Probabilistic Performance Modeling of Virtualized Resource Allocation

Brian J. Watson, Manish Marwah, Daniel Gmach, Yuan Chen, Martin Arlitt, and Zhikui Wang
HP Laboratories
1501 Page Mill Road
Palo Alto, CA
{firstname.lastname}@hp.com

ABSTRACT

Virtualization technologies enable organizations to dynamically flex their IT resources based on workload fluctuations and changing business needs. However, only through a formal understanding of the relationship between application performance and virtualized resource allocation can over-provisioning or over-loading of physical IT resources be avoided. In this paper, we examine the probabilistic relationships between virtualized CPU allocation, CPU contention, and application response time, to enable autonomic controllers to satisfy service level objectives (SLOs) while more effectively utilizing IT resources. We show that with only minimal knowledge of application and system behaviors, our methodology can model the probability distribution of response time with a mean absolute error of less than 6% when compared with the measured response time distribution. We then demonstrate the usefulness of a probabilistic approach with case studies. We apply basic laws of probability to our model to investigate whether and how CPU allocation and contention affect application response time, correcting for their effects on CPU utilization. We find mean absolute differences of 8-10% between the modeled response time distributions of certain allocation states, and a similar difference when we add CPU contention. This methodology is general, and should also be applicable to non-CPU virtualized resources and other performance modeling problems.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: *Modeling Techniques*; G.3 [**Probability and Statistics**]: *Probabilistic algorithms*; I.6.5 [**Simulation and Modeling**]: Model Development – *Modeling methodologies*;

General Terms

Performance, Theory

Keywords

Performance modeling, quantile regression, probability theory.

1. INTRODUCTION

A recent focus on reducing the economic costs of information technology (IT) motivates increased resource sharing and "on-demand" computing. Towards this, virtualization technologies enable IT resources to be dynamically allocated among multiple applications. Such a model empowers organizations to flex their computing resources based on workloads and business needs, and hence improve the efficiency of IT operations. To achieve this goal, a key task is to leverage virtualization technologies to increase the effective utilization of IT resources. A specific problem within this area is how to minimize the allocation of server resources to an IT service (or application), while satisfying Service Level Objectives (SLOs). This objective requires a clear understanding of the impact on application performance of different virtual machine (VM) resource allocations and contention states, and development of performance models that accurately capture these relationships. This is difficult in practice due to the follow challenges.

First, while many previous research efforts have developed performance models and addressed resource allocation, most of them have focused on physical servers [3][5][8][10]. A virtual machine differs from a physical server in that its effective capacity varies with dynamic resource allocation (e.g., CPU shares), which can significantly change application performance. Further, multiple applications sharing the same physical resources may compete with one another for the resource in complicated ways. As a result, the models derived for physical servers may not be directly applied to virtualized server environments.

Second, application performance can be affected by many factors, including resource allocations, workload variations, and application and system behaviors. Many of these factors fluctuate randomly over various time scales, such as request arrival processes, transaction mixes, CPU scheduling, cache misses, and concurrency locking. Directly measuring and modeling these factors requires intimate knowledge of the applications and the systems, as well as invasive instrumentation. Even when technically possible, it is time consuming and often economically infeasible, due to the variety and complexity of modern IT application architectures like multi-tier and Service Oriented Architectures (SOA). Nevertheless, these *hidden factors* can still affect application performance to varying degrees, and capturing their effects without measuring them is a challenging problem.

Lastly, a key goal of the SLOs is to ensure appropriate performance of the corresponding IT service. Most previous work has looked only at average performance (e.g., mean response time). However, average performance guarantees are not sufficient for many applications, in particular interactive ones. Instead, providers of such services prefer percentile performance guarantees [18], such as that 95% of end users receive response times below an agreed upon threshold.

As a solution, we propose *probabilistic performance modeling*, in which we model the probability distribution of an application performance metric conditioned on one or more variables that we can measure or control, such as system resource utilization and allocation metrics. We consider the influence of hidden factors that we cannot readily measure to be implicit in the probability distribution of performance. This is a valid assumption as long as there is no significant change in the probabilities of those factors. With a probabilistic model, we can calculate the probability of satisfying a percentile performance requirement, given knowledge of the conditional variables during a particular time interval.

Our work provides three main contributions. First, we describe a general methodology for constructing probabilistic performance models that requires little special knowledge of the application. Second, we demonstrate the validity of our methodology through experiments in virtualized environments across a range of resource allocation and contention states. We show that our methodology can model the probability distribution of response time with a mean absolute error of less than 6% when compared with the measured response time distribution, even for the most challenging of our test cases. Third, we apply these models to case studies that investigate the impact of CPU allocation and contention on application performance when we correct for differences in CPU utilization. Our results show that allocation and contention do have statistically significant effects. Furthermore, our case studies illustrate how probabilistic models might be applied to other performance-related problems.

The remainder of the paper is organized as follows. Section 2 defines our problem. Section 3 explains our methodology. Section 4 describes our test bed and applications, and Section 5 provides our experimental results. Section 6 discusses related work. Section 7 concludes the paper with a summary of our work and a list of future directions.

2. PROBLEM STATEMENT

2.1 Background and Assumption

We consider a virtualized environment where applications are hosted by a common pool of virtualized server resources. Each application consists of several interacting components, each of which runs in a virtual machine (VM) hosted on a physical server. As an example, consider a 3-tier application with a web tier, application tier and a database tier. Each tier runs in an individual virtual machine, and they can be placed on the same physical server or distributed across different servers. Autonomic controllers can dynamically adjust this layout using VM live migration. Controllers can also dynamically adjust resource sharing by the virtual machines, including their allocation of CPU, disk, and network I/O. Given a specific application and workload, the performance is affected by the resources allocated to the virtual machines hosting the application, as well as less readily observable factors like system scheduling, cache misses, and interactions between application components. Although we do not explicitly consider these hidden factors, they do have an effect on the probability distribution of a performance metric like response time, and some of them may be partially correlated with the variables that we do consider in our model. It is important to collect enough data to capture the fluctuation patterns of these hidden factors, such as non-stationary transaction mixes [5]. Significant changes in their probability distributions can invalidate a probabilistic performance model, although this can be fixed by periodic online training of new models.

Though multiple system resources can affect performance, adjusting a virtual server's CPU allocation is the most mature resource control option available in today's virtualization technology. In the following discussion, we assume that CPU is the only resource to be dynamically allocated among virtual machines. We further assume that the VM scheduler employs a capped mode, such that a virtual machine cannot use more than the CPU time allocated to it. This assures a straightforward guarantee on resource allocation and provides good performance differentiation between applications sharing physical resources. Another assumption that we make to simplify our experimental results is that all three tiers always have the same CPU allocation, but our approach is general and removing this assumption is a straightforward extension of our work. Finally, we assume that the only specific knowledge we have about an application is how to monitor its response time and how to locate the VMs hosting its components.

2.2 Problem Definition and Formulation

For convenience, Table 1 summarizes the notation that we use in this paper.

Table 1: Notation

N	Number of data points or observations
D	Number of variables x under consideration
M	Number of bins for discretizing each variable x
α, a	CPU allocation, a is discretized index
u_{web}, i	Web tier CPU utilization, i is discretized value
u_{app}, j	Application tier CPU utilization, j is discretized value
u_{db}, k	Database tier CPU utilization, k is discretized value
c	CPU consumption (c=$u\alpha$)
τ, t	Response time metric, t is discretized value
T_t	Upper bound of discretized response time t: $T_{t-1} < \tau \leq T_t$
q	Quantile
r	Index for a set of quantiles $\{q\}$
$T_q()$	Quantile model of response time for quantile q
ψ	Observed percentage of data points $\leq$ some value of T

To avoid confusion in terminology, we define a few concepts used throughout this paper. We use resource allocation α to refer to the percentage of a physical resource capacity (e.g., CPU) that is allocated to a virtual machine. Resource consumption c is the actual percentage of the physical resource consumed by a virtual machine during a given time interval. We define the resource utilization u of a virtual machine as the ratio between its resource consumption and resource allocation: u=c/α. For example, if a virtual machine consumes 20% of CPU capacity and is allocated 40%, then its CPU utilization is 50%.

As case studies for our probabilistic performance modeling methodology, we consider the questions of whether and how virtual machine CPU allocation affects the probability distribution of application response time. Although it appears obvious that the answer to the first question is yes, we must be careful to distinguish between the effects of CPU allocation and utilization on application performance. Queuing theory has firmly established a monotonically increasing relationship between the utilization of an available resource and its response time. In other words, response time is a function of CPU utilization, and utilization is defined relative to allocation. Since autonomic controllers can dynamically modify resource allocation in virtualized environments, and queueing theory suggests that a good heuristic for these controllers to maintain acceptable application performance is to strive for a particular utilization target, the question of interest is how allocation directly affects response time if the probability distribution of utilization is fixed.

We consider this problem from a probabilistic perspective. The CPU consumption of an application component changes from one measurement interval to the next, due in large part to random fluctuations in mix and interarrival times of transactions. If we double CPU allocation between measurement intervals, we cannot expect the CPU utilization in the next interval to be half of what it was in the previous interval. A more accurate expectation is that the probability distribution for CPU utilization will be scaled by a factor of ½ relative to what it was in the previous interval, as we show in Section 5.1. Even this might be an oversimplification if we consider that changing the allocation of one application component while leaving the other components unaffected could result in coupling effects that changes the joint distribution of CPU utilization for all application components in complex ways, but in this paper we only consider the same allocation at every tier. Response time also has a probabilistic relationship with CPU utilization, because of the factors discussed in Sections 1 and 2.1. In this probabilistic performance modeling approach, we consider the joint distribution of response time, CPU utilization, and allocation derived from readily measureable data.

We consider a five dimensional state space for this problem ($D=5$), because it involves five metrics: response time τ, CPU allocation α, and CPU utilization at each of three tiers (u_{web}, u_{app}, u_{db}). We simplify this problem by assuming that all three tiers have the same CPU allocation at any given time, which is why there is only one CPU allocation metric rather than three. Furthermore, we examine four distinct values of allocation: 100%, 70%, 40%, and 25%. Note that these values were chosen to roughly cover the allocation range. Additional allocation values and combinations across tiers will be explored in future work.

We discretize the variables using fixed-width binning. For any given metric x (e.g., web tier CPU utilization), equal width binning divides the variable range into M sub-ranges of the same size, and assigns a discrete value d to a continuous value x.

$$d = \left\lceil M \cdot \frac{x - x_{\min}}{x_{\max} - x_{\min}} \right\rceil \ , \quad x_{\min} < x \le x_{\max} \tag{1}$$

$x_{\min}$ and $x_{\max}$ are set to selected low and high percentiles of the variable x; we indicate these ranges for each analysis in Section 5. We discard data points outside the range ($x_{\min}$,$x_{\max}$], but an alternative approach to handle outliers would be to extend the ranges of the first and last bins to ($-\infty$,$x_{\min}$] and [$x_{\max}$,$+\infty$), respectively.

If we assume the same number of bins M in each dimension, then the total number of bins for a full joint distribution is $M^D=M^5$. Since we are considering only four values of allocation, this is actually $4M^4$, but it is easy to see that this number grows exponentially with an increase in either M or D, so we must carefully select the granularity of our analysis M and the number of metrics D that we wish to consider.

As previously discussed, we are interested in the dependency between CPU allocation and response time. Our approach is to characterize response time as a probability distribution conditioned on allocation: $p[t|a]$, where t and a are the discretized variables for response time and allocation, respectively. As shown in equation (2), it is straightforward to derive an expression for $p[t|a]$ using basic rules of probability and marginalizing, or summing out, the utilizations. The notations i, j, and k are the discretized variables for u_{web}, u_{app}, and u_{db}, respectively. We call the first component of equation 2, $p[t|a,i,j,k]$, the *conditional performance model*, and the second component, $p[i,j,k|a]$, the *joint utilization distribution* for a given allocation.

This expression for $p[t|a]$ can be further summed to compute a cumulative probability distribution, for example, to estimate the probability that response time is less than or equal to a specified threshold, as shown in equation (3). Note that we use lower-case $p[]$ to indicate a probability distribution, upper-case $P[]$ for a cumulative distribution, and d as a summation index for t.

$$\begin{aligned} p[t \mid a] &= \frac{p[t,a]}{p[a]} = \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{M} \frac{p[t,a,i,j,k]}{p[a]} \\ &= \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{M} \frac{p[t \mid a,i,j,k] \cdot p[i,j,k \mid a] \cdot p[a]}{p[a]} \\ &= \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{M} p[t \mid a,i,j,k] \cdot p[i,j,k \mid a] \end{aligned} \tag{2}$$

$$\begin{aligned} P[\tau \le \mathrm{T}_t \mid a] &= \sum_{d=1}^{t} p[d \mid a] \\ &= \sum_{d=1}^{t}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{M} p[d \mid a,i,j,k] \cdot p[i,j,k \mid a] \end{aligned} \tag{3}$$

An intuitive approach to estimating the conditional performance model is to count the corresponding instances in the data. This approach has two main drawbacks: (1) it is not scalable since the required amount of data N grows exponentially with an increase in the number of variables D (the so called curse of dimensionality [23]); (2) even when data is sufficient, it may not be uniformly distributed over the state space of all relevant variables, resulting in pockets where the sparseness of data renders invalid the probability computed there.

Quantile regression [1] is a simple alternative that does not rely on exhaustive counting to determine the conditional performance model, and thus avoids the need to restrict M and D solely because of the number of data points N. Its drawback is that it is a parametric method for which we must specify a functional form that might not exactly match the underlying relationships in the data. However, we find the error to be acceptable for the performance data we modeled (see Section 5.2). Although non-parametric quantile regression techniques exist [1], which partition the input space and perform local fitting, they are prone to overfitting. We discuss the details of quantile regression modeling in Section 3.4.

3. METHODOLOGY

In this section, we present a generalized methodology that can be fully automated for deriving probabilistic performance models. This process comprises five steps, which are depicted in Figure 1.

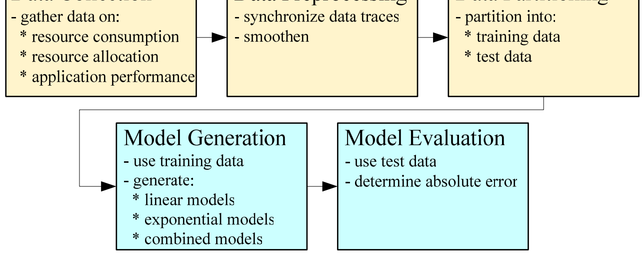

Figure 1: Automatic model generation and evaluation process.

3.1 Data Collection

We employ monitoring agents to periodically collect CPU consumption and allocation values for each virtual machine in which an application component is running. We also deploy a performance monitor to measure the client-side response time of each individual transaction. Our measurement interval is 1 second. We describe our test bed in more detail in Section 4.

3.2 Data Preprocessing

Since application components are executed on different physical servers in most test cases, we keep the clocks on the servers synchronized. However, the sampling are not synchronized which can cause the measurement timestamps on difference servers to be off from each other by as much as half a measurement interval. In this work, we employ a simple moving average technique to increase the effective size of the measurement interval, and to reduce the relative synchronization error. We note that increasing the effective size of the measurement interval might not always be desired and that other techniques like linear interpolation could also synchronize the data. For each trace in our implementation, we calculate a ten second moving average that slides forward in one second increments, reducing the synchronization error to 5% or less.

Our performance metric is the 95th percentile response time of all transactions for each previously determined 10 second interval. We select this metric, since the long tail of performance is more important to end users than average values, and percentiles are not handled well by classical performance modeling techniques. We emphasize that our selection of the 95th percentile should not be confused with the quantiles that we model in Section 3.4. Quantiles indicate the probability that 95th percentile response time will be less than or equal to a given value for a ten second window, not the percentile of the response time metric itself.

3.3 Data Partitioning

We then partition the preprocessed trace into training and test data sets. We use the training data for model generation and reserve the test data to evaluate their accuracy. In our implementation, we randomly sampled 80% of the data points for the training set, and the remaining 20% for the test set.

3.4 Model Generation

Quantile regression [1] is a technique for modeling quantiles. As with conventional regression analysis, it optimizes the coefficients for a specified functional form such that the function models a certain characteristic of the data. In this case, we are modeling the response time threshold T for a given combination of CPU utilization values such that there is a probability q of response time being less than or equal to T, as shown in equation (4). A quantile model is only valid for a particular quantile q, and, for this instantiation, a given allocation a. We use an ensemble of quantile models for different values of q and a to define the conditional performance model, as we discuss later in this section.

$$\mathrm{T}_q(u_{web}, u_{app}, u_{db}) : q \approx P[\tau \le \mathrm{T}_q(\ldots) \mid a, u_{web}, u_{app}, u_{db}] \quad (4)$$

While conventional regression analysis minimizes the sum of squared residuals (or errors) to generate a model of the mean conditioned on a set of variables, an alternative is to minimize the sum of absolute residuals to yield a model of the median (i.e., 0.5 quantile). To obtain models for other quantiles, asymmetric weights are applied to the absolute value of positive and negative residuals. For example, weighting positive residuals three times as much as negative residuals produces a model for the 0.75 quantile (75th percentile). This optimization problem can be solved using linear programming methods like simplex [26]. As with conventional regression, the more data points N, the more accurately the model reflects real-world relationships. To fit our experimental data, we evaluated three parametric functional forms of increasing complexity: linear (5), exponential (6), and a combination of linear and exponential (7). We chose these forms since response time versus utilization tends to show linear behavior at low utilizations and exponential at high utilizations [4]. Bodik et al. [12] use a model similar to (7) for modeling response time and workload. However, it is more time-consuming and challenging to derive coefficients for (7), due to its non-linearity. Note that (6) can be reduced to a linear equation by taking the natural logarithm of the response time data.

$$\mathrm{T}_q(u_{web}, u_{app}, u_{db}) = \beta_0 + \beta_1 \cdot u_{web} + \beta_2 \cdot u_{app} + \beta_3 \cdot u_{db} \quad (5)$$

$$\mathrm{T}_q(u_{web}, u_{app}, u_{db}) = e^{(\beta_0 + \beta_1 \cdot u_{web} + \beta_2 \cdot u_{app} + \beta_3 \cdot u_{db})} \quad (6)$$

$$\mathrm{T}_q(u_{web}, u_{app}, u_{db}) = \beta_0 + \beta_1 \cdot u_{web} + \beta_2 \cdot u_{app} + \beta_3 \cdot u_{db} + e^{(\beta_4 + \beta_5 \cdot u_{web} + \beta_6 \cdot u_{app} + \beta_7 \cdot u_{db})} \quad (7)$$

In the remainder of this section, we describe how we use quantile regression to derive a conditional performance model. First, we generate an ensemble of these functions for the CPU allocations a under consideration and a set of quantiles $\{q\}$ uniformly distributed throughout the range [0,1]. In our implementation, we define 49 values of q at uniform increments of 0.02, as in $\{q{:}0.02, 0.04, \ldots, 0.98\}$. Then for a given bin of response time, utilization, and allocation values (t,i,j,k,a), we identify the response time threshold T_t that demarcates the upper boundary of t, and we find two adjacent values of q such that their corresponding quantile models return values that straddle T_t.

$$\mathrm{T}_{q_{r-1}}(\bar{u}_i, \bar{u}_j, \bar{u}_k) \le \mathrm{T}_t \le \mathrm{T}_{q_r}(\bar{u}_i, \bar{u}_j, \bar{u}_k) \quad (8)$$

The overbar notation indicates that we use the geometric center of each utilization bin (i,j,k) as the inputs to the quantile model, and r is an index for the set of quantiles $\{q\}$. Next we estimate the cumulative conditional probability of T_t using linear interpolation.

$$P[\tau \le \mathrm{T}_t \mid a,i,j,k] \approx q_{r-1} + (q_r - q_{r-1}) \cdot \frac{\mathrm{T}_t - \mathrm{T}_{q_{r-1}}}{\mathrm{T}_{q_r} - \mathrm{T}_{q_{r-1}}} \quad (9)$$

We repeat this for T_{t-1}, and now we can calculate the conditional probability of bin (t,i,j,k,a).

$$p[t \mid a,i,j,k] = P[\tau \le \mathrm{T}_t \mid a,i,j,k] - P[\tau \le \mathrm{T}_{t-1} \mid a,i,j,k] \quad (10)$$

Repeating this procedure for all bins gives us the conditional performance model described in Section 2.2. Note that this methodology is general and does not presume any domain knowledge regarding its variables. Therefore, we can add, remove, and change the metrics to consider different problems. We demonstrate this generality in Section 5, where we not only consider the effects of allocation on response time, but also those of CPU contention when we consolidate our application components onto a single physical server. We do this by substituting allocation a with the CPU contention state.

3.5 Model Evaluation

A straightforward approach for measuring the accuracy of the conditional performance model is to use it to calculate a cumulative probability distribution for response time, per equation (3), and then to compare that against a set of previously unseen data points, such as the test partition described in Section 3.3. The unseen requirement ensures that the model is sufficiently general, and does not overfit its training data set.

We derive the conditional performance model using the procedure described in Section 3.4, and we estimate the joint probability distribution $p[i,j,k|a]$ by counting the number of data points in each bin (i,j,k,a). We similarly determine an observed quantile $\psi_{t|a}$ for each value of T_t and a by counting the percentage of data points with response times less than or equal to T_t. If the model is accurate, then $\psi_{t|a}$ should approximately equal $P[\tau \leq \mathrm{T}_t|a]$, plus or minus some statistical drift. Therefore, we can measure the accuracy of our model in terms of the mean absolute error (MAE) between these values, as defined in (11). We show these results in Section 5.2.

$$MAE = \frac{1}{M}\sum_{t=1}^{M}\left|P[\tau \leq \mathrm{T}_t \mid a] - \psi_{t|a}\right| \qquad (11)$$

We also use the term mean absolute difference (MAD) in Section 5 to measure the difference between two measured or modeled distributions. For this we compare the two values of $P[\tau \leq \mathrm{T}_t|a]$, rather than determining $\psi_{t|a}$ from the data.

4. TEST BED AND APPLICATIONS

We have conducted extensive experiments to collect the necessary data using the virtualized test bed shown in Figure 2. It consists of three physical servers, each with one Intel Pentium D 3.2 GHz dual core CPU and 4GB RAM, and three virtual machines (VMs). All servers run SLES 10 SP2 Linux with a Xen 3.2 kernel [24].

We used a modified 3-tier RUBiS e-commerce application [2] and transaction traces adapted from a real application for our experiments. Our RUBiS implementation consists of a front-end Apache web server (version 2.2), a JBoss application server (version 4.0.2), and a MySQL database server (version 5.0). The RUBiS implementation defines 26 interactions and has 1,000,000 registered users and 60,000 items stored in the database.

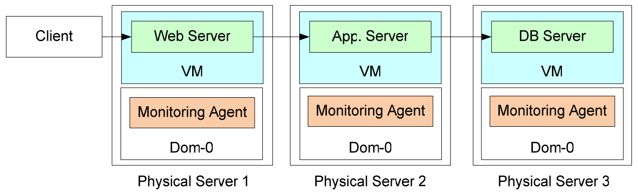

Figure 2: RUBiS test bed implementation.

We deployed the RUBiS instances on three VMs in the virtualized test bed and each application component was run inside one virtual machine. Additionally, a request-based open workload generator was running on a separate machine, and it replayed a pre-defined trace of request rates. To emulate a real application with a highly dynamic workload, we used transaction traces from a globally-distributed business-critical application called *VDR* [5]. We replaced *VDR* transactions in the original traces with RUBiS transactions of the same popularity rank.

We evaluated our models with the following scenarios. First, we employed RUBiS as shown in Figure 2 and varied the allocation of the three RUBiS VMs to use *25%, 40%, 70%*, and *100%* of a physical CPU, which we call the *25%, 40%, 70%,* and *100%* cases, respectively. In one more case, called *25% Consolidated*, we consolidated the three RUBiS VMs onto one physical server, and allocated 25% of the CPU to each VM. To prevent performance interference, we restricted the management domain (dom-0) to use the other of the two CPU cores in the physical machine. For all cases, the Xen credit scheduler was used to allocate CPU shares to the VMs. We played the same transaction trace on the workload generator for the same length of time for each test case to ensure comparable results.

A monitoring agent ran in the management domain of each physical server. It collected the CPU, memory, network, and disk I/O utilization once per second for each hosted VM, although we focus on the CPU data for this paper. Further, the client monitored the response time of each individual transaction. We ran each test case for about ten hours, and collected and pre-processed more than 35,000 data points for each test case. After partitioning, we have about 28,000 data points for training the models, and approximately 7,000 for evaluation.

5. EXPERIMENTAL RESULTS

In Section 5.1, we examine the data that we collected from our test bed. In Section 5.2 we evaluate our models on the test data partition and select the most accurate model. Sections 5.3 and 5.4 apply this model to assess the response time effects of CPU allocation and contention, respectively.

5.1 Analysis of Data

Figure 3 shows the actual distributions of (a) web, (b) application, and (c) database tier CPU utilizations and (d) 95th percentile response time for the pre-processed data from our five test cases. The markers for the *25% Consolidated* case are for the purpose of distinguishing the lines when viewed in grayscale, and are not intended to indicate the spacing of data points used to determine the curves. The same applies to all figures in the section.

We observe that the web utilizations for all test cases are low with little dispersion, the largest being a mean of 7.8% and a standard deviation of 1.8% for the *25% Consolidated* test case. Queueing theory indicates that with consistently low utilization, the web tier CPU is never a bottleneck, and that its utilization has a much smaller effect on response time than either the application or database tier CPU utilizations. Therefore, although we consider the full distribution of web utilization in our modeling, as indicated in equation (2), we simplify the presentation of our results with a focus on application and database utilizations.

Our next observation is that the distributions of all three CPU utilizations for the four non-consolidated test cases (i.e., *100%*, *70%*, *40%*, and *25%*) look like linearly scaled versions of each other. This is intuitive if we recall that CPU utilization equals CPU consumption divided by allocation, and we assume that the distribution of CPU consumption is independent of allocation, since it is mostly a function of workload and we used the same workload for all five test cases. We test this hypothesis by comparing the three-dimensional joint distributions of CPU consumption for each pairing of these four test cases. We discretize the distributions into 50^3=125,000 bins in the range of [0,25]% CPU consumption for all three dimensions, which includes more than 99% of the CPU consumption values for all test cases. Note that Figure 3(a)-(c) shows CPU utilizations, which is why it has values in excess of 25%. We then calculate the mean absolute difference (MAD) between the cumulative probabilities for each bin, similar to the procedure we describe in Section 3.5. We find the mean absolute difference for these consumption distributions to be no more than 0.0055, so we conclude that the CPU consumption is independent of allocation. On the other hand, in Figures 3(a)-(c) we can see that the utilization distribution for the *25% Consolidated* case is substantially different from that for the identically allocated *25%* case. In particular, we notice that the distribution of application utilizations is trimodal for the *25% Consolidated* case, and bimodal for the other four cases. The same is true for the database tier, albeit with less sharply defined peaks. Applying the same technique to measure the MAD between their joint CPU consumption distributions (utilization multiplied by an allocation of 0.25), we find a mean absolute difference of 0.0951. Therefore, CPU contention due to resource sharing has a noticeable effect on the resource consumption patterns of this application.

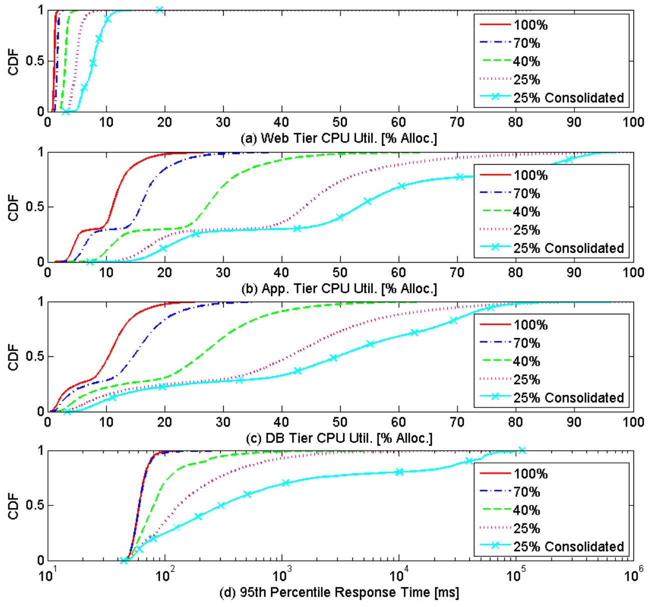

Figure 3: CDFs of metrics for all test cases.

The cumulative response time distributions in Figure 3(d) can be interpreted as the probability of satisfying a given 95th percentile response time requirement during a 10 second interval. Although the *100%* and *70%* test cases do appear to be very similar, the response time distributions for the other allocation and consolidation states vary greatly from one another. In Sections 5.3 and 5.4, we use our probabilistic performance modeling approach to examine how much of this variation is due to the different CPU allocation and contention states versus the different utilization distributions for these five test cases. As we showed in this section, CPU contention in particular can have a significant effect on the distribution of CPU utilization.

5.2 Evaluation and Selection of Models

In this section, we compare the three model types discussed in Section 3.4, and select the most accurate one for more detailed analysis and application to the problems in Section 5.3 and 5.4. First, we derive conditional performance models using the 80% training data partition and the linear, exponential, and combined linear-exponential quantile regression functional forms defined in equations (5), (6), and (7), respectively. We discretize utilization and response time data within the ranges shown in Table 2, because they encapsulate most of the data for each test case. Due to the large range of values for response time, we use equal logarithmic width bins in that dimension, in which the width of each bin is measured as an exponent of 10. The number of bins in each dimension M is equal to 50, for a total of $50^4 = 6.2$ million bins; we choose this number for a good balance between the granularity and computational run time of our analysis. We then use equation (3) to combine the conditional performance model of each type with the discretized joint utilization distribution from the 20% testing data partition for each test case to determine the modeled cumulative response time distributions. Next we assess the accuracy of each model by comparing its estimated distribution against the previously unseen distribution of response time in the 20% testing data partition. Table 2 shows comparisons between each test case and model type, quantified by mean absolute error (MAE) per equation (11).

Table 2. Mean absolute error between modeled and measured distributions for the 20% test data partitions

Test Case	Util. Range [%]	Resp. Time Range [ms]	Model Type	MAE
100%	[0,25]	$[10^1,10^3]$	Lin.	0.0025
			Exp.	0.0023
			Lin-Exp	0.0240
70%	[0,30]	$[10^1,10^3]$	Lin.	0.0030
			Exp.	0.0026
			Lin-Exp	0.0038
40%	[0,50]	$[10^1,10^3]$	Lin.	0.0124
			Exp.	0.0076
			Lin-Exp	0.0237
25%	[0,100]	$[10^1,10^4]$	Lin.	0.0397
			Exp.	0.0232
25% Cons.	[0,100]	$[10^1,10^5]$	Lin.	0.1112
			Exp.	0.0544

Our first observation is that model error monotonically increases as CPU allocation decreases, and the model error is greatest for the *25% Consolidated* case. Further, the CPU utilization distributions in Figure 3 increase according to the same pattern. There may be a correlation between CPU utilization and the accuracy of performance models, as evidenced by the focus on low CPU utilization scenarios (<50% mean utilization) in the performance modeling literature [5][9]. Along similar lines, we observe that the difference in error between the linear and exponential models is negligible for the *100%* and *70%* test cases, but increases dramatically after that. For the *25% Consolidated* case, the linear model has twice the error of the exponential model. The exponential model has consistently lower MAEs for all five test cases, so we select that model type for our remaining analysis. The success of the exponential model is not surprising when we consider that queueing theory also derives non-linear relationships between response time and resource utilization, particularly for high utilization values.

Our second observation is that the combined linear-exponential model performs poorly on the *100%* and *40%* test cases. It performs better for the *70%* case, but still has the greatest error of the three model types. In theory, this model type should be able to better fit the underlying data, due to its larger number of coefficients, but it can be challenging in practice to identify coefficient values for a non-linear equation. We did not generate this model type for the *25%* and *25% Consolidated* test cases after seeing the poor results for the first three test cases. We also spent some time training neural networks as quantile models, with similarly disappointing results. Although we see low errors in general for the simpler models, there is room for improvement for the *25%* and *25% Consolidated* test cases, which further investigation of more sophisticated model types might address.

Now that we have validated the models and selected the exponential type as the most accurate, we use all of the data for our remaining analyses. Table 3 establishes the MAE values for the exponential model when evaluated against all data, so that we may compare these errors against results in later sections that are also based on all of the data. For this evaluation, we use the same ranges as shown in Table 2. We see that the errors in Table 3 are mostly smaller than those for Table 2, which makes sense considering that the data used to generate Table 3 includes the 80% training set against which these models were optimized. The fact that the errors in Table 2 are not much bigger suggests that the exponential model was not overfit to the training data, and that it works well with unseen data, as long as variations in workload and other unobserved factors continue to follow probability distributions similar to what they were when the training data was collected (see discussion in Section 2.1).

Table 3. Mean absolute error between modeled and measured distributions for exponential model type and all data

Test Case	MAE
100%	0.0021
70%	0.0023
40%	0.0066
25%	0.0234
25% Consolidated	0.0540

Figure 4 depicts the modeled and measured cumulative response time distributions for selected test cases. As the low error for the *100%* case suggests, its modeled and actual distributions are almost indistinguishable. The 0.0234 error for the *25%* case is noticeable but small. Only the 0.0547 error for the *25% Consolidated* case is visually significant, and it appears that the limited number of parameters for the exponential form constrains this model from fully capturing the more complicated shape of the actual response time distribution, but it is reasonably accurate.

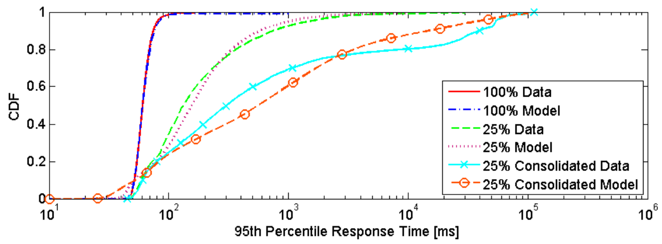

Figure 4: CDFs of response time for exponential models vs. actual data for selected test cases.

The coefficients for the exponential quantile models appear in Figure 5. The form of this function is as defined in equation (6). The value of β_0 in (a) is plotted against the 49 modeled quantiles $\{q{:}0.02,0.04,\ldots,0.98\}$ for each test case. Similarly, 5(b) shows the values of β_1, (c) displays β_2, and (d) depicts β_3. We see that the coefficients for the *100%* and *70%* quantile models are similar to each other and change only slightly with quantile. This matches what we see in Figure 3(d), in which their response time distributions are similar and have a narrow range of values. The *40%* models are similar to those for the first two test cases, except at quantiles above 0.85, in which the significance of web tier CPU utilization becomes substantially more pronounced. This suggests that web utilization at 40% allocation has a strong influence on the tail of the response time distribution. The *25%* models indicate that application tier CPU utilization has a negative relationship with 95th percentile response time. Although this may seem counter-intuitive, it is likely due to this metric being correlated with some other metric that also influences response time for 25% allocation. We also see that response time for the *25%* case has a stronger positive relationship with web and database tier CPU utilizations than it does for the first three test cases, particularly when we consider that the utilizations themselves are higher for 25% allocation. As we have already seen, the *25% Consolidated* case is the most unusual of the five. The strength and direction of its relationships between response time and utilization vary markedly between quantiles.

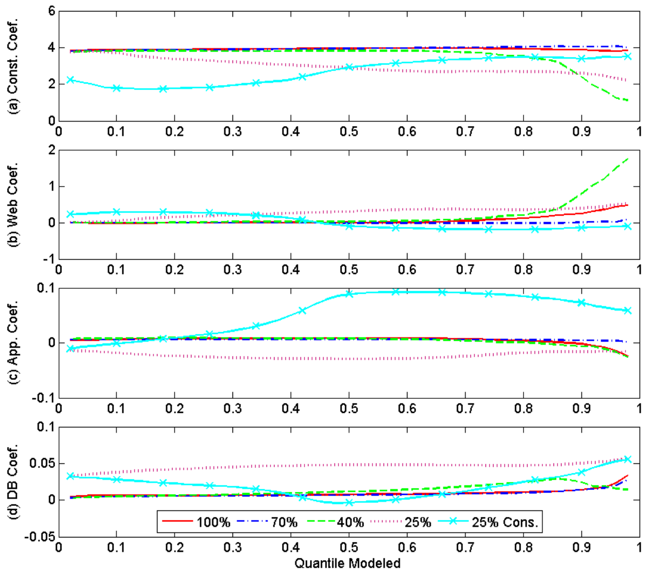

Figure 5: Exponential model coefficients for all test cases.

In summary, we selected the exponential model type, which estimates response time distributions with good accuracy for various scenarios. Modeling high utilization scenarios is challenging, and we show that our model achieves less than 6% error (MAE*100%) for even the most challenging test case.

5.3 (How) Does CPU Allocation Impact Application Response Time?

Now that we have assessed our models, we are ready to investigate the questions of whether and how virtualized CPU allocation affects the distribution of response time. We correct for the effects of CPU utilization by selecting a common joint utilization distribution for the four allocations under consideration, and we use equation (3) to estimate cumulative response time distributions for each allocation. We then calculate mean absolute difference (MAD) statistics for each pair of distributions, and compare them against the model errors in Table 3 to determine that the differences in response time due to allocation are statistically significant.

To achieve valid modeling results, the common joint utilization distribution should be representative of the distributions used to train the model for the *100%*, *70%*, *40%*, and *25%* test cases, which includes being within the range of utilization values for each test case. This is because the model is only intended for interpolation, not extrapolation. If we simply select the observed utilization distribution for the *100%* test case to use as our common distribution, then the utilization values for most of its data points would be significantly less than those for the *25%* test case. For example, the median (quantile 0.5) application utilization for the *100%* case is 11.1%, whereas the quantile of the same value for the *25%* allocation case is only 0.0032. Rather than directly using the utilization distribution for the *100%* case, we linearly scale and translate it such that the resulting 0.005 and 0.995 quantiles are within the range of the same quantiles for each of the observed utilization distributions. However, this criterion is not possible for web utilization, due to its narrow variance and low overlap between test cases, so we fix web utilization at an intermediate value of 3.25% for our common distribution. Table 4 shows the range of values for our common utilization distribution (*Comm.*), and their corresponding quantiles for the actual utilization distributions of each test case.

Table 4. Common utilization ranges and corresponding quantiles for actual utilization distributions

Test Case	Web Util	App Util 0.005	App Util 0.995	DB Util 0.005	DB Util 0.995
Comm.	*3.25%*	*11.48%*	*25.82%*	*3.28%*	*24.11%*
100%	0.9996	0.5670	0.9950	0.1913	0.9950
70%	0.9991	0.2973	0.9599	0.1346	0.9344
40%	0.7208	0.1784	0.4080	0.0405	0.4278
25%	0.0004	0.0051	0.2852	0.0046	0.2657

Figure 6 shows the modeled cumulative response time distributions for each allocation, and Table 5 lists the mean absolute difference (MAD) between each pair of response time distributions, sorted in order of increasing MAD.

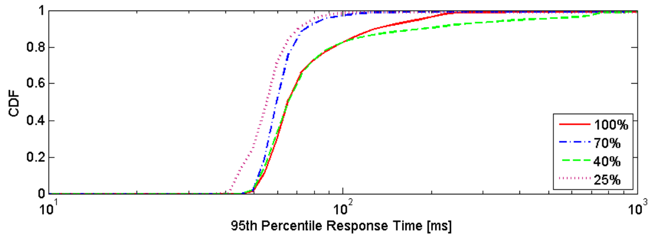

Figure 6: Modeled response time CDFs for different allocations with common joint utilization distribution.

For this analysis, the utilization range is [0,25]%, the response time range is [10^1,10^3] ms, and the number of bins in each dimension *M* is 50. We observe that 25% allocation results in the lowest response time distribution, whereas 40% allocation has the highest. The MAD for the *25%* and *40%* distributions is 0.1064, which greatly exceeds either of their MAEs in Table 3 (0.0066 for 40% allocation, and 0.0234 for 25% allocation), suggesting that the differences between their response distributions in Figure 6 are mostly real and not modeling artifacts. We similarly observe that the MADs for every other pair of distributions also exceed their respective model errors in Table 3.

Another observation is that 40% allocation has a much longer tail above the 0.8 quantile. This is likely due to the increase in the web utilization coefficient above quantile 0.85 for the *40%* case, as shown in Figure 5, combined with our relatively high choice of web utilization for the common joint utilization distribution (quantile 0.7208 of the *40%* data). A third observation is that the response time distribution for *25%* starts at 40 ms, whereas the other three distributions begin close to 50 ms. Since the bottom of the distribution likely represents the shortest service time of the application transaction types, and the other three distributions are in agreement, it is possible that the low starting value for the *25%* case is a modeling artifact. A longer test run with more data points or a more rigorous statistical analysis could confirm that.

Table 5. Comparison of modeled cumulative response time distributions for different allocation pairings

Allocation 1	Allocation 2	MAD
100%	40%	0.0290
70%	25%	0.0290
100%	70%	0.0580
70%	40%	0.0823
100%	25%	0.0837
40%	25%	0.1064

The pattern of influence between allocation and response time is not obvious from these results. A fine-grained survey of more allocation levels could reveal additional patterns and insight, as would a study of different allocations at the various application tiers. We also caution that these results depend on our choice of application, workload, common joint utilization distribution, virtualization layer, scheduler, and possibly the underlying hardware architecture. A more comprehensive investigation of these factors is needed before drawing general conclusions for how allocation affects response time.

In summary, we demonstrated an approach for assessing the impact of virtualized CPU allocation on response time, and we observed differences as high as 8-10% (*MAD**100%) between several pairs of allocations. This is much higher than the model errors, which shows that CPU allocation is a significant factor in virtualized application performance.

5.4 (How) Does CPU Contention Impact Application Response Time?

To demonstrate the generality of our methodology, we now investigate the questions of whether and how CPU contention between the three virtual machines comprising our application affects the distribution of response time when we correct for its effects on the joint CPU utilization distribution. We apply the same analysis as in the preceding section to compare the modeled response time distributions for the *25%* and *25% Consolidated* test cases. Selecting a common joint utilization distribution is easier for this comparison, due to the high degree of overlap between the measured utilization distributions for these two test cases. We simply select the joint utilization distribution from the *25%* case, because it overlaps with 86% of the web utilization values for the *25% Consolidated* case, and more than 99% of the values for application and database utilization. We use a utilization range of [0,100]%, a response time range of [10^1,10^5] ms, and the number of bins in each dimension *M* is again 50.

Figure 7 shows the modeled cumulative response time distributions for these two test cases with our selected common joint utilization distribution. We saw in Figure 3(d) that the *25% Consolidated* case has a much longer response time tail than the non-consolidated *25%* case. Correcting for their different utilization distributions, we see that the difference in tail length is less extreme, but still substantial. The MAD between them is 0.0892, which exceeds the sum of the model errors for these test cases in Table 3. This suggests that the difference between these distributions is statistically significant, but the actual degree of the difference depends on how the non-trivial model errors in Table 3 stack up for this analysis. Summing them gives a simple and conservative estimate, but a more rigorous statistical analysis like bootstrapping could better answer this question.

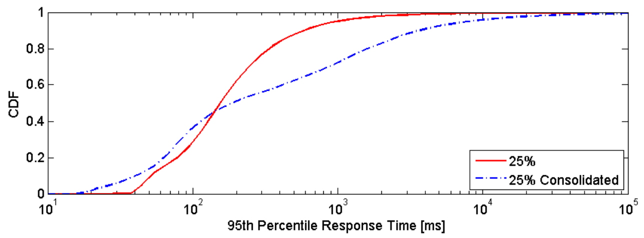

Figure 7: Modeled response time CDFs with and without CPU contention for common joint utilization distribution.

We conclude that CPU contention increases response time, with the corollary that there is incomplete performance isolation between Xen VMs sharing the same CPU, even with a capped scheduler mode. To better understand these effects, we encourage a more comprehensive study using this methodology.

6. RELATED WORK

Modeling has garnered considerable attention within the computer performance community for decades. Thus, the history of modeling is too extensive to cover in detail. Instead, we briefly consider three topics: a persistent challenge, a comparison with traditional queueing performance models, and research leveraging percentiles to improve modeling accuracy and usefulness.

6.1 The Theory and Practice Chasm

An aim of our research is to develop an accurate and intuitive method of modeling the performance of the system. Such features are important for instilling confidence in system administrators and managers (i.e., practitioners) that we can automatically control the managed system. However, convincing practitioners has proven to be a persistent problem. For example, Bhat reported in 1969 that one of the main challenges for the Queueing Theory community was that practitioners found little use for it [15]. Howard noted two reasons for this: "ridiculous assumptions" used by researchers, and a lack of communication between researchers and practitioners [16]. Howard noted one cause of the communication gap is a lack of understanding of the formal concepts by the practitioners. Lee noted that many practitioners do in fact adopt aspects of theoretical work, but choose not to discuss their work with researchers for fear of criticism [17]. Hence, a more intuitive approach could help bridge the gap.

6.2 Queueing Performance Model

In this section we consider a number of recent papers that use queueing theory to model the behavior of modern computer systems. A lot of recent research efforts have developed queueing performance models for interactive and enterprise applications, such as multi-tier Internet applications [3][5][6][8][10]. Urgaonkar et al. proposed one of the most recent and well-known models [3], which uses a closed queueing network model and Mean Value Analysis (MVA) algorithm for multi-tier applications. Though these models work well for certain scenarios, most of them require detailed service demand information that is not readily available from traditional monitoring. Further, the model parameters were normally derived from certain workloads and hence the results are expected to work well only for systems with a similar transaction mix. Stewart et al. employed an open queueing network model to explicitly model a non-stationary transaction mix [5]. Their model parameters can be derived from passive monitoring data through regression analysis, but they require sufficient application knowledge to measure the transaction mix. Chen et al. generalized the idea to develop models for both request-based and session-based workloads [8].

Most queueing theoretic approaches ignore high resource utilization. Urgaonkar et al. take congestion effects into account in their model [3], but how well it performs in general is not clear. In addition, most prior work focused on physical server platforms; only a few considered performance in virtualized environments [9]. Virtualization layers increase the challenges of performance modeling, because different applications may compete with one another for resources in complicated ways across many different servers. Compared to previous solutions, our methodology evaluates and models the tail distributions of performance and works well for different scenarios with low and high CPU utilizations in virtualized environments. Also, our model is calibrated from data that is generally available from regular system resource and application performance monitoring. Though our approach can incorporate multiple resources, we have not included memory, network, and disk I/O in our current evaluation, as Stewart et al. did in [10].

6.3 Other Performance Models

There are increasing efforts to apply statistical learning and probability modeling to performance management. Cohen et al. presented a probabilistic approach for correlating SLO violations with system signatures [13]. Kumar et al. proposed an approach that partitions the system's state-space into homogeneous sub-spaces, creates a micro-model for each subspace, and then uses these micro-models to translate higher-level objectives into component-level objectives [14]. Though these statistics-based black box approaches exist, none is focused on general performance modeling.

6.4 Use of Percentiles

A current criticism of traditional performance modeling is the use of moments (e.g., mean response time), as practitioners often prefer percentiles for modern, popular IT services. For example, Amazon uses the 99.9th percentile to impose stringent requirements on the latency of their platform [18].

In fact, the weakness of using moments in the modeling of computer systems was observed as early as 1967, when Gaver noted that such models oversimplified the variability and randomness of data [25]. In 1976, Price recommended the use of percentiles instead of moments [19]. The following year, Lazowska provided additional evidence of the benefits of percentiles over moments, and pointed out that matching a sufficient number of percentiles would capture the moments as well [20]. However, relatively few research efforts adopted percentiles instead of moments. Two examples that did are [12] and [21]. Other researchers looked for opportunities to instead improve the accuracy of queueing models (e.g., [22]). As our work shows, there are benefits beyond improved accuracy in modeling percentiles rather than moments.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a probabilistic performance modeling methodology for virtualized environments. Our approach models the probability distributions of performance metrics, in terms of percentiles, based on variables that can be readily measured and controlled. This is one of the few performance modeling approaches to address virtualized systems. We showed how to implement this methodology to generate a conditional performance model, as well as how to apply that model to estimate response time distributions for a variety of scenarios. We validated our model using the RUBiS benchmark application in a Xen virtualized environment. The results show that our model is accurate with mean absolute errors of less than 6%, even for our most challenging test cases. We further showed that virtualized CPU allocation and contention are significant factors in determining application response time. We can combine this methodology with a more rigorous statistical analysis to increase confidence in its results, and apply it to a more comprehensive survey of allocation and contention states to better understand the relationships between allocation, contention and response time. This can aid in the performance management of virtualized applications, and can possibly help software designers to improve the performance isolation of virtualized resource schedulers.

Our methodology is general and can be applied to non-CPU resources such as memory, I/O, network, etc. As future work, we will adapt it to different metrics, evaluate the effect of different workloads on model accuracy, incorporate additional inputs for improved model performance, and experiment with different allocation levels and configurations. To demonstrate the practical value of our models, we will integrate them with autonomic control systems. We will also investigate how to fully automate our methodology, and periodically generate new models to ensure their continued accuracy in a production environment, even as workloads, applications, and systems change over time.

8. REFERENCES

[1] R. Koenker, "Quantile Regression", Cambridge University Press, 2005.

[2] RUBiS: Rice University Bidding System. http://www.cs.rice.edu/CS/Systems/DynaServer/rubis

[3] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An Analytical Model for Multi-tier Internet Services and its Applications". In Proc. of ACM SIGMETRICS, June 2005.

[4] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, "Quantitative System Performance: Computer System Analysis Using Queueing Network Models". Prentice-Hall, Inc., 1984.

[5] C. Stewart, T. Kelly, and A. Zhang, "Exploiting Nonstationarity for Performance Prediction". In Proc. of EuroSys 2007.

[6] Q. Zhang, L. Cherkasova, and E. Smirni. "A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications". In Proc. of the 4th Int. Conf. on Autonomic Computing and Communications (ICAC), 2007.

[7] V. Gupta, M. Harchol-Balter, A. Scheller Wolf, and U. Yechiali. "Fundamental Characteristics of Queues with Fluctuating Load". In Proc. of SIGMETRICS 2006, 2006.

[8] Y. Chen, S. Iyer, A. Sahai, and D. Milojicic, "A Systematic and Practical Approach to Generating Policies from Service Level Objectives". In Proc. of the 11th IFIP/IEEE Int. Symposium on Integrated Network Management , June 2009.

[9] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser, "AppRAISE: Application-level Performance Management in Virtualized Server Environment". IEEE Transactions on Networking and Service Management, Vol. 6, No. 4, pp. 240-254, 2009.

[10] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services". In Proc. of USENIX NSDI, 2005.

[11] E. Ipek, S. McKee, B. Supinski, M. Schultz, and R. Caruana, "Efficiently exploring architectural design spaces via predictive modeling". In ASPLOS, 2006.

[12] P. Bodik, C. Sutton, A. Fox, D. Patterson, and M. Jordan, "Response-Time Modeling for Resource Allocation and Energy-Informed SLAs". In Workshop on Statistical Learning Techniques for Solving Systems Problems (MLSys), Whistler, Canada, 2007.

[13] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering and retrieving system history", 20th ACM Symposium on Operating Systems Principles (SOSP), 2005.

[14] V. Kumar, K. Schwann, S. Iyer, Y. Chen, and A. Sahai, "A State Space Approach to SLA based Management". In Proc. of the IEEE/IFIP NOMS, 2008.

[15] U. Bhatt, "Sixty Years of Queueing Theory", Management Science, Vol. 15, No. 6, pp. B280—B294, 1969.

[16] R. Howard, "The Practicality Gap", Management Science, Vol. 14, No. 7, pp. 503—507, 1968.

[17] A. Lee, "Applied Queueing Theory", Macmillan, 1966.

[18] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store", SOSP, Stevenson, WA, 2007.

[19] T. Price, "A Note on the Effect of the Central Processor Service Time Distribution on Processor Utilization in Multiprogrammed Computer Systems", J. ACM, Vol. 23, No. 2, pp. 342—346, 1976.

[20] E. Lazowska, "The Use of Percentiles in Modeling CPU Service Time Distributions", Computer Performance, North Holland Publishing Company, 1977.

[21] U. Lublin and D. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs", Journal of Parallel and Distributed Computing, 2003.

[22] D. Eager, D. Sorin, and M. Vernon, "AMVA Techniques for High Service Time Variability", ACM SIGMETRICS, Santa Clara, CA, 2000.

[23] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," J. Machine Learning Research 3, 1157—1182, 2003.

[24] Xen. http://bits.xensource.com/Xen/docs/user.pdf

[25] D. Gaver, "Probability Models for Multiprogramming Computer Systems", J. ACM, Vol. 14, No. 3, pp. 423—438, 1967.

[26] S. Boyd and L. Vandenberghe, "Convex Optimization", Cambridge University Press, 2004.