RC24887 (W0910-185) October 29, 2009
Computer Science

IBM Research Report

Real-Time Performance Modeling for Adaptive Software Systems

Dinesh Kumar, Asser Tantawi, Li Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Real-Time Performance Modeling for Adaptive Software Systems

Dinesh Kumar, Asser Tantawi, Li Zhang
IBM T.J. Watson Research Center, Hawthorne, NY, USA
{kumardi,tantawi,zhangli}@us.ibm.com

ABSTRACT

Modern, adaptive software systems must often adjust or reconfigure their architecture in order to respond to continuous changes in their execution environment. Efficient autonomic control in such systems is highly dependent on the accuracy of their representative performance model. In this paper, we are concerned with real-time estimation of a performance model for adaptive software systems that process *multiple* classes of transactional workload. Based on an open queueing network model and an Extended Kalman Filter (EKF), experiments in this work show that: 1) the model parameter estimates converge to the actual value very slowly when the variation in incoming workload is very low, 2) the estimates fail to converge quickly to the new value when there is a step-change caused by adaptive reconfiguration of the actual software parameters. We therefore propose a modified EKF design in which the measurement model is augmented with a set of constraints based on past measurement values. Experiments demonstrate the effectiveness of our approach that leads to significant improvement in convergence in the two cases.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

General Terms

performance,model

Keywords

estimation,filter,queueing theory,parameters

1. INTRODUCTION

Today's software systems must continuously self-reconfigure their components to adapt to run-time changes in the host and network environments [1]. This is especially the case for Internet based online applications that operate in a highly dynamic environment with fast changing user workloads and browsing patterns. Changes may also occur in the virtualized system platform that runs the software application [2].

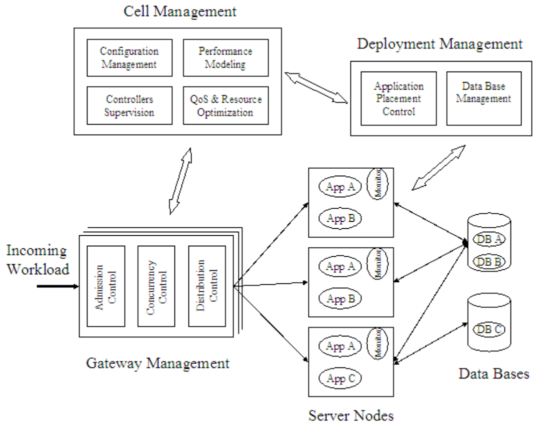

Figure 1: Architecture of Adaptive Software System

This paper considers such *Adaptive Software* (AS) systems [3, 4, 5] that process transactional user workload of request/ response type such as HTTP workload. Each transaction in an AS system that uses the server's resources differently can be classified into different classes. The quality of a software system is often measured in terms of its performance which can be for example, end to end response time from a user's point of view. A performance model of a system can be used for predictive analysis of the system, e.g., for response time prediction at hypothetical workloads. Performance model of an AS system can be useful for autonomic control, if it is updated in real-time to reflect the changes in the software system parameters [1]. However, performance modeling of an AS system is a challenging task. Classical queueing theory based performance models require the knowledge of parameters such as *service times* and *network queueing delays* for different classes of transactions. These parameters are used to compute and predict performance metrics such as average transaction response time, average number of jobs/transactions waiting to be processed, etc. There are existing techniques that make use of simulations and manual caliberations to compute similar performance metrics [6]. However, none of these techniques can be practically applied if the service times and network queueing delays are unknown. Instrumenting software applications with probes in order to actually measure the service time and delay parameters can be intrusive, requires extensive manual coding [7] and is time consuming. In fact, the source code of a standard, commercialized e-commerce software system may not even be accessible. Moreover, instrumentation is an iterative procedure and is difficult to pursue in a dynamically changing environment [7]. This is often the case for an AS system that undergoes continuous changes that can lead to *time-varying* service times and delays. These system parameters must therefore be estimated using only readily available measurement data. AMBIENCE [8, 9] which is a research prototype tool developed at IBM Research, makes use of a powerful Inferencing algorithm to estimate a service time and network queueing delay based performance model. Inferencing allows one to compute the service time and delay parameters from readily available measurement data on end-to-end response times, CPU utilizations and workload arrival rates. It however models service time and delay using *stationary* model parameters and cannot be used for AS systems with time-varying parameters.

Performance models can play an important role in accurately driving the necessary dynamic changes in an AS system. For instance, at runtime, software systems can better adapt to the changes in execution environment if an underlying performance model of the system is known. A performance model updated in *real-time* can be combined with model predictive control [10] to achieve autonomic control of a software system. Figure 1 shows the architecture of an example AS system studied in this paper. Reliable control of a software system in order to achieve the desired objective is critically dependent on the service time and queueing delay parameters that characterize the system. While study of optimal control strategies is a separate research area in itself, robust control can only be achieved if the system model parameters accurately reflect changes in the software system at runtime. Since autonomic control of a software system may lead to reconfiguration of its architecture at run-time, the underlying model parameters may not remain constant and can vary with time. It is thus important to accurately track the time-varying parameters of an AS system in real-time. Note that we *do not* study any kind of control mechanism in this work and focus only on model parameter estimation.

1.1 Related Work

A prototype implementation of a performance management system for multi-tiered web applications deployed on clustered web servers is decribed in [11]. The management system allocates server resources dynamically in order to optimize the expected value of a system-wide utility function. Authors there use a closed queueing network model to predict the response time of requests for different resource allocations. However, the model parameters (number of clients and think time) are estimated by solving independent, *static* optimization problems at each measurement point. Urgaonkar et al. [12] develop another closed queueing network model of a multi-tiered system. Estimates of the model parameters, such as visit ratios and load-dependent service times, are obtained *off-line* through analyzing various measurement logs in the system. Their methodology lacks a sound analytical model for parameter estimation. Pacifici et al. [13] consider the problem of dynamically estimating CPU demands of applications using CPU utilization and throughput measurements. Using a linear model, they formulate the problem as a multivariate linear regression problem and analyze measurement data. However, their experimental results demonstrate that the approach is viable only for a *rough* estimation of the dynamically changing CPU demands.

Real-time performance modeling using Kalman filters [14] is an emerging area of research and only few related papers can be found in literature. To the best of our knowledge, only Zheng et al. [15, 16, 17] have demonstrated the effective use of an Extended Kalman Filter (EKF) [14] for tracking time-varying parameters of an AS system, for the purpose of real-time performance modeling. Though they provide theoretical insights for system models based on both closed and open multiclass queueing networks, their experimental results are only for the *single* class case. Colleagues of Zheng in [1] have studied real-time adaptive control of an autonomic computing environment. Two types of controllers are studied: threshold controller and feed-forward controller based on a Kalman filter model. The later is again based on the EKF design for performance modeling proposed in [15, 16, 18]. Once again only a single class of workload traffic is considered for the experimental results in [1]. Other colleagues of Zheng in [2] have studied model based autonomic server virtualization, but use system identification techniques instead of a Kalman filter. To summarize, none of the existing related work has investigated performance of Kalman filters in a *multi-class* workload environment.

1.2 Main Results and Contribution

The contributions of this work are *twofold*. First, we demonstrate that straight-forward application of EKF is inadequate in a multi-class setting. We present experiments to show that for an open queueing network model of the software system, an EKF design in the lines of the work in [15, 16, 17] performs poorly for two important scenarios: 1) the model parameter estimates converge to the actual value very slowly when the variation in incoming workload is very low, 2) the estimates fail to converge quickly to the new value when there is a step-change in software parameters caused by adaptive reconfiguration of the software architecture. We argue that these anomalies occur due to the under-determined nature of the estimation problem for multiple classes of workloads. We therefore address this issue by modifying the EKF design by augmenting the measurement equation with a set of constraints based on past measurement values. Experiment results demonstrate that use of this modified EKF leads to significant improvement in convergence in the two cases. This is thus the second main contribution of our work.

2. SYSTEM DESCRIPTION

A high level description of a three-tiered AS system and its management is depicted in Figure 1. The first tier hosts gateway management components, the second tier hosts a web server and the adaptive software applications and the third tier comprises database services. The second tier consists of several server nodes that host the various applications. An adaptive software is deployed as an application in an application server that provides the necessary runtime environment. Several application servers may run on a given node and several instances of a given application may coexist at the same time. Incoming requests (workload) from users arrive at the first tier where they are routed to an appropriate node in the second tier. Servicing these requests by an application may trigger further requests to the database tier. The completion of a request triggers a response back to the user.

Two important performance concerns in such an AS system are resource utilization and quality of service in terms of response time. Several mechanisms are usually in place to alleviate such concerns. Let us focus on two broad categories of mechanisms that we call *Cell Management* and *Deployment Management*. The latter is concerned with placement of application instances in nodes, as well as database partitioning and replication. Cell management is concerned with the overall operation of the multi-tier system cell. Efficient cell and deployment management requires knowledge of performance models in order to predict performance and ensure optimal resource utilization through control. Control mechanisms such as configuration management, QoS and resource optimization, application placement control and database management are integral components of the cell and deployment managers. Other control mechanisms such as admission control, concurrency control and distribution control (or load balancing) are components of the first tier gateway manager. Monitoring agents running on various server nodes collect measurements that are fed to the cell and deployment management components. These measurements can be used to update the performance model of the system, which can in turn be employed by the various controller components.

The 'performance modeling' component is a key aspect of the cell manager since most of the control operations and resource optimization settings are based on a representative model of the system. Such models need to be dynamically updated in real-time to reflect the changing cell characteristics, be it resources, applications or user demand. In other words, real-time control of an AS system requires a real-time update of the performance model.

3. EXTENDED KALMAN FILTER DESIGN

We present here an EKF design in the lines of the work in [15, 16, 17]. It is assumed that the reader is aware of the general theory of Kalman filtering. The reader is referred to [14] for the same. Kalman filter provides the required framework for estimating model parameters in a real-time fashion by representing them as the state of an AS system. Kalman filter is a minimum mean square error (MMSE) estimator that estimates the state from a series of incomplete and noisy measurements. It minimizes the covariance of estimation error and operates by propagating both the mean and covariance of state estimates through time. We propose an open queueing model based EKF design for estimating service time and network queueing delay parameters of a single node, running a single application and processing three different classes of traffic. It is sufficient to consider a single node and three traffic classes to demonstrate the problems that are encountered in the AS system of Figure 1 with multiple classes of workload. Such a single node system is shown in Figure 2.

The system in Figure 2 is treated as a dynamical system. Measurements on workload arrival rate and transactional response times for each of the three user classes and CPU utilization of the node are gathered at a sampling interval of T seconds. These time series data are the input to the filter. Workload arriving at the software system may be fast changing and non-stationary. Assume that T is small enough (few seconds) so that the arriving workload can be considered stationary during this sampling interval. Then the stationary, queueing theory based performance models hold during this sampling interval. If we consider the state of the system to comprise service time and network delay, then an EKF can be used to compute a time series estimate of this state.

For three classes indexed as class a, b and c and a single node, we define the system state $\boldsymbol{x}$ as,

$$\boldsymbol{x} = \begin{bmatrix} s^a & s^b & s^c & d^a & d^b & d^c \end{bmatrix}^T, \tag{1}$$

where, s^a, s^b and s^c are service times at the server node for classes a, b and c, respectively, and d^a, d^b and d^c are network delays for the three classes. Based on an $M/G/1$ open queueing network model with processor sharing (PS) service discipline [19], the measurement model [14] $\boldsymbol{z} = h(\boldsymbol{x})$ is defined as,

$$\begin{bmatrix} R^a \\ R^b \\ R^c \\ u \end{bmatrix} = \begin{bmatrix} \frac{s^a}{1-u} + d^a \\ \frac{s^b}{1-u} + d^b \\ \frac{s^c}{1-u} + d^c \\ \frac{1}{P}(\lambda^a s^a + \lambda^b s^b + \lambda^c s^c) \end{bmatrix} \tag{2}$$

where, R^a, R^b and R^c are response times of the three classes, u is the CPU utilization averaged over all CPUs of the only node and P is the number of CPUs. If each sampling interval is denoted by k then we may assume the following dynamics for state evolution,

$$\boldsymbol{x_k} = \boldsymbol{F_k x_{k-1}} + \boldsymbol{w_k},$$

where, F_k is the state transition model which is applied to the previous state $\boldsymbol{x_{k-1}}$ and $\boldsymbol{w_k}$ is the process noise which is assumed to be drawn from a zero mean, multivariate normal distribution with covariance $\boldsymbol{Q_k}$, i.e., $\boldsymbol{w_k} \sim \mathcal{N}(0, \boldsymbol{Q_k})$. The iterative measurement equation [14] for Kalman filter is taken to be,

$$\boldsymbol{z_k} = \boldsymbol{H_k x_k} + \boldsymbol{v_k},$$

where, $\boldsymbol{H_k}$ is the observation model which maps the true state space into the observed space and $\boldsymbol{v_k}$ is the observation noise which is assumed to be zero mean, Gaussian white noise with covariance $\boldsymbol{R_k}$, i.e., $\boldsymbol{v_k} \sim \mathcal{N}(0, \boldsymbol{R_k})$. Since the measurement model in Equation 2 is non-linear in terms of the system state parameters (due to utilization u in the denominator), we must use the 'Extended' version of the Kalman filter [14]. The corresponding Jacobian matrix of the measurement model is given by,

$$\boldsymbol{H} = \frac{\partial h}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{1-u+\frac{\lambda^a s^a}{P}}{(1-u)^2} & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1-u+\frac{\lambda^b s^b}{P}}{(1-u)^2} & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1-u+\frac{\lambda^c s^c}{P}}{(1-u)^2} & 0 & 0 & 1 \\ \frac{\lambda^a}{P} & \frac{\lambda^b}{P} & \frac{\lambda^c}{P} & 0 & 0 & 0 \end{bmatrix}$$

and $\boldsymbol{H_k}$ can be computed as,

$$\boldsymbol{H_k} = \left[\frac{\partial h}{\partial \boldsymbol{x}}\right]_{\hat{\boldsymbol{x}}_{k|k-1}}.$$

We may now use the standard EKF theory [14] to track the system state over time. One of the major advantages of Kalman filter is that it is a recursive estimator. This means that only the estimated state from the previous time step and the current measurements are needed to compute the estimate for the current state. In the following EKF algorithm, the notation $\hat{\boldsymbol{x}}_{n|m}$ represents the estimate of $\boldsymbol{x}$ at time n given observations up to and including time m. The state of the filter itself is represented by two variables:

1. $\hat{\boldsymbol{x}}_{k|k}$ is the estimate of state at time k given observations up to and including time k.

2. $\boldsymbol{P}_{k|k}$ is the error covariance matrix (a quantitative measure of estimated accuracy of the state estimate).

The Kalman filter algorithm has two distinct phases: *Predict* and *Update*. The predict phase uses state estimate from the previous time interval to produce an estimate of the state at current time interval. In the update phase, measurement information at the current time interval is used to refine this prediction to arrive at a new, more accurate state estimate, again for the current time interval. These two phases are given as,

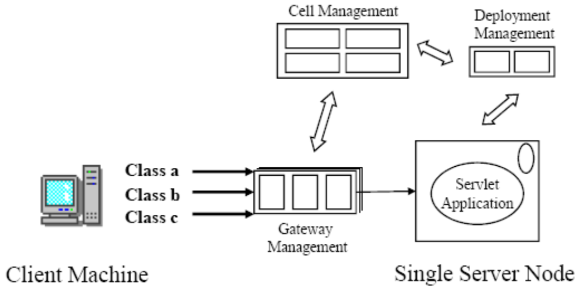

Figure 2: AS system with single server node and three workload classes

Predict:

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

Update:

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1})$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

A more detailed description of the two phases and notation can be found in [14]. The filter design proposed above was implemented as a stand-alone Java application. Measurements gathered during experiments were fed in to this application to obtain service time and delay estimates. The experimental results are presented later in Section 5.

4. EXPERIMENTAL SETUP

In this section, we describe the experimental setup that was used to conduct the experiments. Figure 2 shows the single-node AS system with three user workload classes considered for our experimental setup. It consists of a web-based environment including a synthetic HTTP traffic generator at the client machine and an IBM WebSphere Virtual Enterprise cell [20] with a single server node. The architecture of an IBM WebSphere cell is similar to that shown in Figure 1. The actual application under consideration is a micro-benchmarking servlet that simulates an adaptive software application.

4.1 Simulated Adaptive Software

Details of how the servlet simulates an adaptive software are as follows. Each HTTP request/transaction is served by the micro-benchmarking servlet that alternates between *computing* and *sleeping*. Servlet parameters controlling the behavior of the execution of a request are: the total amount of computation (specified in terms of the number of loops over some arithmetic computation), duration of computation between sleeps and the duration of sleep. These parameters may be fixed or drawn from some probability distribution. The parameters can be provided in the HTTP request and are configurable through the synthetic traffic generator tool. Given values of these servlet parameters translate into values of service time and delay parameters. The actual translation function is not of importance as it may vary for different software systems. Adaptivity of a software system was simulated by manually changing the servlet parameters through an HTTP request, that resulted in modified values for the service time and delay parameters.

4.2 Traffic Generator

The traffic generator is written in Java and generates HTTP requests of different types. A configurable number of parallel threads simulate the web clients. The think time, defined as interval between the receipt of a response from server node and submission of a subsequent request, is specified by a probability distribution. We used the sum of a fixed bias (125 msec) and an exponentially distributed time with mean 125 msec. Load on the system can by altered by varying the number of clients. Changing the parameters may be performed manually or programmatically. In the LVW experiment of Section 5.1, the parameters were not changed and kept fixed, resulting in a stationary workload. In the SSP experiment of Section 5.2, an additional component of the traffic generator was activated that resulted in sinusoidal waves in the workload. This additional component took as input the amplitude (maximum number of clients), the phase in degrees (allowing different request flows to have different phases) and the periodic length (for the time duration of a sinusoidal cycle).

4.3 System Architecture and Hardware

The micro-benchmarking servlet was deployed as an application in the IBM WebSphere cell (see Figure 1) that consisted of a single server node and additional management nodes: Cell, Deployment and Gateway manager. Monitoring agents ran on the server node for collecting statistical measures in a non-intrusive manner, without the need to instrument the application. In our experiments we measured workload arrival rates, response times of requests and CPU utilization of the server node. Once the micro-benchmarking servlet is deployed, the server node ceases to interact with the Deployment Manager. This ensures that management nodes do not interfere with request/transaction processing activity at the server node. IBM WebSphere platform version 6.1 (ND 6.1.0.17) (XD 6.1.0.3) was used for our experiments.

The traffic generator tool ran on a client machine with a single Xeon 2.8 GHz Intel processor, 512 KB cache and 2 GB RAM. Three different types of transaction classes were generated during the experiments. Each class was denoted by a different class ID in the HTTP request and was generated using different values for the traffic generator parameters. The classes were simply named as Class a, Class b and Class c. The servlet application ran on a server node with a single Xeon 3.06 GHz Intel processor, 512 KB cache and 2 GB RAM. The servlet processed different classes of transactions differently, through unique values of servlet parameters that resulted in unique values of service times and delays for each class.

5. PROBLEMS WITH MULTIPLE CLASSES OF WORKLOAD

It is demonstrated here that the use of EKF (in the lines of the work in [15, 16, 17]) for tracking model parameters of an AS system with multiple classes of workload has convergence problems. For this purpose, two different types of experiments were conducted:

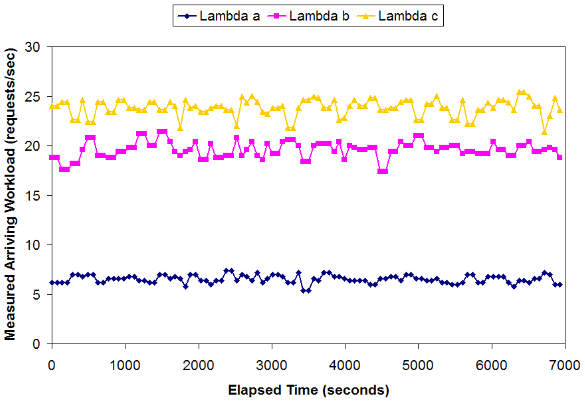

Figure 3: Workloads for LVW experiment

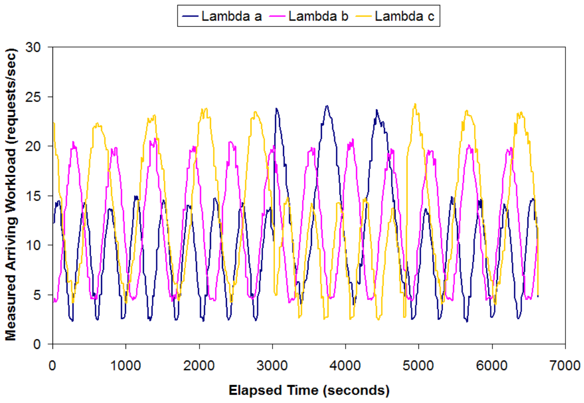

Figure 6: Workloads for SSP experiment

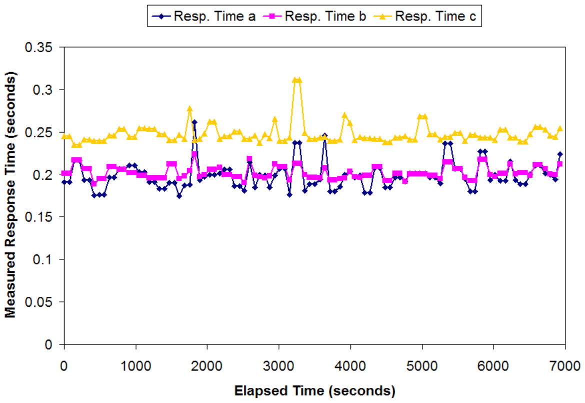

Figure 4: Response times for LVW experiment

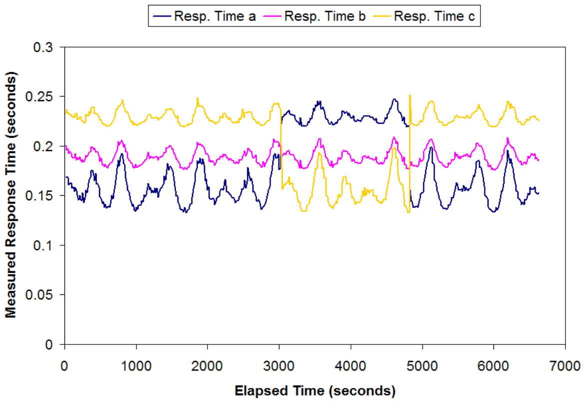

Figure 7: Response times for SSP experiment

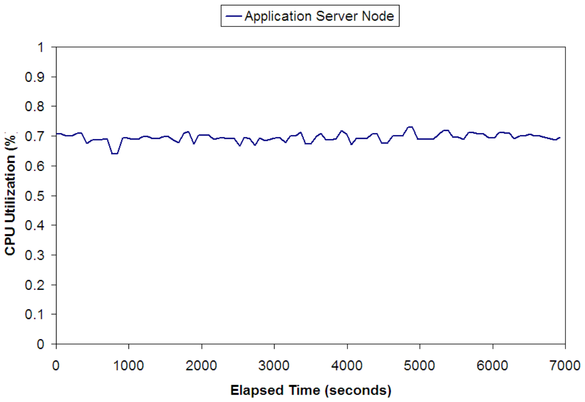

Figure 5: CPU utilization for LVW experiment

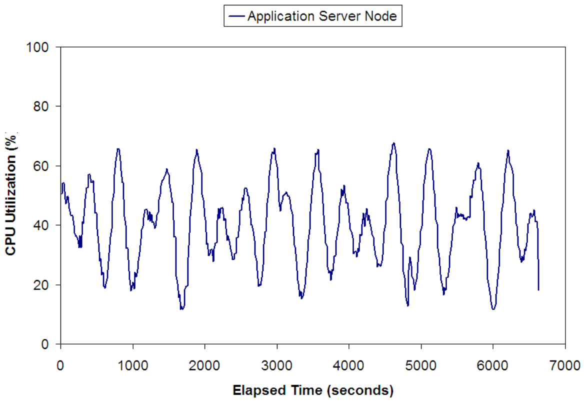

Figure 8: CPU utilization for SSP experiment

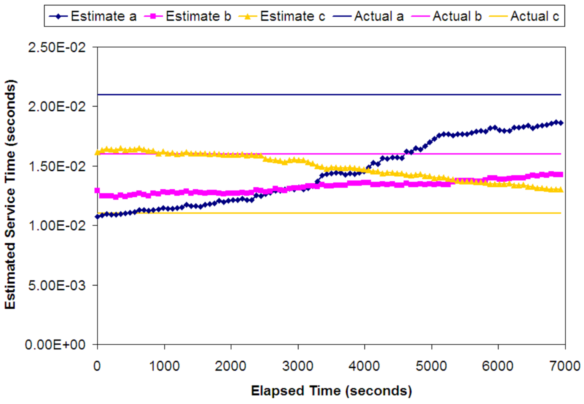

Figure 9: Service time estimates for LVW experiment

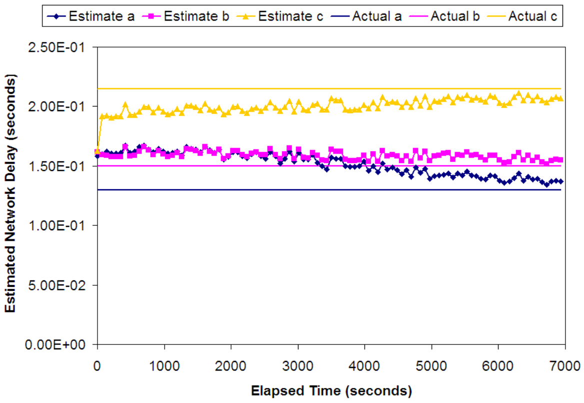

Figure 10: Network delay estimates for LVW experiment

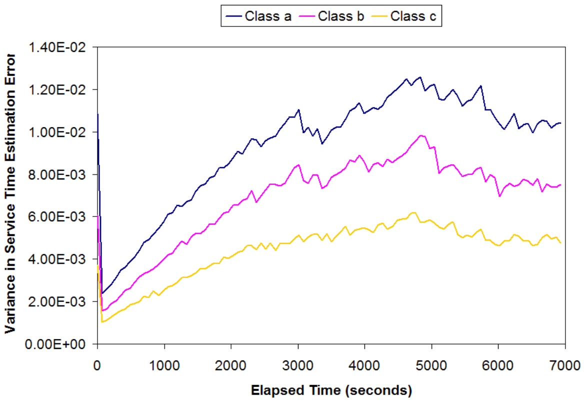

Figure 11: Service time estimation error for LVW experiment

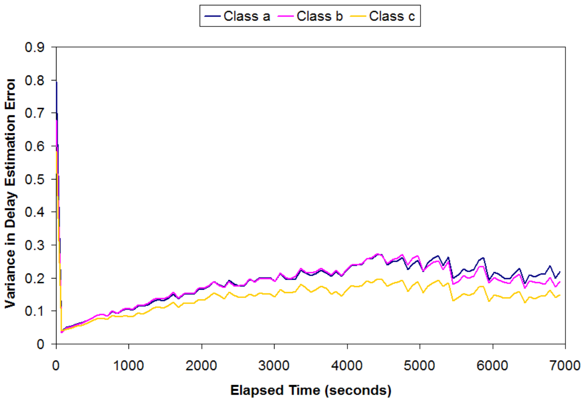

Figure 12: Network delay estimation error for LVW experiment

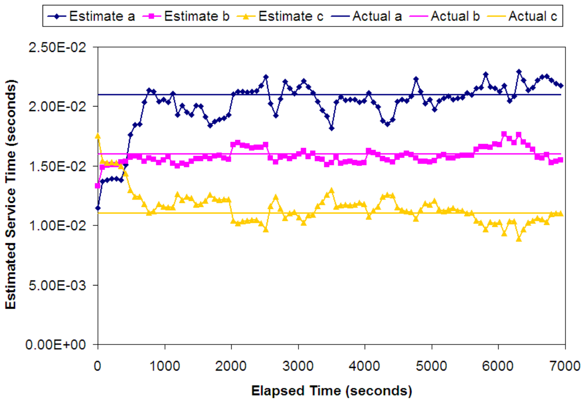

Figure 13: Improved service time estimates (LVW)

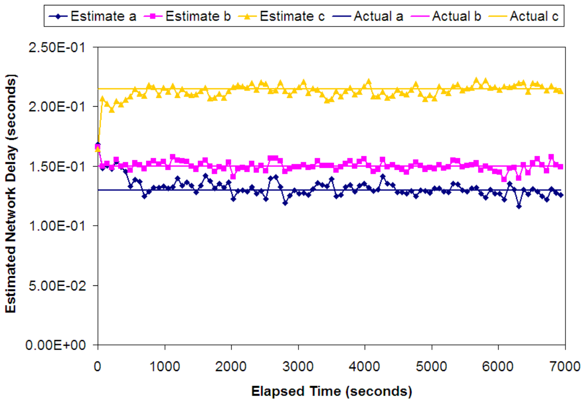

Figure 14: Improved delay estimates for LVW experiment

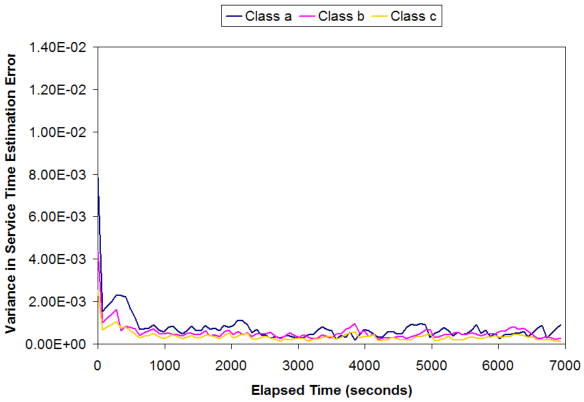

Figure 15: Improved service time estimation error for LVW experiment

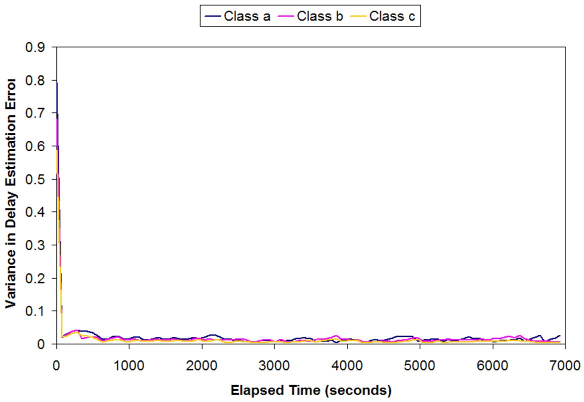

Figure 16: Improved delay estimation error for LVW experiment

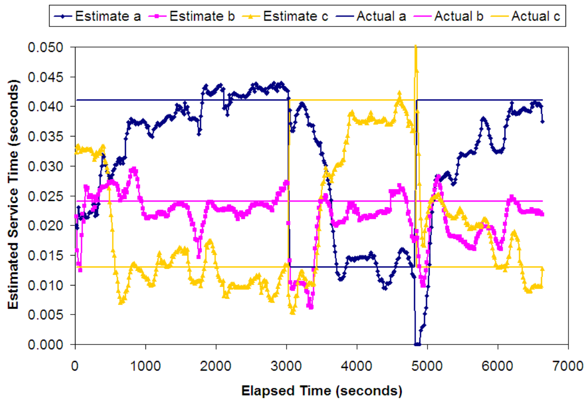

Figure 17: Service time estimates for SSP experiment

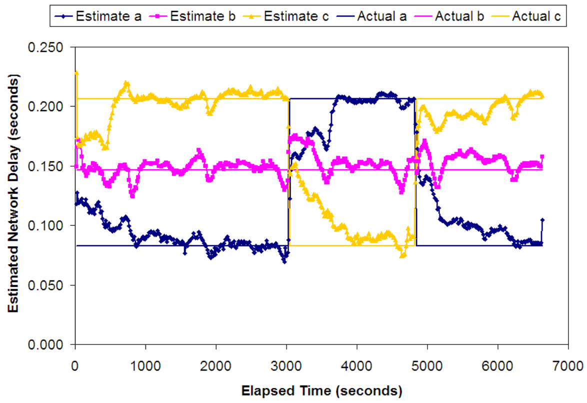

Figure 18: Network delay estimates for SSP experiment

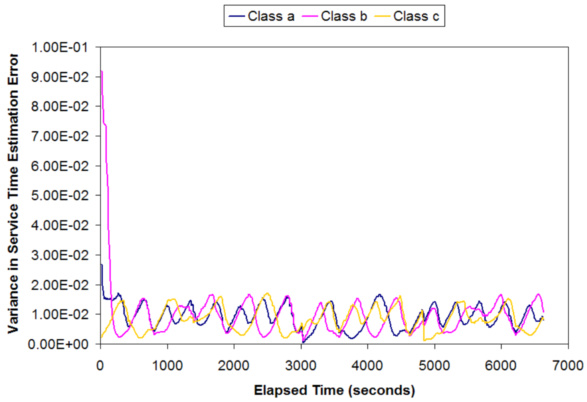

Figure 19: Service time estimation error for SSP experiment

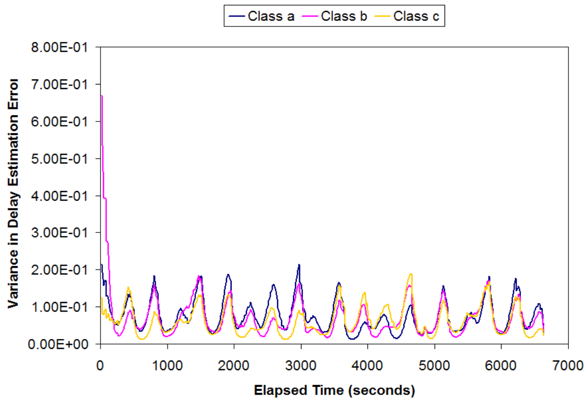

Figure 20: Network delay estimation error for SSP experiment

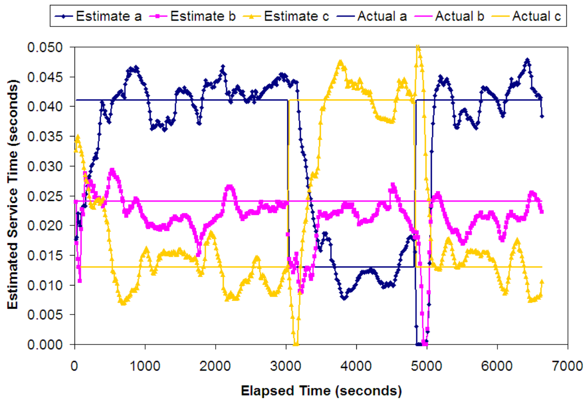

Figure 21: Improved service time estimates (SSP)

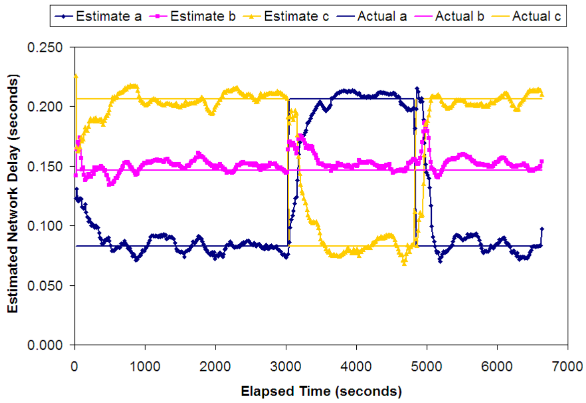

Figure 22: Improved delay estimates for SSP experiment

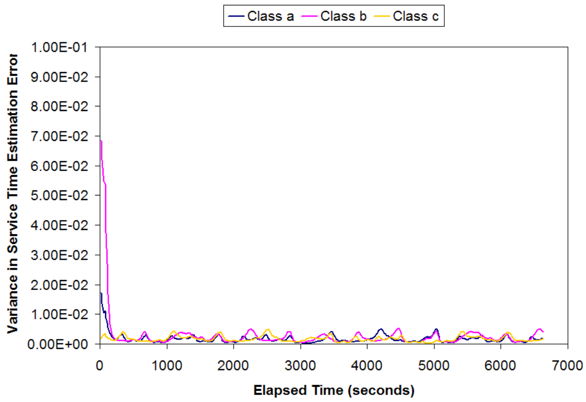

Figure 23: Improved service time estimation error for SSP experiment

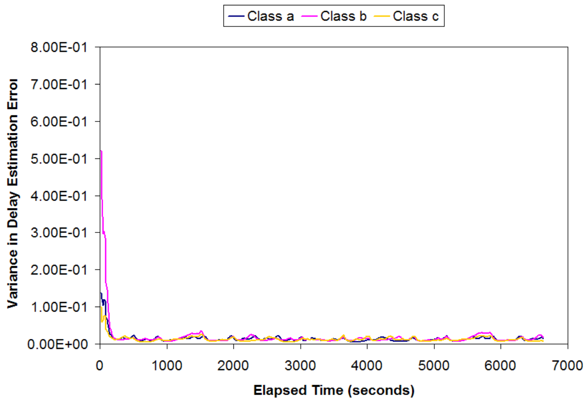

Figure 24: Improved delay estimation error for SSP experiment

1. *Low Variation in Workload (LVW) experiment*: As part of gateway management in an AS system (Figure 1), one of the goals of a typical admission control mechanism is to ensure a smooth workload profile (see [21] and references there-in). The mechanism would attempt to remove spikes in the incoming user request rate leading to a smooth workload with low variations in the admitted request rate. This was the motivation for conducting the LVW experiment in which the admission control lead to a workload with low coefficients of variation for the three classes.

2. *Step-change in System Parameters (SSP) experiment*: Reconfiguration of the software architecture, components and functions in an AS system in order to accommodate sudden changes in the execution environment can lead to step-changes in the service time and queueing delay parameters. This was the motivation for conducting the SSP experiment in which the servlet parameters were altered to simulate adaptive reconfiguration.

5.1 Low Variation in Workload (LVW) Experiment

Consider the scenario when the coefficient of variation for inter-arrival time of incoming workload is much less than one. Such was the case for the LVW experiment whose workload profile is shown in Figure 3. The three different time series depict the transactional workload for classes a, b and c. Their coefficients of variation are 0.176, 0.143 and 0.104 for classes a, b and c, respectively. Figures 4 and 5 show the response time and CPU utilization measurements. All three measurements were taken as per recommendations for the sampling time T provided in [15, 16, 17] papers. The three sets of data were fed in to the Java implementation of the filter proposed in Section 3 to estimate service time and network delay parameters for the three classes. Figures 9 and 10 show the computed estimates. The flat horizontol lines in both plots are the expected, actual values of service time and delay, as a result of the chosen servlet parameters for the compute-sleep micro-benchmarking servlet. The filter was tuned based on the recommended values for Q and R matrices provided in [15, 16, 17] papers. In spite of following the tuning recommendations, both service time and delay estimates in Figures 9 and 10 tend to converge to the actual values very slowly. Even after 6000 seconds of elapsed time the estimates for all classes have not reached the actual values. As a quantitative measure of the performance of the filter, Figures 11 and 12 show the variances in estimation error which are essentially the diagonal elements of $\boldsymbol{P}_{k|k}$ matrix as time step k evolves. Clearly the variances do not converge and instead gradually increase and only slightly decrease thereafter. Qualitatively, non-convergence of the variances to a low steady-state value indicates the 'badness' of the estimates [14].

5.2 Step-change in System Parameters (SSP) Experiment

Consider here the scenario when there is a step-change in the actual parameters caused by adaptive reconfiguration of the software. Such an adaptive reconfiguration allows the software to adapt to changes in the incoming workload, execution environment, etc. We carried out the SSP experiment in which this step-change was simulated by manually exchanging the servlet parameters twice between two of the classes a and c. Instead of a mere exchange, the parameters for the two classes could have also been changed to other different values. However, exchange of parameters was done for the sake of simplicity and easy presentation.

Figures 6, 7 and 8 show the workload, response time and CPU utilization measurements for this experiment. Notice the periodic and high varying nature of the workload. Figures 17 and 18 show the service time and delay estimates using the EKF proposed in Section 3 along with their expected actual values depicted by the flat lines. Notice the manually introduced step-change in the actual values, twice. The filter was again tuned based on the recommended values for Q and R matrices and measurement sampling interval chosen as per [15, 16, 17]. In Figures 17 and 18 it is seen that in the beginning of the experiment the service time and delay estimates converge to their expected values much faster than in the LVW experiment discussed previously. Thus, having a high varying workload improves the tracking of parameters by the filter. The service time and delay estimates also follow the switch in actual values at around 3000 and 4800 seconds of elapsed time. However, after the switch the estimates take a while to get close to the new values. Though the Kalman filter detects and tracks the switch in parameters, the convergence of estimates to new values is quite slow. Figures 19 and 20 show the variances in estimation error that converge to low values, but exhibit a saw-tooth type increase and decrease pattern. The variances do not converge to a steady-state value and could be further improved.

5.3 Problem Analysis

Let us carefully analyze the convergence problems discussed in the foregoing discussion. For the single class case, Zheng et al. [15, 16, 17] demonstrated through experimental results that their Kalman filter is well able to track the step-change in parameters without any convergence problems. For a single class they used response time and CPU utilization measurements to estimate service time and user think time (instead of network delay). Thus they used two different measurements to estimate two different unknowns. This would work fine since they were trying to estimate as many unknowns as the known measurable quantities.

In general, if we have c classes then we would have $2c$ unknowns to be estimated, i.e., service time and delay (or think time) parameters for each class. Whereas, the number of measurements available would be only $c+1$, i.e., response time for each class and CPU utilization for the single server node. User request rate measurements can not be used in the left hand side of the measurement model (Equation 2) to increase the total number of knowns to $2c+1$ instead of $c+1$. Each measurable or known quantity corresponds to a constraint on the state in the measurement model. For a single class ($c = 1$), there would be sufficient measurable knowns to estimate the unknown variables. However, for multiple classes ($c \geq 2$) this would lead to an *underdetermined* system since the number of measurable knowns would be less than the number of unknowns. This would result in lack of unique solution for the filter estimates at each time step. The filter would fail to compute a unique estimate for the service time and delay parameters and instead propose feasible but undesirable solution estimates. This explains the undesirable estimates obtained in both the experiments, leading to slow convergence towards the actual expected values.

Comparing the LVW experiment with the SSP experiment, we observed in Section 5.2 that high varying workload in the latter improved the tracking of parameters. High varying workload has the possibility to generate higher number of *linearly independent* set of measurements at each time step k, as compared to low varying workload (see [22] and references there-in). This would work towards reducing the under-determined nature of the estimation problem and eventually lead to a reduced set of feasible solutions. This explains the improvement in parameter tracking in the SSP experiment.

6. MODIFIED FILTER DESIGN & IMPROVED RESULTS

The under-determined nature of the estimation problem discussed in previous section can be addressed by increasing the number of observations in the measurement model. This can be done by using measurements from recent past to construct *constraints* on the current state $\boldsymbol{x_k}$ and augmenting them as perfect measurements (see Section 7.5.2 in [14]) to the measurement model. Assuming that the state vector remains stationary over the past $l_1 + l_2 + \cdots + l_N$ sampling intervals, the state at time k must perfectly satisfy (i.e., without noise) the measurement equation based on measurements from the last $l_1 + l_2 + \cdots + l_N$ sampling intervals. Here, N is the number of constraints and each $l_i, i = 1..N$ is the number of sampling intervals whose measurements are averaged to build each of the N constraints. The measurement model for time step k can thus be augmented with a set of constraints $D\boldsymbol{x_k} = d$ in the following manner,

$$\begin{bmatrix} \boldsymbol{z_k} \\ d \end{bmatrix} = \begin{bmatrix} \boldsymbol{H_k} \\ D \end{bmatrix} \boldsymbol{x_k} + \begin{bmatrix} \boldsymbol{v_k} \\ 0 \end{bmatrix} \qquad (3)$$

where,

$$d = [d_1 \; d_2 \; \cdots \; d_N]^T \quad \text{and} \quad D = [D_1 \; D_2 \; \cdots \; D_N]^T.$$

Here,

$$d_i \;=\; \bar{\boldsymbol{z}}_{\boldsymbol{l_i}} \;=\; \frac{1}{l_i} \sum_{j=p}^{q} \boldsymbol{z_{k-j}} \qquad \forall\, i = 1..N$$

where,

$$p = 1 + \sum_{r=1}^{i-1} l_r \quad \text{and} \quad q = \sum_{r=1}^{i} l_r.$$

Similarly,

$$D_i \;=\; \bar{\boldsymbol{H}}_{\boldsymbol{l_i}} \;=\; \frac{1}{l_i} \sum_{j=p}^{q} \boldsymbol{H_{k-j}} \qquad \forall\, i = 1..N$$

where, p and q are defined above. Instead of augmenting the constraints as perfect measurements with zero measurement noise, it is also possible to generalize further and consider them as noisy measurements with a non-zero noise term in Equation 3. Also note that this augmented measurement model of Equation 3 is different from the standard theory on constrained Kalman filtering in Section 7.5.2 of [14]. The difference being that here we use actual measurements from recent past to construct the constraints instead of relying on any a-priori knowledge about constraints on the state space. The EKF design proposed in Section 3 was modified to incorporate the augmented measurement model of Equation 3. LVW and SSP experiments were repeated with this modified EKF design. The obtained results are presented after we provide some insights for choosing the values of N and each l_i.

6.1 Insights for choosing N and each l_i

The main purpose of incorporating additional constraints is to have at least as many linearly independent knowns or observations as the number of unknowns. The number of unknowns in our problem will always be $2c$. With one more additional set of constraints, i.e., $N = 1$, the number of observations will become $2(c + 1)$. This is sufficient to have a determined system if at least $2c$ of these $2(c + 1)$ observations are linearly independent. From our experience with various experiments we have observed that this is usually the case, i.e., $N = 1$ additional constraint is usually sufficient. In some experiments it was observed that $N = 1$ was not enough and two additional set of constraints were required for improved results.

Choice of each l_i is very specific to a given experiment. It depends on the rate of change of the software system parameters (due to their time-varying nature) and the sampling interval T. In fact, some systems may require updating the value of l_i at each time step k. Deriving an explicit expression for the optimal choice of l_i is a separate research problem in itself and is outside the scope of this paper.

For the experimental results that follow, values of N and l_i were chosen intuitively through empirical observation of results. They are given in this table,

Experiment	N	l_1	l_2
LVW	2	4	3
SSP	1	3	n/a

Table 1: Values chosen for the constrained EKF

6.2 Improved LVW Experiment Results

Figures 13 and 14 show service time and delay estimates for the LVW experiment using the modified EKF. $N = 2$ additional constraints were used for results in these figures. Observe the relatively fast convergence of estimates to the actual values as compared to Figures 9 and 10. The additional constraints in the augmented measurement model increase the number of linearly independent knowns. This tends to reduce the under-determined nature of the estimation problem and the filter converges to the desirable, unique solution much faster. The fluctuations of estimates around the actual values in Figures 13 and 14, reflect changes in the server node due to any background processes, context switching and fluctuating CPU cycles consumed for memory management.

With the modified EKF, estimates reach close to their actual expected values within around 700 seconds. Compare this number with Figures 9 and 10 where the estimates do not converge to their actual values even after 6000 seconds. The order of improvement here is more than 8X in terms of the time to converge. This is substantial improvement with the modified EKF design. Figures 15 and 16 show the improvement quantitatively, in terms of the variances in estimation error that converge to very low and steady-state values.

6.3 Improved SSP Experiment Results

Figures 21 and 22 show service time and delay estimates for the SSP experiment using the modified EKF. $N = 1$ additional constraint was sufficient this time since the high varying workload already contributed towards generating higher number of linearly independent set of measurements at each time step k [22]. Service time estimates here converge to the new actual values only about 200 seconds after the 2^{nd} switch in parameters which occurs at around 4800 seconds of elapsed time. Compare this with Figure 17 where it takes more than around 1200 seconds. Similarly, delay estimates converge to the new actual values in only about 300 seconds in Figure 22 as compared to more than approximately 1400 seconds in Figure 18. Thus, for SSP experiment the order of improvement is about 4X to 6X in terms of the time to converge. Figures 23 and 24 show the quantitative improvement in terms of the variances in estimation error that converge to very low and steady-state values.

The results presented here confirm that our modified EKF design is very effective in solving the convergence problems encountered with the original EKF design. Our modified design is crucial for a successful implementation of EKF for real-time performance modeling of AS systems that process multiple classes of workload.

7. CONCLUSION

Real-time performance modeling for AS systems is an emerging area of research. Prior work has demonstrated the effective use of Extended Kalman filters for tracking (estimating) model parameters in a system with single class of workload. However, in this work we have presented experiments that demonstrate the in-effectiveness of prior approaches for multiple classes of workload. We have further proposed a simple yet powerful modification to the existing filter design. Our modification eliminates the problems encountered with the original filter design, with a very high degree of improvement. To the best of our knowledge, ours is the first attempt to propose such a modification. Results presented in this paper can have a significant impact in improving the performance of Kalman filters for the purpose of real-time performance modeling of AS systems.

8. REFERENCES

[1] Solomon, B., Ionescu, D., Litoiu, M., Mihaescu, M.: A real-time adaptive control of autonomic computing environments. In: CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research. (2007) 124–136

[2] Ionescu, D., Solomon, B., Litoiu, M., Mihaescu, M.: A robust autonomic computing architecture for server virtualization. In: INES 2008: International Conference on Intelligent Engineering Systems. (2008) 173 – 180

[3] : All papers. In: SEASS '07: Proceedings of First IEEE International Workshop on Software Engineering for Adaptive Software Systems. http://conferences.computer.org/compsac/2007/workshops/SEASS.html. (July 2007)

[4] Hamann, T., Hübsch, G., Springer, T.: A model-driven approach for developing adaptive software systems. (2008) 196–209

[5] Chen, W.k., Hiltunen, M.A., Schlichting, R.D.: Constructing adaptive software in distributed systems. In: Proceedings of the 21st International Conference on Distributed Computing Systems, IEEE Computer Society (2001) 635–643

[6] Zhang, L., Liu, Z., Riabov, A., Schulman, M., Xia, C., Zhang, F.: A comprehensive toolset for workload characterization, performance modeling, and online control. Computer Performance Evaluations, Modelling Techniques and Tools (Springer-Verlag LNCS) **2794** (2003) 63–77

[7] : Application Resource Measurement - ARM. http://www.opengroup.org/tech/management/arm/

[8] Zhang, L., Xia, C., Squillante, M., III, W.M.: Workload Service Requirements Analysis: A Queueing Network Optimization Approach. In: 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS). (2002)

[9] Liu, Z., Xia, C.H., Momcilovic, P., Zhang, L.: AMBIENCE: Automatic Model Building using InferEnce. In: Congress MSR03, Metz, France (Oct. 2003)

[10] Bemporad, A.: Model-based predictive control design: New trends and tools. In: Proc. 45th IEEE Conf. on Decision and Control. (2006) 6678Ű6683

[11] Pacifici, G., Spreitzer, M., Tantawi, A., Youssef, A.: Performance management for cluster based web services. IEEE Journal on Selected Areas in Communications **23**(12) (December 2005) 2333–2343

[12] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: Analytic modeling of multitier internet applications. ACM Transactions on The Web **1**(1) (May 2007) 1–35

[13] Pacifici, G., Segmuller, W., Spreitzer, M., Tantawi, A.: Cpu demand for web serving: Measurement analysis and dynamic estimation. Performance Evaluation **65**(6–7) (June 2008) 531–553

[14] Simon, D.: Optimal state estimation: Kalman, h infinity and nonlinear approaches, John Wiley and Sons

[15] Zheng, T., Yang, J., Woodside, M., Litoiu, M., Iszlai, G.: Tracking time-varying parameters in software systems with extended kalman filters. In: CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research, IBM Press (October 2005) 334–345

[16] Zheng, T., Woodside, M., Litoiu, M.: Performance model estimation and tracking using optimal filters. IEEE Transactions on Software Engineering **34**(3) (May–June 2008) 391–406

[17] Woodside, M., Zheng, T., Litoiu, M.: The use of optimal filters to track parameters of performance models. In: QEST '05: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems, Torino, Italy (September 2005) 74

[18] Woodside, M., Zheng, T., Litoiu, M.: Service system resource management based on a tracked layered performance model. In: ICAC '06: Proceedings of the third International Conference on Autonomic Computing, IEEE Press (June 2006) 175–184

[19] Gross, D., Harris, C.M.: Fundamentals of Queueing Theory (Third Edition), Wiley-Interscience (1998)

[20] IBM: Ibm websphere extended deployment. http://www.ibm.com/software/webservers/appserv/extend/

[21] Zhang, Z., Kurose, J., Salehi, J., Towsley, D.: Smoothing, statistical multiplexing and call admission control for stored video. IEEE Journal on Selected Areas in Communications **15** (1997) 1148–1166

[22] Stewart, C., Kelly, T., Zhang, A.: Exploiting nonstationarity for performance prediction. SIGOPS Oper. Syst. Rev. **41**(3) (2007) 31–44