Predictive Model for Dynamically Provisioning Resources in Multi-Tier Web Applications

Saurav Nanda
Purdue University
Department of Computer and Information Technology
nandas@purdue.edu

Thomas J Hacker
Purdue University
Department of Computer and Information Technology
tjhacker@purdue.edu

Yung-Hsiang Lu
Purdue University
Department of Electrical and Computer Engineering
yunglu@purdue.edu

***Abstract*—In the competitive world of modern web applications, performance plays a crucial role. An e-commerce company estimated that every $100ms$ delay reduces sales by 1 percent, and a popular search engine reported that every $500ms$ delay in search reduces earnings by 20 percent. The demands from users for these services can vary widely based on factors such as the time-of-day and unexpected events that can trigger flash crowds. To meet these demands web applications can be organized using a multi-tier architecture to make them modular and scalable in a cloud environment. However, a highly dynamic workload and different types of resource requirements in each tier can make it difficult to model the behavior of these applications. This presents two significant challenges to infrastructure providers: 1) to model the behavior of an application workload and provide responsive resources using dynamic resource provisioning; and 2) to maintain performance-based (response time) Service Level Agreements (SLAs). In this paper, we formulate a convex optimization problem for resource allocation, and offer a strict SLA for performance. We adopt an SLA violation cost model to formulate our optimization problem and derive the solution for dynamic resource provisioning. To achieve a strict SLA for the response time of an application, we propose a predictive model that seeks to dynamically provision resources using a Feedback-based Control System (FCS). Our model is applicable for a broad range of multi-tier applications. We demonstrate the effective use of our model through experiments that analyze the behavior of an online auction application using a common workload benchmark.**

***Keywords*—Feedback Control System; Bandpass Filter; ARIMA; Dynamic Resource Provisioning; Multi-tier Applications**

I. Introduction

Application software vendors are increasingly migrating to cloud platforms motivated by its *pay-as-you-go* model, where they do not have to worry about the purchase, installation, and maintenance of a physical server infrastructure. They can scale up and down the resources (CPU, memory, storage, network, etc.) in response to increasing workload, and functional complexity to maintain the performance of an application. Hence, application owners only pay for the amount of resources and services they use. On the other hand, cloud hosting companies seek to maximize profits by serving customer requirements using a minimal amount of resources and maximizing resource utilization to avoid idle resources. Cloud computing offers on-demand scaling or dynamic resource provisioning. This phenomenon of dynamic resource provisioning helps cloud providers offer SLAs related to performance-based metrics such as response time. Kohavi et. al [1] described the experimental results of two popular web applications that demonstrates the direct impact of increasing response time on their revenues. Moreover, Kohavi et. al [1] also reported that a leading e-commerce estimated that every $100ms$ delay reduces sales by 1 percent, and a popular search engine reported that every $500ms$ delay in search reduces earning by 20 percent. However, it is difficult for cloud providers to guarantee a response time SLA as described by Islam et al. [2].

9781-5090-1445-3/16$31.00 © 2016 IEEE (CloudCom'16)

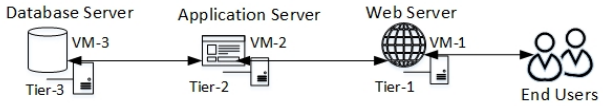

Fig. 1. Three-tier Application Deployment

Modern web applications usually have three major components: a web server, an application server, and a database server. Their performance demands do not scale proportionally, and they are usually connected in a three-tier architecture so that resources can be allocated to each tier independently. Figure 1 shows a three-tier architecture (described by Huang et al. [3]) commonly used for web applications running on *Virtual Machines* (VMs). *Tier-1* represents a web server between the end users and an application server at *Tier-2*. *Tier-3* is a database server. For our study, we assigned one VM for each tier because we are considering only the vertical scaling of VMs, and not the horizontal scaling of VMs in which multiple VMs can be assigned to each tier. Since every tier has different functionality, their corresponding resource requirements (CPU, memory, storage, etc.) may also differ. This presents a major problem for cloud providers seeking to find the optimal resource allocation at each tier and to offer a strict response time SLA to customers. A response time SLA is primarily defined as a threshold value of the response time that is part of an agreement between the customers and the cloud providers regarding the application performance. The challenge is to model the behavior of an application workload so that the resource allocation can be completed in advance based upon a predictive function. The motivation for this paper is to guide application owners in their efforts minimizing resource allocation and its related costs, and to help cloud providers to maximize resource utilization while still maintaining the response time SLAs.

This paper acknowledges the prior work done in dynamic resource provisioning by Islam et al. [2], Zhang et al. [4];

2330-2186/16 $31.00 © 2016 IEEE
DOI 10.1109/CloudCom.2016.56

response time SLA guarantees by Iqbal et al. [5], Padala et al. [6]; and multi-tier web applications using a cloud environment by Calheiros et al. [7], Han et al. [8], and some other relevant works that are discussed in Section 2. The prior work used different approaches to model the behavior of application workloads and to predict the resources needed to guarantee performance. Most of the aforementioned work used CPU and memory resource utilization data as prime inputs to their predictive models because the performance of a web application is mainly affected by these two resources. However, one of the major challenges is to remove noise from the CPU and memory resource utilization data of VMs that is caused by several processes running on the system.

Noise present in the data collected from CPU and memory utilization of each VM is defined as the *unwanted fluctuations in data that are caused by a large number of background processes running on the system, and the frequent context switching between processes*. If such a noisy input is given to a time series predictive model, it affects the prediction accuracy specifically in a dynamic environment. The presence of higly fluctuating positive and negative signal components (*noise*) requires the use of a filtering technique that will only allow signal components within a particular frequency band. We acknowledge the significance of *Moving Average* (MA) smoothing filters that have an impulse response of finite length, and involves averaging the adjacent values while moving through the data. MA filters are generally used with time series data to handle small fluctuations. Since MA is a kind of convolution, it can be seen as a type of low-pass filter used in signal processing that has a limitation of only filtering out high frequency noise for a small number of data points.

In this paper, we propose a solution for dynamic resource provisioning in a multi-tier web application hosted in the cloud with the target of minimizing the allocated resources while maintaining a response time SLA. We formulate a convex optimization problem to allocate resources (CPU and memory only). We then assess the necessary optimality conditions of the dynamic resource provisioning problem and propose a *Feedback-based Control System* (FCS) to solve a constrained discrete-time optimal control problem. We also address the issue of removing noise from the CPU and memory resource utilization data by using a Bandpass Filter [9]. CPU and memory resource usage data (obtained from different tiers of the application) is time series data. Hence, to apply a bandpass filter we transform this time series data to the frequency domain using a Fourier Transform This approach has not been well explored for use in cloud systems. We apply a bandpass filter to the transformed data to remove the unwanted frequencies. The filtered data is then passed through an Inverse Fourier Transform to bring it back to the time domain. Finally, the time series data is used as input to an Autoregressive Integrated Moving Average (ARIMA) model [10] to predict the resource requirements of different tiers of the application. Based on this prediction, our control system adjusts the CPU resources and memory at each tier so that the application's response time does not violate the SLA.

An evaluation of the implementation of our model shows that we are able to maintain a 95% confidence interval for average response time and a response time SLA of $30ms$ for our test application. We performed a rigorous test of our model using a standard workload called RUBiS [11], which is a prototype of an e-commerce and online auction website (based on eBay.com) that is used to assess application performance under varying workloads. We deployed our customized version of RUBiS in a 3-tiered architecture as shown in Figure 1. By varying the type (static/dynamic) and level (clients per session) of the workload on each tier, we can simulate the activity of real time users on this application. We continuously monitor the resource usage on every tier because each tier may experience resource congestion. The main contributions of this paper include:

- To the best of our knowledge, a bandpass filter has not been used for multi-tier applications in a cloud environment. We are able to reduce the noise in the resource utilization data of the VMs used for deploying the different tiers of the web application.
- We propose, design, and develop a predictive model for our control system using an ARIMA model. Based on the feedback received from our predictive function, our control system performs optimal dynamic resource provisioning in different tiers of the web application.
- We evaluate our approach to assess its ability to maintain the response time of the application within a 95% confidence interval and avoid SLA violations.

The remaining sections of this paper are arranged as follows. We review related work in Section 2, and then we formulate and solve our convex optimization problem in Section 3. Section 4 describes our control system architecture that includes our bandpass filter and predictive function. Section 5 describes the design of our adaptive controller with algorithms. Our experimental results are analyzed in Section 6, and concluding remarks are given in Section 7.

II. Related Work

Various algorithms have been developed that seek to address the problem of allocating resources for web applications that are multi-tier. Huang et al. [3] surveyed tiered scaling of computing resources to meet QoS requirements. They categorized the approaches as: rule-based and model-based, and concluded that the model-based approach was the most promising resource management mechanism to provide QoS guarantees. Iqbal et al. [5] proposed a reactive model for detecting and fixing the resource bottlenecks automatically to offer a fixed response time SLA. Han et al. [8] developed an adaptive scaling technique using a G/G/n queuing model to minimize the pricing of cloud infrastructure for the users. Bi et al. [12] used a mixture of M/M/c and M/M/1 queuing models for dynamically managing resource allocations to VMs running on cloud platforms. Control theory can be used to manage the uncertainty and disturbance of a system, as described by Maggio et al. [13]. However, it is not well explored in the context of multi-tier applications. Padala et al. [6] rely on control theory and discuss CPU utilization in terms of average response time. As mentioned by Zhu et al. [14], a control theory approach for systems provides a rigorous technique for modeling, designing, analyzing, and evaluating system performance. Karma et al. [15] and Liu et al. [16] demonstrated the effectiveness of resource management, but not in context of a cloud computing. Kalyvianaki et al. [17] proposed a self-adaptive controller that used a Kalman filter for the dynamic allocation of CPU resources to virtualized applications in a cloud environment. Kalyvianaki et al. [17] mention fluctuations in CPU utilization data, however, removing noise from the data is not a part of their core objective, and they do not

describe the improvements achieved from their filter. None of the aforementioned papers address the effect of noise in the CPU and memory resource utilization data used as an input to the control system.

When applying control theory to manage dynamic resource provisioning, we can use both queuing theory and regression models. We already discussed some of the prior work based on queuing theory techniques. Now we present a brief description of prior work based on regression models.

TABLE I. COMPARISON OF PRIOR WORK WITH OUR APPROACH BASED ON RESPONSE TIME (RT) SLA, NOISE IN DATA, APPROACH (CONTROL THEORETIC), AND TYPE OF APPLICATION (MULTI-TIER)

(Gaps are designated by '**blank boxes**')

Authors	RT SLA	Noise in Data	Control Theoretic	Multi-tier Apps
Iqbal et al. [5]	$\surd$ $(1sec)$		$\surd$	$\surd$
Bi et al. [12]	$\surd$ $(0.8sec)$		$\surd$	$\surd$
Padala et al. [6]	$\surd$ $(\leq 1sec)$		$\surd$	$\surd$
Wang et al. [18]	$\surd$ $(0.31sec)$		$\surd$	
Zhu et al. [19]			$\surd$	
Zhang et al. [4]			$\surd$	$\surd$
Calheiros et al. [7]	$\surd$ (NA)		$\surd$	
Kalyvianaki et al. [17]	$\surd$ $(\leq 1sec)$		$\surd$	$\surd$
Our Paper	$\surd$ $(30ms)$	$\surd$	$\surd$	$\surd$

Using an Autoregressive Moving Average (ARMA) model, Diao et al. [20] proposed a method to maintain system performance by controlling the maximum number of connections permitted by the server and monitoring CPU utilization. Wang et al. [18] analyzed the relationship between the allocation of CPU resources and the Mean Response Time using vertical scaling. Zhu et al. [19] proposed a theory based on the Autoregressive Moving Average model with exogenous inputs (ARMAX) to predict CPU and memory resource needs and to scale vertically resource configurations for virtual machines that include cores, memory, types, and speed. Roy et al. [21] also proposed an ARMA model for workload prediction to minimize cost. Although the above papers [19] [21] are close to the goal of our paper, we address the problem of removing noise from the resource utilization data and then apply an ARIMA model to predict the required resources in each tier of the application. Explicit domain knowledge is not required in the ARMA model so there are high chances of modeling errors that can cause performance degradation. Thus, to increase the utility of our control system, we use an ARIMA model for predictions to dynamically manage resources in multi-tier applications.

The ARIMA model is an advanced form of the ARMA model. Tran et al. [22] proposed an ARIMA model based prediction approach for server workloads. The downside of this approach is that it targets a long-time prediction window. We targeted a much smaller prediction window so that our prediction model is faster in terms of processing time and more suitable for multi-tier applications on a cloud platform. Calheiros et al. [7] used an ARIMA based approach for prediction to evaluate the effect of accuracy on resource efficiency and QoS. Zhang et al. [4] proposed a Model Predictive Controller (MPC) to perform an energy-aware resource allocation in cloud platforms. The above papers [4] [7] use an ARIMA model, however our work is focused on the problem of dynamically provisioning resources for use by multi-tier web applications, with the aim of maintaining a response time within SLA limits. We also address the issue of removing noise from resource utilization data, using a bandpass filter, before using it as input to any kind of prediction model. Table I summarizes the comparison of our approach with other related works in terms of: the core objective of maintaining Response Time (RT) SLA, the issue of noise in resource utilization data, the approach taken (Control Theoretic or something else), and the type of hosted applications (Multi-tier Applications or others).

TABLE II. VARIABLE DEFINITIONS

Symbol	Definitions
R_τ^c, R_τ^m	CPU and memory capacity of a VM at τ_{th} tier.
X_τ^t, Y_τ^t	Number of active CPUs and memory at τ_{th} tier.
C_t^τ, M_t^τ	CPU and memory utilization at τ^{th} tier and at time t, for ease we remove the superscript τ and simply write C_t and M_t.
μ_t^X, μ_t^Y	Amount of change in the CPUs and memory resources at time t.
BT_t^{CPU}, BT_t^{Mem}	Bottleneck resource (highest possible of resource utilization) at τ_{th} tier and time t.
W_t	Weight of severity factor of SLA violation at time t.
g^{SLA}	Unit cost for SLA violation.
F_t^{CPU}, F_t^{Mem}	Unit cost for provisioning resources dynamically at time t.
j_1, j_2	Number of times the dynamic resource provisioning is performed for the CPU and memory resources respectively.

III. PROBLEM FORMULATION

This section describes the parameters and assumptions related to our dynamic resource provisioning approach. The derivation steps we present below for our multi-step ARIMA resource usage prediction and optimization follows the derivation developed and described by Zhang et al. [4] used for predicting CPU and memory usage from the Google cluster dataset with the aim of optimizing power consumption. Although we follow Zhang's derivation procedure, our work differs in that we seek to optimize CPU and memory usage with the objective of controlling the average response time of the application rather than optimizing power consumption. We assume that a web application has *P-tiers* and that every tier consists of one VM at time t. Resources can be dynamically provisioned to any of the VMs in any tier on-demand, as the number of users increases. Every VM has different types of resources, and we focus on the dynamic provisioning of CPU and memory only, as previously discussed. Let R_c^τ and R_m^τ denote the CPU and memory capacity of a VM at the τ^{th} tier. X_t^τ denotes the number of active CPUs, Y_t^τ denotes the amount of memory currently allocated (such that $X_t^\tau \leq R_c^\tau$ and $Y_t^\tau \leq R_m^\tau$), and let (μ_t^X, μ_t^Y) represent the corresponding change in these resources. If μ_t^X is positive, it means that a greater number of CPUs will be added to the VM, and if it is negative then it means that some CPUs will be removed from that VM. Similarly, if μ_t^Y is positive, it means that some amount of memory will be added to the VM and if it is negative then it means that some amount of memory will be removed from that VM. Using above statements we form the following equations:

$$X_{t+1}^\tau = X_t^\tau + \mu_t^X \quad and \quad Y_{t+1}^\tau = Y_t^\tau + \mu_t^Y \tag{1}$$

Now, we present our cost model for SLA violation and dynamic resource provisioning using the resource utilization parameters and the response time of the application hosted to formulate our optimization problem.

A. Optimization Problem

The aim is to dynamically provision CPU and memory to maintain a response time SLA. The optimal way of maintaining the response time of the application without violating the SLAs is:

- Resource (CPU and memory) allocation should be as little as possible but it should be sufficient to maintain the response time below its threshold value.
- Resource (CPU and memory) utilization should be always maximized but at the same time it should never go beyond the total resource capacity of the VM.

To maintain the SLA requirements, the average response time RT_t at time t should not exceed RT^{thresh}. The CPU and memory utilization, at τ^{th} tier at time t, are represented with following equations:

$$U_t^{CPU} = \frac{C_t^\tau}{X_t^\tau} \quad and \quad U_t^{Mem} = \frac{M_t^\tau}{Y_t^\tau} \tag{2}$$

where C_t^τ and M_t^τ represents the CPU and memory utilization, and X_t^τ and Y_t^τ represents the number of active CPUs and memory at τ^{th} tier and at time t as defined in Table II.

1) Response Time SLA Violation Cost: In this work, the SLA is represented as a threshold value for the response time RT^{thresh} of the application, that is, the average response time cannot exceed the threshold value RT^{thresh} to avoid the SLA violation. The response time is inversely proportional to the bottleneck resource usage value (BT_t) [4] that is the highest possible value of resource utilization. If the bottleneck value $BT_t \in \phi_t$ of resource (CPU and memory) utilization at the t^{th} time step increases, then the response time of the web application will also decrease. Similarly, if these bottleneck values are lower, then the response time will be higher. We can calculate the bottleneck resource utilization value using the following equations:

$$\begin{aligned} BT_t^{CPU} &= \underset{C_t^\tau}{\operatorname{argmax}} \left\{U_t^{CPU}\right\} \\ BT_t^{Mem} &= \underset{M_t^\tau}{\operatorname{argmax}} \left\{U_t^{Mem}\right\} \end{aligned} \tag{3}$$

So, the average response time at the t^{th} time step as a function $p_b\left(\right)$ of the bottleneck resource usage can be shown as:

$$\begin{aligned} RT_t &= p_b\left(BT_t^b\right), \forall b \in BT \\ p_b\left(BT_t^b\right) &= p_b\left(\lambda BT_t^{CPU} + (1-\lambda) BT_t^{Mem}\right) \\ &\leq \lambda p_b\left(BT_t^{CPU}\right) + (1-\lambda) p_b\left(BT_t^{Mem}\right) \end{aligned} \tag{4}$$

where, $\lambda \in \{0,1\}$ and $p_b\left(BT_t^b\right)$ represents the average response time given the current resource utilization BT_t^b for the bottleneck resources b, which includes both CPU and memory resources. We have obtained this function $p_b\left(\right)$, which is assumed to be a convex function, by fitting a linear equation to the observed data using the technique described by Zhang et al. [4]. To formulate the optimization problem, we adopt a *SLA violation cost model*. Our SLA aims to maintain the average response time to stay below the threshold value. The SLA violation cost G_t^{SLA} at time t is proportional to the severity of the SLA violation if the average response time increases above the threshold value. We formulate the SLA violation cost as:

$$G_t^{SLA}\left(BT_t\right) = W_t g^{SLA}\left(p_b\left(BT_t^b\right) - RT_t^{thresh}\right)^+ \tag{5}$$

where W_t is the weight of severity factor of SLA violation at time t, g^{SLA} is the unit cost for SLA violation and "+" sign in the superscript signifies that we consider non-negative values.

2) Dynamic Resource Provisioning Cost: The resource provisioning module of our control system allocates CPU and memory on demand to maintain the response time SLA. Up-scaling or down-scaling these resources brings a small but considerable lag which we consider as an overhead to the system. We denote F_t to be the unit cost for provisioning resources dynamically at time t. The total resource provisioning cost (F_t^{Total}) can be shown as:

$$F_t^{Total} = j_1 X_t F_t^{CPU} + j_2 Y_t F_t^{Mem} \tag{6}$$

where j_1 and j_2 represents the number of times the dynamic resource provisioning is performed for the CPU and memory resources respectively.

3) Formulating the Optimization Problem: We formulate our optimization problem by minimizing the sum of the SLA penalty cost and dynamic resource provisioning cost shown by the following equation:

$$\begin{aligned} \underset{X_t \in R_c^\tau,\, Y_t \in R_m^\tau}{min} \Bigg[& W_t g^{SLA} \Bigg(p_b \Bigg((\lambda) \underset{C_t^\tau}{\operatorname{argmax}} \left\{ \frac{C_t^\tau}{X_t^\tau} \right\} + \\ & (1-\lambda) \underset{M_t^\tau}{\operatorname{argmax}} \left\{ \frac{M_t^\tau}{Y_t^\tau} \right\} \Bigg) - RT_t^{thresh} \Bigg)^+ + \\ & j_1 X_t F_t^{CPU} + j_2 Y_t F_t^{Mem} \Bigg] \end{aligned} \tag{7}$$

where, $0 \leq X_t \leq R_c^\tau$ and $0 \leq Y_t \leq R_m^\tau$. Furthermore, we define $c_t = (\lambda)\operatorname{argmax}\{C_t^\tau\}$ and $m_t = (1-\lambda)\operatorname{argmax}\{M_t^\tau\}$. We can re-formulate our optimization problem as:

$$\begin{aligned} \underset{X_t \in R_c^\tau,\, Y_t \in R_m^\tau}{min} \Bigg[& W_t g^{SLA} \left(p_b \left(\frac{c_t}{X_t^\tau} + \frac{m_t}{Y_t^\tau} \right) - RT_t^{thresh} \right)^+ \\ & + j_1 X_t F_t^{CPU} + j_2 Y_t F_t^{Mem} \Bigg] \end{aligned} \tag{8}$$

As assumed by Zhang et al. [4], $p_b()$ is a decreasing function of X_t and Y_t. Consequently, the average response time decreases as the number of active CPUs and the amount of memory increases. Hence, the optimal solutions, X_t^* and Y_t^*, of this optimization problem is bounded by the inequalities:

$$X_t^* \leq \frac{c_t}{p^{-1}(RT^{thresh})} \quad and \quad Y_t^* \leq \frac{m_t}{p^{-1}(RT^{thresh})} \tag{9}$$

where, $p^{-1}\left()\right.$ represents inverse of function $p_b\left()\right.$. We can re-write our optimization problem as:

$$\begin{aligned} & min \ \left\{ W_t g^{SLA} \left(p_b \left(\frac{c_t}{X_t^\tau} \right) - RT_t^{thresh} \right) + j_1 X_t F_t^{CPU} \right\} \\ & subject\ to: \ 0 \leq X_t \leq \frac{c_t}{p^{-1}(RT^{thresh})} \end{aligned} \tag{10}$$

$$\begin{aligned} & min \ \left\{ W_t g^{SLA} \left(p_b \left(\frac{m_t}{Y_t^\tau} \right) - RT_t^{thresh} \right) + j_2 Y_t F_t^{Mem} \right\} \\ & subject\ to: \ 0 \leq Y_t \leq \frac{m_t}{p^{-1}(RT^{thresh})} \end{aligned} \tag{11}$$

B. Optimal Solution

To solve the optimization problem, we use Lagrange multipliers with the Karush Kuhn Tucker (KKT) conditions using the procedure as described by Boyd et al. [23]. We transform the optimization problem of Equations (10) and (11) into a set of equations and inequalities. We formulate our Lagrangian function as:

$$L_1(X_t, \lambda_1) = W_t g^{SLA}\left(p\left(\frac{c_t}{X_t}\right) - RT_t^{thresh}\right) + \lambda_1\left(X_t - \frac{c_t}{p^{-1}(RT^{thresh})}\right) + \mu_1(0 - X_t) + j_1 X_t F_t^{CPU} \tag{12}$$

$$L_2(Y_t, \lambda_2) = W_t g^{SLA}\left(p\left(\frac{m_t}{Y_t}\right) - RT_t^{thresh}\right) + \lambda_2\left(Y_t - \frac{m_t}{p^{-1}(RT^{thresh})}\right) + \mu_2(0 - Y_t) + j_2 Y_t F_t^{Mem} \tag{13}$$

Based on the above Lagrangian equations, the KKT conditions are:

$$\frac{\mathrm{d}L_1}{\mathrm{d}X} = -\left(\frac{W_t g^{SLA} c_t}{X_t^2}\right)\frac{\mathrm{d}p_b\left(\frac{c_t}{X_t}\right)}{\mathrm{d}X} + \lambda_1 - \mu_1 + j_1 F_t^{CPU} = 0,$$
$$\mu_1 X_t = 0,$$
$$\lambda_1\left(\frac{c_t}{p^{-1}(RT^{thresh})} - X_t\right) = 0,$$
$$0 \le X_t \le \frac{c_t}{p^{-1}(RT^{thresh})},\ \lambda_1, \mu_1 \ge 0.$$

$$\frac{\mathrm{d}L_2}{\mathrm{d}Y} = -\left(\frac{W_t g^{SLA} m_t}{Y_t^2}\right)\frac{\mathrm{d}p_b\left(\frac{m_t}{Y_t}\right)}{\mathrm{d}Y} + \lambda_2 - \mu_2 + j_2 F_t^{Mem} = 0,$$
$$\mu_2 Y_t = 0,$$
$$\lambda_2\left(\frac{m_t}{p^{-1}(RT^{thresh})} - Y_t\right) = 0,$$
$$0 \le Y_t \le \frac{m_t}{p^{-1}(RT^{thresh})},\ \lambda_2, \mu_2 \ge 0.$$

There are three cases to be considered: 1) $\lambda_1 \ge 0, \lambda_2 \ge 0$, 2) $\mu_1 \ge 0, \mu_2 \ge 0$, and 3) $\lambda_1 = \lambda_2 = 0, \mu_1 = \mu_2 = 0$. Analyzing the first two cases we calculated the boundary conditions:

$$X_t^* = \frac{c_t}{p^{-1}(RT^{thresh})} \ or \ X_t^* = 0$$
$$Y_t^* = \frac{m_t}{p^{-1}(RT^{thresh})} \ or \ Y_t^* = 0 \tag{14}$$

By analyzing the third case and using the first KKT condition, assuming the $p_b()$ function is convex, we can calculate the optimal solutions X_t^* and Y_t^*. From the empirical evaluation, as mentioned above, we can evaluate the function $p_b()$. For example, if $p_b()$ is a linear function: $p_b(A) = l_1\frac{A}{1-A} + l_2$, then we can evaluate the above equations and derive the optimal solutions as:

$$X_t^* = c_t + \sqrt{\frac{g^{SLA} W_t l_1 c_t}{j_1 F_t^{CPU}}}$$
$$Y_t^* = m_t + \sqrt{\frac{g^{SLA} W_t l_1 m_t}{j_2 F_t^{Mem}}} \tag{15}$$

From the above optimal solutions we conclude that the optimal amount of active resources (CPU and memory) depends upon the rate at which CPU and memory resource utilization is changing, shown by c_t. We can also conclude that these optimal solutions X_t^* and Y_t^* also depend upon the lag (F_t) created during the resource provisioning. Based on the analysis of the optimality conditions, and the solution for the optimal number of CPU and memory resources, we propose a Feedback-based Control System (FCS) architecture in the next section.

IV. Control System Architecture

We propose a control system that consists of self-adaptive methods such as a Monitor, Analyze, Plan and Execute (MAPE) [12] closed-loop control system architecture for dynamic resource provisioning in virtualized multi-tier applications. Our control-theoretic approach aims to fulfill the response time SLA requirements of these virtualized web applications by tuning CPU and memory resources (our focus is on these two parameters) continuously. Figure 2 shows the

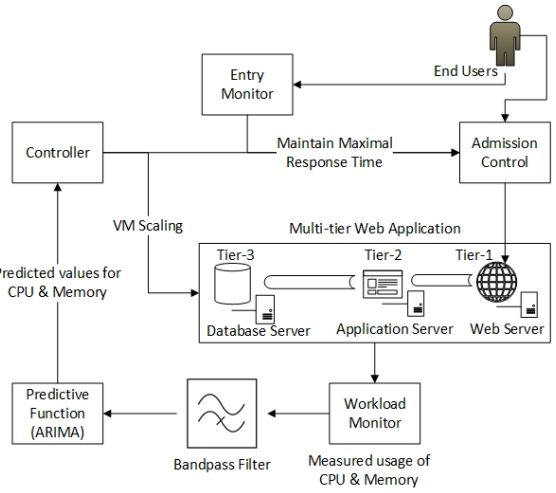

Fig. 2. Control System Architecture

overall depiction of the control system with these components:

- The *Entry Monitor* module collects all the performance metrics such as the average response time and the request arrival rate.
- The *Workload Monitor* captures all the workload related data, i.e. CPU and memory utilization, at different tiers.
- The *Bandpass Filter* helps to improve prediction accuracy as it tends to make noise look periodic or at least quasi-periodic. In the data collected by the Monitoring module we first apply a Fourier transform. Using this transformation, we convert the time series data into the frequency domain and then apply a bandpass filter to remove both high frequencies and low frequencies leaving only frequencies in a band in the middle.
- The *Predictive Function* uses an ARIMA model to forecast the CPU load and memory utilization at different tiers of the web application. Based upon the resource prediction we maintain response time below a threshold value and offer a strict response time SLA.
- Based on the feedback from our prediction function, the

Controller Module performs the vertical scaling of the VMs, by dynamically adjusting the allocation of CPU and memory resources to the VM.

A. Bandpass Filter

This section describes the filtering process that is an oft-used method for accentuating some particular frequencies while removing the remaining frequencies. Low-pass and high-pass filters are commonly used to remove high frequency noise or low frequency signals and leave unaltered the frequencies of interest. A bandpass filter will remove both high and low frequencies and leave only frequencies within a particular band. When we collect CPU and memory resource usage data from the VMs corresponding to each tier of the multi-tier web application, the data contains a lot of noise. There are multiple reasons for noise in the data such as: a large number of processes running in the system, frequent context switch between processes, etc. If this time series data of CPU and memory usage is directly used for resource usage prediction, it cannot be as accurate as needed in such a dynamic environment. We add a bandpass filter in the control system to remove the noise from the raw data measurement before sending it to the predictive function. This approach is widely used in digital signal processing or image processing but it is not explored much in cloud computing. As shown

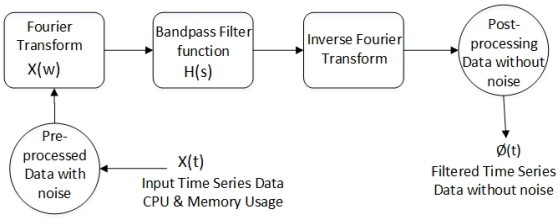

Fig. 3. Steps for Bandpass Filtering

in Figure 3, we first apply a Fourier transform to the time series data of CPU and memory usage to convert it into the frequency domain. We then apply a bandpass filter to remove high frequency noise or low frequency noise trends, and then perform an inverse Fourier transform to convert the data back to the time domain. In this work, we used a Fast Fourier transform (FFT) function which calculates the Discrete Fourier transform (DFT) of time series data, or its inverse as described in Shenoi [9]. The Fourier transform converts a signal from its original time domain to the frequency domain and vice versa. Our solution uses Fourier spectral analysis with the FFT algorithm as it has been the most prominent technique to handle spectral estimation on discrete data set due to its lower time complexity. We acknowledge some modern spectral estimation techniques that can provide better resolution $(1/n)$ with respect to FFT only for small to moderate values of N, and with added amount of computational complexity. Chen [24] described some of these high resolution spectral estimation algorithms: 1) Burg's Algorithm [25] (that is one of the most efficient AR modeling techniques and we compare it with our proposed technique); 2) Least-squares Algorithm; and 3) Marple Least-squares Algorithm. The disadvantage of aforementioned techniques is that they are biased in the location of spectral peaks, line splitting for sine waves at high Signal-to-Noise Ratio (SNR), and frequency shifting for the waves at low SNR.

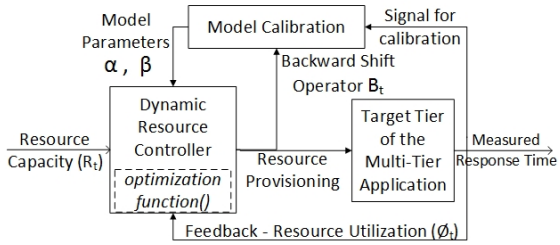

Fig. 4. Adaptive resource controller

B. Predictive Function

This section describes our model for forecasting CPU and memory utilization in a multi-tier virtualized application. Our prediction algorithm uses an ARIMA model to predict C_t^τ and M_t^τ, which represents the CPU and memory use at the τ^{th} tier and at time t. We used the multi-step ARIMA(n,d,q) prediction techniques, as described by Zhang et al. [4], in our model to maintain the accuracy of our prediction. The prediction method has two different variations: the first is known as the Iterating Multi-Step technique (IMS) and the other one is called the Direct Multi-Step technique (DMS) as described by George et al. [10]. We used the IMS approach where the one-step ahead prediction is called recursively. Our objective is to forecast resource (CPU and memory) usage over a prediction time window $Z \in N^+$. In other words, we need to predict resource (ϕ_t) usage $z \in [1, Z]$ steps ahead based on the data collected up to time t. In our experiments, by varying different workloads we found that 400 seconds is a suitable prediction time window. Also, we assume that all the re-configurations of VMs will be performed at the end of these prediction time windows.

V. Adaptive Controller Design

Our adaptive resource controller manages dynamic resource provisioning (CPU & memory) in the three different tiers of a multi-tier web application. Figure 4 shows the working of our feedback-based closed loop control system. The model calibration and dynamic resource controller are the two main modules of our control system. To use this control system we first train our model with a portion of the input data and calibrate the prediction parameters. Later on the model calibration module calibrates the prediction parameters regularly in every prediction cycle. Based upon the feedback (calibrated parameter values) from the model calibration module, the dynamic resource controller module computes the optimization function and can scale (up or down) the resources on each tier of the application accordingly.

Algorithm 1 shows the design of our controller which aims to deal with our constrained (KKT conditions) optimization problem. We start the controller by loading the initial value of the resource allocations for τ_{th} tier at time t. In the previous sub-section, we defined Z as the prediction time window. So, at the end of each time window Z our controller re-calibrates itself. After the calibration is done, the controller predicts the values for the CPU and memory resource changes. Finally, it solves the optimization problem by minimizing the violation of KKT conditions and calculate X_t^* and Y_t^* and then adjusts the number of the resources (CPU and memory) as per the prediction. Algorithm 2 shows the calibration module. It first

Algorithm 1 Adaptive Controller Algorithm

for each tier τ of the hosted application **do**
  Load initial resource allocation X_t and Y_t at time t
  loop
    At the end of controller every time window Z
    Call Calibration()
    Predict the CPU and memory resource change
    Minimize the violation of KKT conditions
    Calculate X_t^* and Y_t^*
    Do the resource provisioning as per prediction
    $Z \leftarrow Z + 1$
  end loop
end for

Algorithm 2 Model Calibration Algorithm

procedure CALIBRATION
  for each tier τ of the hosted application **do**
    Call Bandpass()
    Train the predictive model for Z_c time window
    Set the predictive model parameters
  end for
end procedure

Algorithm 3 Bandpass Filter Algorithm

procedure BANDPASS
  Load X_t and Y_t at end of a time window Z
  for each tier τ of the hosted application **do**
    Fourier Transform of X_t and Y_t.
    Pass the output to Bandpass filter
    Inverse Fourier Transform of filtered output.
  end for
end procedure

removes the noise from the data using the Bandpass filter, then trains the model for a calibration time window Z_c, and finally sets the parameters used in the ARIMA prediction model. Algorithm 3 is the definition of the bandpass filter, where we first take the Fourier Transform of the current time series data (resource utilization) to take it into the frequency domain. The data returned from Fourier transform function is then passed as input to the bandpass filter function to remove the unwanted frequencies (both low and high). Our bandpass filter uses a cutoff frequency of $3dB$, a commonly used threshold value as described by Shenoi [9]. To briefly describe the significance of this value, consider a graph of Frequency (Hz) (abscissa) vs Power (dB) (ordinate). We calculate the boundary frequency points (f_{low} and f_{high}) for our passband where the power drops by 3dB from the maximum power value. Additional details about the choice of a 3dB cutoff frequency is given in Shenoi [9]. Finally, we use an Inverse Fourier Transform function to derive the filtered time series data.

VI. EXPERIMENTAL RESULTS

In this section, we describe our experiments that demonstrates the results of our adaptive control system which we deployed with a customized version of RUBiS [11], an open-source online auction application service, in a three-tier architecture. We also describe the performance and accuracy of our self-adaptive controller, performing dynamic resource provisioning, based on upon our prediction algorithm. Our goal is to maximize the average response time, without violating the SLA, with an optimal number of resources. We are focusing on the vertical scaling of CPU and memory resources.

TABLE III. SERVER (XEON® X5650 2.67GHZ ×19) HARDWARE INFORMATION AND CONFIGURATION OF VMS AT EACH TIER

CPUs: 24	**memory**: 18GB
Storage: 2.9TB	**Network**: 1G Ethernet
Hypervisor: KVM	**OS**: Fedora20 64-bit

	CPU		memory(GB)	
Tier	**Initial**	**Max**	**Initial**	**Max**
Web	2	5	2	3
App	2	6	3	4
DB	1	5	2	4

A. Implementation Details

We used a computer with Kernel-based Virtual Machine (KVM) as the hypervisor and virt-manager (version 1.0.1), which uses libvirt, to manage the VMs. The RUBiS application was deployed in a 3-tier architecture in which the first tier consists of the Apache web server that mainly handles all incoming requests, then forwards all of them to the application server, receives an acknowledgement back from the application server and lastly responds back to the clients. The application (App) server (PHP) on the second tier uses server-side scripts to process the requests received, fetch the required information from the database tier, and finally send output to the Apache web server. The third tier consists of a MySQL database (DB) server carrying all the test records ($1.1GB$), and handles all the data processing queries.

1) Testbed Setup: We developed a small testbed on the same physical machine of a prototype cloud environment, in which all the three tiers of the RUBiS application were hosted in 3 different VMs and the resources on each of them can be easily managed via libvirt. The hardware information of server machine and the initial configuration of each of three VMs is shown in Table III. We used our customized RUBiS emulator which generates workloads for a particular time window using a fixed rate of users entering the website every second. Each session initiated by a simulated user emulates a customer browsing different items from various categories. Our customized script keeps increasing the level of workload randomly while switching between different pages of the web application to emulate a real time web traffic. Different levels of workload consists of a different amount of user sessions per second and every user session consists of some static resource request, some dynamic resource request and some small pause moments, time spent by a user while browsing a page. RUBiS uses PHP pages with read-only MySQL database queries that shows the dynamic resource content of the application. Every new user session is started irrespective of system's capacity to handle them because our workload generator always waits for an acknowledgment before generating the next request. Hence, the response time keeps fluctuating mainly because of the change in the number of user sessions. To conduct our simulations, we made a flat file for storing page transitions called $transitions.txt$.

2) Controller Setup: We developed a feedback-based control system, whose controller was placed on the first tier (web server), which continuously collects resource utilization from all three tiers of the application using $sysstat$, a standard Linux utility, installed on each VM. The resource usage data collected at the end of every prediction time window Z (400 seconds) is passed through the Bandpass filter and the filtered data is sent into the predictive model which performs the

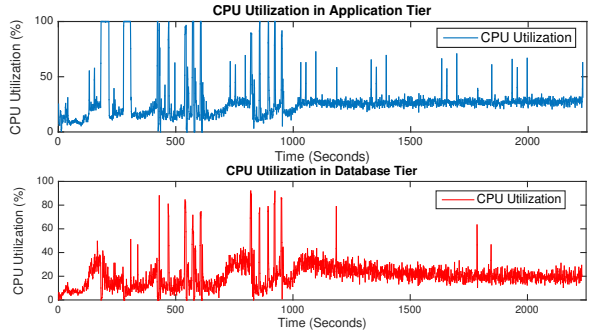

Fig. 5. CPU utilization at the application and database tier that includes measurement noise over a period of randomly selected 2300 seconds

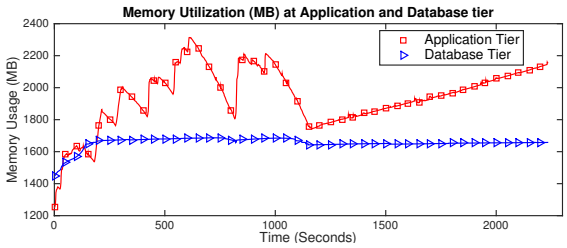

Fig. 6. memory utilization at the application and database tier that includes measurement noise over a period of randomly selected 2300 seconds

dynamic resource provisioning. Our control system keeps re-calibrating the control parameters to address the dynamic traffic changes on the web application with different workloads. Another part of the controller keeps collecting the response time as well and avoids SLA violation.

B. Results and Discussion

We evaluated our system by conducting a set of simulations using a standard workload generator which creates a random load that consists of both static and dynamic contents so that we can stress both CPU and memory resources. We first analyzed the behavior of the RUBiS application in terms of CPU and memory usage at the application and database tier. Figures 5 and 6 show the significant amount of noise present in CPU and memory utilization data collected from two different tiers of the application, which complicates analysis of the underlying trends of these measurements. Hence, it supports

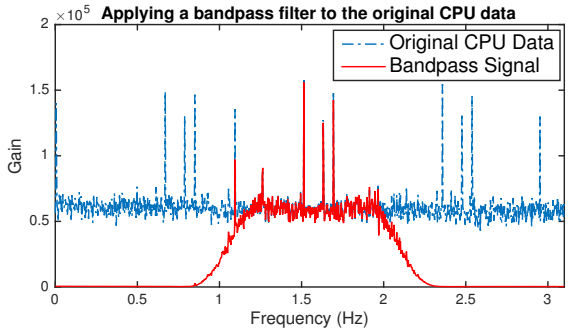

Fig. 7. Bandpass filter applied to CPU utilization data with 3dB as the cut-off frequency

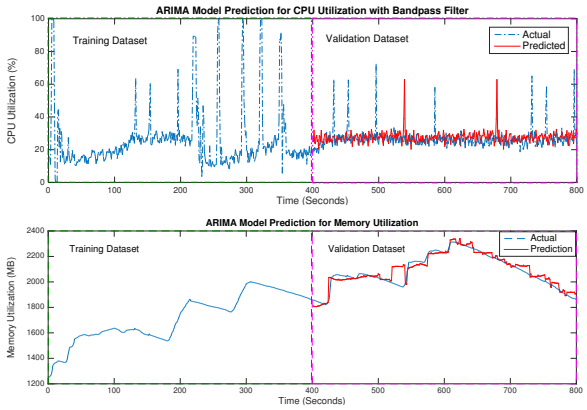

Fig. 8. ARIMA with bandpass filter based prediction of CPU and memory utilization at application tier using 400 seconds of training and validation data

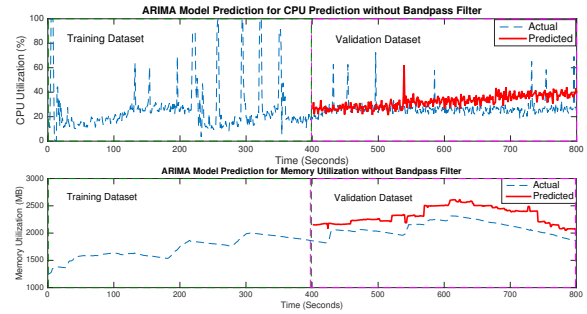

Fig. 9. ARIMA without bandpass filter based prediction of CPU and memory utilization at application tier using 400 seconds of training and validation data

our motivation to use a bandpass filter to remove the high and low frequency noise and leave only frequencies within a band. Figure 7 shows the results of applying a bandpass filter to raw CPU utilization data which has significant noise.

To evaluate the prediction accuracy of our control system we divided the data into two parts: the training dataset and the validation dataset. We set the prediction time window Z to 400 seconds, as explained in Section 4.2.2. We train our prediction model with the data collected in 400 seconds, and then we predict the data for next 400 seconds. Our prediction model requires a minimum of 35 seconds, which was fast enough to re-calibrate itself before prediction. Figure 8 shows the prediction of CPU and memory resources resulting from our analysis, and highlights that the first 400 seconds is used as training dataset, and the second half shows the next 400 seconds for which our system validates the predicted values with original resource usage values. We applied the multi-step ARIMA(2,1,1) model on the filtered data and validated the prediction results by calculating the *Mean Absolute Percentage Error* (MAPE) [7] as shown in the below Equation 16.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{\phi_t - \phi_{t+z|t}}{\phi_t}\right| \tag{16}$$

where ϕ_t is the real value of the resource utilization over a prediction time window $Z \in N^+$ (400 seconds in our experiments), $\phi_{t+z|t}$ represents the *z-step* ($z \in [1, Z]$) prediction of ϕ_t and n represents the total count of input data.

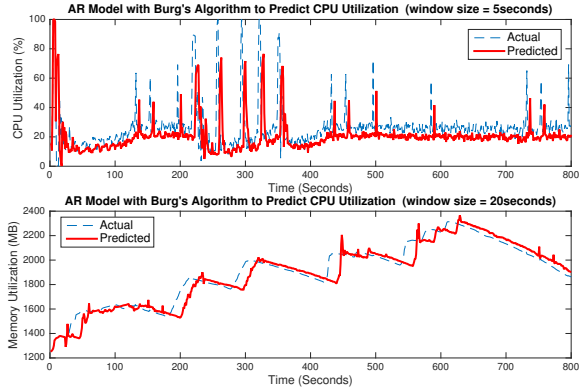

Fig. 10. Example of AR model based on Burg's algorithm to predict CPU (window size = 5sec) and memory (window size = 20sec) utilization at application tier using smaller time window of training and validation dataset

TABLE IV. TRADEOFF ANALYSIS FOR CHOOSING HIGH PREDICTION ACCURACY ($100 - MAPE$) WITH VARYING PREDICTION TIME WINDOW FOR ARIMA MODEL (WITH AND WITHOUT BANDPASS FILTER) AND AR MODEL USING BURG'S ALGORITHM BASED ON DIFFERENT TIME WINDOWS

	% Accuracy of ARIMA with BP Filter		% Accuracy of ARIMA without BP Filter	
Time (sec) Window	CPU	memory	CPU	memory
10	76.02	91.99	60.24	81.83
50	80.84	96.51	60.72	85.34
100	84.08	97.71	62.56	87.38
200	85.26	98.27	63.57	87.63
300	85.93	98.63	64.95	87.87
400	86.76	98.63	69.81	88.28

	% Accuracy of AR with Burg's Algo	
Time (sec) Window	CPU	memory
1	86.6	99.87
5	75.74	99.38
10	60.54	98.75
20	37.84	97.37
30	23.27	96.32
40	14.29	95.50

Hence, $(100 - MAPE)$ measures the percentage of accuracy of our prediction model. The *Mean Absolute Percentage Error* (MAPE) values for the CPU and memory usage prediction (using Bandpass Filter) was recorded as 13.23% and 1.37%. Thus, the prediction accuracy $(100 - MAPE)$ of our model for the CPU and memory usage are 86.77% and 98.63%. To check the significance of bandpass filter, we conducted two more experiments: 1) ARIMA(2,1,1) based prediction without using a bandpass filter as shown in Figure 9, and 2) AR(4) with Burg's algorithm as shown in Figure 10, using the built-in functions of MATLAB [26].

Table IV shows a *tradeoff analysis* for choosing a method with higher predicton accurracy and optimal prediction time window. The choice of an optimal prediction time window is equally important as the higer prediction accuracy because if that window is too small then it is not practical to perform all the dynamic resource adjustments in that short span of time, and if that window is too large then the prediction becomes meaningless because of rapid changes in the data. Table IV clearly shows the significance of using a bandpass filter, as the ARIMA model with a bandpass filter has higher prediction accuracy for both CPU and memory utilization for the different time windows that we tried. We also found that the ARIMA model with a bandpass filter has a higher prediction accuracy for both CPU and memory utilization for average to larger (200-400 seconds) time window. However, the AR model using Burg's algorithm acheives a high prediction accuracy of 86.6% for CPU utilization when time window was *1 second*, and has a higher prediction accuracy for memory utilization when time

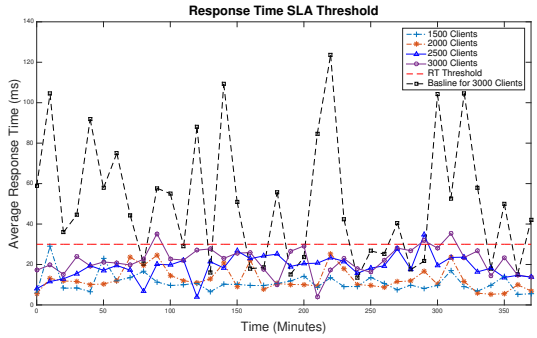

Fig. 11. Average response time with with 4 heaviest workloads for a period of 380 minutes

window is $\leq$ *100 seconds*. Our experimental results show that *400 seconds* is an optimal time window that allows sufficient time to adjust the CPU and memory resources, and helps our model (based on ARIMA with bandpass filter) to acheive high prediction accuracy. Finally, we conclude that accuracy of our model for memory usage prediction is much better than for CPU usage prediction, mainly because of much more noise in the CPU usage data compared to the memory usage data.

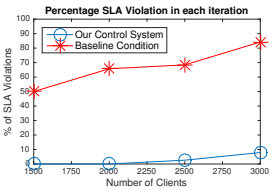

Fig. 12. Comparison of response time SLA violations with baseline criteria for 4 heaviest workloads

Fig. 13. 95th percentile of average response time for 4 heaviest workloads

The performance of the application is inversely proportional to the response time, i.e., lower response time shows better performance. Based on our prediction accuracy, and simulations with different workloads we sought to achieve a response time SLA of $30ms$. Table I (see Section II) shows that response time SLA achieved by our control system is very low compared to the SLA chosen by some prior work such as: Padala et al. [6] and Kalyvianaki et al. [17] who maintained a mean response time SLA of ≤ 1 second; Wang et al. [18] who presented a response time SLA of 0.31 second; Iqbal et al. [5] who imposed a 1 second response time SLA; and Bi [12] who achieved a 0.8 second response time SLA. We conducted 30 experiments by increasing the workload, however, we are highlighting the results from four heaviest workloads for the average response time. We varied the number of clients per node as: 1500, 2000, 2500 and 3000, and captured the average response time for each iteration, along with the baseline condition we used for comparison, i.e. when our control system is not applied, for 3000 clients as shown in Figure 11. From the graph of the percentage of SLA violations, shown in Figure 12, we can see that there were no SLA violations for workloads with 1500 and 2000 clients per node, but after we increased that value to 3000 clients we see a minor violation of 2.63% and then it increases to 7.89% in case of a fourth workload conducted with 3000 clients per

node. The tendency of increasing the percentage of violations is expected because of the hardware configuration limits and the length of prediction time window. Figure 12 also shows that the percentage of SLA violations is still very low compared to the baseline condition, where the violation is 50% with 1500 clients, and goes up to 84.21% for 3000 clients. The baseline condition describes the scenario in which our control system is not applied to the test application i.e. the default behavior of the application. The box-plot shown in the Figure 13 displays the nature of average response time with variation of these four heavy workloads that we achieved using our control system. Taking the average response time from all the experiments we conducted we are able maintain a 95% confidence interval of $14.9676ms$ to $17.2803ms$ which is well within the SLA range.

VII. Conclusion

With an increasing number of migrations of web applications from dedicated hosting to cloud hosting, it is becoming important for cloud providers to choose an efficient way to manage dynamic resource provisioning. In this paper, we described a feedback-based control system that we developed to maintain a response time SLA and to perform on-demand scaling of VMs, whenever needed. One of the major issues we addressed in this paper is to deal with the noise in resource utilization data by using a Bandpass filter. Generally, people have ignored this factor of real time noise while doing experiments with trace-driven simulations. For example, if you use traces from Google Cluster dataset [4], then you may not be dealing with the real time noise in the resource usage data that is caused either due to large number of processes running in the system, or due to the frequent context switch between processes. We modified and deployed the RUBiS application in a multi-tier architecture to perform all of the experiments. The workload we used in our experiments effectively created static and dynamic requests to vary the resource demands in both the application and database tiers of the application. We validated our adaptive controller on a testbed, and maintained a response time SLA of $30ms$ with a maximum of 7.89% of SLA violation. Hence, using this control system we are able achieve the main goal of this paper, i.e., performing dynamic resource provisioning more accurately to avoid SLA violations.

References

[1] R. Kohavi and R. Longbotham, "Online Experiments: Lessons Learned," *Computer*, vol. 40, pp. 103–105, Sept. 2007.

[2] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, pp. 155–162, Jan. 2012.

[3] D. Huang, B. He, and C. Miao, "A survey of resource management in multi-tier web applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1574–1590, 2014.

[4] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *Proceedings of the 9th international conference on Autonomic computing*, pp. 145–154, ACM, 2012.

[5] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive Resource Provisioning for Read Intensive Multi-tier Applications in the Cloud," *Future Gener. Comput. Syst.*, vol. 27, pp. 871–879, June 2011.

[6] 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 Operating Systems Review*, vol. 41, pp. 289–302, ACM, 2007.

[7] R. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2014.

[8] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, vol. 32, pp. 82–98, Mar. 2014.

[9] B. A. Shenoi, *Introduction to Digital Signal Processing and Filter Design*. John Wiley & Sons, Nov. 2005.

[10] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[11] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "Performance and Scalability of EJB Applications," in *Proceedings of the 17th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '02, (NY, USA), pp. 246–261, ACM, 2002.

[12] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 370–377, IEEE, 2010.

[13] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Decision Making in Autonomic Computing Systems: Comparison of Approaches and Techniques," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, (NY, USA), pp. 201–204, ACM, 2011.

[14] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What Does Control Theory Bring to Systems Research?," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 62–69, Jan. 2009.

[15] A. Kamra, V. Misra, and E. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered Web sites," in *Twelfth IEEE International Workshop on Quality of Service, 2004. IWQOS 2004*, pp. 47–56, June 2004.

[16] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive Control of Multi-Tiered Web Applications Using Queueing Predictor," in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pp. 106–114, Apr. 2006.

[17] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th international conference on Autonomic computing*, pp. 117–126, ACM, 2009.

[18] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions," in *Ambient Networks* (J. Schonwalder and J. Serrat, eds.), no. 3775 in Lecture Notes in Computer Science, pp. 133–144, Springer Berlin Heidelberg, 2005.

[19] Q. Zhu and G. Agrawal, "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC, (NY, USA), pp. 304–307, ACM, 2010.

[20] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung, "Self-managing systems: a control theory foundation," in *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the*, pp. 441–448, Apr. 2005.

[21] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 500–507, July 2011.

[22] V. Tran, V. Debusschere, and S. Bacha, "Hourly server workload forecasting up to 168 hours ahead using Seasonal ARIMA model," in *2012 IEEE International Conference on Industrial Technology (ICIT)*, pp. 1127–1131, Mar. 2012.

[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[24] C.-h. Chen, *Signal processing handbook*, vol. 51. CRC Press, 1988.

[25] J. P. Burg, "A new analysis technique for time series data," in *Modern Spectrum Analysis (Edited by D. G. Childers), NATO Advanced Study Institute of Signal Processing with emphasis on Underwater Acoustics*, (NY, USA), IEEE Press, 1968.

[26] "Estimate parameters of AR model for scalar time series - MATLAB." http://www.mathworks.com/help/ident/ref/ar.html, visited 2016-09-19.