINT-VM SIZING

This section describes how to estimate the capacity (hence after referred to as c) needed for provisioning n VMs. In a nutshell, this method applies both timeseries forecasting and statistical modeling to infer future workload patterns, then it finds c by solving the SLA constraint in (2). The inputs to this method are the historitic workload timeseries and the parameters in the SLA model for each VM. The output is the estimated c. Specifically, this method first uses Definition 2 to determine T and β that should be used in the joint-VM SLA constraint. The total workload for all VMs is then computed. The total workload is further projected to the future by standard timeseries forecasting techniques. Since any forecasting technique is subject to error, the forecasting error is modeled by statistical distributions. The forecasted workload plus the forecast error model constitutes a complete description of the future workload, which is further used to compute c based on the joint-VM SLA constraint. The flowchart of this method is provided in Figure 4.

The remainder of this section addresses three major steps in the above procedure: Section 6.1 discusses workload forecasting (lines 2-6 in Figure 4) . Section 6.2 presents two approaches to deriving the forecast error distribution (lines 7-11). The final step, computing c (lines 12-14), is presented in Section 6.3.

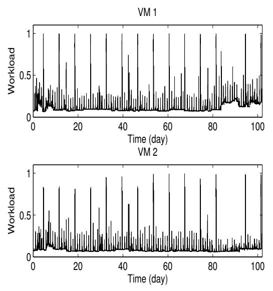
```
Input: $n$ VM workload timeseries $x_i(t)$
       $n$ VM constraint parameters $(T_i,\beta_i)$
1.  Derive constraint parameters $T$ and $\beta$ by Defintion 2
2.  For each VM $i$
3.      Decompose $x_i(t) \longmapsto x_i(t) = \hat{x}_i(t) + \tilde{x}_i(t)$
          $\hat{x}_i(t)$: trend and seasonal components
          $\tilde{x}_i(t)$: irregular fluctuations
4.  $\hat{x}(t) = \sum_i \hat{x}_i(t), \tilde{x}(t) = \sum_i \tilde{x}_i(t)$
5.  Forecast $\hat{x}(t)$ in future time frame $[1, L]$
6.  Forecast $\tilde{x}(t)$ in future time frame $[1, L]$
7.  $\tilde{x}_e(T)$: forecasting error accumulated in $T$ timeslots
8.  If forecast method has explicit error model
9.      Compute probability density $f(\tilde{x}_e(T))$
10. else
11.     Infer $f(\tilde{x}_e(T))$ by past forecasting errors
12. For every $[1+sT, (s+1)T](s = 0, 1, \ldots)$ in $[1, L]$
13.     Compute minimum $c(s)$ s.t.
        $Prob\{\sum_{t=1+sT}^{(s+1)T}(\tilde{x}(t) + \hat{x}(t)) + \tilde{x}_e(T)) > c(\tau)T\} \leq \beta$
14. Find $c = \max_s c(s)$
```

Figure 4: Flowchart for Joint-VM sizing

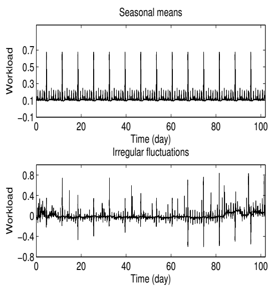
6.1 Workload forecasting

Prior to forecasting, the workload for each VM is decoupled into regular and irregular fluctuating components. The regular workload refers to those deterministic patterns such as trends, cycles and seasonality. Various timeseries techniques [14] can be applied to extract such patterns. The irregular fluctuating workload is the residual after removing the regular ones. For ease of illustration, let $x_i(t)$ denote the workload of VM i $(i = 1, 2, \ldots, n)$, $\hat{x}_i(t)$ and $\tilde{x}_i(t)$ denote the regular and fluctuating workload respectively. The next step is to forecast the aggregate regular workload $\sum_i \hat{x}_i(t)$ and the aggregate fluctuating workload $\sum_i \tilde{x}_i(t)$. These two types of workload are forecasted separately. Forecasting $\sum_i \hat{x}_i(t)$ is done by simply assuming the regular patterns will preserve in the future, e.g., a steadily increasing trend keeps increasing at the same rate; a daily seasonality continues to hold. On the other hand, forecasting $\sum_i \tilde{x}_i(t)$ is performed by a timeseries forecasting technique such as linear regression, ARMA and neural networks. Because our method is compatible with any forecasting technique and there is no single technique widely agreed to be the best one, we take a best practice approach in which we keep comparing the forecasting accuracy of those popular predictors based on historic workload patterns and choosing the better one periodically.

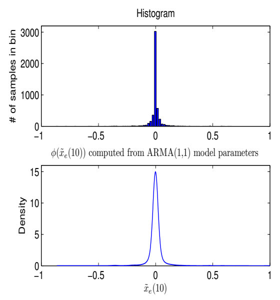
An alternative strategy is to directly apply trend, seasonality detection and forecasting to $\sum_i x_i(t)$ without decomposing $x_i(t)$. This brings a potential drawback that a regular pattern in $x_i(t)$ could become difficult to extract after multiple $x_i(t)$ are aggregated. For example, $x_i(t)$ has a regular pattern but the average workload rate is small, $x_j(t)$ contains only fluctuating workload with much higher average rate. After aggregating $x_1(t)$ and $x_2(t)$, the regular pattern in $x_1(t)$ could easily be unrecognizable.



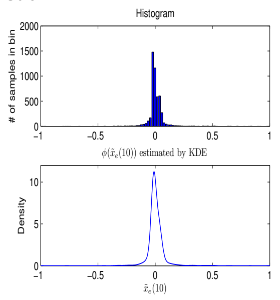
(a) Two VMs' CPU workload



(b) Decompose VM 1's workload



(c) $\phi(\tilde{x}_e(10))$ computed from ARMA(1,1)



(d) $\phi(\tilde{x}_e(10))$ estimated by KDE with FNN

Figure 5: An example of joint-VM sizing by using two VMs' CPU utilizations

Figure 5(a) shows two VMs' CPU workload in a 100-day period (hereafter referred to as VM 1 and 2). Figure 5(b) shows the regular and fluctuating patterns extracted from VM 1's workload. Clearly VM 1's workload has daily and weekly seasonality. Compared with the original workload of VM 1 (top of Figure 5(a)), the fluctuating workload (bottom of Figure 5(b)) shows less variability and easier to forecast.

6.2 Modeling forecasting error

The above forecasting process decomposes $\sum_i x_i(t)$, the aggregate workload at a future time t, into

$$\sum_{i=1}^{n} x_i(t) = \sum_{i}^{n} \hat{x}_i(t) + \sum_{i}^{n} \tilde{x}_i(t) + e(t) \tag{3}$$

where $e(t)$ is the forecasting error (or uncertainty) at t. Because the SLA constraint (see Inequality 2) is concerned about the sum of workload accumulated in a time interval $[\tau, \tau + T]$, we need to compute the following

$$\sum_{t=\tau}^{\tau+T} \sum_{i=1}^{n} x_i(t) = \sum_{t=\tau}^{\tau+T} \sum_{i=1}^{n} \hat{x}_i(t) + \sum_{t=\tau}^{\tau+T} \sum_{i=1}^{n} \tilde{x}_i(t) + \sum_{t=\tau}^{\tau+T} e(t) \tag{4}$$

where the third term on the right side is the forecast error accumulated over the time interval. In general, forecast error is assumed to be a stationary stochastic process. Thus we can use $\tilde{x}_e(T)$ to denote the accumulated forecast error in any time interval of length T, i.e.,

$$\tilde{x}_e(T) = \sum_{t=\tau}^{\tau+T} e(t) \ (\forall \tau) \tag{5}$$

The next step is to derive the probability density function for $\tilde{x}_e(T)$. Depending on whether the used forecasting method provides an explicit model for $e(t)$, we have the following two approaches.

6.2.1 With explicit forecast error model

Some forecasting algorithms such as ARMA provide an explicit model for the forecasting error e. Their common approach is to model $e(t)$ as

$$e(t) = \epsilon(t) - \theta_1\epsilon(t-1) - \theta_2\epsilon(t-2) - \ldots - \theta_m\epsilon(t-m) \quad (6)$$

where $\epsilon(t)$ is normal white noise with variance σ^2. Combining (6) with (5), $\tilde{x}_e(T)$ becomes

$$\tilde{x}_e(T) = \sum_{t=\tau-m}^{\tau+T-m} \alpha_t\epsilon(t) \quad (7)$$

where α_t is a linear combination of θ_i ($i \in [1, \ldots, m]$) and can be easily computed from (6).

Because $\epsilon(t)$ is normal white noise, $\tilde{x}_e(T)$ is the sum of a set of normal random variables. Thus $\tilde{x}_e(T)$ is a normal random variable with mean 0 and variance $\sum_{t=\tau-m}^{\tau+T-m} \alpha_t^2\sigma^2$. Back to the example in Figure 5(a) and 5(b), we use ARMA(1,1) to forecast $\tilde{x}(t)$. The top of Figure 5(c) is the histogram for all $\tilde{x}_e(10)$ samples, which are obtained by subtracting forecasted values from the actual values in the model training phase. The bottom of Figure 5(c) is the computed normal distribution which fits the histogram well.

6.2.2 Without explicit forecast error model

For many forecasting methods such as neural network based algorithms, they do not provide an explicit model for the forecast error. Our strategy is to collect many realizations of $\tilde{x}_e(T)$ and to use them to obtain an empirical density function. Specifically, we apply such a forecasting method to historical workload and compute forecasting error at every past time point. The forecasting error summed over any interval of length T constitutes a realization for $\tilde{x}_e(T)$. Next, we apply the classical Kernel Density Estimation (KDE) technique [21] to construct an empirical density function by using all the collected realizations for $\tilde{x}_e(T)$. KDE is chosen because it is a non-parametric method so there is no need to make any assumption on the distribution of $\tilde{x}_e(T)$. The KDE implementation used in this work is from Ihler and Mandel's Toolbox [12]. To illustrate this procedure by an example, we apply the Feed-forward Neural Network (FNN) algorithm [17] to forecast $\tilde{x}(t)$ which is obtained from the aforementioned VM 1 and 2. While the top of Figure 5(d) is the histogram of collected realizations for $\tilde{x}_e(T)$, the bottom shows the corresponding density function estimated by KDE. Again, the estimated density function fits the histogram well.

6.3 Computing joint size c

The final step is to compute c based on Inequality (2). For this purpose, we need to compute the aggregate workload in a future time interval with length T. Let $[\tau, \tau+T]$ denote such an interval, according to (4), the aggregate workload in this interval consists of three components, among which the last one, $\tilde{x}_e(T)$, is probabilistic. Thus, Inequality (2) is

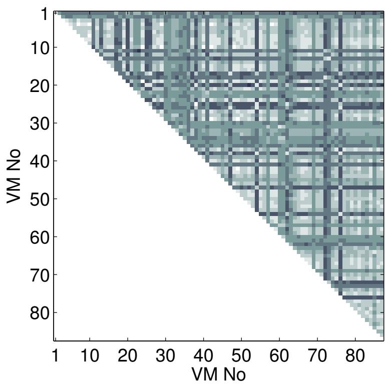


Figure 6: Initial correlation matrix

equivalent to

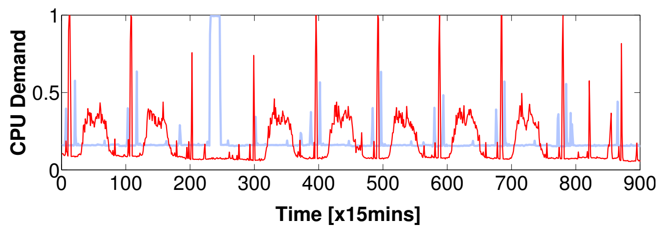
$$c \geq \sum_{t=\tau}^{\tau+T}\sum_{i=1}^{n}\hat{x}_i(t) + \sum_{t=\tau}^{\tau+T}\sum_{i=1}^{n}\tilde{x}_i(t) + \phi^{-1}(1-\beta) \quad (8)$$

where Φ^{-1} is the quantile-function for $\tilde{x}_e(T)$. Here a quantile-function is defined as the inverse of the CDF. For the right side of (8), the first two terms are known after the workload forecasting phase, and the last term is also known after computing the density function for $\tilde{x}_e(T)$. Thus, for every future time interval of length T, we use (8) to determine a lower bound for c. In each future time interval of length T, we obtain such a lower bound. By iterating all such time intervals that are of interests, we find out the maximum lower bounds and use it as the estimated c.

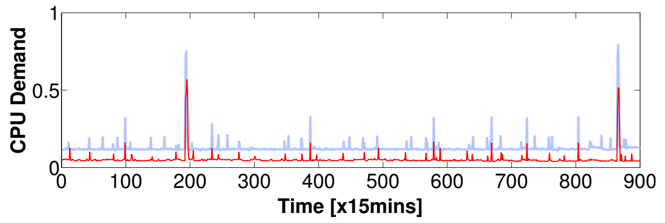
7. VM SELECTION FOR JOINT SIZING

The key advantage of joint sizing stems from the fact that the workload peaks and troughs of different VMs do not exhibit identical temporal patterns. Obviously an important factor contributing to the realized capacity savings with VM multiplexing is the way VMs are combined: a method that combines VMs with similar temporal peak and trough patterns cannot achieve savings compared to a method that identifies and favors VMs with *complementary* temporal behavior. Therefore, in this section we describe a method to identify VMs with complementary demand patterns. As a relevant problem, virtualized clusters often employ additional resource allocation constraints such as affinity and anti-affinity rules that describe which entities need to be or cannot be placed together. Their techniques can also be incorporated into our VM selection process.

Our VM selection method uses the correlations among VMs as the indicator of their compatibility. For a given set of VMs, we first build *correlation matrix* C based on the historical or forecasted demand behavior of each VM. Such a correlation matrix is commonly employed in similarity analysis techniques such as principle component analysis and factor analysis [6]. Each entry $C(i, j)$ is the Pearson's correlation coefficient between the workload timeseries of the two VMs i and j. We consider VM pairs with strong negative correlations as good candidates for multiplexing because a negative correlation indicates that these VMs' demand behavior changes in opposite directions. Figure 6 demonstrates the upper-diagonal correlation matrix for an example actual cluster of 87 VMs. Here, the correlation matrix is shown as an intensity plot with higher (pos-



(a) VMs with Strong Negative Correlation



(b) VMs with Strong Positive Correlation

Figure 7: Demand patterns two VM pairs.

itive)correlations shown brighter and lower (negative) correlations shown darker. Therefore, good candidate VMs for joint sizing lie in the intersections of dark points. Our VM selection method simply finds the matrix entries $C(k,l)$ with the lowest correlation coefficients and multiplexes the two VMs k and l. Once the two VMs are chosen, the k^{th} and l^{th} rows and columns of the matrix are invalidated so that the next set of compatible VMs can be identified.

Figure 7 depicts two sets of VMs, selected from the virtualized cluster used for the displayed correlation matrix. Figure 7(a) shows two VMs with a strong negative correlation, while Figure 7(b) plots two VMs with a very high positive correlation entry. The two figures distinctly show the contrasting characteristics of the two sets of VMs. While the first set of VMs complement each other very well with their peaks temporally distributed in a non-overlapping manner, the latter two VMs—with high positive correlation—have highly overlapping demand characteristics with strongly aligned peaks. Therefore, the joint capacity requirement of the latter set of VMs is close to the sum of their individual capacity requirements, while significant capacity savings are realizable by joining the first set of VMs. These show that the correlation-based approach provides a reasonably good method for VM selection.

We also evaluate the quality of our method against a random selection approach, where VMs are randomly picked for joint sizing. For this evaluation, Figure 8 demonstrates the achievable capacity savings - over individual VM sizing - with both selection methods for 441 VMs. We assume oracle knowledge of future VM demand, thus evaluating the selection method is independent from the underlying forecasting technique. The capacity requirement for each VM is considered as the peak demand. The joint capacity is defined in a similarly conservative manner, i.e., it equals the peak demand of all multiplexed VMs. Figure 8 depicts how the total required capacity decreases as we multiplex VMs. Both random and correlation-based selection schemes achieve significant capacity reduction, which demonstrates the strong potential of our joint sizing approach. However, the correlation-based approach performs distinctly better

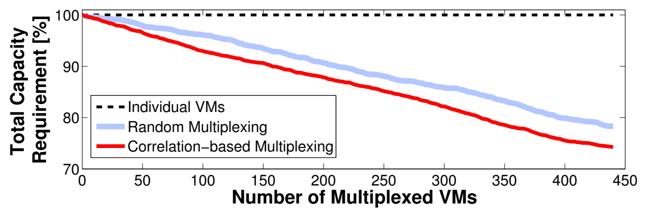


Figure 8: Capacity improvement with VM multiplexing and correlation-based VM selection.

than the random selection, thus corroborating the benefit of our VM selection technique.

While not the scope of our work, there are several avenues of improvement for our VM selection technique. First, as the data center sizes grow, the scalability of the correlation matrix becomes an issue. Here the simplest workaround is to consider a hierarchy of VM clusters that are evaluated disjointly. Second, so far we only identify pairs of compatible VMs. In principle, we can extend this to more than two VMs by using an approach similar to factor analysis, in which VMs with similar correlation characteristics are clustered into sets of arbitrary sizes.

8. EVALUATION OF VM MULTIPLEXING

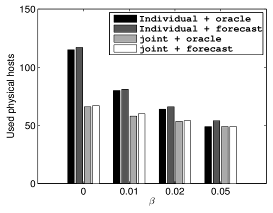
As discussed in the previous sections, the proposed SLA model, joint-VM sizing and VM selection constitute the three building blocks for VM multiplexing. These building blocks can be directly plugged into existing resource provisioning applications. In this section, we demonstrate and evaluate two use cases for VM multiplexing with two common cloud management applications, and demonstrate its substantial benefits with dramatic improvements in resource use efficiency in each use case.

8.1 VM consolidation

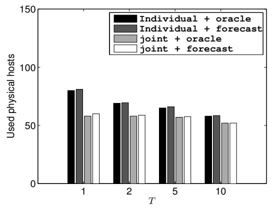
VM consolidation is performed when a VM controller needs to create and deploy a set of VMs on a set of physical hosts. The goal of VM consolidation is to determine a mapping of VMs to physical hosts such that the minimum number of hosts are used. Existing VM consolidation schemes consist of two steps: estimating the future size for each VM, and placing VMs on physical hosts. The first step, estimating VM size, is usually solved by first forecasting the future workload, then finding a capacity size that can sufficiently cover the forecasted workload. The second step, VM placement, usually requires solving a bin packing type of problem. Specifically, since each VM carries a size and each physical host has fixed capacity, the VM placement problem is equivalent to packing items (VMs) into the smallest number of bins (hosts) without violating the size limit on each bin. In practice, VM placement is tackled by either heuristics or solving an integer programming problem [1].

By exploiting VM multiplexing, it is possible to achieve even more compact consolidation. The only necessary change is to replace the first step in the above procedure with the proposed three building blocks. Briefly speaking, the VM controller first applies the proposed SLA model to describe the performance requirement for each VM. It then run the VM selection algorithm to partition VMs into VM groups. For each VM group, the joint-VM sizing algorithm is employed to determine the capacity being allocated.

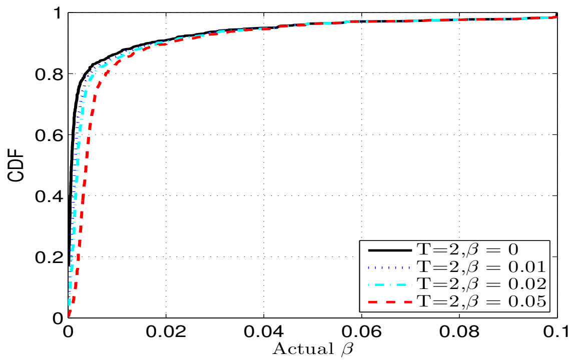
We conduct an experiment to compare capacity savings



(a) $T = 1$, β varies



(b) $\beta = 0.01$, T varies



(c) CDF for actual β with forecasting-based joint sizing

Figure 9: VM consolidation by different schemes

between using and without using VM multiplexing. From the aforementioned dataset, we extract the server information for one regional data center, which contains 266 physical hosts and 1418 VMs. The purpose of this experiment is to try various VM sizing schemes by consolidating these 1418 VMs into the 266 hosts. The VM placement method we used is FFD (First-Fit Decreasing) [1], a popular approximation algorithm for solving bin packing problem. We consider a homogeneous scenario in which all VMs have the same SLA constraint parameters T and β. Therefore, these two parameters are also taken in the joint SLA model. We compare four VM sizing schemes, coming from two choices that whether individual or joint sizing is used, and whether an oracle or an actual forecasting is used. The oracle-based joint sizing schemes represent the potential capacity savings with perfect VM demand knowledge. The measurement period for our data lasts for 3 months. We always use the first half for training forecasting model, and the second half for conducting forecasting, sizing and packing. We vary T and β and compute the number of physical hosts needed to accommodate the 1418 VMs. Figure 9(a) and 9(b) shows the number of required hosts for several settings of T and β.

The two figures show that the extra capacity savings by joint-VM provisioning varies at different combinations of T and β. The largest capacity saving occurs when $T = 1$ and $\beta = 0$, which corresponds to the peak-load based sizing strategy mentioned in Section 3. In this case, while individual sizing with forecasting uses 117 hosts, joint sizing with forecasting only uses 67 hosts. This is a 45.3% extra saving. A clear trend is that when β increases, the gain margin of joint sizing shrinks. An intuitive explanation is that when β is larger, more capacity violation is allowed, and the computed VM size tends to be more affected by the long-term workload mean instead of spikes. Therefore, the benefit of multiplexing decreases.

Because joint sizing relies on forecasting algorithms to infer the future workload, the forecasted capacity also incurs some errors. Thus, when the estimated joint size is enforced in practice, the original SLA might be violated. To examine this aspect, we measure the *actual* β defined as the percentage of intervals with capacity violations when the estimated joint size is enforced. Essentially, the actual β is calculated from the left side of Inequality (2). If the actual β is significantly greater than the required β, the joint sizing becomes less usable. We calculate the actual β for all the 709 VM pairs and plot the CDF in Figure 9(c) for $T = 2$ and various β. When β varies from 0.01 to 0.05, the figure shows that 88%-96% of VM pairs have their actual β less than the given β. Across all the VMs, more than 90% of the VM pairs have the actual β within 0.02 of their performance measure. The figure also shows that as β increases, the actual β tends to be higher. This is explained by the fact that for smaller β, the joint sizing is more conservative and the estimated size is higher, thus the negative impact of potential forecasting error is better covered.

8.2 Providing resource guarantees for VMs

Current cloud management and virtualization tools provide mechanisms that expose explicit controls on the distribution of resources to VMs. These control methods include providing resource guarantees for VMs in the form of *reservations* or *mins*, enforcing resource limits with *limits* or *maximums* and manipulating dynamic resource scheduling priorities with *shares* [2, 30]. With reservations, the service providers or end users can explicitly specify the resources that are reserved for the deployed VMs. These reservations guarantee that the VM is entitled to the specified resources, and will receive *at least* the specified amount as long as it demands it. Resource guarantees are commonly employed to impose service levels, similar to our performance-based capacity provisioning method. Moreover the described control mechanisms can also be dynamically modulated and applied to runtime, autonomic management [4].

The critical downside of employing VM reservations to provide resource guarantees is that they create a hard limit on the number of VMs that can be admitted to and powered on in a particular cluster. Each time a VM is to be powered on in the cluster, the hypervisor and the management endpoint check whether there are enough unreserved resources to satisfy the VM's reservations. The VM is admitted to the cluster for power on, only if it passes this condition. This process is known as *admission control* [30]. In addition to VM power on, VMs pass through admission control when resource configurations are changed, and during dynamic placement actions that include migration of VMs across resources such as hosts, resource pools or clusters.

Individual VM-level reservations share the same inherent inefficiency as individual-VM-based capacity planning, as the reservations present a strictly additive constraint over VMs sharing the resources, without considering inter-VM interference. Here, VM multiplexing again provides a highly useful model for considering multi-VM characteristics for providing VM resource guarantees, while improving overall cluster utilization. Conceptually, we use VM multiplexing to define *joint reservations* for the sets of "compatible" VMs. While there is no direct resource management mechanism to specify joint reservations, we use a commonly-overlooked abstraction of virtualized clusters, i.e., *resource pools*, to this effect. Resource pools are defined within individual hosts or a cluster to provide a virtual abstraction layer that divides

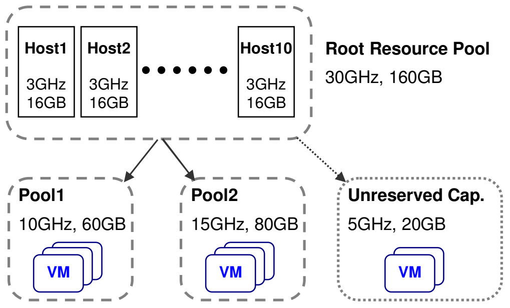
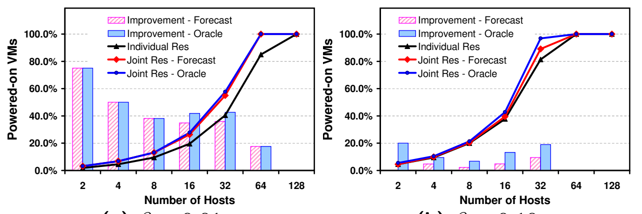


Figure 10: Cluster-level resource pools.

resources into multiple subsets. At the cluster level, one can consider the entire resource capacity of the cluster as a single monolithic *root* resource pool. Then, additional child resource pools can divide the cluster capacity into exclusive partitions [22, 29]. Resource control mechanisms such as reservations and limits can also be defined at the pool level. Figure 10 exemplifies a resource pool hierarchy, including the root resource pool for a ten-host cluster. Two child resource pools with different reservations are defined under the root pool. The remaining unreserved capacity is managed as part of the root resource pool. VMs powered on in different child pools are constrained by the child pool resources, while VMs powered on directly under the root share the resources available to the unreserved capacity pool.

We leverage resource-pool-level reservations in place of VM-level reservations to define joint reservations. Using VM multiplexing, we first use the SLA model to derive the corresponding β values for the individual VM reservations. We use $T = 1$ in the SLA constraint for reservations to match the reservation model used in current hypervisors, where reservations are enforced for each instantaneous sample ($T = 1$) rather than over a tunable time window ($T > 1$). We employ the VM selection method to identify compatible VMs and group them within the same resource pools. We then apply joint-VM sizing with the same β values to define the reservation levels for each resource pool. We consider joint-VM sizing both with oracle knowledge of future demand and by using a forecasting technique. In our evaluation, we only consider VM pairs in each resource pool. While the derived joint reservations are generally higher than individual reservations, we make sure that they are substantially smaller than individual host capacities to avoid creating any artificial placement constraints.

We evaluate the benefits of VM multiplexing with joint reservations for a pool of 441 VM traces obtained from the aforementioned global cloud. We consider several clusters with varying number of hosts. We measure the number of VMs that pass admission control—that can be powered on—for each cluster size with both individual VM-level reservations and joint, resource-pool level reservations. We consider varying levels of reservations for VMs corresponding to the β values in the range of $[0.0, 0.2]$. Figure 11 shows the percentage of VMs that can be powered on in different sized clusters for two β values. Here, $\beta = 0.01$ represents stringent reservation levels for each VM, which are highly "peak sensitive". That is, the specified reservations are close to the peak demand levels of VMs. In contrast, $\beta = 0.10$ represents more relaxed reservation levels, where the reservations are closer to the average demand levels. Therefore, reservations corresponding to lower β values exhibit higher potential for improvement that can be exploited by joint



(a) $\beta = 0.01$ **(b)** $\beta = 0.10$

Figure 11: VMs that are admitted for power on with individual and joint reservations.

reservations. This effect is clearly visible between the two plots of Figure 11. For $\beta = 0.01$, the average increase in number of powered-on VMs with joint reservations is 44%. This reduces to 14% with $\beta = 0.10$. The effectiveness of joint reservations diminishes (to 3%) as we further reduce reservations to reach $\beta > 0.20$.

Figure 11 shows the benefits of joint reservations using both oracle knowledge of future demand and using forecasting. The oracle results show the actual potential benefits of joint reservations. The forecasting-based joint reservations show the achieved benefits with a real runtime implementation. In this case, the number of powered-on VMs is generally slightly smaller than the oracle. The bars in Figure 11 show the improvement over individual VM-level reservations for each cluster size with both approaches.

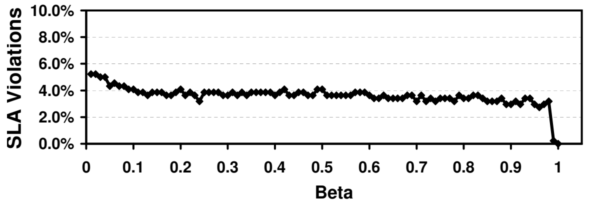


Figure 12: Percentage of VMs that violate their SLA with forecasting-based joint reservations.

With forecasting, an additional consideration is the violation of the SLA specified by the VM-level reservations. As the oracle approach assumes complete knowledge of future VM behavior, the chosen joint reservations completely satisfy all individual reservations. With forecasting, the actual β in future time horizon could be potentially greater than the given β and thus leads to SLA breach. Figure 12 shows the percentage of VMs with the actual β exceeding the given β, for the entire range of β . Overall, more than 95% of the VMs still meet their SLA. based on VM-level reservations.

Overall, VM multiplexing, in conjunction with resource pool level joint reservations, significantly improves cloud utilization efficiency, and the achievable consolidation level on virtualized hosts while respecting the same VM-level SLA requirements. Across the $[0.0, 0.2]$ range of β, the average improvement in the number of VMs that are admitted for power on is 16%, with up to 75% improvements for stringent reservations and small cluster sizes.

9. CONCLUSION

This paper advocates leveraging VM multiplexing to improve resource utilization in compute clouds. The benefit of VM multiplexing is that when the peaks and troughs in multiple VMs are temporally unaligned, these VMs can be consolidated and provisioned together to save capacity. This paper presents three design modules that enable the concept in practice. Specifically, a new SLA model reflects application performance requirements; a joint-VM sizing technique that estimates the aggregate capacity needs for multiplexed VMs; and a VM selection algorithm for identifying most compatible VM combinations. The proposed design modules can be seamlessly plugged into existing resource provisioning applications. VM multiplexing is evaluated with two example applications: VM capacity planning and providing VM resource guarantees via reservations. Experiments based on data from an operational cloud demonstrate that the proposed joint-VM provisioning significantly outperforms traditional approaches. In capacity planning, joint provisioning uses 45% less physical machines for hosting the same number of VMs. With joint-VM reservations, VM multiplexing improves the ratio of admitted VMs by 16% on average, and up to 75% with stringent SLA requirements.

10. REFERENCES

[1] Y. Ajiro and A. Tanaka. Improving packing algorithms for server consolidation. In *Proceedings of the International Conference for the Computer Measurement Group (CMG)*, 2007.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Nineteenth ACM symposium on Operating systems principles (SOSP)*, 2003.

[3] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2007.

[4] M. Cardosa, M. Korupolu, and A. Singh. Shares and Utilities based Power Consolidation in Virtualized Server Environments. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2009.

[5] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam. Profiling, prediction and capping of power consumption in consolidated environments. In *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS)*, 2008.

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification. Second Edition*. Wiley Interscience, New York, 2001.

[7] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Capacity management and demand prediction for next generation data centers. In *IEEE Intl. Conference on Web Services*, pages 43–50, 2007.

[8] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, pages 317–330, 2009.

[9] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: communication-aware CPU scheduling for consolidated xen-based hosting platforms. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pages 126–136, 2007.

[10] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2008.

[11] IBM WebSphere CloudBurst. http://www-01.ibm.com/software/webservers/cloudburst/.

[12] A. Ihler and M. Mandel. Kde toolbox http://www.ics.uci.edu/ ihler/code/kde.html.

[13] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *International Conference on Autonomic Computing (ICAC)*, pages 117–126, 2009.

[14] S. M. Kendall and J. K. Ord. *Time Series*. Oxford University Press, New York, third edition edition, 1990.

[15] D. Kusic and N. Kandasamy. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. In *IEEE International Conference on Automatic Computing (ICAC)*, 2006.

[16] Lanamark Suite. http://www.lanamark.com/.

[17] S. Mohammadi and H. A. Nejad. A matlab code for univariate time series forecasting.

[18] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *IEEE International Conference on Automatic Computing (ICAC)*, 2005.

[19] Novell PlateSpin Recon. http://www.novell.com/products/recon/.

[20] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007.

[21] E. Parzen. On estimaton of a probability density function and mode. *Ann. Math. Stats*, 33:1065–1076, 1962.

[22] J. Rolia, L. Cherkasova, M. Arlit, and A. Andrzejak. A capacity management service for resource pools. In *International Workshop on Software and Performance*, 2005.

[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct 2002.

[24] J. Sonneck and A. Chandra. Virtual putty: Reshaping the physical footprint of virtual machines. In *HotCloud Workshop in conjunction with USENIX Annual Technical Conference*, 2009.

[25] B. Urgaonkar, B. Urgaonkar, P. Shenoy, P. Shenoy, T. Roscoe, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *the 5th symposium on Operating systems design and implementation (OSDI)*, 2002.

[26] A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and Migration Cost Aware Placement of Applications in Virtualized Systems. In *Proceedings of the ACM Middleware Conference*, 2008.

[27] VMware Inc. VMware Capacity Planner, http://www.vmware.com/products/capacity-planner/.

[28] VMware Inc. VMWare vCenter CapacityIQ, http://www.vmware.com/products/vcenter-capacityiq/.

[29] VMware Inc. Resource Management with VMware DRS. Whitepaper, VMware Inc., 2006.

[30] VMware Inc. vSphere Resource Management Guide. Whitepaper, VMware Inc., 2009.

[31] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the denali isolation kernel. In *5th Symposium on Operating systems design and implementation(OSDI)*, 2002.

[32] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource usage of virtualized applications. In *ACM International Conference on Middleware*, 2008.

[33] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2009.