Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers

Wei Xu
University of California, Berkeley
Berkeley, CA 94720, USA
xuw@cs.berkeley.edu

Xiaoyun Zhu Sharad Singhal Zhikui Wang
Hewlett-Packard Laboratories
Palo Alto, CA 94304, USA
{xiaoyun.zhu, sharad.singhal, zhikui.wang}@hp.com

Abstract*—It is challenging to reduce resource over-provisioning for enterprise applications while maintaining service level objectives (SLOs) due to their time-varying and stochastic workloads. In this paper, we study the effect of prediction on dynamic resource allocation to virtualized servers running enterprise applications. We present predictive controllers using three different prediction algorithms based on a standard auto-regressive (AR) model, a combined ANOVA-AR model, as well as a multi-pulse (MP) model. We compare the properties of the predictive controllers with an adaptive integral (I) controller designed in our earlier work on controlling relative utilization of resource containers. The controllers are evaluated in a hypothetical virtual server environment where we use the CPU utilization traces collected on 36 servers in an enterprise data center. Since these traces were collected in an open-loop environment, we use a simple queuing algorithm to simulate the closed-loop CPU usage under dynamic control of CPU allocation. We also study the controllers by emulating the utilization traces on a test bed where a Web server was hosted inside a Xen virtual machine. We compare the results of these controllers from all the servers and find that the MP-based predictive controller performed slightly better statistically than the other two predictive controllers. The ANOVA-AR-based approach is highly sensitive to the existence of periodic patterns in the trace, while the other three methods are not. In addition, all the three predictive schemes performed significantly better when the prediction error was accounted for using a feedback mechanism. The MP-based method also demonstrated an interesting self-learning behavior.

Keywords-utility computing, virtualization, resource allocation, predictive control, feedback control

I. INTRODUCTION

The recent IT industry initiatives in utility computing envision that today's enterprise data centers will become computing utilities that can provide infrastructure on demand to business critical applications such as enterprise resource planning applications, database applications, customer relationship management applications, and general e-commerce applications. A key enabler for this vision is server virtualization that allows applications to be hosted inside virtual servers instead of physical ones. A challenge that comes with virtualization is how to effectively manage capacities of such virtual servers to increase resource utilization while ensuring that the hosted applications can meet their service level objectives (SLOs). As opposed to capacity planning that deals with long-term allocation of resources to virtual servers, dynamic resource allocation responds to changes in demands in real time. Control-theory based techniques have been applied to deal with this challenge. In this paper, we study both predictive and feedback control-based techniques that can be used to periodically determine the amount of resource needed for each virtual server.

Standard feedback controllers such as the PI (proportional and integral) controllers compare the measured attributes with their desired values and use the errors to compute the actuation levels for the next interval. Such a controller is reactive in the sense that it relies on the feedback mechanism to handle disturbances in the system caused by uncontrollable factors such as changes in the workload. It cannot respond to such a change until its effect on the output metric has been observed, which normally results in at least a one step delay, which in turn may lead to SLO violations due to buffer overflows, lost connections, or long latencies. In this paper, we explore the use of time series prediction to exploit repeatable patterns or short-term correlation in the workload so that more proactive control actions can be taken to avoid the potential SLO violations in face of workload changes.

Time series prediction is a well-understood technique used in the financial or other industries to handle changes and uncertainties in an environment. A commonly used prediction approach uses auto-regressive (AR) models that account for temporal correlations between the current value of an attribute and its recent history [20]. Another method, ANOVA, analyzes long-term repeatable patterns in a time series [21]. A combination of ANOVA analysis and short-term AR-based correlation analysis is another common technique used for prediction [17]. We introduce yet another prediction algorithm, a multi-pulse (MP) model, first used in speech processing [22], which performs searches for both long-term and short-term patterns in an online fashion and therefore eliminates the need for offline analysis as in the ANOVA approach.

Predictive control is a technique that determines the actuation levels for the next control interval(s) based on prediction(s) of certain attributes for the next interval(s). It is a proactive approach as opposed to the reactive, feedback-driven approach, e.g., in the standard PI control. In this paper, we introduce a simple predictive controller that directly sets the actuation level for the next control interval based on the target value for the output metric and the predicted value for a related variable. We then compare the performance of the predictive controller using AR-, ANOVA-, and MP-based algorithms to that of an adaptive integral (I) controller using a case study.

Our case study is based on a hypothetical virtual server environment using a set of CPU utilization traces collected on 36 servers from an enterprise data center. Due to the fact that these traces were collected in an open-loop environment, we use a simple queuing algorithm to simulate the closed-loop CPU usage based on the original data and the computed CPU allocation. We also emulate the workload on a test bed using a Web server hosted inside a Xen virtual machine [27] to re-create the original utilization traces inside the virtual servers. The CPU allocation to a virtual server was controlled using the two classes of controllers. We compare the results of these predictive or feedback controllers from all the servers. We find that the MP-based predictive controller performs slightly better statistically than the other two predictive controllers by achieving a better balance among multiple metrics. It also demonstrates an interesting self-learning behavior. The ANOVA-based approach that derives the fixed model offline does not perform well compared to the other methods that adapt their models or parameters online. Finally, all the three predictive schemes performed significantly better when the prediction error was considered using a feedback mechanism.

The remainder of this paper is organized as follows. Section II discusses related work. Section III reviews the three different prediction algorithms. Section IV introduces a standard feedback controller and a predictive controller. Section V describes the data for our case study and the utilization control system. Section VI presents the results from our simulation study. Section VII describes our test bed set up and presents the emulation results. Finally, we conclude in Section VIII.

II. RELATED WORK

A. *Feedback control in computer systems*

Feedback control theory has been applied to solve a number of performance or quality of service (QoS) problems in computing systems in the past several years (see [1] [2] and the references therein). In these applications, we see two major challenges for appropriate system modeling and effective controller designs: the complex behaviors of computing systems themselves, and the time-varying demands placed on these systems by stochastic and sometimes bursty workloads.

Most early work in this area assumed that the system under control is linear, and the parameters can be identified offline, e.g., in [3], where a MIMO controller was used for automated configuration tuning for a Web server. However, due to the wide variation of demands observed in computing systems, the parameters or even the structure of the models could change over time. To deal with the time-varying parameters in linear models, adaptive control theory has been applied to systems in the context of, for instance, caching services [4], storage systems [5] and resource containers [6]. This approach allows the parameters of the model to automatically adapt to changes in operating conditions using online system identification.

Nevertheless, it is usually not enough to consider only the linear behavior since much of systems' behavior exhibits clear nonlinearity, for instance, the nonlinear relationship between the response time and CPU allocation [7], the saturation of the control actuators and bounded buffer sizes [8], and the intrinsic time-varying delay in queuing systems. In [2] the authors modeled the relationship between the content adaptation level and the CPU utilization as a time-varying static gain with no dynamics but with a time delay. In [7], the system's nonlinear and bimodal behavior in different operating regions was studied quantitatively, and controllers that could adapt to the bimodal behavior were developed.

B. *Proactive control algorithms*

Many algorithms have been described in the literature for proactive control. For instance, the derivative operation [8] in PID controllers anticipates potential changes and improves the responsiveness of the closed-loop system. However, it is very sensitive to noise, therefore is often unsuitable for many computing systems. Model Predictive Control [9] is a widely applied methodology, which uses a model to predict the system's behavior over a finite future horizon and chooses the control action that optimizes a cost function subject to constraints. This approach was used in [10] to control CPU utilization in distributed real-time systems. It requires online solution to a constrained optimization problem, which may not be feasible for all systems. It is also usually limited to fixed linear models that are known a priori.

Another technique is feed-forward control [8], which sets the actuator directly based on the predicted behavior so that the system can react to disturbance before it takes effect. This scheme can be combined with feedback control to correct the steady-state errors from the estimation. In [11], queuing theory was applied to predict the future queuing delay based on observation of request arrival rate and estimation of service rate, from which the queuing delay in the steady state can be computed with a simple formula [12]. In [13], this predictor was improved to respond to sudden and transient workload changes and the impact on the latency of future requests were estimated through heuristic flow-level approximation. Similar techniques and control structures were also applied in [14] for relative delay guarantees in Web servers. In [15], resource allocations to meet certain SLOs were computed based on the predicted workloads using on-line measurements of the request arrival process, service demand distribution and queue length.

The queuing theory based predictors can better model the queuing behavior and anticipate the impact of the workload changes on the controlled target such as the response time. However, it does require additional implementation of schemes to model and estimate the arrival process and service demand. The relationship between these workload metrics and SLO metrics such as response time and throughput has to be analyzed. In contrast, we study in this paper the possibility of predicting future resource demands through resource utilization metrics such as CPU consumption. This metric is intuitive, easy to measure, and applicable to a wider class of workloads.

C. *Workload modeling and capacity planning*

Workload forecasting has also been widely studied in capacity planning [23] and fault detection. In [16], application demand profiles were computed from historic resource utilization traces that demonstrated clear daily and weekly patterns. Admission control algorithms for a data center were then designed so that statistical assurances can be offered to the hosted applications. The workload model in [17] consists of a number of regular calendar patterns and an AR(2) process with model parameters estimated from historic data. These models were then used to predict the probability of threshold violations (in HTTP operations per second) so that proactive actions can be taken by the service provider. This approach was further studied in [18] for analyzing the trend and distribution of the residual process. Case studies on the application of predictive algorithms to computing system management were presented in [19] to deal with system failures in long-term or short-term. As opposed to these approaches that require offline training, we explore in this paper some online prediction schemes that can adapt to changes in workload patterns.

III. Review of Three Prediction Algorithms

In this section we provide an overview of three prediction methods and discuss their relationship.

A. Auto-Regressive (AR) Model for Prediction

Let $\{x(k)\}$ be the time series of an attribute x that is of interest to a particular problem, where $x(k)$ represents the measured value of x during time interval k. At the beginning of every interval k, a standard auto-regressive model predicts an attribute's value for the current interval using a linear combination of its measured value in the past several intervals,

$$\hat{x}(k) = \sum_{i=1}^{m} a_i x(k-i), \tag{1}$$

where $\hat{x}(k)$ is the predicted value for $x(k)$, a_i's are the predictor coefficients, and m is the order of the model that indicates the number of past samples used for the prediction. This model is useful for systems with some amount of memory and therefore an attribute's value is strongly correlated to its recent past.

The predictor coefficients can either be computed offline using the least-squares method on training data, or estimated online using the recursive least-squares (RLS) algorithm [24]. The latter approach allows the AR model to be updated periodically, adapting to possible changes in the system.

The standard AR model is not sufficient to represent long-term repeatable patterns in an attribute. For example, if the attribute x demonstrates certain behavior at the fixed time every day, while the sampling interval for x is one minute, then such periodic pattern will not be captured in the above model.

B. Prediction based on ANOVA Decomposition

If we assume that the measured attribute x is affected by certain independent factors, such as time-of-day, or day-of-week, then x can be modeled with two processes based on ANOVA decomposition [21]:

$$x(k) = c(k) + r(k), \tag{2}$$

where $c(k)$ captures the periodic patterns and $r(k)$ represents the residual process. For a pattern based on two factors, for instance, $c(k)$ can be modeled as,

$$c(k) = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij}, \tag{3}$$

where μ is the overall mean of the series, i and j indicate the location of index k in each individual period, α_i and β_j represent the main effects of the two factors, and the last term represents the interactive effect between the two factors.

If time dependency still exists after factoring out the known effects, the residual process can be further decomposed as

$$r(k) = y(k) + \varepsilon(k), \tag{4}$$

where $y(k)$ represents the non-stationary process and $\varepsilon(k)$ is a stationary process with zero mean. Sometimes $y(k)$ can be described with an AR model [17]:

$$y(k) = \sum_{i=1}^{m} b_i y(k-i)\,. \tag{5}$$

By identifying repeatable patterns and estimating parameters from historic data, we can predict the value of $x(k)$ based on the patterns and its past values as

$$\hat{x}(k) = u + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \sum_{i=1}^{m} b_i y(k-i). \tag{6}$$

We refer to such a combined approach as the ANOVA-AR approach later in the paper. The shortcomings of using the ANOVA approach are two fold. First, for the time series to be predicted, it requires a large amount of historic data so that repeatable patterns can be identified and built into the model. Second, different data traces may have different periodicity. Therefore, a model with fixed periodic pattern may not be applicable to all traces. The second point will be further illustrated in the case study.

C. A Multi-Pulse (MP) Model for Prediction

Here we introduce another prediction model that attempts to track both long-term patterns and short-term correlations in a time series. It maintains the online learning capability of the AR model and eliminates the need for offline analysis as in the ANOVA approach. The model follows:

$$\hat{x}(k) = \sum_{i=1}^{m} a_i x(k-n_i) \tag{7}$$

The only difference between model (1) and model (7) is that rather than using the samples immediately preceding the attribute value to be predicted, (7) can use sample values from much earlier in the history for prediction. The model in (7) was first used in [22] to compute excitation signals in high-fidelity speech coding using a closed-loop analysis-by-synthesis technique called multi-pulse analysis. We refer to it as the MP model in this paper. Unlike the AR model in (1), both the predictor coefficients $\{a_i\}$ and the predictor delays $\{n_i\}$ have to be computed dynamically in this model. The predictor delays are computed by minimizing the mean-squared error between the predicted value and the samples over an optimization window, while searching for the predictor delays over some history. The computational aspects of this predictor are described further in [22]. For the purpose of this paper, we define two parameters, the history length H and the size of the optimization window W, besides the number of coefficients m.

IV. Predictive and Feedback Control

A classical feedback control system uses an input-output relationship to represent a system, where the variable y represents one or more system metrics that the designer or user of the system cares about, such as the throughput or response times of an Internet service, and the variable u represents one or more control knobs that the controller can tune, such as some configuration parameters of the Web server, so that y can satisfy certain requirements. A standard feedback loop consists of an input-output system as described above, a controller, a sensor, and an actuator. Typically, discrete-time control is used for controlling computing systems. A discrete-time control system works around the notion of a sampling interval. At the beginning of each sampling interval k, the controller obtains $y(k-1)$, the measured value of y in the last sampling interval, from the sensor, and decides the value for u for the current interval, $u(k)$, and passes it to the actuator for execution. The parameters of the control algorithm can be designed offline, or online while the system is in operation. The latter is referred to as adaptive control, which is particularly useful for systems whose model parameters vary over time due to changes in the operating conditions.

This section reviews a simple adaptive feedback controller, and introduces a predictive controller that uses predictive methods as the basis of control actions. Finally, we discuss the relationship between these two types of controllers.

A. An Adaptive Feedback Controller

A common design goal for a feedback control system is to ensure that the output variable, $y(k)$, remains within the neighborhood of a target value, y_{ref}. This is referred to as a regulation problem. An integral controller is a commonly used feedback controller for a regulation problem, because it ensures zero steady state error, i.e., convergence of $y(k)$ to y_{ref} if the closed-loop system is stable. The following equation defines an adaptive integral (I) controller:

$$u(k) = u(k-1) + K_I(k)(y_{ref} - y(k-1)), \tag{8}$$

where $K_I(k)$ is the integral gain that measures the aggressiveness of the control actions. Note that this controller is different from a standard integral controller because K_I is a function of time, meaning that it can adapt to varying operating conditions of the system over time. The following diagram illustrates the key components of the closed-loop system.

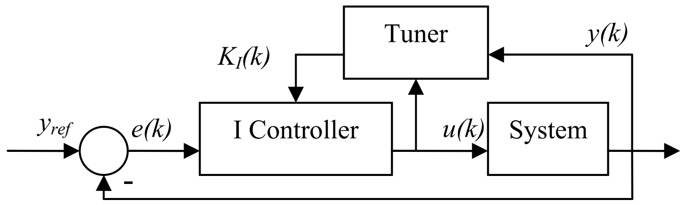


Figure 1. Block diagram of an adaptive integral controller

Note the sensor and actuator are omitted to simplify the diagram. In [7] we presented an adaptive I controller for regulating the relative utilization of a resource partition, which outperformed both I controllers and PI controllers with fixed gain values. Other adaptive controllers, such as adaptive PI or PID controllers can be designed with more parameters to be self-tuned online.

B. A Direct Predictive Controller

Suppose that the output, y, is a function of both the control action, u, and some state of the system, x:

$$y(k) = g(x(k), u(k)), \tag{9}$$

where the functional form $g(\cdot)$ is known. Also assume that $x(k)$ is predictable using the prediction algorithms described in Section III, and $\hat{x}(k)$ is the predicted value of $x(k)$. A simplest possible predictive controller has the following form:

$$u(k) = u^* \quad \text{s.t.} \quad g(\hat{x}(k), u^*) = y_{ref}, \tag{10}$$

This controller directly sets the value of $u(k)$ to be such that the predicted value, $\hat{y}(k)$, for the next interval equals to its target value. Therefore, we refer to it as the "direct predictive controller". The key components of the control loop are shown in the following diagram.

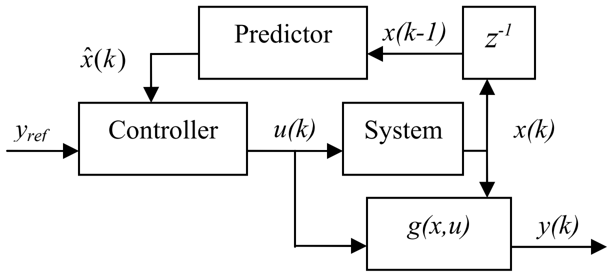


Figure 2. Block diagram of the direct predictive controller

Note the backward-shift operator z^{-1} is used because the predicted value $\hat{x}(k)$ only depends on past values for $x(k)$. An underling assumption for the direct predictive controller is that such u^* value exists. We will see later that this assumption is valid for the case study we performed.

C. Relationship between Predictive and Feedback Control

The controllers in (8) and (10) appear quite different, because (8) performs incremental corrective control actions based on the gap between the measured output value and its target, while (10) may be more aggressive because it attempts to reach the target value in one interval assuming that the prediction is accurate. However, as we will show in the next section, when the adaptive gain K_I takes a particular form, the adaptive controller in (8) can be viewed as a special case of the predictive controller in (10).

In general, the two controllers are different. Intuitively, when the predictor does a perfect job, the predictive controller should work better, especially if the system experiences large spikes in the workload periodically. On the other hand, when the prediction error is large, then the predictive controller may make poor decisions or even cause the system to be unstable. These observations can be further demonstrated in our case study.

V. Virtual Server Utilization: Case Study

This section presents a case study that demonstrates how the above predictive and feedback controllers may be applied in practice to manage capacities of virtual servers in a consolidated environment. Through both simulation and emulation studies, we explore the impact of using predictive techniques and compare the results with those from the feedback-driven adaptive integral controller. For the purpose of our study, we consider CPU utilization information from 48 servers in an enterprise data center. In this section, we describe our hypothetical virtual server environment, the architecture and goal for the closed-loop control system, the design of our simulations and the results we obtained.

A. Description of Data

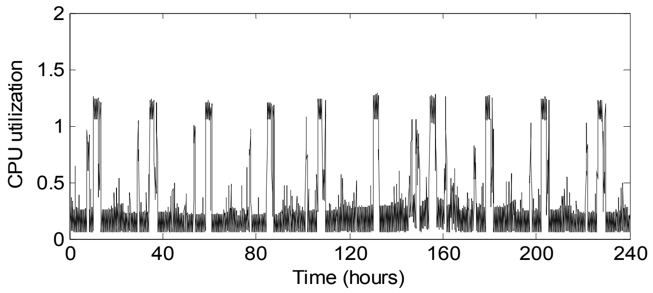
We obtained CPU utilization information from a collection of 48 servers hosting enterprise applications. The servers have between 2 and 8 CPUs each, with the majority having either 4 or 6 CPUs. The data was collected between September 5, 2001 and October 24, 2001 for seven weeks. For each server, the average CPU utilization across all processors on the server was reported for each five-minute measurement interval. The information was collected using *MeasureWare* (Openview Performance Agent) [25].

We conduct a hypothetical server consolidation practice where the above servers are consolidated onto a virtual server environment. We interpret the CPU utilization of each physical server as the CPU "demand" within a virtual server. For instance, if a physical server consumed an average of 1.5 CPUs in a five-minute interval, then the CPU demand for the corresponding virtual server for that interval is also 1.5 CPUs. We exploit the fact that changes in server utilization reflect real changes in the activities of the applications hosted on these servers. Therefore, our virtual servers demonstrate the same temporal patterns and dynamic characteristics of the workloads that resulted in the original utilization traces. However, since our purpose is simply to evaluate our predictive and feedback control techniques, we are not concerned about scaling the loads with respect to processor speed, memory or I/O capacity.

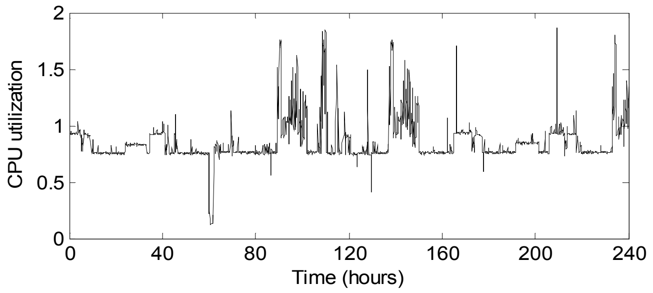
We reviewed the utilization data from each server, and found 12 servers with significant missing data or with an almost constant load. We disregard the traces for these servers, and found that the remaining 36 servers can be classified into two groups:

1. Group A (24 servers): with fairly strong periodicity based on spectrum analysis of the time series. The cyclic patterns in the utilization may be hourly, daily, or weekly, with daily patters being the most dominant.
2. Group B (12 servers): with no periodicity or very weak daily or weekly patterns.

The following figure demonstrates two example traces from the two different groups respectively. We only show the data for the first ten days so that we can clearly see the daily pattern in the first trace. There is no visible periodicity in the second trace, which is verified by spectrum analysis of the trace. Note that both servers had 6 CPUs, therefore neither was heavily utilized when the data was collected. This is true for the majority of the 48 servers we observed. This further motivates our study of more efficient resource allocation schemes so that higher resource utilization can be reached.



(a) A server from Group A, shows strong daily pattern



(b) A server from Group B, shows no periodicity

Figure 3. Measured CPU utilization from two servers for a week

B. Utilization Control for Virtual Servers

The CPU utilization of a server is a commonly watched metric to determine whether more or less CPU resource should be allocated to the server. Controlling CPU utilization of a virtual server in a consolidated environment is an effective way of ensuring sufficient allocation of CPU resources to hosted applications as well as increasing overall resource utilization of the host machine. For example, the usage-based operation mode in the HP-UX Workload Manager [26] allows the relative CPU utilization of a resource partition to be controlled within a user-specified range. Compared to SLO-based metrics such as response times, CPU utilization is easier to measure on the server side and is more intuitive to control because its relationship to the CPU allocation of the virtual server is more straightforward. The downside is that the relationship between a given relative utilization level and the client-perceived service level varies with the demand of the workload. Therefore, headroom is often provided to handle variations in the demands of the hosted applications. For example, 50% can be a control target for highly interactive applications, and 80% may be used for applications with more predictable demands. In this section, we study how effective the predictive and feedback controllers we introduced in Section IV are by simulating their application to utilization control of these virtual servers.

Before presenting the simulation algorithms, we need to define some notation for this particular case study. For any virtual server, let *d(k)* be the average CPU demand of its hosted applications in (possibly fractional) number of CPUs for sampling interval *k*. Let *u(k)* be the control knob, i.e., the (possibly fractional) number of CPUs allocated to that virtual server for interval *k*. The measured CPU usage by the virtual server, *x,* should be a function of both *d* and *u,* which will be described later. We define relative utilization of a virtual server, *y(k),* as the ratio of the measured CPU usage to the CPU allocation. That is,

$$y(k) = g(x(k), u(k)) = x(k) / u(k) . \quad (10)$$

The goal of the controller is to maintain the relative utilization of each virtual server at a specified level, y_{ref}. In this case, an adaptive I controller has the following form:

$$u(k) = u(k-1) - K_I(k)(y_{ref} - x(k-1)/u(k-1)) . \quad (11)$$

Note that in this particular case, a negative sign is used before $K_I(k)$ to ensure a negative feedback loop. We have tested different adaptation rules for $K_I(k)$, among which we found the following to be the most effective:

$$K_I(k) = \lambda x(k-1) / y_{ref} , \quad (12)$$

where λ is a positive constant that affects the aggressiveness of the control actions (see [7]) for analysis of this controller). In our simulations, we set $\lambda = 1.5$.

C. Predictive Control for Tracking Utilization

Now let us see if we can apply the predictive controller described in Section IV to track a utilization target on a virtual server. Based on the relationship in (10), if the measured CPU usage, *x(k),* is predictable using its values from the previous intervals, and $\hat{x}(k)$ is its predicted value, then the direct predictive controller simply determines *u(k)* as follows:

$$u(k) = \hat{x}(k) / y_{ref} . \quad (13)$$

Now the question is whether x(k) can be accurately predicted. From the data center CPU utilization data described earlier, we do observe that many of them demonstrate long-term or short-term correlation that can be exploited by some prediction algorithm. That reflects certain degree of predictability in the CPU demand, *d(k),* of the applications on a virtual server. The measured CPU usage in a closed-loop system, *x(k),* depends on both *d(k)* and the CPU allocation *u(k).* In a memoryless system, the following relationship holds:

$$x(k) = \min\{u(k), d(k)\} . \quad (14)$$

When *d(k)<=u(k)*, i.e., the demand is less than the allocation, the virtual server is underloaded, therefore the CPU usage equals the demand, i.e., *x(k) = d(k).* In this case, the application demand is observable through the measured CPU usage, and its timely patterns can be learned by our prediction models. However, when *d(k)>u(k)*, the virtual server is in an overload state, then the resulting CPU usage is capped by the CPU allocation, i.e., *x(k) = u(k),* and the original application demand becomes unobservable. The hope is, if the controller is successful in tracking the target utilization, which is typically less than 100%, the virtual server should always be underloaded so that *d(k)* remains observable. This assumption will be tested in our simulation and emulation studies.

The two controllers in (11) and (13) appear drastically different since the former uses feedback while the latter relies on prediction to handle changes in the application demand. However, if we make $K_I(k) = u(k-1) / y_{ref}$ in (11), we have:

$$u(k) = x(k-1) / y_{ref} . \quad (15)$$

Comparing the control laws in (13) and (15), it is easy to see that, with a special way of adapting K_I, the adaptive I controller in (11) is indeed the simplest possible direct predictive controller, where the measured CPU usage for the last sampling interval is used as the prediction for the next interval, i.e., $\hat{x}(k) = x(k-1)$. In this paper, we will not show results for this particular controller because it did not perform as well as other controllers in our simulations for obvious reasons.

D. Simulation of Closed-loop Usage

The simulations were run using each 5-minute utilization trace from a physical server as the CPU demand *d* for a virtual server, so that it captures the patterns and dynamics of real enterprise applications' resource demand. We then apply one of the formerly described controllers to compute the CPU allocation *u* for that virtual server. One caveat is, the original traces were collected in an open-loop fashion. Therefore, we need to simulate the closed-loop CPU usage, *x*, that would have been measured on a real controlled server, as a function of the CPU allocation, *u*, and the application demand, *d*.

The model in (14) is attractive for its simplicity. However, various queuing and timeouts in software systems can cause systems to violate the memoryless assumption, and the excess demand from the last period may have a strong impact on the current performance. We use the algorithm in Fig. 4 to simulate the queuing behavior of our virtual server, where *L* is the total queue capacity (in number of CPUs), and *q(k)* represents the queue length at the end of interval *k*. Basically, the queue keeps all the excess demand during overload periods up to a total capacity of L, and submits work to the CPU during subsequent underload intervals. When the total excess demand exceeds the queue capacity, the extra demand is dropped.

```
Inputs: $u(k), d(k), q(k-1), L$
Outputs: $x(k), q(k)$
Algorithm:
    1. if $u(k) <= d(k)$
    2.         $\Delta q = \min(d(k)-u(k), L-q(k-1))$;
    3.         $x(k) = u(k)$;
    4. else
    5.         $\Delta q = -\min(u(k)-d(k), q(k-1))$;
    6.         $x(k) = d(k) - \Delta q$;
    7. $q(k)=q(k-1) + \Delta q$.
```

Figure 4. Pseodu code for simulation of closed-loop utilization

This simulation model may appear simplistic. One assumption here is that CPU demand can be delayed without additional cost on CPU (e.g. to manage the queue, or to do scheduling). Also, we choose L to be 1 in our simulations, which is arbitrary. However, we find that it is sufficient to compare the difference between these controllers. It is also consistent with our emulation results that will be described in the next section.

VI. SIMULATION RESULTS

In this section, we present simulation results from applying both the direct predictive controller and the adaptive integral controller to the control of virtual server utilization. Inside the predictive controller, each of the three prediction algorithms, referred to as AR, ANOVA-AR, and MP, was used. In both the AR and MP-based predictive controllers, a fourth-order model (m=4) was estimated online. The search algorithm for the MP model also used a history of H=1152 (4 days) and an optimization window of W=288 (1 day). The ANOVA-AR model was estimated offline based on the first three weeks of trace data using a combination of a weekly pattern and a fourth-order AR model. It was then applied to the remaining four weeks of data in the predictive controller in the simulation. In each case, after simulating the closed-loop CPU usage, x, we calculated the relative utilization of a virtual server, $y=x/u$. The target utilization, y_{ref}, was set at 75% in all the simulations.

Fig. 5 demonstrates a statistical evaluation of the four controllers for the two server groups A (top) and B (bottom) as described in Section V.A. Each figure shows the cumulative distribution function (CDF) of the achieved relative utilization, $y(k)$, during the simulated 7-week period from all the servers in the respective group.

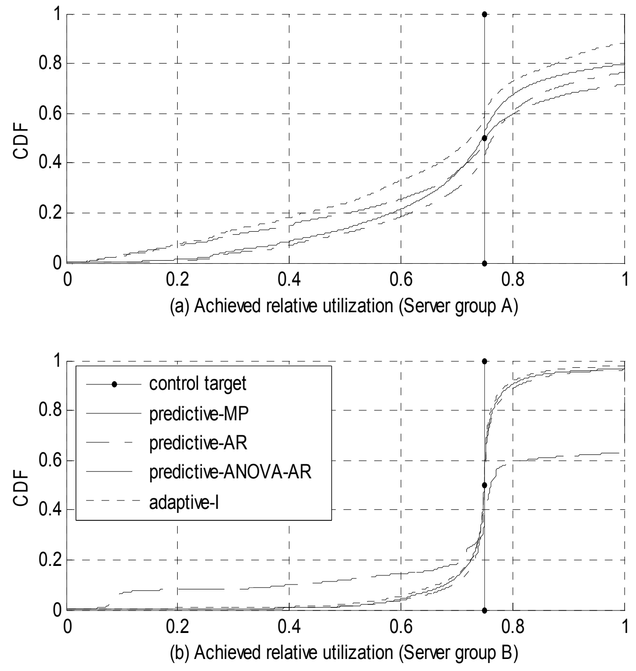


Figure 5. CDF of relative utilization from four controllers for server group A (top) and server group B (bottom): predictive-MP (solid), predictive-AR (dash-dot), predictive-ANOVA-AR (dashed), adaptive-I (dotted)

For each server group, we can compute the following statistical metrics from the CDF of the relative utilization y: mean (μ_y), standard deviation (σ_y), the percentage of time when the controller did not allocate enough CPU capacity to meet the target (P[$y>y_{ref}$]), and the percentage of time when the system is overloaded (P[$y=100\%$]). We can also compute the mean CPU allocation (μ_u) and the mean simulated queue length (μ_q). Table I and II show the values of these metrics from server group A and B, respectively.

TABLE I. STATISTICAL METRICS COMPUTED FROM SERVER GROUP A (NO CONSIDERATION OF PREDICTION ERROR)

Metrics	MP	AR	ANOVA-AR	Adaptive I
μ_y	0.73	0.76	0.72	0.66
σ_y	0.21	0.20	0.27	0.25
P[$y>y_{ref}$]	0.48	0.58	0.51	0.41
P[$y=100\%$]	0.20	0.24	0.28	0.12
μ_u	0.63	0.63	0.63	0.74
μ_q	0.09	0.09	0.18	0.04

TABLE II. STATISTICAL METRICS COMPUTED FROM SERVER GROUP B (NO CONSIDERATION OF PREDICTION ERROR)

Metrics	MP	AR	ANOVA-AR	Adaptive I
μ_y	0.75	0.75	0.77	0.74
σ_y	0.09	0.09	0.26	0.10
P[$y>y_{ref}$]	0.46	0.51	0.64	0.47
P[$y=100\%$]	0.03	0.04	0.37	0.02
μ_u	1.44	1.44	1.21	1.46
μ_q	0.02	0.02	0.35	0.01

For servers in group A that have strong periodicity, the performance from the three predictive controllers is fairly comparable, although the MP-based approach in general offers the best balance between multiple metrics, resulting in a mean utilization close to the target (μ_y=73%) and a smaller variance (σ_y=0.21), lower probability of under-allocation (P[$y>y_{ref}$]=48%), lower overload probability (P[$y=100\%$]=20%), as well as less queuing activity (μ_q=0.09), without requiring higher average CPU allocation (μ_u=0.63). The adaptive I controller on average allocated more CPU capacity (0.11 more CPU than the predictive controllers) and resulted in lower average utilization of the virtual servers (μ_y=66%). Consequently, its values for the other four metrics are better than those from the three predictive controllers. This is because this particular controller was designed to be fast-increasing (when under-allocating) and slow-decreasing (when over-allocating) to reduce oscillation between the two operating regions.

For servers in group B that have no or fairly week periodicity, the ANOVA-AR-based approach performed particularly poorly, resulting in an overloaded virtual server for 37% of the time across all the servers in group B. This is understandable given that the model was computed offline based on the underlying assumption of a weekly pattern in the demand. Because there are a fair number of traces with multiple cyclic patterns, a multi-period analysis along with online-adaptation of the AR model may improve the performance of the ANOVA-AR-based approach.

Interestingly, the MP and AR-based predictive controllers and the adaptive I controller performed almost equally well, tracking the utilization target closely and resulting in less than 5% of overload time and very minimum queuing. It may seem counter-intuitive that the two predictive controllers performed better for servers with less periodicity as in group B. This is because predictability in a demand can be due to shorter-term correlations as well as longer-term cyclic patterns. The AR model identifies the former, and the MP model can recognize both. When only the former exists or is dominant, the MP model acts like an AR model that utilizes samples from the recent past to predict the future. In addition, the servers in group B also happen to have less variable hence more predictable demands, therefore they are generally easier to control than those in group A.

The predictive controllers used in the previous simulations did not take into prediction errors. In principle, any predictive scheme should be able to use how accurate the prediction was in previous intervals as feedback to help improve the prediction accuracy for the current interval. To evaluate this, we consider a modified predictive controller that keeps track of the past prediction errors: $e_p(k-l) = \hat{x}(k-l) - x(k-l),\ l = 1,2,...$ It then estimates the mean, μ_e, and standard deviation, σ_e, of the prediction error in a sliding window of size W_{pe}, and computes the next CPU allocation as follows,

$$u(k) = (\hat{x}(k) - \mu_e + \sigma_e) / y_{ref}\,. \tag{16}$$

Note that σ_e is added onto the prediction so that there is a higher probability of over-allocating than under-allocating the CPU resource. This is usually preferred by data center operators since the latter may lead to performance degradation.

We reran the simulation on the 36 virtual servers with the modified predictive controller (with W_{pe}=4) using the same three prediction algorithms. Fig. 6 shows a comparison between the adaptive I controller and the predictive-MP controller with and without feedback of prediction errors (noted as "PEF") for server group A. These three controllers represent three different but related approaches for dynamic resource allocation: feedback-based, prediction-based, and prediction-feedback-combined. The results from the AR-based and ANOVA-AR-based predictive controllers are fairly similar to that from the MP-based controller when the CPU allocation was compensated with estimated prediction error. Statistical metrics as those in Table I can be computed from the results for server group A and are listed in Table III.

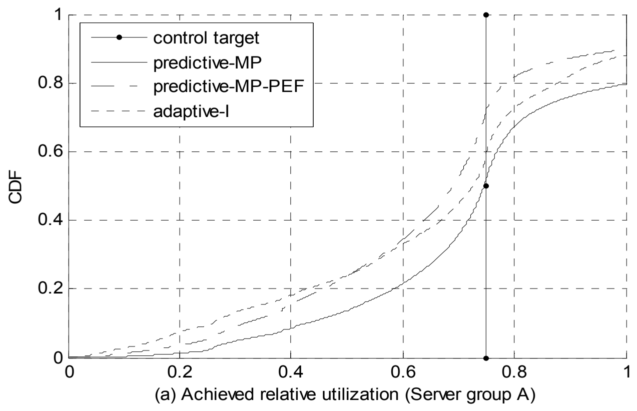


Figure 6. CDF of relative utilization from three controllers for server group A: predictive-MP (solid), predictive-MP-PEF (dash-dot), adaptive-I (dotted)

TABLE III. STATISTICAL METRICS COMPUTED FROM SERVER GROUP A (WITH CONSIDERATION OF PREDICTION ERROR)

Metrics	MP-PEF	AR-PEF	ANOVA-AR-PEF
μ_y	0.65	0.67	0.64
σ_y	0.22	0.22	0.21
P[y>y_{ref}]	0.28	0.36	0.25
P[y=*100%*]	0.10	0.14	0.08
μ_u	0.77	0.74	0.83
μ_q	0.04	0.05	0.03

As we can see from both Fig. 6 and Table III, due to the intended conservativeness of the modified predictive approach, all the three predictive controllers resulted in the mean utilization lower than the target (μ_y~65%), and higher average CPU allocation (μ_u=0.74~0.83). Compared to the unmodified predictive controllers and even the adaptive I controller (as in Table I), the new scheme offers lower probability of under-allocation (P[y>y_{ref}]=0.25~0.36), lower probability of overloading the system (P[y=*100%*]=0.08~0.14), and less queuing (μ_q~0.04). The results from server group B are quite similar, therefore are not shown here. One observation, though, is that, in spite of the poor performance of the predictive-ANOVA-AR controller for the server group B, the same scheme with compensation of prediction error delivered much better performance, similar to those from the other controllers. These results suggest that an approach combining prediction with even a simple feedback mechanism can be very powerful. Even when the prediction model being used is not accurate to start with, when observed prediction errors are fed back into the system and used to correct the control actions, the closed-loop control system may still perform reasonably well.

One important point we discussed in Section V.C is that, changes in the CPU demand, *d(k),* may be unobservable in the measured CPU usage, *x(k),* when a virtual server is overloaded, i.e., *u(k) < d(k).* Since *x(k)* is ultimately what we use in our prediction algorithms, one may worry that the predictive scheme may not work. However, we observed in our simulations an interesting self-learning behavior of the MP-based predictive controller. Fig. 7 demonstrates the simulation result from both the predictive-MP and predictive-MP-PEF controllers on the utilization trace as shown in Fig. 3(a). The CPU demand demonstrates a strong daily pattern. To make the figure more readable, we only show the first seven periods (i.e., the first week) here.

Fig. 7(a) shows the CPU demand from the original trace (dark dot solid) and the computed CPU allocation (light solid). The predictive algorithm started from a blank memory. Therefore, when there was a surge in the CPU demand in the first period (as in the first square wave), the controller did not allocate enough CPU capacity to handle the load. As a result, the virtual server was overloaded and the measured CPU usage was capped by the insufficient CPU allocation. In the next few periods, the allocation was still insufficient because what the predictive-MP algorithm observed from the earlier weeks was the truncated demand instead of the original square wave, which was unobservable during overload periods. However, the demand pattern was gradually being learned over time as the controller always allocates more CPU than what was predicted. In the last period, the allocation had almost caught up with the demand resulting in much better tracking performance. Interestingly, the predictive-MP-PEF controller with compensation of prediction error enhances this self-learning behavior because it intentionally allocates slightly more capacity at every interval to avoid overload. As shown in Fig. 7(b), the learning process was sped up and the controller was allocating enough capacity starting from the 3rd period. We will see confirmation of this learning behavior in the emulation results in the next section.

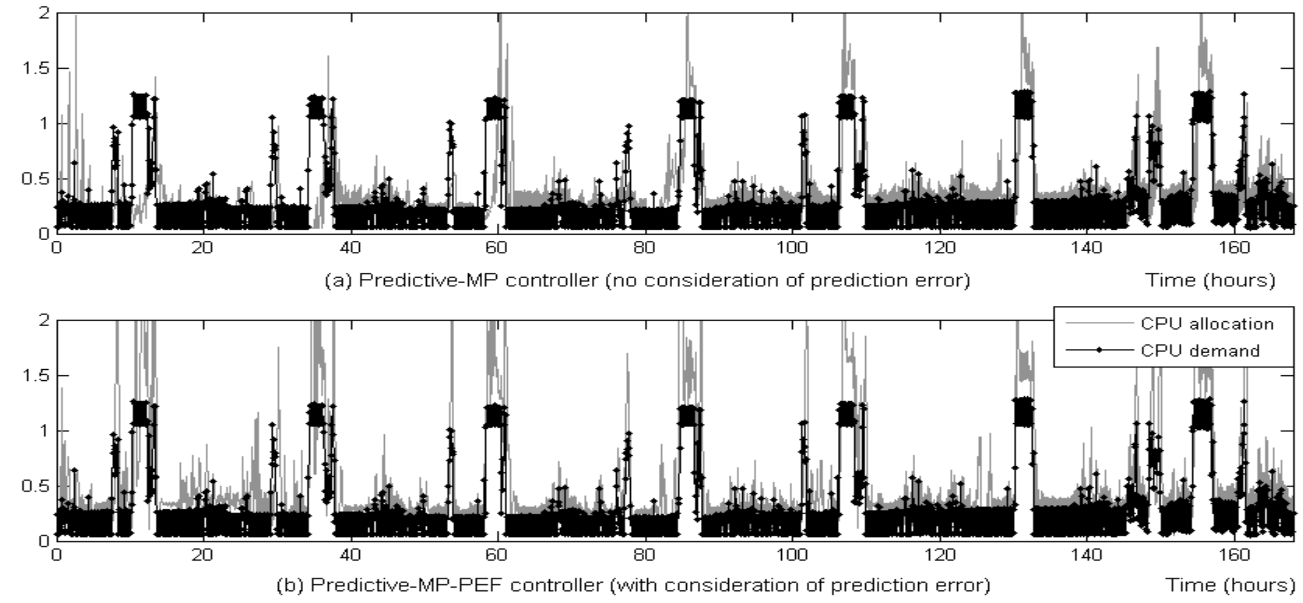


Figure 7. Simulation results by predictive-MP (top) and predictive-MP-PEF (bottom) controllers for one virtual server for the first week

VII. EMULATION ON A TEST BED

A. Experiment Setup

In addition to the simulations, we built a test bed to emulate the application demands seen in the data center utilization traces. We used Xen virtual machine [27] with an SEDF (Simple Earliest Deadline First) scheduler as the virtual server. The EDF algorithm, like most real-time scheduling algorithms, is based on reservation. An application is guaranteed to have certain share of CPU time during a period of time, even if the time allocated is not used, and an application cannot use more CPU time than allocated. Thus, this scheduling technique is a good fit for our abstraction of a sizable virtual server.

We used an Apache Web server [28] as the test application to be hosted inside a Xen virtual machine. The Apache server was configured to run with multiple *pthread,* and serves a trivial CGI program, which starts a process, does some random calculation, and returns the result in HTML. The reason for using this CGI workload is that it is CPU intensive, and has minimal impact of memory or I/O contention.

On the client side, we used a slightly modified version of *httperf* [29], an open source Web server workload generator. It was modified so that it can generate variable number of concurrent connections.

The experiments were performed on an HP ProLiant DL360 Generation 4 server, with two 3GHz Xeon processors and 3GB memory. We used the Xen unstable version, checked out on Jun 9th, 2005 and Linux kernel version 2.6.11 (patched for Xen), in all Xen domains. (Domain is a Xen terminology for virtual machine.) The domain with the Apache server runs on one processor, shared with one other competing domain. *Domain-0*, the controlling domain, runs the controller on a dedicated processor. We reserved 750MB memory for the Web server domain.

In order to reproduce the CPU utilization from the observed traces, we first ran experiments to obtain the average CPU time a CGI request uses, and thus calculated the rate at which the requests should be generated to produce a given level of CPU utilization. In each trace, each sample is a five-minute average utilization. To shorten the time span of the emulation, we change the sampling interval to 10 seconds, which can almost reproduce the same utilization changes on the virtual machine. Fig. 8 shows CDFs of the CPU utilization from both the emulation and the real trace.

This emulator has some advantages over a pure simulation algorithm, such as the one in Fig. 4. First, the emulator models the queuing behavior and degradation of application performance during overloading periods more accurately. Second, it captures the random disturbances in the system. Third, it also captures the inaccuracy in sensing and actuation. For example, the CPU utilization may not be accurate at every point, especially when overloaded, due to network I/O, thread scheduling and measurement errors. We can see the cost of control in the emulation as well.

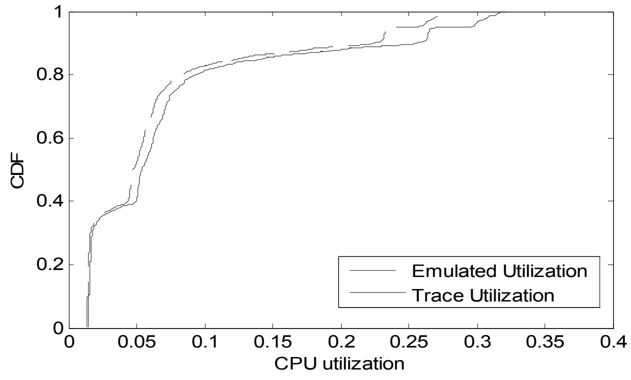


Figure 8. CDFs of emulated utilization (dashed) and real utilization (solid)

Of course, the limitations of this approach are also obvious. First, we only emulate one type of workload on a single application, which may not be the application(s) from which the original trace was generated. For example, our emulation only represents the behavior of the Apache Web server. Our workload is purely CPU intensive, and generates very few page faults, so there may be disturbances that were not captured in the emulation. Second, we could only emulate the average utilization value, while the distribution of the real values is unknown, and may be application dependent. However, since we only care about the average behavior of the system in this study, we believe the emulation is a reasonable approach.

B. Results

Due to the time required for emulating the entire data set (36 servers with 7 weeks of data), we emulated a small subset of the traces used for the simulation study. In particular, we used traces where the simulation showed interesting dynamics to validate the transient behavior of different controllers. In this section, we describe the emulation results on one of the traces (shown in Fig. 3(a)) using a hand-tuned PI controller and the MP-based predictive controller, with m=4, H=1052, W=288 (same as those used in the simulations).

Fig. 9 shows two periods in the emulation log. The figures in the left column represent the first peak in Fig. 3(a), while those on the right column are the 7th peak in Fig. 3(a). The two peaks in the trace look almost the same. Fig. 9(a) shows the behavior of the PI controller. The PI controller, which forgets history quickly, responded to the two peaks in a very similar way, as expected. Like all feedback-driven controllers, it is reactive to changes in the CPU demand, thus causing some delay in the corresponding changes in the CPU allocation.

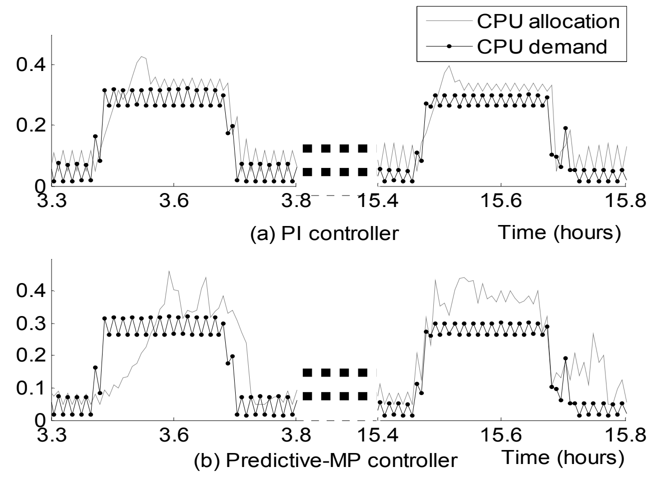


Figure 9. Transient response in CPU allocation (light solid) to step changes in CPU demand (dark solid-dot) from the PI controller (top) and the predictive-MP controller (bottom).

Fig. 9(b) shows the result of the predictive-MP controller. It shows a clear self-learning behavior similar to what we saw in the simulations (as in Fig. 7). It performs worse than the PI controller in the beginning (slower tracking of the first peak in the CPU demand). However, at the 7th peak, it actually increased the CPU allocation before the demand surged, causing no delay, therefore resulting in better application performance.

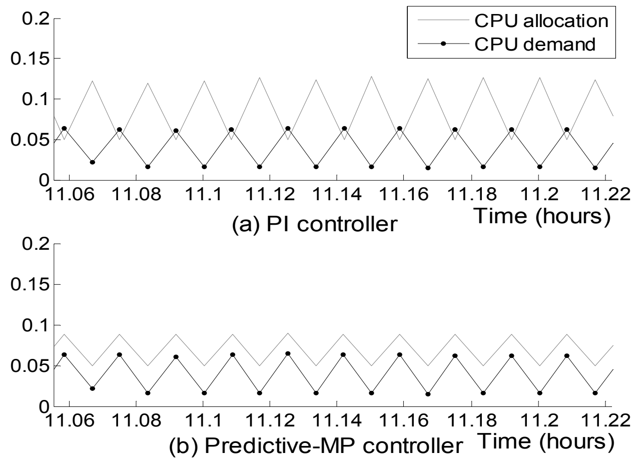


Figure 10. Transient response in CPU allocation (light solid) to small periodic oscillations in CPU demand (daek solid-dot) from the PI controller (top) and the predictive-MP controller (bottom).

Fig. 10 shows another interesting comparison of these two controllers. In a real system, small oscillations are often seen in many metrics at different time scales. If the oscillations have a period close to the control interval, reactive controllers, such as PI controllers, can 'hunt' by alternately setting the allocation either too high, or too low. This is visible in Fig. 10(a), where the PI controller is unable to track the changes in the demand. In contrast, as shown in Fig. 10(b), the predictive-MP controller can recognize this pattern quickly and predict the next demand accurately

However, prediction can also lead to problems as illustrated in the next example. Fig. 11 shows the behavior of the PI controller and the predictive-MP controller for the 7th, 8th and 9th peak of Fig. 3(a).

The CPU demand from the trace shows a wide peak, followed by a small peak. The PI controller, shown in Fig. 11(a), reacts to all the wide or small peaks in a similar fashion. In contrast, the MP-based predictor, as shown in Fig. 11(b), fails to predict the first small peak since such pattern did not exist in its history. At the second small peak, the predictive-controller was able to allocate more CPU due to its self-learning behavior. In the third period, the predictor increased the allocation where the small peak had occurred before, in anticipation of the increase in demand. However, this time, the small peak shifted to a different time, thus causing misalignment between the allocation and the demand.

The reason for the failure of the predictive-MP controller in Fig. 11 is the large optimization window (1 day) used by the prediction algorithm. Thus, narrow peaks that have a significantly shorter duration do not cause the predictor to adjust its model immediately to focus only on the recent history, while the PI controller only looks at the immediate history (the last interval) and adjusts to it. We are currently exploring ways of combining the benefits of prediction with rapid feedback-based adjustment in case of prediction errors, such as what we described in the last section.

VIII. CONCLUTIONS AND FUTURE WORK

In this paper, we study predictive closed-loop control techniques for systems management. Three predictive algorithms based on AR, ANOVA-AR and MP models and one adaptive I controller are evaluated in a case study on virtual server consolidation through simulations as well as emulation done on a Xen virtual machine. From both the simulation and emulation results, we found that, the predictive controller can deal with time-varying demands in a more proactive way once the demand pattern is learned and the prediction is accurate, but may also result in poor performance when the prediction error is big. In contrast, feedback-based adaptive controller always incurs some delay in responding to changes due to its reactive nature, but its behavior was more consistent across different demand pattens. We have explored in our simulations one way to improve the performance of the predictive controllers by taking into account estimated prediction error in future control actions. Further research and more experiments are required to evaluate other possible ways of combining predictive algorithms and feedback-based mechanisms to achieve better closed-loop performance in face of changes in demands or other system conditions.

It will also be interesting to apply the controllers presented here to other resource utilization traces for possibly different applications collected at different time scales. We would like to validate that the observations we made in this paper about the relative performance of different prediction algorithms, the self-learning behavior of the predictive-MP controller, and the interaction between prediction and feedback are also applicable to other dynamic resource control problems in data centers.

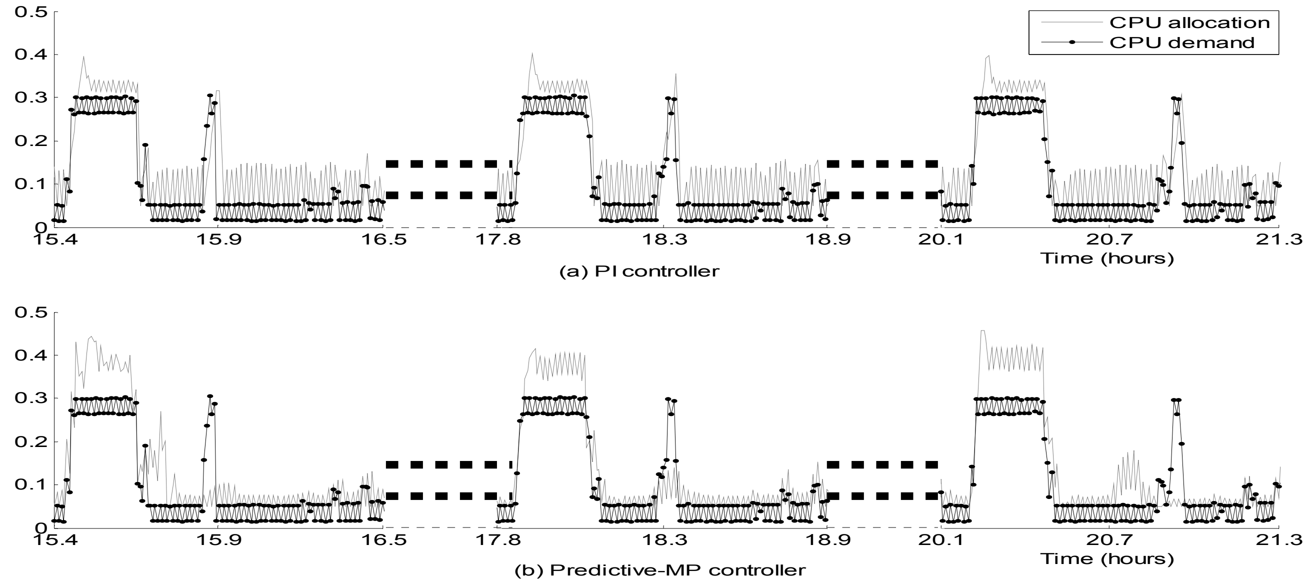


Figure 11. Section of a trace showing the impact of prediction errors by comparing the PI controller (top) with the predictive-MP controller (bottom). For each controller, the top curve (light solid) is the CPU allocation, and the bottom curve (dark solid-dot) is the CPU demand.

REFERENCES

[1] J.L. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, Feedback Control of Computing Systems, Wiley-Interscience, 2004.

[2] T.F. Abdelzaher, K.G. Shin, and N. Bhatti, "Performance guarantees for Web server end-systems: A control-theoretical approach," IEEE Transactions on Parallel and Distributed Systems, vol. 13, 2002.

[3] Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "MIMO control of an Apache Web server: Modeling and controller design," American Control Conference, 2002.

[4] Y. Lu, C. Lu, T. Abdelzaher, and G. Tao, "An adaptive control framework for QoS guarantees and its application to differentiated caching services," 10th IEEE International Workshop on Quality of Service, May, 2002.

[5] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance isolation and differentiation for storage systems," 12th IEEE International Workshop on Quality of Service, June, 2004.

[6] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource partitions on shared servers," 9th International Symposium on Integrated Network Management, May, 2005.

[7] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-based control for dynamic sizing of resource partitions," 16th IFIP/IEEE Distributed Systems: Operations and Management, October, 2005.

[8] K. Astrom and T. Hagglund, PID Controllers: Theory, Design, and Tuning (2nd Edition), Instrument Society of America, 1995.

[9] E.F. Camacho and C. Bordons, Model Predictive Control, Advanced Textbooks in Control and Signal Processing, Springer-Verlag, 2004.

[10] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time Systems with End-to-End Tasks," IEEE Transactions on Parallel and Distributed Systems, 16(6):550-561, June 2005.

[11] L. Sha, X. Liu, Y. Lu and T. F. Abdelzaher, "Queueing model based network server performance control", 23rd IEEE International Real-Time Systems Symposium, Austin, Texas, December 2002.

[12] L. Kleinrock, Queueing Systems Theory, Volume 1, John Wiely & Sons, January, 1975.

[13] D. Henriksson, Y. Lu, T. Abdelzaher, "Improved prediction for Web server delay control," 16th Euromicro Conference on Real-Time Systems, Catania, Italy, July 2004.

[14] Ying Lu, Tarek F. Abdelzaher, Chenyang Lu, Lui Sha, and Xue Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in Web servers," 9th IEEE Real-Time/Embedded Technology and Applications Symposium, May 2003.

[15] A. Chandra, W. Gong and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements", 11th IEEE International Workshop on Quality of Service, June 2003.

[16] J. Rolia, X. Zhu, M. Arlitt, A. Andrzejak, "Statistical service assurances for applications in utility grid environments", Performance Evaluation Journal, Special Issue on Distributed Systems Performance, edited by S. Majumdar and A. Boukerche, Vol. 58/2-3, pp.319-339, 2004.

[17] J. L. Hellerstein, F. Zhang, P. Shahabuddin, "A statistical approach to predictive detection," Computer Networks, January, 2000.

[18] D. Sheng and J. L. Hellerstein, ``Predictive models for proactive network management: Application to a production Web server," IEEE/IFIP Network Operations and Management Symposium, April 2000.

[19] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, S. M. Weiss, ``Predictive algorithms in the management of computer systems", IBM Systems Journal, Vol. 41, No. 3, 2002.

[20] L. Ljung, System Identification: Theory for the User (2nd Edition), Prentice Hall, 1999.

[21] R. L. Mason, R. F. Gunst, J. L. Hess, Statistical Design and Analysis of Experiments with Applciations to Enginneering and Science, Second Edition, John Wiely & Sons, 2003.

[22] S. Singhal and B.S. Atal, "Amplitude optimization and pitch prediction in multipulse coders," IEEE Trans. On Acoustics, Speech, and Signal Processing, vol. 37, No. 3, March 1989.

[23] D. A. Menasce, L. W. Dowdy, V. A.F. Almeida, Performance By Design : Computer Capacity Planning By Example, Prentice Hall, 2004

[24] K.J. Astrom and B. Wittenmark, Adaptive Control (2nd Edition), Prentice Hall, 1994.

[25] HP Openview Performance Agent, http://www.openview.hp.com/products/performance.

[26] HP-UX Workload Manager, http://h30081.www3.hp.com/products/wlm/index.html

[27] Xen Virtual Machine, http://www.xensource.com/xen/about.html.

[28] Apache Web Server, http://www.apache.org

[29] D. Mosberger and T. Jin, "Httperf --- A tool for measuring Web server performance," Performance Evaluation Journal, Vol. 26, No. 3, pp. 31-37, December 1998.