Efficient Server Provisioning with End-to-End Delay Guarantee on Multi-tier Clusters

Palden Lama and Xiaobo Zhou
Department of Computer Science
University of Colorado at Colorado Springs, CO 80918, USA
Email: {plama, xzhou}@uccs.edu

***Abstract*—Dynamic server provisioning is critical to quality-of-service assurance for multi-tier Internet applications. In this paper, we address three important and challenging problems. First, we propose an efficient server provisioning approach on multi-tier clusters based on an end-to-end resource allocation optimization model. It is to minimize the number of servers allocated to the system while the average end-to-end delay guarantee is satisfied. Second, we design a model-independent fuzzy controller for bounding an important performance metric, the 90_{th}-percentile delay of requests flowing through the multi-tier architecture. Third, to compensate for the latency due to the dynamic addition of servers, we design a self-tuning component that adaptively adjusts the output scaling factor of the fuzzy controller according to the transient behavior of the end-to-end delay. Extensive simulation results, using one representative customer behavior model in a typical three-tier web cluster, demonstrate that the provisioning approach is able to significantly reduce the server utilization compared to an existing representative approach. The approach integrated with the model-independent self-tuning fuzzy controller can efficiently assure the average and the 90_{th}-percentile end-to-end delay guarantees on multi-tier server clusters.**

I. Introduction

Internet applications pose great challenges including scalability and quality-of-service guarantee to the underlying networked systems. Popular Internet applications employ a multi-tier architecture, with each tier provisioning a certain functionality to its preceding tier and making use of the functionality provided by its successor to carry out its part of the overall request processing [2], [3], [6], [7], [11], [12], [14], [15], [18]. For load sharing, various tiers of an Internet application are often replicated and clustered. Today, a typical e-commerce application usually consists of three tiers; a front-end Web tier that is responsible for HTTP request processing, a middle application tier that implements core application functionality say based on Java Enterprise platform, and a backend database that stores product catalogs and user orders. In this context, an incoming user request undergoes HTTP processing, application server processing, and triggers queries or transactions at the database. Note that the database tier may or may not be replicated on-demand depending if the database employs a shared or shared-nothing architecture [2], [14].

Dynamic server provisioning is critical to quality-of-service assurance for Internet applications. The problem was well studied in the context of single-tier servers [4], [13], [19]. It is however non-trivial or even infeasible to extend the mechanisms designed for a single-tier architecture to a multi-tier architecture. The challenges include the inter-tier interaction, concurrency limits and cross-tier performance dependencies [3]. For example, adding servers to one tier does not necessarily increase the effective system performance due to cross-tier performance dependencies [14]. End-to-end system delay is the major performance metric of multi-tier Internet applications. It is the response time of a request that flows through a multi-tier computer system [14], [17] (we use the terms "delay" and "response time" interchangeably). In this paper, we address three important but challenging problems. The first is what is an efficient server provisioning scheme with end-to-end delay guarantee in multi-tier server clusters. The second is how to bound the 90_{th}-percentile end-to-end delay. The third is how to reduce the delay oscillation effect due to the dynamic addition of servers.

Recently, an important dynamic provisioning approach was proposed in [14] for multi-tier Internet applications. Figure 1 illustrates an example on a typical three-tier server cluster. The approach decomposes an end-to-end delay guarantee into the per-tier average delay targets (i.e., d_1, d_2, and d_3). Then per-tier server provisioning is conducted based on a queueing model to meet the per-tier delay target. Its key problem, however, is on how to determine those decomposition percentages, while the dynamic behavior of an Internet application shifts the performance bottleneck from one tier to another. The important performance metric is the end-to-end system delay, not the per-tier delay. In this paper, we propose an efficient server provisioning scheme that aims to minimize the total number of servers allocated to a multi-tier cluster while the end-to-end delay guarantee is satisfied.

We next address the second important problem, that is, how to bound the 90_{th}-percentile end-to-end delay of requests flowing through the multi-tier service. People envision that the metric of the 90_{th}-percentile delay, compared to the average delay, has the benefit that is both easy to reason about and to capture the user's perception of Internet service performance [17]. The queueing model based approaches in [3], [15] can monitor the average delay of requests, but have no control on the 90_{th}-percentile delay of requests. The work in [14] proposed an innovative approach for assuring the 95_{th}-percentile delay guarantee (applicable to the 90_{th}-percentile delay guarantee). It uses an application profiling technique to determine a service time distribution whose 95_{th}-percentile

978-1-4244-3876-1/09/$25.00 ©2009 IEEE

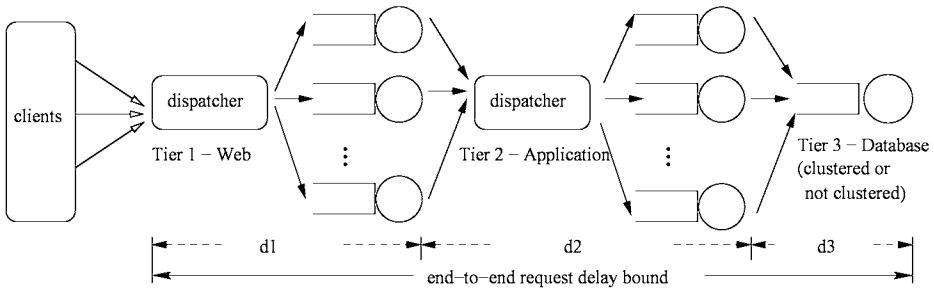


Fig. 1. End-to-end delay guarantee on a multi-tier server cluster.

is the delay bound. The mean of that distribution is used as the average end-to-end delay constraint. It then applies the constraint for the per-tier delay target decomposition and per-tier server provisioning based on a queueing model. There are two key problems, however. One is that the approach is model dependent. The second is that the application profiling has to be done each time before the server replication and allocation, which could be time consuming and complex due to the dynamic nature of Internet applications. We use a model-independent fuzzy controller to bound the 90_{th}-percentile delay of requests on a multi-tier server architecture. Fuzzy control has been recently applied for performance guarantee on single-tier Internet servers [8], [16]. We apply it for end-to-end system delay guarantee on multi-tier architecture, together with a resource allocation optimization model.

Furthermore, we consider a practicability issue with the new server provisioning approach. Addition (or switching) of a server to a tier introduces non-negligible latency to a multi-tier service [2], which will affect the perceived end-to-end delay of users. The system instability also occurs due to the time spent by the newly added server adapting to the existing system. For example, an addition of database replica goes through a data migration and system stabilization phase [2], during which the delay will be higher than expected. Provisioning of servers during an adaptation phase will cause oscillations in server allocation. We design a self-tuning control component to reduce potential oscillation due to server additions. The latency introduced by addition of a server is considered as a process delay. To compensate for the process delay, the self-tuning component adaptively adjusts the output scaling factor according to the transient behavior of the end-to-end delay. The scaling factor is chosen due to its significant influence on the performance and stability of the controller system.

We build a simulation model for a typical three-tier web cluster and conduct extensive simulations to evaluate the new server provisioning approach. We use a representative session-based customer behavior model from the work in [10]. Experimental results demonstrate that the new approach is able to significantly reduce the server utilization compared to the representative server provisioning approach designed in work [14]. Integrated with the model-independent self-tuning fuzzy controller, it can effectively and efficiently assure the average and the 90_{th}-percentile end-to-end delay guarantees on a multi-tier cluster. Results also show that the use of the self-tuning component can provide fine granularity fuzzy control in end-to-end delay, great efficiency in resource utilization, and fast convergence to the steady state.

The major contributions of our work lie in the modeling and analysis of the efficient server provisioning scheme on multi-tier server clusters, the design and implementation of a model-independent self-tuning control system, and the evaluation and conveyed insights about provisioning end-to-end delay guarantees by the server provisioning approach with fuzzy control for multi-tier Internet services.

The structure of this paper is as follows. Section II reviews related work. Section III presents the optimization modeling and analysis. Section IV gives the design of a model-independent self-tuning fuzzy controller. Section V is on the performance evaluation. Section VI concludes the paper.

II. Related Work

Resource management for quality-of-service provisioning in multi-tier Internet applications is a very important and active research topic. A few studies focused on the modeling and analysis of multi-tier servers with queueing foundations. For instance, Diao *et al.* described a performance model for differentiated services of multi-tier applications [3]. Per-tier concurrency limits and cross-tier interactions were addressed in the model based on a $M/M/1$ queueing model. The work in [11] designed a coordinated admission control approach based on a machine learning technique.

The work in [15] studied an optimization for allocating servers in the application tier that increase a server provider's profits. That single-tier provisioning method does not consider the end-to-end delay constraint. Recently, Urgaonkar et al. designed an innovative dynamic provisioning technique on multi-tier server clusters [14]. It sets the per-tier average delay targets to be certain percentages of the end-to-end delay constraint. Based on a queueing model, per-tier server provisioning is executed at once for the per-tier delay guarantees. The work provides important insights on dynamic provisioning for multi-tier clusters. There is however no guidance nor optimization about the decomposition of end-to-end delay to per-tier delay targets. It relies on a queueing model with application profiling for the 95_{th}-percentile delay guarantee. Our work aims to find an efficient server provisioning scheme based on an end-to-end resource allocation optimization model. We further design a

model-independent self-tuning fuzzy controller to address the lack of an accurate workload model.

Feedback control was used for service differentiation and performance guarantee on Internet servers [1], [6], [9], [16]. Linear control techniques were applied to control the resource allocation in single-tier Web servers [1]. However, the performance of the linear feedback control is often limited [16]. Recent work applied adaptive control for performance guarantee [6]. For instance, a multi-tier e-commerce application was modeled as one $M/G/1$ server [6]. A proportional integral (PI) controller based admission control proxy was developed to maintain the end-to-end response time target. However, using the average response time as the performance metric is unable to represent the shape of a response time curve. For the end-to-end delay guarantee, the inherent process delay needs to be considered and addressed as well.

Fuzzy control was applied for performance guarantee. In [8], fuzzy control was used to determine an optimal number of concurrent child processes to improve the aggregated server performance. In [16], a fuzzy controller was used for provisioning guarantee of user-perceived response time of a web page. It demonstrated that due to the model independence, the approach significantly outperforms linear PI controllers. The work was done on a single server. We use fuzzy control for dynamic server provisioning with end-to-end delay guarantee in a multi-tier server architecture, together with a resource allocation optimization model. We also consider the use of the self-tuning capability for fine-granularity control of the system performance.

III. An Efficient Server Provisioning Approach

We propose a resource allocation optimization based server provisioning scheme that minimizes the total number of servers allocated to a multi-tier cluster while the end-to-end delay guarantee is satisfied. The basic idea is to divide the provisioning process into a sequence of intervals. In each interval, based on the measured resource utilization, end-to-end delay, and the predicted workload, the servers are dynamically allocated to the tiers at once.

Popular multi-tier Internet applications are session based [14], [20]. Consider sessions arrive at a rate λ to a n-tier server cluster. Let v_i denote the average number of visits of a session to tier i. The request arrival rate to tier i becomes λv_i. A request in different tiers usually demands different processing resource usages [10], [20], [14]. Let r_i be the normalized resource demand of a request in tier i. Let m_i be the number of servers allocated to tier i in the current interval. With a load balancer, the workload at a tier is shared by the allocated homogeneous servers. Let ρ_i be the resource utilization of tier i, $\rho_i = \lambda v_i r_i / m_i$. The requests will traverse multiple tiers. Like many works in [3], [15], [20], we assume that request arrivals to a tier meet a Poisson process. We model the workload at each tier by an $M/G/1$ queueing system. Let X_i be the service time distribution at tier i.

Let d_i denote the average delay of a request in tier i. According to Pollaczek-Khinchin formula of the queueing theory, we have $d_i = E[W_i] + E[X_i] = \frac{\rho_i E[X_i^2]}{2r_i(1-\rho_i)} + E[X_i]$, where $E[W_i]$ is the expected queueing delay at tier i, $E[X_i]$ and $E[X_i^2]$ are the first moment and second moment of the service time distribution X_i at tier i, respectively.

We formulate the dynamic server provisioning with the end-to-end delay guarantee as the optimization problem:

$$\text{Minimize} \quad \sum_{i=1}^{n} m_i \tag{1}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} d_i = U - \bar{U} \tag{2}$$

$$d_i = \frac{\rho_i E[X_i^2]}{2r_i(1-\rho_i)} + E[X_i] \tag{3}$$

$$m_i = \frac{\lambda v_i r_i}{\rho_i} \tag{4}$$

$$0 \le \rho_i < 1. \tag{5}$$

Eq. (1) gives the optimization objective. It is to minimize the total number of servers allocated to the multi-tier server cluster. We assume the servers are homogeneous with the same cost and each tier is replicable on-demand [2]. If one wants to differentiate the deployment costs at different tiers, the objective function can be a weighted sum. Eq. (2) describes that the average end-to-end delay of a request is bounded. $\bar{U}$ is an important parameter used for end-to-end delay guarantee. If it equals to ϵ, the constraint is that the average end-to-end delay of a request be less than the bound U. We later use it with a model-independent fuzzy controller for the 90_{th}-percentile end-to-end delay guarantee. Eq. (3) describes the predicted average delay of a request at a tier. Eq. (4) describes the relationship between the server allocation and the utilization of each tier. It assumes a load balancer which can equally distribute the workload to each server of a tier. Eq. (5) describes the resource allocation constraint.

The equations (1) and (4) lead to the objective function

$$L(\rho_1, \cdots, \rho_n) = \sum_{i=1}^{n} \frac{\lambda v_i r_i}{\rho_i}. \tag{6}$$

The equations (2) and (3) lead to the constraint function

$$C(\rho_1, \cdots, \rho_n) = \sum_{i=1}^{n} \left(\frac{\rho_i E(X_i^2)}{2r_i(1-\rho_i)} + E[X_i]\right). \tag{7}$$

Let ν be the Lagrange multiplier. The optimal solution to (1) occurs when the first order derivatives of the Lagrangian objective function $L(\rho_1, \cdots, \rho_n)$ over variables ρ_i are equivalent to the first order derivatives of the constraint function $C(\rho_1, \cdots, \rho_n)$ over variable ρ_i multiplied by ν. That is

$$\frac{\partial L(\rho_1, \cdots, \rho_n)}{\partial \rho_i} = \nu \frac{\partial C(\rho_1, \cdots, \rho_n)}{\partial \rho_i}. \tag{8}$$

Deriving ν and using it with (4) leads to the solution

$$\rho_i = \left(1 + \frac{\sum_{i=1}^{n} \sqrt{\lambda v_i E[X_i^2]}}{U - \bar{U} - \sum_{i=1}^{n} E[X_i]} \frac{E[X_i^2]}{2\lambda v_i r_i}\right)^{-1} \tag{9}$$

$$m_i = \lambda v_i r_i + \frac{\sum_{i=1}^{n} \sqrt{\lambda v_i E[X_i^2]}}{U - \bar{U} - \sum_{i=1}^{n} E[X_i]} \frac{\sqrt{\lambda v_i E[X_i^2]}}{2}. \tag{10}$$

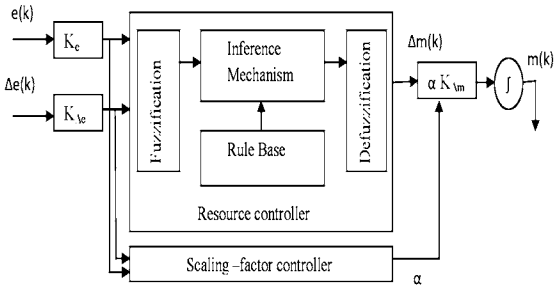


Fig. 2. A self-tuning fuzzy controller.

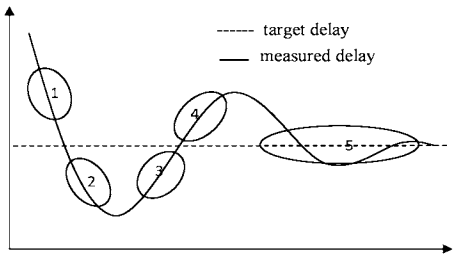


Fig. 3. Fuzzy control effect.

"Δm(k)"		"Δe(k)"						
		NL	NM	NS	ZE	PS	PM	PL
"e(k)"	NL	PH	PH	PH	PH	PL	PS	ZE
	NM	PH	PH	PH	PL	PS	ZE	NS
	NS	PH	PH	PL	PS	ZE	NS	NL
	ZE	PH	PL	PS	ZE	NS	NL	NH
	PS	PL	PS	ZE	NS	NL	NH	NH
	PM	PS	ZE	NS	NL	NH	NH	NH
	PL	ZE	NS	NL	NH	NH	NH	NH

Fig. 4. The fuzzy rule base.

Internet workload measurements indicate that a heavy-tailed distribution is often an accurate model for service time distribution for many Web applications. Like work in [5], [19], we use a bounded pareto distribution for modeling a heavy-tailed distribution. Our work in [19] found closed-form expressions of $E[X]$ and $E[X_i^2]$ for a bounded pareto distribution, which are used in the optimization model.

IV. A Self-tuning Fuzzy Controller

The server provisioning scheme based on the optimization however is model dependent. We enhance it with a self-tuning fuzzy controller that is model independent. The fuzzy controller determines the number of servers to be allocated to each tier at once without relying on an accurate workload model. It is to guarantee the average end-to-end delay on a multi-tier system, to bound the 90_{th}-percentile end-to-end delay, and to integrate the fuzzy controller with the optimization model. To provide fine granularity control on the delay and the resource utilization efficiency, we consider the use of uniform membership functions in the fuzzy controller. The self-tuning capability is to compensate for the process delay due to the addition of a server to a tier.

A. The architecture of the fuzzy controller

Figure 2 illustrates the block diagram of the self-tuning fuzzy controller. The controller has two inputs; $e(k)$ is the difference between the measured value and target value of the end-to-end delay in the $(k)th$ sampling period, and $\triangle e(k)$ is the change in error. The output of the controller is the resource adjustment $\triangle m(k)$ for the next sampling period. The scaling factors K_e, $K_{\triangle}e$ and $\alpha K_{\triangle}m$ are used to tune the controller's performance. The output scaling factor α is automatically adjusted by the scaling factor controller. Thus, the total number of servers allocated to the multi-tier clusters during the $(k+1)th$ sampling period is

$$m(k+1) = m(k) + \alpha K_{\triangle m} \triangle m(k) = \int \alpha K_{\triangle m} \triangle m(k) dk. \quad (11)$$

The fuzzy controller consists of four components. The rule base is the core component. It contains a set of rules based on which fuzzy control decisions are made. The fuzzification interface converts numeric values of controller inputs into equivalent fuzzy values. It determines the certainties of fuzzy values based on input membership functions. The inference component applies pre-defined rules according to the fuzzified inputs and generates fuzzy conclusions. The defuzzification interface combines fuzzy conclusions and converts them to a single output, i.e., the resource allocation adjustment in a numeric value.

B. The fuzzy rule base

Designing the rule base for a fuzzy controller is based on heuristic control knowledge. The rules are defined using linguistic variables "$e(k)$", "$\triangle e(k)$" and "$\triangle m(k)$" corresponding to the numeric values of control inputs and outputs. The linguistic variables "$e(k)$" and "$\triangle e(k)$" have linguistic values NL, NM, NS, ZE, PS, PM, and PL. The linguistic variable "$\triangle m(k)$" has linguistic values NH, NL, NM, NS, ZE, PS, PM, PL, and PH. The rules are in the form of If-Then statements. For example, *If error "$e(k)$" is NL and change in error "$\triangle e(k)$" is PL, then the resource adjustment $\triangle m(k)$ is ZE.* To design the fuzzy control rules, we analyzed the behavior of end-to-end delay due to changes in resource allocation. We identified five zones of the end-to-end delay as shown in Figure 3. Since $e(k)$ and $\triangle e(k)$ have opposite signs in zones 1 and 3, the error is self-correcting. If $e(k)$ is small, $\triangle m(k)$ needs to be adjusted to slow down the current trend so as to avoid any overshoot. Whereas, if $e(k)$ is large, $\triangle m(k)$ needs to be adjusted to speed up the current trend. In zones 2 and 4, $e(k)$ and $\triangle e(k)$ have the same sign. That is, the measured end-to-end delay is moving away from the target value. Therefore, $\triangle m(k)$ should be adjusted to reverse the current trend. Zone 5 indicates the steady state since $e(k)$ and $\triangle e(k)$ have small magnitudes. In this case, $\triangle m(k)$ should be adjusted to maintain the current state and to correct steady state errors. The control rules designed for each of the analyzed zones are illustrated in Figure 4.

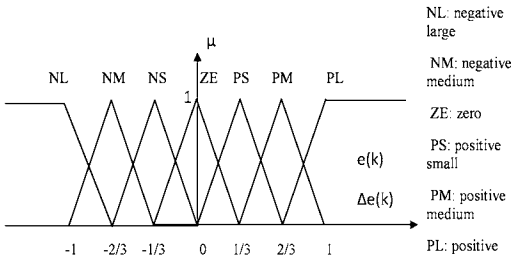


Fig. 5. Membership functions for control inputs.

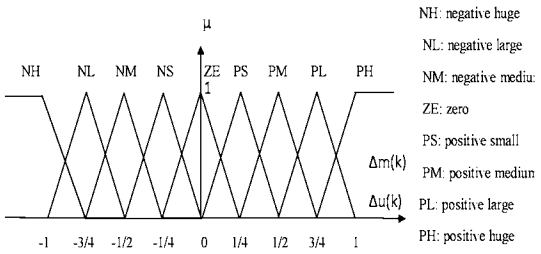


Fig. 6. Membership functions for control outputs.

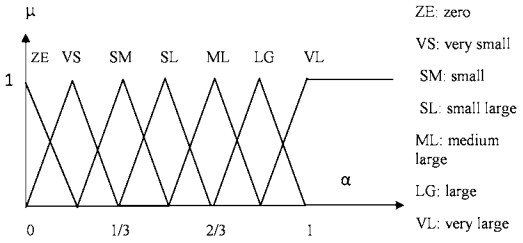


Fig. 7. The membership function for α.

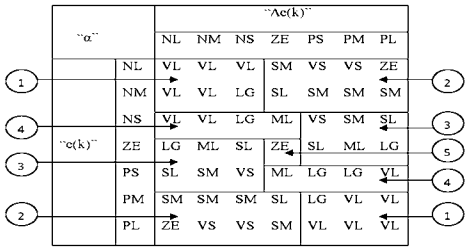


Fig. 8. The fuzzy rule base for α.

C. Fuzzification, inference, defuzzification

Fuzzification is the process of converting the numeric input values into corresponding linguistic values and calculating the certainty of those linguistic values. The effective inputs to fuzzy controller are $e(k)$ multiplied by input scaling factor K_e and $\triangle e(k)$ multiplied by input scaling factor $K_{\triangle e}$. The linguistic values are represented by membership functions. As work in [16], we choose triangle membership functions due to the simplicity and wide usage. We consider both uniform and non-uniform membership functions. The membership functions for both scaled inputs and output are defined within the common interval [-1,1]. The values of the inputs $e(k)$ and $\triangle e(k)$ are mapped into [-1,1] by the input scaling factors K_e and $K_{\triangle e}$, respectively. The output value $\triangle m(k)$ is mapped into [-1,1] by the output scaling factor $\alpha \cdot K_{\triangle m}$.

Figure 5 shows uniform membership functions for fuzzy control inputs $e(k)$ and $\triangle e(k)$. The fuzzification process assigns linguistic values to an input and determines their certainties by using input membership functions. The certainty of linguistic value m assigned to an input is denoted by $\mu(m)$. For example, if $e(k)$ is 1/3, the linguistic variable "$e(k)$" is assigned a value PS and $\mu(PS)$ is 1. If $\triangle e(k)$ is 1/6,"$\triangle e(k)$" is assigned values ZE and PS with certainties $\mu(ZE)$ and $\mu(PS)$ as 0.5. Based on the fuzzified inputs, the inference mechanism determines which rules should be applied to reach fuzzy conclusions. Let $\mu(m,n)$ denote the premise certainty of $rule(m,n)$ where m and n are membership functions. Following the standard max-min inference mechanism, the rules to be activated are defined as set of $rule(m,n)$ such that $\mu(m,n) > 0$, where $\mu(m,n) = \min(\mu(m),\mu(n))$. For example, if $e(k)$ = 1/3 and $\triangle e(k)$ = 1/6, the certainties of rule(PS,ZE) and rule(PS,PS) are $\mu(PS,ZE) = 0.5$ and $\mu(PS,PS) = 0.5$, respectively.

The defuzzification component combines the rules activated by the inference mechanism using the "center average" method and calculates the fuzzy controller output. Let $b(m,n)$ denote the center of the membership function of the result of $rule(m,n)$. The fuzzy control output is calculated as $\triangle m(k) = \frac{\sum_{m,n} b(m,n)\cdot\mu(m,n)}{\sum_{m,n}\mu(m,n)}$.

The membership function of fuzzy control output determines the value of $b(m,n)$. As shown in Figure 6, we use a large number of uniform membership functions for the fuzzy control output $\triangle m(k)$ to increase the flexibility.

D. A scaling factor controller for self-tuning

One major challenge in controlling a physical process is its inherent process delay [16]. For the dynamic server provisioning process, it is the latency between allocating servers and accurately measuring the effect of the server provisioning on the end-to-end delay. To compensate for the process delay, we design a scaling-factor controller which adaptively adjusts the output scaling factor $\alpha \cdot K_{\triangle m}$ according to the transient behavior of the end-to-end delay. The output scaling factor is chosen due to its significant impact on the performance stability of the system.

The scaling factor controller works in similar way as the fuzzy controller. The output of the controller is the gain updating factor α. The membership function of α is defined within the interval [0,1], as shown in Figure 7. Figure 8 shows the rule base for the scaling-factor controller. It is designed in association with that of the fuzzy controller. A few important considerations for the rule design are as follows:

1) When the error is large but has the same sign as the change in error, α should be made very large to prevent from further worsening the situation.
2) To avoid large overshoot and reduce the settling time, α is set at a small value when the error is big but has

the opposite sign as compared to the change in error. If the process delay is high, the controller may not achieve expected output after allocating required number of servers, and hence, tries to overreact by assigning too many servers in the next sampling period. This is compensated by adjusting the output scaling factor to a small value.

3) When the error is small, there should be a wide variation of the gain, depending on the process trend to avoid large overshoot and undershoot. This will avoid an excessive oscillation.
4) To improve the controller performance under load disturbance, α should be sufficiently large around the stead state.
5) At a steady state, when the error is small and the change in error is also small, α should be very small to avoid oscillation problem around the equilibrium point.

E. Stability analysis

A system is said to be stable if it would come to its equilibrium state after any external input, initial conditions or disturbances that have impressed the system. We analyze the stability of the self-tuning fuzzy control system by using Lyapunov's direct method. It is a time domain method suitable for analyzing the stability of a non-linear system.

We follow an approach similar to [16] for finding a suitable Lyapunov function. We first define the difference between the equilibrium resource value and current one as

$$\tilde{m}(k) = M(k) - m(k). \tag{12}$$

Equilibrium value refers to the resource allocation value for which the end-to-end delay reaches or is close to the target value. From (11) and (12), we get

$$\tilde{m}(k+1) = \tilde{m}(k) - \alpha K_{\triangle m} \triangle m(k). \tag{13}$$

We choose $V(\tilde{m}(k)) = \tilde{m}^2(k)$ as the Lyapunov function in the discrete time. It gives the distance of the allocated resources from its equilibrium value. By applying the Lyapunov stability theorem, we find the stability constraint as follows,

$$|\triangle m(k)| < \frac{2\epsilon}{\alpha K_{\triangle m}}. \tag{14}$$

The stability constraint is intuitive as it suggests that increasing the output scaling factor reduce the stability of the system.

F. Integration with the optimization

We have designed a model-independent fuzzy controller for 90_{th}-percentile end-to-end delay guarantee in multi-tier server clusters. However, the server allocation with the fuzzy controller does not promise the efficiency of the end-to-end resource provisioning. We integrate the fuzzy controller with the optimization model. The integrated approach uses the same fuzzy controller, except that the output of the controller is the adjustment value of the parameter $\bar{U}$ used in the optimization model Eq.(10). Increasing $\bar{U}$ reduces the value of target response time, thus increases the resource allocation and vice versa. The advantage of controlling $\bar{U}$ is that the number of servers allocated to each tier is based on the optimization in each sampling interval of the fuzzy control process.

TABLE I
THE CHARACTERISTICS OF WORKLOAD A AND B.

Workload A	WebTier	AppTier	DBTier
$E[X_i]$	24.163 ms	48.709 ms	34.403 ms
$E[X_i^2]$	591.864	2667.88	1191.77
Workload B	**WebTier**	**AppTier**	**DBTier**
$E[X_i]$	13.593 ms	67.962 ms	44.536 ms
$E[X_i^2]$	192.226	4805.66	1991.71

V. PERFORMANCE EVALUATION

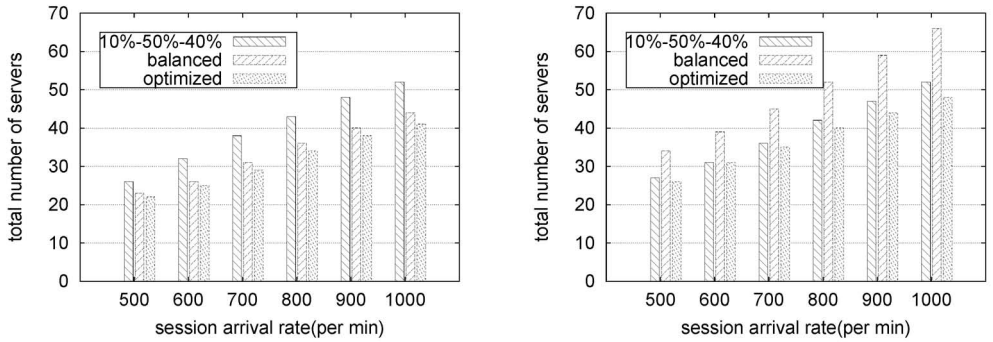
We evaluate the server provisioning approach based on the optimization model alone, the model-independent fuzzy control system, and the integrated approach in a typical three-tier server cluster with extensive simulations. As others in [2], we assume that the database tier can be replicated on-demand as it employs a shared architecture. As our previous work in [20], we use a synthetic session-based workload generator derived from a customer behavior model in [10]. It allows us to perform sensitivity analysis in a flexible way. A session generator produces head requests that initiate sessions. The subsequent requests of a session are generated according to the customer behavior model. The think time was generated by an exponential distribution with a mean of 5 seconds.

As work in [5], [19], we use bounded pareto distributions for modeling the service time distribution. We adopt two such sets of characteristics as shown in Table I. During the simulation, the end-to-end delay was measured periodically with a sampling interval of 4 minutes. Each representative result reported in the following is an average of 100 runs.

A. Impact of the provisioning optimization

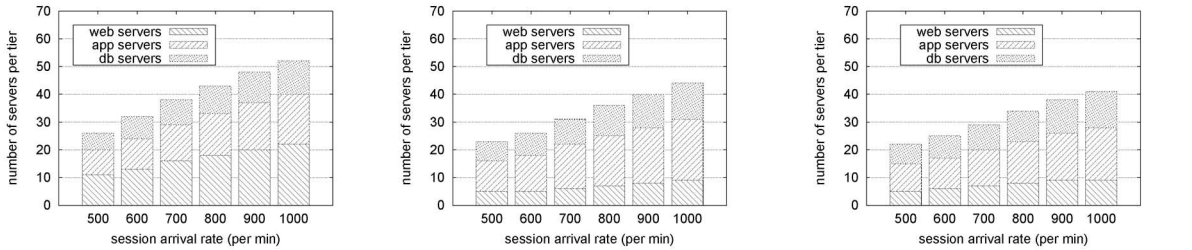
The first experiment is to demonstrate that the optimization-based provisioning approach is able to significantly reduce the overall server usage while the end-to-end delay guarantee is still satisfied. For comparison, we experimented the per-tier decomposition based approach in [14] that sets the per-tier average delay targets to be 10%, 50%, and 40% of the end-to-end delay target, respectively. We also experimented a balanced decomposition based approach that sets each tier's average delay target to be 1/3 of the end-to-end delay target. In this experiment, we set the target average delay to be 300 msec and 500 msec for the characteristics A and B, respectively.

Figure 9 shows the total number of servers required by three approaches for the average end-to-end delay guarantee at varying session arrival rate. It shows that the optimization-based approach uses the minimum number of servers for the delay guarantee. It also shows that the impact of the delay decomposition on the server usage is dependent on the workload characteristics. Figure 9(a) shows that by using the characteristic A, the balanced approach uses less servers than the 10%-50%-40% approach. Figure 9(b) shows that by using the characteristic B, the 10%-50%-40% approach uses less servers than the balanced approach. Overall, the experimental results show that the optimization-based approach can reduce the total number of servers allocated by about 20% compared



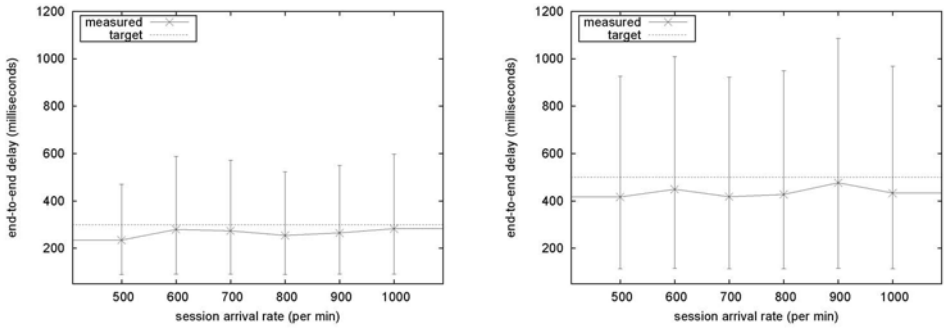
(a) By using the workload characteristic A. (b) By using the workload characteristic B.

Fig. 9. Impact of three server provisioning approaches on the server usage.



(a) The 10%-50%-40% approach. (b) The balanced approach. (c) The optimization-based approach.

Fig. 10. The server allocation at each tier by three server provisioning approaches using the workload characteristic A.



(a) With the workload A. (b) With the workload B.

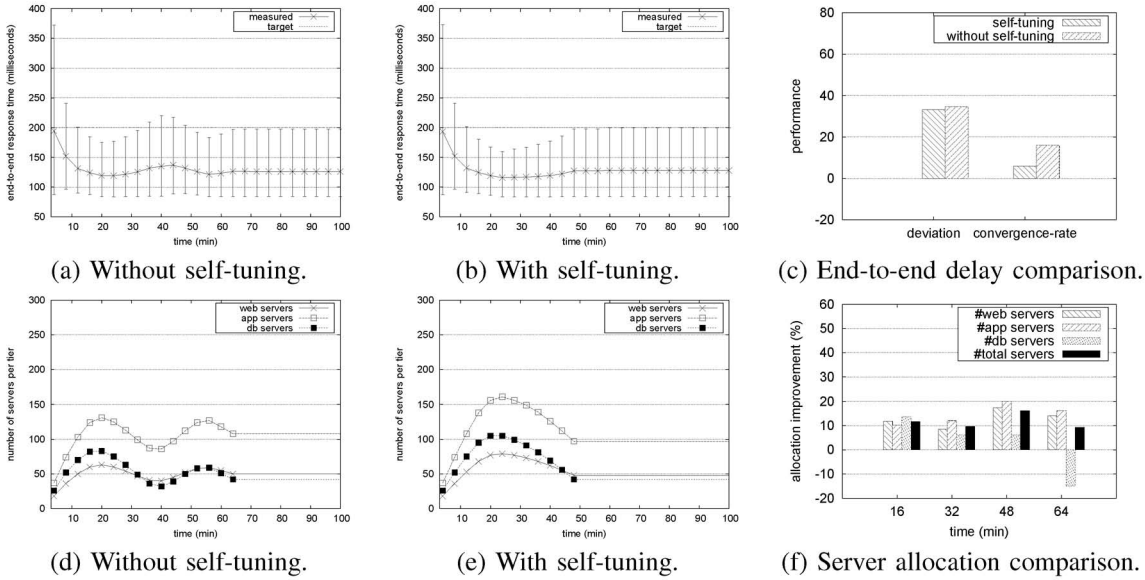
Fig. 11. End-to-end delay due to the optimization-based allocation.

to the 10%-50%-40% decomposition approach, and by about 25% compared to the balanced decomposition approach.

Figure 10 shows the number of servers at each tier required by three approaches for the average end-to-end delay guarantee due to the use of characteristic A. Results of characteristic B are omitted due to the space limitation. Figure 11 shows the average end-to-end delay with its 90_{th} and 10_{th} percentiles at varying session arrival rate due to the use of the optimization-based approach. It shows while the average end-to-end delay is within the target, there is no control on the 90_{th}-percentile end-to-end delay.

B. Impact of the self-tuning controller

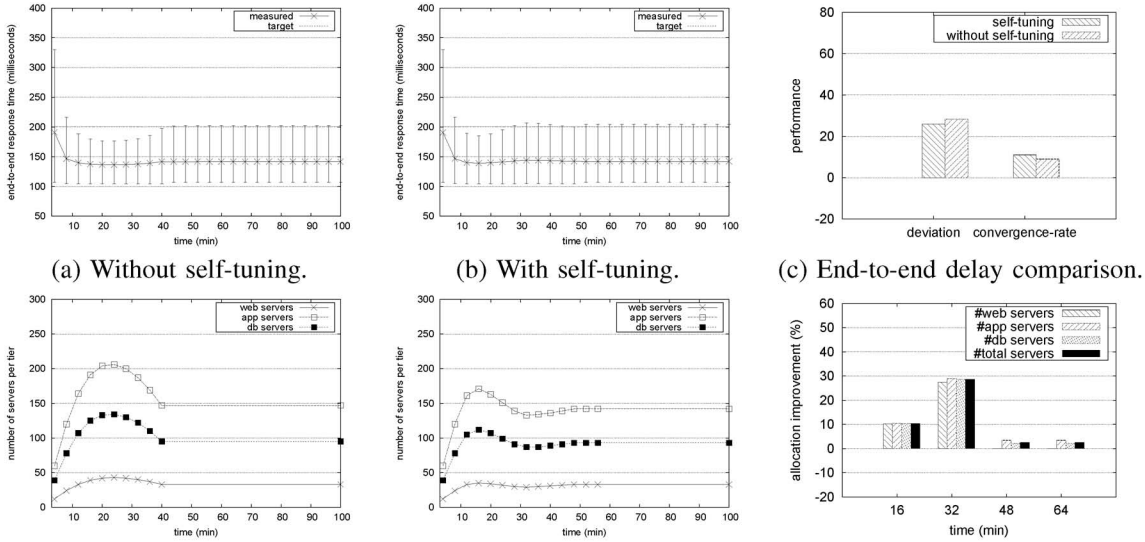
To evaluate the impact of the self-tuning capability on the controller performance, we simulate the instability of end-to-end delay during the addition of servers in a tier. Figure 12 shows the performance difference of the fuzzy control system with and without the self-tuning capability with workload characteristic A. The simulation was performed with a workload of 1600 sessions per minute, the characteristic A and the target 90_{th}-percentile end-to-end delay of 200 msec. From 12(a) and (b), we observe that the target end-to-end delay is reached much earlier by the use of the self-tuning fuzzy controller that reduces unnecessary oscillations during the server allocation processes. It leads to less number of server switchings as compared to the fuzzy control system without the self-tuning capability. Figure 12(c) shows that the self-tuning capability can significantly speed up the convergence rate for the end-to-end delay guarantee and slightly reduce the delay deviation. Moreover, the total number of servers allocated at the steady state is less by the use of the self-tuning fuzzy controller, as shown in Figures 12(d)-(f). It is due to the fact that the



(a) Without self-tuning. (b) With self-tuning. (c) End-to-end delay comparison.

(d) Without self-tuning. (e) With self-tuning. (f) Server allocation comparison.

Fig. 12. Impact of the self-tuning controller on the system performance workload characteristic A. (a-c) Impact on the end-to-end delay deviation and convergence rate; (d-e) Impact on the number of servers allocated to the multi-tier system.



(a) Without self-tuning. (b) With self-tuning. (c) End-to-end delay comparison.

(d) Without self-tuning. (e) With self-tuning. (f) Server allocation comparison.

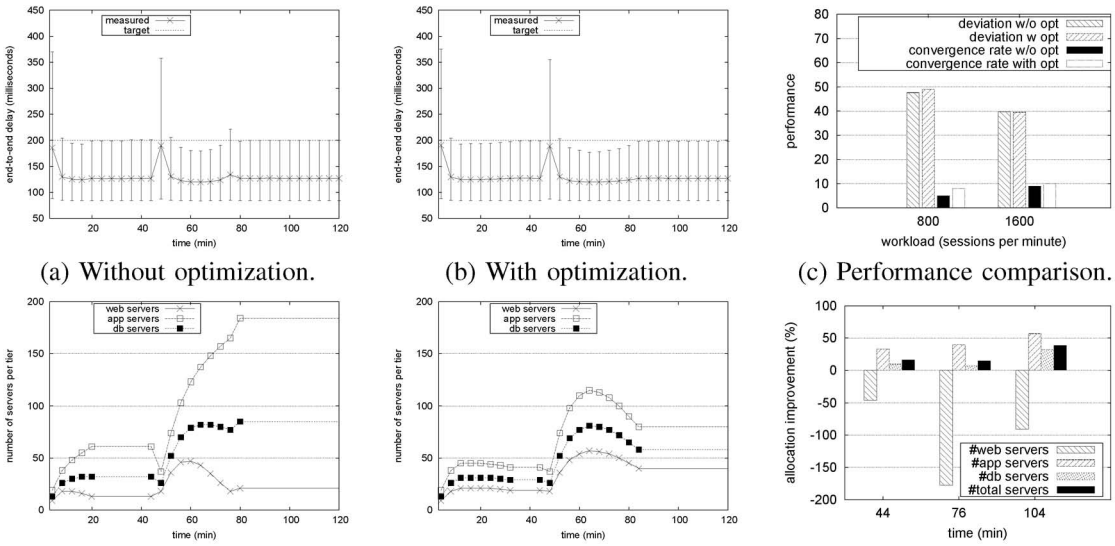
Fig. 13. Impact of the self-tuning controller on the system performance workload characteristic B. (a-c) Impact on the end-to-end delay deviation and convergence rate; (d-e) Impact on the number of servers allocated to the multi-tier system.

control process is more finely and adaptively tuned with the self-tuning capability. Note that the impact of the self-tuning capability on the server usage at different tiers is different. Similar results were obtained for workload characteristic B, as shown in Figure 13.

C. Impact of optimization and fuzzy controller integration

In the previous experiments, the optimization model was applied just once when the initial system workload is measured. We now apply the optimization model at every sampling instance of fuzzy control action. This is achieved by adjusting the controllable parameter $\bar{U}$ as discussed in section IV-F. We study the performance impact of the integrated approach when subjected to a dynamic workload. The session arrival rate changes from 800 sessions per minute to 1600 sessions per minute. The workload characteristic A is used with the target 90_{th}-percentile end-to-end delay of 200 msec.

Figure 14(a) show that the integrated approach has little impact on the end-to-end delay. It has a similar deviation of the end-to-end delay and a similar convergence rate to the steady state. During the interval minutes 44-48, there are spikes in the end-to-end delay due to both approaches. Results show that the self-tuning fuzzy controller is able to quickly adjust the server allocations and assure the 90_{th}-percentile end-to-end delay guarantee. Figures 14(b) show the efficiency impact of the optimization on the number of servers allocated



(a) Without optimization. (b) With optimization. (c) Performance comparison.

(d) Without optimization. (e) With optimization. (f) Performance comparison.

Fig. 14. Impact of the integrated approach on system performance with workload characteristic A. (a-c) on the end-to-end delay deviation and convergence rate; (d-e) on the number of servers allocated to the multi-tier system.

to the multi-tier system. Though the impact on the server usage at different tiers is different, the integrated approach can significantly reduce the total number of servers needed for the end-to-end delay guarantee. We also conducted the experiment with the workload characteristic B. We found similar efficiency impact of the integrated approach.

VI. Conclusion and Future Work

In this paper, we proposed an efficient server provisioning approach based on an end-to-end resource allocation optimization model. Compared to an existing representative approach, it is able to significantly reduce the number of servers allocated for the end-to-end delay guarantee of multi-tier Internet applications. We then designed a model-independent self-tuning controller that is able to provide both average and the 90_{th}-percentile end-to-end delay guarantees under dynamic workloads. The integration of the optimization model and the model-independent fuzzy controller provides superior performance in resource utilization efficiency and end-to-end delay assurance. Our future work will be on the impact of non-uniform membership functions on the delay guarantee and the resource allocation efficiency, and on the implementation and evaluation of the approach in a prototype data center.

Acknowledgement: This research was supported in part by US National Science Foundation grant CNS-0720524 and an internal research grant from the College of EAS at UCCS.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: a control-theoretical approach. *IEEE Trans. on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content Web servers. In *Proc. IEEE ICAC*, 2006.

[3] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaihk, and M. Surendra. Controlling quality of service in multi-tier Web applications. In *Proc. IEEE ICDCS*, 2006.

[4] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat. Model-based resource provisioning in a web service utility. In *Proc. USITS*, 2003.

[5] M. Harchol-Balter. Task assignment with unknown duration. *Journal of ACM*, 29(2):260–288, 2002.

[6] A. Kamra, V. Misra, and E. M. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In *Proc. of IEEE IWQoS*, 2004.

[7] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *Proc. ACM WWW*, 2006.

[8] X. Liu, L. Sha, and Y. Diao. Online response time optimization of apache web server. In *Proc. of IEEE IWQoS*, 2003.

[9] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Trans. on Parallel and Distributed Systems*, 16(6), 2005.

[10] D. A. Menascé, R. Fonseca, V. A. F. Almeida, and M. A. Mendes. Resource management policies for E-commerce servers. *ACM Performance Evaluation*, 27(4):27–35, 2000.

[11] S. Muppala and X. Zhou. CoSAC: Coordinated session-based admission control for multi-tier Internet applications. *Prof. of IEEE ICCCN*, 2009.

[12] J. Rao and C. Xu. Online measurement of the capacity of multi-tier websites using hardware performance counters. In *Proc. IEEE ICDCS*, 2008.

[13] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based Internet services. In *Proc. USENIX OSDI*, 2002.

[14] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems*, 3(1), 2008.

[15] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. on Internet Technology*, 7(1):1–23, 2007.

[16] J. Wei and C.-Z. Xu. eQoS: provisioning of client-perceived end-to-end QoS guarantee in Web servers. *IEEE Trans. on Computers*, 55(12):1543–1556, 2006.

[17] M. Welsh and D. Culler. Adaptive overload control for busy Internet servers. In *Proc. 4th USITS*, 2003.

[18] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier Internet applications. In *Proc. IEEE ICAC*, 2007.

[19] X. Zhou, J. Wei, and C.-Z. Xu. Processing rate allocation for proportional slowdown differentiation on Internet servers. In *Proc. IEEE IPDPS*, 2004.

[20] X. Zhou, J. Wei, and C.-Z. Xu. Resource allocation for session-based two-dimensional service differentiation on e-commerce servers. *IEEE Trans. on Parallel and Distributed Systems*, 17(8):838–850, 2006.