Application-level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications

Wei Wang, Xiang Huang, Xiulei Qin, Wenbo Zhang, Jun Wei, Hua Zhong
Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R. China
{wangwei, huangxiang, qinxiulei08, zhangwenbo, wj, zhongh}@otcaix.iscas.ac.cn

***Abstract*—Performance isolation is a key requirement for application-level multi-tenant sharing hosting environments. It requires knowledge of the resource consumption of the various tenants. It is of great importance not only to be aware of the resource consumption of a tenant's given kind of transaction mix, but also to be able to be aware of the resource consumption of a given transaction type. However, direct measurement of CPU resource consumption requires instrumentation and incurs overhead. Recently, regression analysis has been applied to indirectly approximate resource consumption, but challenges still remain for cases with non-determinism and multicollinearity. In this work, we adapts Kalman filter to estimate CPU consumptions from easily observed data. We also propose techniques to deal with the non-determinism and the multicollinearity issues. Experimental results show that estimation results are in agreement with the corresponding measurements with acceptable estimation errors, especially with appropriately tuned filter settings taken into account. Experiments also demonstrate the utility of the approach in avoiding performance interference and CPU overloading.**

Keywords-performance isolation; multi-tenancy

I. Introduction

Multi-tenancy is a key feature in cloud computing that enables the sharing of resources and costs across a large pool of users, thus allowing for centralization of infrastructure in locations with lower costs. In a multi-tenancy enabled cloud system, user requests from different organizations and companies (tenants) are served concurrently by one or more hosted applications [1]. As illustrated in Figure 1, there are generally three kinds of multi-tenancy deployment approaches: the shared infrastructure approach, the shared middleware approach and the shared application approach.

The shared application approach is a high level multi-tenancy enabling approach which hosts multiple tenants on a single application instance, such as Salesforce [5]. This approach shares more resources compared to the shared infrastructure approach which deploying one OS process per tenant, and the shared middleware approach which deploying one application instance per tenant, and is therefore much more scalable for additional tenants. Other advantages include cost effectiveness because the infrastructure is shared by all tenants and a lower overhead [1].

One of the open questions in the application-level sharing hosting environments is the performance isolation between tenants. Tenants could be untrusted or mutually antagonistic.

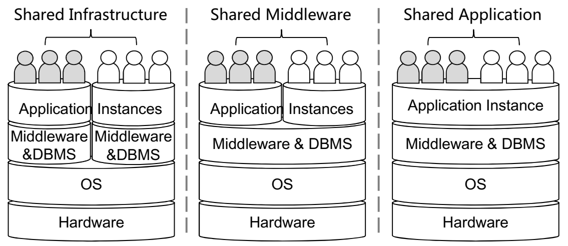


Figure 1. Three kinds of multi-tenancy deployment approaches

Even in workgroup environments where there is more trust between tenants, tenants could misbehave or get overloaded and affect the performance of other tenants. Consequently, this affects infrastructure resources as well as application and services that are hosted on shared resources but need to be made available in multiple isolated instances and need to be protected from possible performance interference caused by other instances. We call this requirement performance isolation. In performance-critical applications such as online trading and e-commerce, failing to meet the performance isolation requirements may result in a loss of customers, serious financial damage, or legal liabilities.

Controlling the resource consumption (such as the CPU and memory) of each tenant is extremely valuable for performance isolation. A key requirement is the knowledge of online resource consumption of the various tenants, thus to quickly detect performance problems and to tune the system according to the workload [10]. We assume that the workload of transactional Web applications is composed of several transaction types. It is of great importance not only to be aware of the resource consumption of a tenant's given kind of transaction mix, but also to be able to be aware of the resource consumption of a given transaction type. We call this requirement application-level resource estimation.

CPU consumption is one of the most challenging resource types to measure [14]. Prevailing approaches to assessing CPU cycles consumed rely on native code libraries, program transformations, or regression techniques. The native code based approach requires application instruments at the source or binary levels and incurs overhead in the request execution path. An alternative is the kernel-based measurement which may proved to be more efficient but still intrusive [2][12][23]. The program transformation based approach measures the CPU consumption by tracking the number of executed bytecode instructions and then transforms this measurement into CPU consumption [14]. This approach is independent of the underlying operating system. However, these transformations incur more than a 30% performance overhead at runtime [15]. Recently, the regression approach has been applied to multi-tiered systems to extract the mean CPU demand of application transactions or services [9]. The regression techniques rely only on high-level measurements such as total CPU usage and transaction throughputs. However, the accuracy of regression techniques suffer if the CPU demands are not deterministic due to resource competition [4]. Furthermore, previous studies also shown that the many regression techniques suffer from the problem of multicollinearity caused by correlated workload flow [3].

In this paper, we propose a novel approach to dynamically estimating application-level CPU consumption of the multi-tenancy Web application. The proposed approach is based on a Kalman filter [16] which estimates hidden CPU consumptions indirectly from easily observed data, in an approximately optimal way. Kalman filter has been used previously to track the CPU utilization of virtualized servers [25] and to estimate the parameters of a queueing model [7]; however, to our knowledge, this is the first time that Kalman filter has been used to track the application-level CPU utilization of multi-tenant applications and to guide their performance isolations. We also present techniques to deal with the non-determinism and the multicollinearity issues.

We have developed a tenant resource estimation engine [8]. The effectiveness of the methodology is illustrated via a detailed set of experimentation using the TPC-W e-commerce suite [17]. The numerical results show the good accuracy of the proposed approach and its competitiveness with respect to regression based approach. We also demonstrate the utility of our approach for fine-grained performance isolation and CPU overload protection. The contribution of this paper can be summarized as follows:

1) We establish the feasibility of dynamically estimating CPU consumption of the application-level sharing multi-tenancy Web applications using a Kalman filter based approach. Our approach is based on monitoring data that are typically available in enterprise production environments and therefore is nonintrusive and incurs much less performance overhead. Meanwhile, experimental results show that, in contrast to regression techniques, the proposed approach does not suffer from the problem of non-determinism and multicollinearity, especially with appropriately tuned filter settings taken into account.

2) We conduct an experimental study using our estimation engine to demonstrate the utility of our approach for the purposes of performance isolation and CPU overload avoidance. Our experiments demonstrate that, in contrast to coarse-grained performance isolation strategy based on the tenant-level resource estimation, the fine-grained strategy based on the application-level resource estimation is more efficient and agile.

The rest of this paper is organized as follows. Section II describes the problem formalization and discusses the challenges. Section III presents our approach. Then, Section IV evaluates the effectiveness of the approach with a TPC-W business service benchmark. Section V discusses related work and Section VI concludes the paper.

II. Problem Statement

In this section, we first formalize the problem of estimating CPU consumption for a multi-tenancy application and then discuss the limitations of using regression techniques to estimate the application-level CPU demands.

A. Formalizing the Problem

During a sequence of monitoring windows, we collect the server log that reflects the number of completed different transactions for each tenant and the total CPU utilization of the server. Assuming that there are totally L tenants and M transaction types processed by the server, we use the following notation:

- T is the length of the monitoring window;
- $N_{i,k,t}$ is the number of completed transactions of the i-th type of the k-th tenant during the t-th monitoring window, where $1 \le i \le M$ and $1 \le k \le L$;
- U_t is the mean CPU utilization of the server during the t-th monitoring window;
- $U_{k,t}$ is the mean CPU utilization of k-th tenant during the t-th monitoring window, where $1 \le k \le L$;
- $D_{i,t}$ is the mean CPU demand of the transactions of the i-th type during the t-th monitoring window, where $1 \le i \le M$;

From the utilization law [20], one can easily obtain (1.a) and (1.b) for each monitoring window.

$$U_{k,t} = \sum_i N_{i,k,t} D_{i,t} \quad (1.a)$$

$$U_t T = \sum_k U_{k,t} \quad (1.b)$$

Because it is almost infeasible to get an accurate CPU demand $D_{i,t}$, let $C_{i,t}$ denote the approximated value of $D_{i,t}$. Then, an approximated utilization U'_t can be calculated as

$$U'_t = \frac{\sum_k \sum_i N_{i,k,t} C_{i,t}}{T} \quad (1.c)$$

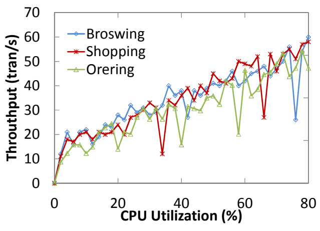
To solve for $C_{i,t}$, one can choose a statistical analysis method from a variety of known methods in the literature. In the present paper, we are interested in the estimation error between the estimated CPU demand $C_{i,t}$ and the measured CPU demand $D_{i,t}$.

B. Limitations of Using Regression Techniques

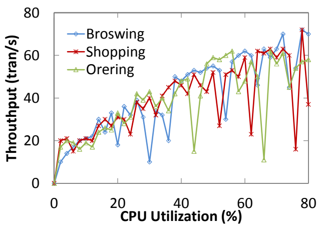
A multiple linear regression technique can be used to solve for $C_{i,t}$. For example, Zhang et al. apply a nonnegative least squares regression (Nonnegative LSQ) technique to derive the mean CPU demand for application transactions. For each monitoring window T, it monitors the $N_{i,k,t}$ and U_t and obtains a set of equations in a form of (1.b). Then, using the non-negative LSQ, one can solve this set of equations for $C_{i,t}$ and compute the approximated U'_t .This nonnegative LSQ regression minimizes the error $\epsilon = \sqrt{\sum (U'_t - U_t)^2}$, such that $C_{i,t} \ge 0$.

However, there are two reasons that forced us to choose a different approach.

First, the existence of anomalies in the resource demand caused by uncontrolled operations makes regression solutions difficult. For example, the industry-standard



(a) Middleware server

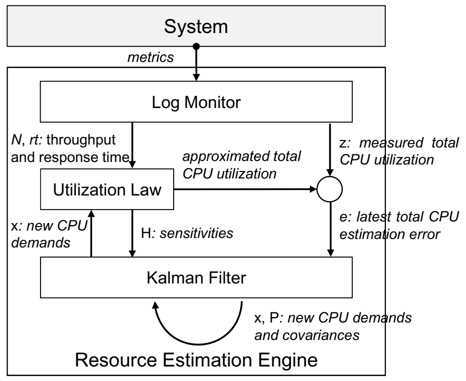


(b) Database server

Figure 2. The TPC-W *Home* transaction throughput profiles

benchmark for e-commerce sites TPC-W [17] includes 14 types of transactions, all of which access an online bookstore and execute a shopping action such as browsing, query, or order. Figure 2(a), (b) illustrate the examples of throughput profiles of the TPC-W *Home* transaction at the middleware server and the database server. The three curves correspond to the three workload types in TPC-W. For each 1-minute monitoring window, we generate the throughput profile by classifying the throughput into the corresponding CPU utilization. For example, if during the current monitoring window there are N_k transactions of type k completed under observed CPU utilization of 10%, then N_k goes in the CPU utilization U_{10}. Finally, for each CPU utilization U_r, where $r \in \{1, 2, ..., 100\}$, we compute the overall transaction throughput $N_{k,r}$. In such a way, we create a throughput profile in the following format $[U_r, N_{k,r}]$. As shown in Figure 2(a), (b), the transaction throughput profiles are similar under the three workloads. However, there are many "outliers" in these curves, especially at the database server. Typically, the "outliers" correspond to some anomalous CPU utilization with few transaction occurrences due to resource competition or lock. As a result, these non-deterministic outliers could significantly affect the accuracy of the regression solution, as regression generally aims to minimize the absolute error across all points, while these outliers are non-representative points.

Second, many regression techniques suffer from the problem of multicollinearity [3] which can lead to unreliable predictions for demands. The workloads of the Web applications are composed of sessions. A session is defined as a sequence of transactions of different types made by a single customer during a single visit to the applications. It is quite common to find transactions with visit rates that are proportional to each other, which means the client behaviour links these transactions in a certain workload type. Different transition probabilities between transactions represent different workload type. For example, the TPC-W defines 3 types of workloads: browsing, shopping and ordering [17]. Under a certain type of workload, the multicollinearity issue arises when some tuples of "independent" transactions are in fact correlated during session. There are several techniques for the detection and handling of multicollinearity by replacing groups of correlated variables with a single variable [13]. However this will not satisfy the requirement of application-level resource estimation because it reduces the number of transactions that can describe a workload.


System
metrics
Log Monitor
N, rt: throughput and response time
approximated total CPU utilization
z: measured total CPU utilization
Utilization Law
x: new CPU demands
H: sensitivities
e: latest total CPU estimation error
Kalman Filter
x, P: new CPU demands and covariances
Resource Estimation Engine
Figure 3 Logical architecture of the resource estimation engine

III. ADAPTING THE KALMAN FILTER TO CPU CONSUMPTION ESTIMATION

A. The Kalman Filter Estimator

The Kalman filter [16] has been widely used in the area of autonomous or assisted navigation. One of the main advantages of this filter is that it can estimate hidden parameters indirectly from observed data and integrate data from as many measurements as are available, in an approximately optimal way.

The Kalman filter addresses the general problem of estimating the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation (with a discrete monitoring window indexed by t):

$$x_t = Ax_{t-1} + w_{t-1} \tag{2.a}$$

and with a measurement z that is the following:

$$z_t = H_t x_t + v_t \tag{2.b}$$

in which A is a transform matrix from monitoring window t-1 to t, w_{t-1} represents the process noise which is a random independent normally distributed disturbance vector. H_t is a matrix concerning the relation of z_t and x_t, and v_t represents the measurement noise which is normally distributed and independent.

B. Adapting Kalman Filter to Application-level Estimation

Figure 3 depicts the architecture of a tenant resource estimation engine based on a Kalman filter. We model x as a vector of mean CPU demands of the M transaction types where $x_t = [C_{1,t}, C_{2,t}, \ldots C_{M,t}]$ represents the mean CPU demands of each transaction type within the monitoring window t. The drift w is assumed to be an independent normally distributed vector with covariance matrix Q.

The measured total CPU consumption is modelled by the result of the equation (1.c) calculation with added errors:

$$z_t = \frac{\sum_k \sum_i N_{i,k,t} C_{i,t}}{T} + v_t \tag{3.b}$$

in which z_t represents the measured total CPU utilization, H_t is defined as $\left[\frac{\sum_k N_{1,k,t}}{T}, \frac{\sum_k N_{2,k,t}}{T}, \ldots \frac{\sum_k N_{M,k,t}}{T} \right]$. It is reasonable to assume that the measurement noises are independent from one monitoring window to the next and normally distributed with covariance matrix R[24], which matches the assumptions of the filter.

The estimated CPU demands are computed recursively, beginning from an initial estimate vector $\hat{x}_0$ and an initial error covariance matrix P_0. As shown in Algorithm 1, each recursive step is executed at the end of each monitoring window. First, the algorithm projects the state ahead with $w_{t-1} = 0$ (see line 2). Next, it projects the error covariance P_t^- ahead and computes the Kalman gain K_t (see line 3-4). Then, the algorithm updates the state of x_t and computes the CPU consumption of each tenant (see line 5-8). Finally, it updates the error covariance at the end of each recursive step (see line 9). The Kalman filter dynamically corrects the estimated resource demand using the latest estimation error. Thus, it is more suitable for online estimation of non-deterministic states.

Algorithm 1 Estimating CPU Consumptions of Multiple tenants.

```
1.  for each recursive step do
2.      $\hat{x}_t^- = A_t \hat{x}_{t-1}$;
3.      $P_t^- = A_t P_{t-1} A_t^T + Q_t$;
4.      $K_t = P_t^- H_t^T (H_t P_t^- H_t^T + R_t)^{-1}$;
5.      $\hat{x}_t = \hat{x}_t^- + K_t (z_t - H_t \hat{x}_t^-)$;
6.      for each tenant do
7.          $U_{k,t} = \sum_i N_{i,k,t} C_{i,t}$;
8.      end for
9.      $P_t = (I - K_t H_t) P_t^-$;
10. end for
```

C. The Setting of the Filter Parameters

A robust approach requires an understanding of the filter and its setup.

According to the previous studies [11], the Kalman filter will converge if an *identifiability condition* is met. This condition requires that the measurements be non-correlated, which means that the basic Kalman filter cannot conquer the problem of multicollinearity either. To reduce the impact of the multicollinearity, we use the transform matrix A to dynamically forecast the states based on a simple and intuitive performance model. Our performance model uses the response time of a certain transaction type to forecast its resource demand:

$$C'_{i,t} = rt_{i,t}(1 - U_t) \qquad (4.a)$$

in which $rt_{i,t}$ is the mean response time of *i*-th transaction type within the *t*-th monitoring window. Equation (4.a), which means that the CPU demand is equal to the response time multiplied by the CPU idle rate, is derived from the classic queuing theory [20]. Then, we set

$$A_t = [\, C'_{1,t} / C'_{1,t-1}, ..., C'_{i,t} / C'_{i,t-1}, ..., C'_{M,t} / C'_{M,t-1} \,] \qquad (4.a)$$

If $t = 1$, set the $A_1 = I$, and calculate the initial estimates $\hat{x}_0$ by $C'_{i,0} = rt_{i,1}(1 - U_1)$. Meanwhile, a rule of thumb is to set $P_0 = \mathrm{diag}((C'_{1,0})^2, (C'_{2,0})^2, \ldots, (C'_{M,0})^2)$. We show in Section IV that the state forecasting will not only reduce the estimation error for cases with multicollinearity, but also improve the filter accuracy for cases with non-determinism.

The matrices Q and R affect the gain K and therefore influence how the filter reacts to new data. A strategy is to set Q as diagonal matrix and the value of diagonal elements is set to the square of the largest one-step change in x. Because R is the covariance matrix for the measurement noise of total CPU utilization averaged over a step, R can be determined by standard methods from a sufficiently large set of samples under no workload conditions.

The measurements, i.e., $N_{i,k,t}$, $rt_{i,t}$ and z_t, are averages over the monitoring window. A longer monitoring window gives more accurate measurements, but allows greater drift and more delay before the next update. These trends reduce the adaptability of the approach to CPU demand variation. A shorter monitoring window tracks changes more closely at the cost of inaccurate measurements. Our previous work [21] indicates that a shorter monitoring window (30s-100s) can track changes more closely and still delivers acceptable estimation accuracy. In the following experiment, the monitoring window size is set to 30s.

The time complexity of a recursion in our approach is $O(N^3)$, where N is the number of transaction types. In detail, the complexity of equation in line 4 of Algorithm 1 is only related to the complexity of seeking a reciprocal, as opposed to the complexity of inversing a matrix. Therefore, the most complex calculation left is in equation in line 9, which is a matrix multiplication. Thus, the total complexity of our approach is $O(N^3)$.

IV. Case Study: Applying to TPC-W Benchmark

To verify the effectiveness and the utility of the approach, we conducted case studies using the TPC-W ecommerce suite [17]. This section is organized as follows. Section IV.A describes the experiment setup used for the studies. Section IV.B presents sensitivity analysis of the approach with respect to the filter parameters. We also compare our approach with a regression technique. In Section IV.C, we conduct an experimental study using our estimation engine to demonstrate the utility of our approach for the purposes of performance isolation and CPU overload avoidance.

A. Experimental Environment

We built a test-bed of an e-commerce application that simulates the operation of an online bookstore, according to the industry standard e-business benchmark TPC-W. We also developed a workload generator, which employs the business design of TPC-W and has features that deduce a controllable and flexible representation of complex session-based workloads and that simulate authentic customer behavior [19]. This allows us to evaluate the proposed approach under different experiment settings in a controlled environment. Also, requests from different tenants were identified and classified by tenant ID stored in the session.

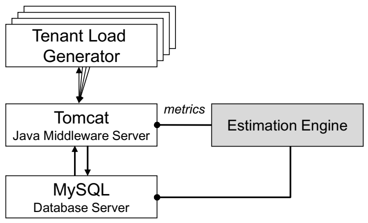


Figure 4 TPC-W experimental environment with multiple tenants

Table 2 Software/Hardware components

Server	Processor	RAM	Number
Tenants (Emulated-Browsers)	Pentium IV/1.8GHz	2GB	4
Middlerware Server-Tomcat6.0	Pentium IV/2.8GHz	2GB	1
Database Server-MySQL5.5	Pentium IV/3.8GHz	2GB	1
Estimation Engine	Pentium IV/1.8GHz	2GB	1

Figure 4 depicts the experimental environment. The system under test is a two-tier architecture paradigm including a middleware server (Tomcat) and a database server (MySQL) run on two servers. Four tenant load generators communicate with an online bookstore service deployed on the Tomcat server via request-reply transactions. A tenant resource estimation engine has been developed to gathers metrics from the Tomcat server and the MySQL server over a sequence of execution intervals [8]. The database size is determined by the number of items and the number of customers. In our experiments, we use the default database setting, i.e., the one with 10,000 items and 1,440,000 customers. All machines run Windows Server 2003 and have a 1Gbps Ethernet interface connected point-to-point full duplex with the switch. We use the PerMon tool provided by the operating system to gather the total CPU utilizations of the Tomcat server and the MySQL server. Specifics of the software/hardware used are given in Table 2.

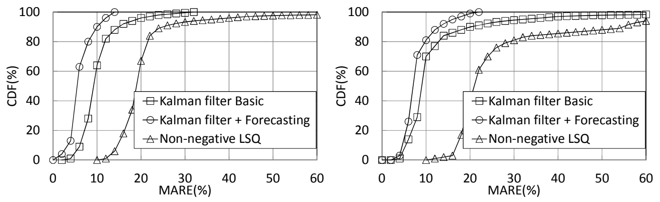
B. Estimation Results

We run experiments under the random workload situation and the standard TPC-W workload situation to explore the impact of non-determinism and multicollinearity, respectively. The random workload situation is constructed from random sessions which are obtained by conducting random walks over the Markov chains specified by TPC-W. The standard TPC-W workload situation is one of the three TPC-W default workload types under which the multicollinearity phenomenon exists. In both situations, the workload size of each tenant changes randomly, but less than 50 concurrent emulated browsers (EBs). Each experiment ran for 6 hours, beginning with a 10 minutes warm-up period and ending with a 10 minutes cool-down period.

We also compare the proposed approach with the basic Kalman filter approach (without state forecasting) and the regression based approach used in [9]. We use a Nonnegative LSQ with software provided by MATLAB. Typically, the accuracy of the regression approach is insensitive to the size of the monitoring window in approximating the overall resource utilization [9]. In this experiment, we use a 5mins monitoring window size.

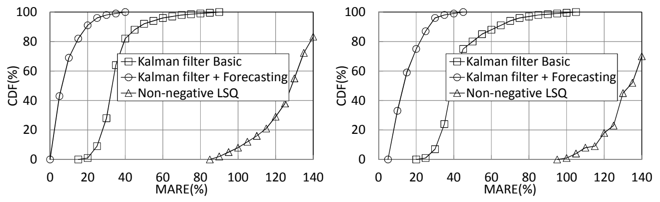
For every 30 seconds during each experiment (error monitoring window), we calculate the absolute relative estimation error of each transaction $error_i = \frac{\left| C_i^{new} - D_i \right|}{D_i}$,

where C_i^{new} is the latest estimated CPU demand and D_i represents the latest CPU demand measured by the approach used in [2]. Then, for every error monitoring window, we validate the accuracy by calculating the mean absolute relative error (MARE) of all transactions.



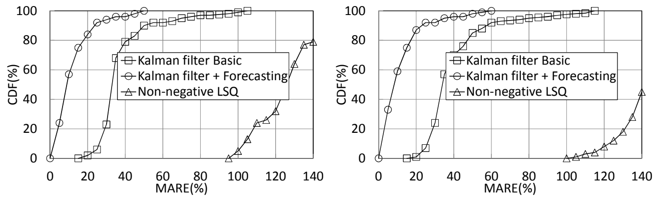
(a) Tomcat server (b) MySQL server

Figure.5 CDF of MARE under random workload



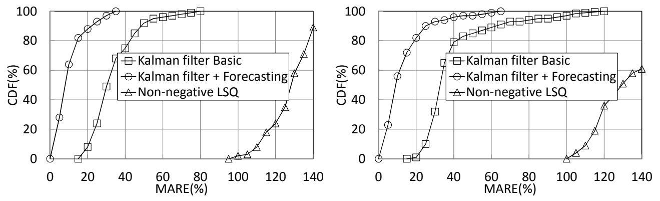
(a) Tomcat server (b) MySQL server

Figure.6 CDF of MARE under browsing workload



(a) Tomcat server (b) MySQL server

Figure.7 CDF of MARE under shopping workload



(a) Tomcat server (b) MySQL server

Figure.8 CDF of MARE under ordering workload

Table 3 The estimated results using regression based approach at the Tomcat server under browsing workload (ms)

Intervals / ***Transactions***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
Home	0	0	0	3.7	0	6.5	6.0	5.3	4.0	3.1	5.7	4.3
Best Sellers	0	0	0	0	0	0	0	0	0	0	0	0
New Products	0	0	0	0	0	0	0	0	0	0	0	0
Product Detail	0	0	0	2.4	0	0	0	0	2.7	3.0	0	2.5
Search Request	3.3	4.2	3.4	0	3.6	0	0	0	0	0	0	0
Search Results	0	0	0	0	0	0	0	0	0	0	0	0
Admin Confirm	0	0	0	0	0	0	0	0	0	0	0	0
Admin Request	0	0	0	0	0	0	0	0	0	0	0	0
Buy Confirm	0	0	0	0	0	0	0	0	0	0	0	0
Buy Request	0	0	0	0	0	0	0	0	0	0	0	0
Shop Cart	3.2	2.0	2.9	0	2.8	0	0	0	0	0	0	0
New Customer	0	0	0	0	0	0	0	0	0	0	0	0
Order Display	0	0	0	0	0	0	0	0	0	0	0	0
Order Inquiry	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5 (a), (b) show the results of experiments for the Tomcat server and the MySQL server under the random workload situation, respectively. The lines labeled "Kalman filter Basic", "Kalman filter + Forecasting" and "Non-negative LSQ" present the Cumulative Distribution Function (CDF) of the MARE caused by the basic Kalman filter, the Kalman filter with state forecasting and the regression, respectively. These performance results can be summarized as follows:

• The Kalman filter with state forecasting does much better than both the basic Kalman filter and the regression at both the Tomcat server and the MySQL server. The results show that the state forecasting improves the filter performance. The reason is probably that it improves the filter convergence rate and tracking performance.

• The regression approach has higher error due to the presence of non-determinism.

• All the three approaches deliver higher accuracy at the Tomcat server than that at the MySQL server. This reflects a higher variance in the CPU demand at the database tier for the same transaction type.

Figure 6, 7 and 8 show the CDF of the MARE for the Tomcat server and the MySQL server for under the three standard TPC-W workload situations (browsing, shopping and ordering). Comparing these results, one can observe that the accuracy of the filter with state forecasting is much better than the other two approaches at both Tomcat server and the MySQL server. This implies that our state forecasting techniques reduce the impact of the multicollinearity. The results also show the MARE values delivered by the regression can be quite large. For example, as show in Figure 6(a), the percentage of error monitoring windows at the Tomcat server that show less than 100% of relative errors are only 8%. To illustrate the impact of multicollinearity on the regression, Table 3 shows the estimation results of the 14 transactions for the Tomcat server under the browsing workload. One can observe most of the derived transaction CPU demands are "0", except for a few "significant" transactions, i.e., *Home*, *Product Detail*, *Search Request* and *Shop Cart*.

C. *Performance Isolation and Overload Control*

Two commonly used techniques of performance isolation and preventing server overload are (1) the concurrency control, which imposes a limit on the number of active requests and (2) the admission control, which imposes a limit on the request admission rate. Finding the ideal technique for the overload problem is outside of the scope of this paper. We conduct an experiment to demonstrate the application of our estimation approach by using the concurrency control to handle performance interference and CPU overload conditions. In our previous work, we proposed a fine-grained concurrency control framework for the Web applications, say a QoS-enabled WorkManager (QWM) [6]. In this experiment, we combine the proposed CPU consumption estimation approach and the QWM to design a fine-grained performance isolation and overload control strategy.

Table 4 Parameters used in the strategy

Parameter	Description	Default value
$T_{control}$	Length of the control interval	10s
a	Smoothing constant	0.7
$U_{threshold}$	CPU overload threshold	85%
adj_{d}	Multiplicative concurrency decrease	0.9
adj_{i}	Additive concurrency increase	1

The basic idea of the strategy is simple and intuitive: for every predefined control interval $T_{control}$, the strategy estimates and records the CPU consumption for each tenant and transaction, and monitors the total CPU consumption of the Tomcat server and the MySQL server. In order to prevent sudden spikes in the CPU consumption from causing large reactions, the CPU consumptions are smoothed using an exponentially weighted moving average with parameter α:

$$U_{cur\_interval} = a \cdot U_{last\_interval} + (1 - a) \cdot U_{cur\_interval}$$

If the total CPU utilization exceeds the predefined CPU overload threshold $U_{threshold}$, the strategy identifies the tenant with the largest CPU consumption during the last interval as an aggressive tenant. Moreover, the strategy also identifies which transaction type of the aggressive tenant consumes the most CPU time. Then, the number of concurrent request of the identified transaction type is reduced by a multiplicative factor adj_{d}. If the total CPU utilization is less than the threshold, and there are pending requests in the QWM due to previous concurrency limits, then the number of current request of the transaction type with pending requests is increased a additive factor adj_{i} until the strategy finds the largest that keeps the total CPU utilization within the desired threshold. Finding optimal parameters are outside the scope of this paper. The parameters used in the strategy implementation are summarized in Table 4.

In the following experiment, we also compare the proposed fine-grained strategy with a coarse-grained strategy which adjusts the concurrency limit for tenant scope based on the tenant-level resource estimation. We simulate a case in which a TPC-W online bookstore application serves 4 tenants, identified as "T1", "T2", "T3" and "T4". All tenants used shopping workload type. Figure 9 shows the workloads of the 4 tenants. The X axis is monitoring interval, and the Y axis is concurrent emulated browsers (EBs). From the estimation results in Section IV.B, we also identified that the *Admin* related transactions and the *Best Seller* transaction are the largest CPU consumers per transaction for the Tomcat server and the MySQL server, respectively. Previous work in [18] also found the same phenomenon. As shown in Figure 9, we simulate an aggressive Tenant "T1" by increasing the number of *Admin* related transactions for the intervals 11-30, and then simulate an aggressive Tenant "T2" by increasing the number of *Best Seller* transaction for the intervals 41-60.

Figure 10(a), (b) show how the throughput of each tenant changes over control intervals with coarse-grained

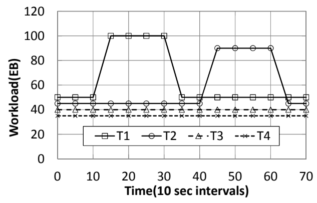
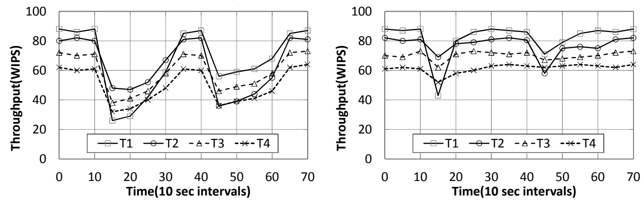
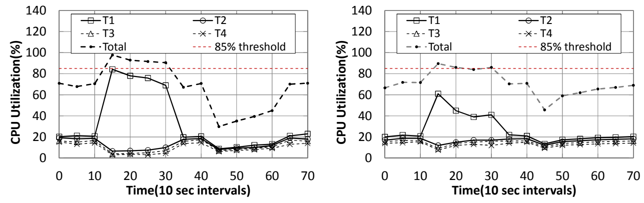


Figure.9 Tenants workloads



(a) coarse-grained strategy (b) fine-grained strategy

Figure.10 Tenant throughputs



(a) coarse-grained strategy (b) fine-grained strategy

Figure.11 CPU utilization of the Tomcat server

strategy and fine-grained strategy. Figure 11(a), (b) show the CPU utilization of each tenant for the Tomcat server with coarse-grained strategy and fine-grained strategy. The same performance trends were observed for the MySQL server. Because of space constraints, we report only the experimental results for the Tomcat server. These performance results can be summarized as follows:

• Coarse-grained strategy yields larger throughput decrease for the aggressive tenant. Figure 10 (a) shows that the coarse-grained strategy causes the average throughput of T1 for the intervals 11-30 intervals decrease 55%. Whereas, Figure 10 (b) shows that the fine-grained strategy causes 18% drop and, especially only 9% drop for the intervals 20-30. The same performance trends were observed for the intervals 41-60. This is due to the requests accessing a certain transaction type that consumes many resources can be identified and conditioned to reduce their concurrency limit, rather than refusing all new requests of the aggressive tenant in a generic fashion.

• Fine-grained strategy based on the application-level resource estimation is more efficient and agile. As shown in Figure 11 (a), during T1 overload (intervals 11-30), an obvious drop occurs in other tenants' utilizations, and the total CPU utilization remains above 85%. This implies that the coarse-grained strategy react slowly to the negatively impacts of T1, and even lead the system into a danger of crashing. Note that, in Figure 11 (b), the overall CPU utilization reduced to 85% within 5 intervals (intervals 15-20). This leads to a small negative impact on the CPU utilizations of the other tenants. The same performance trends were observed during T2 overload (intervals 41-60).

V. RELATED WORK

Native code-based approaches are widely used in previous work on CPU consumption measurements. This approach measures the CPU consumption by a native agent that probes a CPU for the calculation of cycles by sampling. However, with respect to most of the web-based systems, the CPU consumption for the execution of each request is very small. Therefore, we need high precision timestamps to provide accurate results. For example, Magpie [2] uses Event Tracing for Windows to measure CPU usage with a processor cycle counter in the Windows operating system. Similar approaches on different operating systems are the Linux Trace Toolkit [23] and Solaris DTrace [12]. This kind of approach requires an applications instrument at the source or binary levels and relies on kernel agents to collect and correlate events. In contrast, our approach is nonintrusive.

Binder et al. [14] proposed a CPU-management framework for Java. Their proposed approach measures the CPU consumption by tracking the number of executed JVM bytecode instructions and then transforms them to CPU consumption. However, this approach incurs more than a 30% performance overhead at runtime [15]. In contrast, our approach is based on monitoring data, i.e., throughput, response time, total CPU utilization, that are typically available in enterprise production environments and therefore incurs much less performance overhead.

Recently, the use of statistical methods in resource consumption estimation has been proposed. In [9], the authors use multiple linear regression techniques to derive the mean service time for application transactions or services. Previous studies [4] and the experimental results in this paper have shown that the accuracy of regression techniques suffer if the CPU demands are not deterministic due to resource competition or if there exist collinear transactions. There are several techniques for the detection and handling of multicollinearity by replacing groups of correlated variables with a single variable [13], for example by using reduction techniques [22]. However this will not satisfy the requirement of application-level resource estimation because it reduces the number of transactions that can describe a workload.

Kalman filters have been used previously to predict the CPU utilization of virtualized servers [25] and to estimate the parameters of a queueing model [7]. Kalyvianaki et al. [25] use Kalman filter to predict the CPU utilizations of virtualized servers based on past observations and improve the performance of the feedback controllers for dynamically allocating the CPU resources of virtualized servers. Woodside et al. [7] proposed a mechanism to update the multiple performance models' parameter based on the extended Kalman filter. Their approach investigates the use of queuing models to deduce resource consumption. This approach relies on a measured response time from a system and a performance model for the system. Resource consumption is estimated such that the model's mean response time prediction closely matches the mean of the measured response time. The cost of their approach is the product of the number of transaction types and the total time for calculation. The time for calculation is directly related to the complexity of the performance model, which usually takes much more time. In this paper, we propose a linear Kalman filter approach with a time complexity of $O(N^3)$, where N is the number of transaction types processed by the server. This complexity order implies that this method delivers resource consumptions in several milliseconds and thus can be used efficiently as a part of an online resource evaluation method.

VI. CONCLUSIONS

Shared application easily causes performance interference between multiple hosted tenants. A key requirement for effective online performance isolation and resource management is the knowledge of how many resources are used by various tenants during a fine time-scale. However, detailed resource consumption information of tenants is difficult to obtain using direct profiling techniques. In this paper, we established the feasibility of dynamically estimating CPU consumption in multi-tenancy applications using a Kalman filter-based approach. We also propose techniques to deal with the non-determinism and the multicollinearity issues. The end result is a mechanism that extends the current monitoring solutions to multi-tenancy monitoring. A detailed set of experiments were performed under different settings for changing workloads in the controlled TPC-W e-commerce suite to evaluate the effectiveness of the approach. Our experiments show that our approach is insensitive to non-determinism and multicollinearity. We also demonstrate the utility of our approach for the purposes of performance isolation and CPU overload avoidance. Experimental results show that the fine-grained strategy based on the application-level resource estimation is more efficient and agile.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No. 61100068; the National High Technology Research and Development Program of China under Grant No. 2012AA011204; the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320704.

REFERENCES

[1] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang and B. Gao. A Framework for Native Multi-Tenancy Application Development and Management. CEC/EEE, IEEE Computer Society, pp. 551-558, 2007

[2] P.Barham, A.Donnelly, R.Isaacs, R.Mortier. Using Magpie for request extraction and workload modelling December 2004 6th Symposium on Operating Systems Design and Implementation (OSDI'04),2004.

[3] J. Rolia, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley & Sons, 1991.

[4] J. Rolia, V. Vetland. Correlating Resource Demand Information with ARM Data for Application Services. In Proc. of the ACM Workshop on Software and Performance, 1998.

[5] Salesforce, URL: http://www.salesforce.com/

[6] Wei Wang, Wenbo Zhang, Jun Wei, Tao Huang. A QoS-enabled WorkManager Model for Web Application Servers. The 7th International Conference of Quality Software (QSIC 2007), 40-49, IEEE CS Press, October 11- 12, 2007, Portland, Oregon, USA.

[7] M. Woodside, T. Zheng, and M. Litoiu, The Use of Optimal Filters to Track Parameters of Performance Models, Proc. Second Int'l Conf. Quantitative Evaluation of Systems, Sept. 2005.

[8] TRE4J, http://www.trustie.net/projects/project/show/TREforJ

[9] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: A capacity planning and anomaly de-tection tool for enterprise services with live workloads. In Middleware, 2007.

[10] X.H. Li, T.C. Liu, Y. Li, Y. Chen, SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment, Proceedings of the 6th International Conference on Service-Oriented Computing, p.649-663, December 01-05, 2008, Sydney, Australia.

[11] A.H. Jazwinski, Stochastic Processes and Filtering Theory. Academic Press, 1970.

[12] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic instrumentation of production systems. In Proc. USENIX Annual Technical Conference, pages 15–28, June 2004

[13] J.O. Rawlings, Applied Regression Analysis: A Research Tool,Wadsworth & Brooks, Cole Advanced Books & Software, Pacific Grove, CA,1988.

[14] W. Binder and J. Hulaas. A portable CPU-management framework for Java. IEEE Internet Computing, 8(5):74–83, Sep./Oct. 2004.

[15] Hulaas, J., Kalas, D.: Monitoring of Resource Consumption in Java-based Application Servers. In: Proceedings of the 10th HP OpenView University Association Plenary Worksop (HPOVUA 2003), Geneva, Swizerland (2003).

[16] R.E.Kalman, A New Approach to Linear Filtering and Prediction Problems, Transactions of the ASME-Journal of Basic Engineering, 1960.

[17] TPC-W Benchmark: URL http://www.tpc.org/tpcw

[18] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Bottleneck Characterization of Dynamic Web Site Benchmarks. Technical Report TR02-391, Rice University, Feb 2002.

[19] Bench4Q: URL http:// forge.ow2.org/projects/jaspte

[20] Lazowska ED, Zahorjan J, Graham GS, Sevcik KC. Quan-titative System Performance: Computer System Analysis Using Queueing Network Models. Upper Saddle River: Prentice-Hall, Inc., 1984.

[21] W. Wang, X. Huang, Y. Song, W. Zhang, J. Wei, H. Zhong, T. Huang, A Statistical Approach for Estimating CPU Consumption in Shared Java Middleware Server, Proceedings of the 35th compsac, pp.541-546, 2011 Munich, Germany.

[22] G. Pacifici, W. Segmuller, M. Spreitzer, A. Tantawi, CPU Demand for Web Serving: Measurement Analysis and Dynamic Estimation, Performance Evaluation, Vol. 65, No. 6-7, pp.531-553, June 2008.

[23] K. Yaghmour and M. R. Dagenais. Measuring and characterizing system behavior using kernel-level event logging. In Proc. USENIX Annual Technical Conference, June 2000.

[24] A.M. Law and W.D. Kelton, Simulation Modeling and Analysis, third ed. McGraw-Hill, 2000.

[25] E. Kalyvianaki, T. Charalambous, S. Hand, Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters, in ICAC '09: Proceedings of the 6th international conference on Autonomic computing. ACM, 2009, pp. 117-126.