Tracking Time-Varying Parameters in Software Systems with Extended Kalman Filters

Tao Zheng, Jinmei Yang, Murray Woodside
Dept. of Systems and Computer Engineering,
Carleton University, Ottawa K1S 5B6,
{zhengtao | jinmeiy | cmw}@sce.carleton.ca

Marin Litoiu, Gabriel Iszlai
Centre for Advanced Studies,
IBM Toronto Lab, Canada
{marin | giszlai}@ca.ibm.com

Abstract

Autonomic control of a service system can take advantage of a performance model only if a way can be found to track the changes in the system. A Kalman Filter provides a framework for integrating various kinds of measured data, and for tracking changes in any time-varying system. This work evaluates the effectiveness of such a filter in tracking changes in performance parameters of a software system that occur at different rates and amplitudes. The time-varying system is a Web application deployed in a data centre with layered queuing resources, in which parameter variations happen at random instants. The tracking filter is based on a layered queuing model of this system, with parameters representing CPU demands and the user load intensity. Experiments were performed to evaluate the effectiveness of the filter in tracking the changes, and the requirements for the filter settings for fast and slow variations in the parameters. The target application is autonomic control of a service centre.

1. Introduction

The goal of autonomic control[18] of a computer service centre is to make controlled changes in the system configuration to offset disturbances in the workload or the system, and to maintain Service Level Agreements (SLAs). Disturbances in the workload include changes in the load intensity or the types of services requested. Disturbances in the system include failures or load imbalances, or responses to security attacks. A second goal of control is to optimize the use of resources. Control is based on observations of the ongoing Quality of Service (QoS), and of other system measures reflecting system status and activity, interpreted with the help of a performance model.

The model in turn needs to reflect the system structure and behavior and provide means to infer current and future changes in the system. Figure 1 shows an application offering a service interface to system users at the top right. System measures at the bottom drive an autonomic control loop, which tracks and updates a model, makes decisions based on the model, the SLA, and other system goals, and makes the controlled changes.

Control strategies have been described based on different kinds of performance models, including regression functions, queuing models [14] [15], and dynamic models [1][3][6][7][13]. In [12], the present authors described a hierarchical structure of models and controllers, and suggested the use of layered queuing models to quantitatively assess the effect of component and application tuning or provisioning on the performance of the application. Layered queuing models [4][5][17] are extensions of Queuing Network models [8], which capture contention for software resources such as threads and critical sections, as well as for hardware. This paper assumes the use of layered queues, which are described further in Section 2.

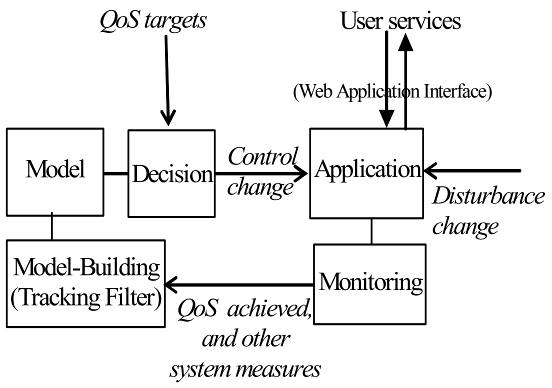
QoS targets
User services
(Web Application Interface)
Model
Decision
Control change
Application
Disturbance change
Model-Building (Tracking Filter)
Monitoring
QoS achieved, and other system measures

Figure 1 Architecture of an autonomic control loop based on a tracking model

An important aspect of the autonomic control loop in Figure 1 is the model-building element. Its role is to maintain accurate model parameters as the system evolves. In tracking changes in the model parameters, various kinds of data give useful but indirect information. Some means is needed to integrate this data, in order to estimate the model parameters. In [20], it was proposed to integrate performance data and track the parameters of queuing models with a tracking filter. A tracking filter updates past estimates of parameters from observations on functions of them (such as performance observations), based on a performance model and a statistical model of the dynamic parameter change. Tracking filters are used in many fields, but not yet in computer system control. This paper evaluates the effectiveness of the filter mechanism for tracking parameters of a model of a time-varying layered system.

The first application of a tracking filter to performance model parameter estimation in [20] used an Extended Kalman Filter to estimate the parameters of a queuing network with unknown but constant parameter values. The transient response of the filter when it first acquired the parameter value was evaluated under a wide range of conditions. The filter showed:

- Almost instantaneous convergence to the transient parameter change
- Low sensitivity to tuning parameters that describe the measurement accuracy and the parameter drift process

This work extends [20] to evaluate the success of the filtering approach to track a time-varying parameter in a layered queuing model. The novel aspects of this work are:

- Evaluation of the effectiveness of approximations needed to make the filter practical
- The use in the filter of an approximate sensitivity matrix for the layered queuing network
- The interaction of the rate of system change, the system measurement accuracy, and the length of the measurement steps, in determining the accuracy of tracking

A Kalman Filter is a model-based estimator for time-varying state values in a dynamic system that can be derived either as an optimal least-squares estimator, or a Bayesian estimator. At each step, the filter compares measured values (in our case, performance measures) to predicted values from a model, to give a prediction error **e**. From **e**, it updates the state estimates **x** with a linear update equation:

$\mathbf{x}_{\mathbf{new}}=\mathbf{x}_{\mathbf{old}}+\mathbf{Ke}$.

The *Kalman gain matrix* **K** is a function of certain properties of the model and of estimates of the accuracy of both the measurements and the model, which are also updated by the filter. **K** is updated at each step to minimize the mean of the square of the prediction error **e** (or the mean of a quadratic norm on a vector **e**). Details are given in Section 2.

Kalman Filters were originally derived for estimating states of a linear dynamic system [10], and were extended to provide an approximately optimal filter for parameter estimation and for estimating states in non-linear systems. The Extended Kalman Filter is heavily used to estimate positions in space from radar data (see [2]). There is a vast literature on Kalman Filters; the references cited provide further background.

The remainder of the paper is organized as follows. Section 2 describes a Web application and its layered queuing performance model; Section 3 explains the Extended Kalman Filter and its implementation for tracking layered queuing parameters; Section 4 presents the experiments and the results. The practical implementation issues of the Kalman Filter are detailed in Section 5, and conclusions are presented in Section 6.

2. Time-varying Web Application

We consider a Web-based application and its associated layered queuing model structure as shown in Figure 2. In terms of Figure 1, the Web application represents the controlled Application while the layered queuing model is the Model element in the autonomic control loop. In this section, we identify the structure of the performance model, the directly measurable data, the indirectly measurable data and the change characteristics of the above.

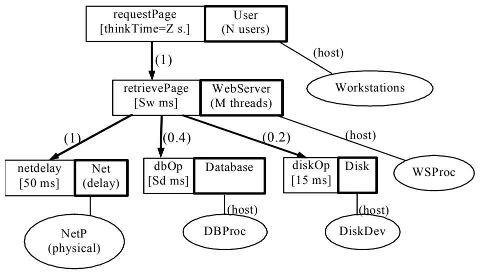

Figure 2. The Layered Queuing Model of a Web application, showing its organization

The User block in Figure 2 represents *N* separate users and their browsers, which alternately send requests to the Web Server every *Z* ms. (as a default, Z = 1000). Z is known as the think time. The WebServer block represents the server software with M threads, running on processor WSProc (indicated by the “host” relationship). The model represents M servers with a single queue of user requests. The box labeled RetrievePage represents the operation done for the users, and requires a CPU demand of S_w ms (default value 5 ms), one network latency of 50 ms, and on average 0.4 database operations and 0.2 disk operations. The disk and the database are here represented as single servers with a queue, running on their own devices DBProc and DiskDev, with CPU demands of S_d (default value 10 ms) and 15 ms, respectively.

We will consider this model as a representation of a Web-based application system with a similar structure and the same structure of requests between its components. In a real system, the behaviour of the components would be more complex than in the model, and the structure could contain more detailed substructures. For instance, a real database server would include threads, concurrency control mechanisms, and perhaps its own storage subsystem.

For this study, we will represent the real system by a simulated system with the same structure as Figure 2. In this way, we can study the success of the tracking filter in tracking the parameters of the simulated system, if they vary.

2.1 Parameter Changes

Figure 2 shows five parameters as variables:

N = number of active users (default 100)

M = number of Web-server threads (default 50)

Z = mean User think time per request (default 1000 ms)

S_w = mean Web server demand per user request (default 5 ms)

S_d = mean database server CPU demand per database request (default 10 ms)

A major challenge to autonomic systems is variation in offered load arriving at a service center. This phenomenon can be modeled by variations in *Z* or in *N*; a larger *N* or a smaller *Z* leads to a higher level of offered load. Variation in the type of transaction being executed by users is a second challenge, and it can be captured by variation in the CPU demand, or the request frequencies. CPU demands in a data centre can also vary because of provisioning (upgrading or downgrading the hardware). We will vary *Z* and the CPU demands, and keep *N* and the request frequencies constant.

We assume that the system has parameters that change over time, according to a parameter change process as follows:

- Changes in some parameter *a* occur at discrete random instants, at a mean rate of α_a changes/s, and the parameter values are constant between changes
- At a change point, the new value *a’* is independent of the previous value *a*, and is governed by a distribution with density function *f(a)*, mean m_a , variance σ^2_a, and coefficient of variation $C_a = \sigma_a / m_a$

The amplitude and frequency of this change process are characterized by $T_a = 1/\alpha_a$, the mean time between changes for a, and by C_a.

There might be other change processes, such as a smooth process of gradual increments over time.

For simplicity, we assume that only Z and S_d change. Thus, for tracking purposes, the parameter vector is $\mathbf{a} = [Z]$ or $\mathbf{a} = [S_d]$.

2.2 Performance Measures

The directly measurable performance data would be taken, in a real system, from instrumentation and operating system counters. In our simulation, we consider:

- Mean response time to users (R)
- Utilization of the Web server processor(U_w)
- Utilization of the database processor(U_b)
- Utilization of the disk(U_d)

Thus the measurement vector is $\mathbf{z} = [R, U_w, U_b, U_d]$, averaged over a measurement time interval of length T.

Other measures might be of interest, such as quantiles of response time, or the probability of exceeding a stated target response time. However, the measures above are well understood and will give us a first view of the capability of tracking.

Values for the think time Z, or the CPU demands S_w or S_d are not directly accessible at run time. They also vary over time, so we compute and track them indirectly by using the Extended Kalman Filter and the layered queuing models. The next section describes the tracking filter, which deduces these hidden measures from the measures that are available.

3. The Tracking Filter

The filter takes the standard form of an Extended Kalman Filter (EKF) as described, for instance, in [2]. It applies to cases where there is a model $\mathbf{x}_{\text{new}} = \mathbf{f}(\mathbf{x}_{\text{old}})$ for the evolution of the desired state-and-parameter vector $\mathbf{x}$, and a model $\mathbf{z} = \mathbf{h(x)}$ for the relationship between the observation vector $\mathbf{z}$ and $\mathbf{x}$. Here, we replace $\mathbf{x}$ by our unknown parameter vector $\mathbf{a}$. In Kalman's classic paper [8] the relationships $\mathbf{f}$ and $\mathbf{h}$ were linear and an optimal (least-squares) estimator was derived; in the extended filter the relationships are nonlinear and the optimality is only approximate. In our case, $\mathbf{f}$ is the identity, but $\mathbf{h}$ is a nonlinear function.

In the discrete time filter used here, time advances in steps of duration T, indexed by a step counter k. The change process of the parameter vector $\mathbf{a}_k$ is modeled as a drift driven by random increments:

$$\mathbf{a}_k = \mathbf{a}_{k-1} + \mathbf{w}_{k-1} \quad (1)$$

The random vector $\mathbf{w}_k$ has a mean of zero and has the disturbance covariance matrix $\mathbf{Q}_k$ (which we assume to be a constant $\mathbf{Q}$), and is independent from one step to the next.

The system observation vector $\mathbf{z}_k$ is modeled as a function $\mathbf{h}(\mathbf{a}_k)$, defined in our case by the relationship that determines the performance from the parameters, including an error of measurement. Thus, it assumes:

$$\mathbf{z}_k = \mathbf{h}(\mathbf{a}_k) + \mathbf{v}_k \quad (2)$$

The assumed random error vector $\mathbf{v}_k$ has a mean of zero, is independent from one step to the next, and has the measurement error covariance matrix $\mathbf{R}_k$.

The filter assumes that the relationship $\mathbf{h}$ is given by a performance model, the layered queuing model for the system, which includes the parameter vector $\mathbf{a}$.

3.1 The Filter Computations

The filter computations are recursive, beginning from an initial estimate $\mathbf{a}_0$, and an initial error covariance matrix $\mathbf{P}_0$. Each recursive step can be summarized as follows:

(1) Based on the most recent parameter estimate $\mathbf{a}_{k-1}$, the filter predicts the measurements as $\mathbf{h}(\mathbf{a}_{k-1})$ (because the assumed drift has zero mean, the predicted parameter value is the same as the previous estimate). From the current observation vector $\mathbf{z}_k$, it computes a prediction error vector $\mathbf{e}_k$:

$$\mathbf{e}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{a}_{k-1}) \quad (3)$$

(2) The core filter calculation is the update of the estimates by the linear feedback equation:

$$\mathbf{a}_k = \mathbf{a}_{k-1} + \mathbf{K}_k \mathbf{e}_k \quad (4)$$

where the "Kalman Gain" matrix $\mathbf{K}_k$ is computed as follows:

(a) Computation begins from an estimate $\mathbf{P_k}$ of the covariance matrix of estimation errors for $\mathbf{a_k}$. $\mathbf{P_k}$ is projected forwards one step, based on the drift covariance matrix $\mathbf{Q}$:

$$\mathbf{P^-_k} = \mathbf{P}_{k-1} + \mathbf{Q} \qquad (5)$$

(b) Then the optimal gain matrix $\mathbf{K_k}$ (which is only suboptimal when $\mathbf{h}$ is a nonlinear function, as it is here) is given by:

$$\mathbf{K}_k = \mathbf{P^-}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P^-}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \qquad (6)$$

In this equation, the matrix $\mathbf{H_k}$ is the matrix of partial derivatives of the performance model function $\mathbf{h}$, with respect to the parameters $\mathbf{a}$ at their current values $\mathbf{a_{k-1}}$. Thus, $\mathbf{H_k}$ is a matrix of sensitivity values for the performance model.

(3) When $\mathbf{a_k}$ is updated, the covariance estimate $\mathbf{P_k}$ is also updated, to take into account the improved accuracy after the filter step:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P^-}_k \qquad (7)$$

Where there is no ambiguity, the step subscript k will be omitted. Figure 3 shows the organization of the filter.

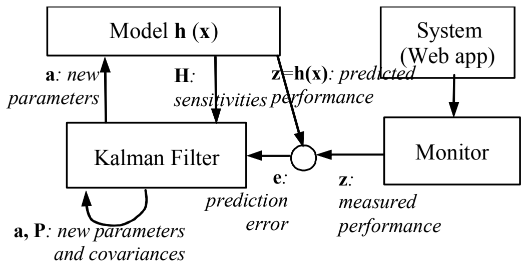

Figure 3. The Kalman Filter architecture

The optimality and convergence properties of the EKF depend on the way the functions are linearized around the current estimate of $\mathbf{a}$ [11]. This Extended Kalman Filter (EKF) [2][19] linearizes $\mathbf{f(a)}$ and $\mathbf{h(a)}$ by a first order Taylor series around the state estimate $\hat{a}^-_{k-1}$ and does not take linearization errors into account. A variant called the Iterative Kalman Filter (IEKF) linearizes $\mathbf{h(a)}$ around the predicted state estimate $\mathbf{a_{k-1}}$. Other variants of the filter, like the Unscented Kalman Filter [9] or the Divided Difference Filter [16] capture the linearization errors in the covariance matrices. They were shown in [11] to provide better estimates when dealing with non-linear $\mathbf{f(a)}$ functions, while EKF and IEKF provide better performance when dealing with non-linear $\mathbf{h(a)}$.

3.2 The Influence of the Filter Parameters

The matrices $\mathbf{Q}$ and $\mathbf{R}$ capture knowledge or assumptions about the disturbances and the measurement errors, and they also influence how the filter reacts to new data. Both $\mathbf{Q}$ and $\mathbf{R}$ can often be assumed to be diagonal, with variance terms for the one step disturbances and the measurement errors, respectively.

Small values in $\mathbf{Q}$ indicate that only small changes are expected, and lead to a small filter gain matrix $\mathbf{K}$ that can only adapt slowly. A large value of $\mathbf{Q}$ leads to large $\mathbf{P}$ and thus large gains that might overreact to measurement errors.

Each diagonal element Q_{ii} should be set to the square of an estimate of the magnitude of the changes to be tracked in parameter $\mathbf{a_i}$:

$$\mathbf{Q_{ii}} = (\text{approx. magnitude of change in } \mathbf{a_i})^2$$

Each diagonal element $\mathbf{R_{ii}}$ should be an estimate of the variance of the measurement error in $\mathbf{z_i}$. If the averaging time T is large enough (which we shall assume is the case) $\mathbf{R}_{ii}$ varies as:

$$\mathbf{R}_{ii} = \text{const}/T \qquad (8)$$

A standard step-length T* was determined (by experiment) that gave a 95% confidence interval of +- 5% in the user response time measure z_1. For the system in Figure 2 with the default values of the parameters, T* = 15.7s. From the asymptotic properties of the t distribution, the confidence interval is 1.96 times the standard deviation. This implies that when T = T*,

$$\mathbf{R}_{11} = (0.05\ (\text{mean of } z_1)/1.96)^2.$$

For other values of T, the ratio of T to T* is denoted by γ_T:

$$\gamma_T = T / T^*$$

and then, approximately:

$$\mathbf{R}_{11} = (0.025\ (\text{mean of } z_1))^2 / \gamma_T$$

This value could be used in the filter, with the model prediction to estimate the mean of z_1.

Further, it can be assumed that the confidence intervals of the other measures have similar accuracy. Thus:

$$\mathbf{R}_{ii} = (0.025\ (\text{mean of } z_i))^2 / \gamma_T \qquad (9)$$

4. Results

To demonstrate the ability of the filter to follow parameter changes, the system in Figure 2 was simulated with deterministic and random parameter changes (disturbance changes). A tracking filter was set up, based on the same model solved by an approximate analytic calculation with the LQNS solver.

The filter was driven by the measurement vector defined above, made up of the user response time and the device utilizations:

$$\mathbf{z} = [R, U_w, U_b, U_d]$$

These are typical of readily available performance measures from a real system.

Changes in a single parameter were tracked, with either Z or S_d.

The goodness of tracking can be measured in two ways, by the performance prediction error E_R or by the parameter tracking error E_A. We will use the RMS (root-mean-square) tracking error measures for both of these quantities.

4.1 Tracking Deterministic Changes in Parameters

The tracking performance was recorded for a series of alternating step changes in value of two parameters:

- User think time Z (which affects the arrival rate; smaller Z gives a higher arrival rate)
- Database service time S_d

Case 1: Z alternates between 500 ms and 2500 ms with a change every 471 s. (30T*). This creates a much larger arrival rate for small Z than for large Z (about 168/s when Z = 500, vs 39/s when Z = 2500). Equivalently, we could have modified the arrival rate directly.

The filter parameters were set to:

- T = T* = 15.7 s. (making $\gamma_T = 1$). This gives an estimation accuracy such that the 95% confidence interval in the mean user response time is +-5% at the base case parameters.
- Q = 4,000,000. Q is a scalar, since there is just one parameter to track, and this is the square of the step change in Z that is applied in going from 500 to 2500.
- $\mathbf{R}_k$ was a diagonal matrix with an element for each performance measure. From Eq. (9), the i^th element is $\mathbf{R}_{k,ii} = (0.025 * h_i(\mathbf{a}_k))^2$

Figure 4 shows a fragment of the record of values taken from the simulation and the tracking filter. The filter tracks the change in Z with a one-step delay plus a few steps to settle to the new value.

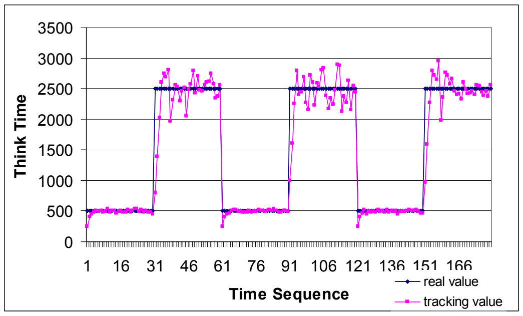

Figure 4 (Case 1) Tracking performance for a deterministic sequence of mean user think times. (T = T*)

The high and low values of Z give a moderate load and a heavy load on the system, respectively. Thus the performance calculations traverse the knee in the performance curves between these two regimes, where the function **h** is the most nonlinear.

The tracking error is visibly greater for Z = 2500. The arrival rate is lower, and there are fewer response times in the averaging period, so the variance of the measurement error is larger than is allowed for by **R** when it is calculated by Eq. (9). The variation of measurement errors with load is quite a complex phenomenon, and Eq. (9) makes the simplifying assumption that the relative accuracy is constant in a neighborhood of the configuration for which it was measured (in establishing T*). This assumption allows constant values to be used for **Q** and **R**.

The assumption is justifiable if the sensitivity to R is low. Figure 4 supports the assumption, in that the tracking errors at both extremes are moderate. Further tuning of **Q** and **R** might give even better performance.

Case 2: The database demand S_d alternates between 10 ms and 40 ms, with changes every 471 s. At the lower value, the system is lightly loaded; the higher value creates a significant load at the database, with a queuing delay that blocks some application threads.

Q was set to 900, and **R** was set as in Case 1.

Figure 5 shows how the filter tracked the changes, corresponding to Figure 4 for Case 1. Again, the filter takes a few steps to track the (very large) change.

In this case, the larger tracking errors are evident for the larger value of S_d, which corresponds to a heavier load (as opposed to the case above in which the larger value of Z gives the lighter load). This time the number of responses in an averaging period decreases with heavy load, since the delays at the server back up the traffic. Also, there is a general tendency for the accuracy of statistics to suffer as system load increases, because of increased correlation of the successive responses. This dependency is complex and was not accounted for in setting **R** according to Eq. (9).

Again the system is traversing through the most nonlinear range of the performance relationships expressed in **h(a)**.

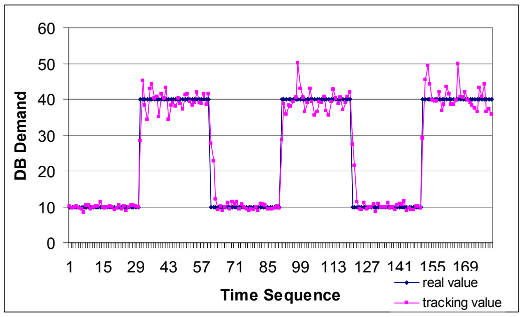

Figure 5. (Case 2) Tracking of deterministic changes in the database service time S_d .($T = T^*$)

4.2 Tracking Random Changes in Parameters

A random change process was generated in the simulation, with step changes of a parameter value occurring at randomly chosen instants (multiples of a common time step), at a mean rate of α changes/s. The change process was applied to one parameter at a time, first to Z and then to S_d.

Figure 6 shows a fragment of a trace of the filter tracking random changes in the mean User think time. Sometimes the "real" mean value used by the simulation (the line with the diamonds) changes in the middle of a measurement step, so there is a point between two values. Generally, the filter follows a change within a few steps.

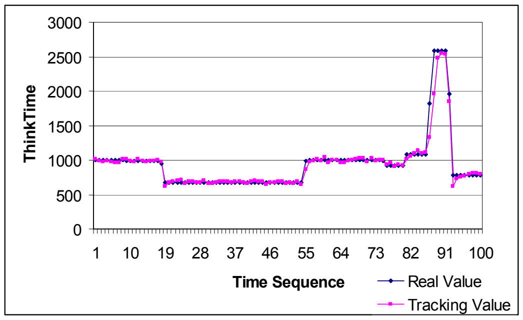

Figure 6. A trace fragment of tracking random changes.

Experiments were done for a range of values of the measurement step time T, the rate of changes α, and the variance of the changes σ^2. The time T*, which is characteristic of the system and designates the time to get a moderately accurate average by measurement, was used to normalize these values.

The *normalized relative parameter change rate,* γ_α, specifies the change rate relative to T*:

$$\gamma_\alpha = \alpha T^*.$$

As already defined, the *normalized relative measurement interval* γ_T specifies the measurement interval T relative to the time T*:

$$\gamma_T = T/T^*$$

Base values of these parameters in the following experiments were $\gamma_\alpha = 0.025$ and $\gamma_T = 4$.

At each change instant, a new value of the parameter (Z or S_d) was chosen independently according to a shifted hyper-exponential distribution (such as: Z = constant + random part), which had a mean value equal to the average value of the parameter and a stated coefficient of variation C. The base value of C was C = 1, but in Case 7, C was varied from 0.1 to 2.

Case 3: Parameter Tuning.

The filter tuning parameters **Q** and **R** might affect the way the filter reacts. Small entries in **Q** make the filter conservative, as it assumes only small changes are possible in a single step. Small entries in **R** make the filter track more aggressively, as it assumes that the measurements are accurate and therefore the filter must react in order to explain them. We must learn how to set these parameters, and also it is important to understand how sensitive the whole filter process is to their values.

To investigate these effects, the entries of **Q** and **R** were multiplied by factors denoted as *QFac* and *RFac* respectively. These two factors were varied over two orders of magnitude. Otherwise, the experiments had the usual base values of γ_α = 0.025, γ_T = 4.0 and CV_a= 1.0. The expressions used for **Q** and **R** were:

$$\mathbf{Q}_{ii} = (QFac(mean\ of\ a_i)CV_{ai})^2 \qquad (9a)$$

$$\mathbf{R}_{ii} = ((RFac)\,(z_i) / 1.96)^2/\gamma_T \qquad (9b)$$

Table 1. (Case 3) The RMS tracking error in the mean user think time Z, as R and Q are varied

RFac	QFac							
	0.01	0.025	0.05	0.1	0.25	0.5	1	2
0.02	124.6	140.0	143.7	143.7	143.7	143.8	144.6	2.72E7
0.05	156.4	124.6	131.2	143.7	143.8	143.8	143.8	143.6
0.1	186.4	145.9	124. 6	131.2	143.8	143.8	143.8	143.8
0.2	221.1	176.1	145.9	124.6	140.0	143.8	143.8	143.8
0. 5	264.8	221.1	186.4	156.4	124.6	131.1	143.7	143.8
1	290.6	255.1	221.1	186.4	145.9	124.6	131.2	143.7
2	317.0	282.6	255.1	221.8	176.1	145.9	124.6	131.0
4	345.1	308.1	282.6	255.1	290.9	176.1	145.9	124.6

The results in Table 1 show that it is the ratio of **Q** to **R** that is important, rather than the values of the parameters. Also, above the diagonal (when **Q** is too large, and the filter over-responds to measurement errors), there is only a modest effect up to the point in the top right corner, where the error explodes. On the other hand, when **Q** is too small (the filter is sluggish), the error increases steadily.

We can conclude that the tracking performance is somewhat insensitive to **Q** and **R**. Around the ideal balance between **Q** and **R**, there is a wide band (more than a factor of 10 up or down) in which the filter is "not bad" (within a factor of 2 in RMS tracking performance). This agrees with the results reported in [20] for transient response and queuing models. Furthermore, it is better if **Q** should be somewhat overestimated (rather than underestimated) relative to **R.**

For the rest of the paper, we set *QFac* =0.1 and *RFac* = 0.2.

Case 4: Measurement Time

The next investigation considers how the measurement step time affects the accuracy of tracking. The mean number of parameter changes per measurement step is given by the ratio αT, for a given parameter change process with mean rate of α changes/s. We expect that low values of this ratio, such as αT << 1, will be necessary for good tracking, but we are also interested in determining the relationship between αT and tracking. Given a measurement step T, the results will show how fast a parameter change process can be tracked. In terms of the normalized values γ_T and γ_α, $\alpha T = \gamma_T \gamma_\alpha$.

The mean user think time *Z*, and thus the request arrival rate, changed at random instants as described above. *Z* was chosen from a distribution with mean 1000 ms. and C_Z = 1 (thus, Q was set to 10^5). The relative change rate was set to γ_α = 0.025, which makes the average time between changes 40T*. The relative measurement interval was varied over the range γ_T = [0.4, 40], so that the smallest value gives measurement intervals of relatively poor accuracy, while the largest value is so long that it equals the mean change time.

Figure 7 plots the RMS tracking error (Ea) in the estimate of *Z* from 1000 measurement and tracking steps and also the RMS prediction error (Er) for the user response time coming from the estimated model. The horizontal axis again gives log10 γ_T, which ranges from 0.4 to 40.

The longer measurement intervals clearly are too slow and fail to keep up with changes; at γ_T = 40 (the right-hand end) there is on average one change per measurement interval, so the filter never has a chance to settle. The shorter intervals give lower measurement accuracy, but have the opportunity to smooth more.

Up to γ_T = 4 (with log = 0.6), the parameter tracking is quite good, with RMS error Ea of about 100, which is 10% of the mean value m_Z = 1000 ms. It is also insensitive to γ_T, which suggests that the key factor for tracking is to make the measurement interval much shorter than the inter-disturbance time (frequent measurements are better than accurate ones). According to Figure 6, there should be 10 measurement and tracking steps between disturbance changes. With fewer, longer measurement intervals, accuracy deteriorates steadily.

The RMS prediction error (Er) has the same trend as the RMS tracking error. In the "good tracking" regime at the left, Er is about 5, which is less than 10% of the mean response time of this system in light loads.

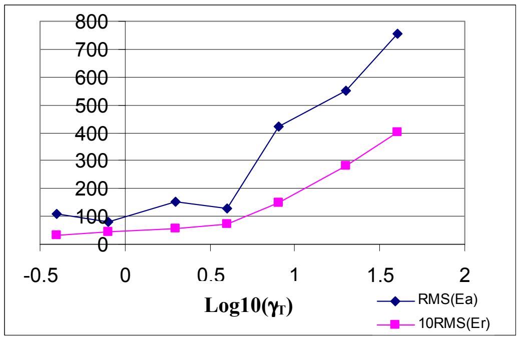

Figure 7. (Case 4) RMS errors in tracking random changes in user think time Z, for different lengths of the measurement interval.

Case 5: Rate of Disturbance Changes

The rate of change of the User think time was varied, so the normalized rate $\gamma_\alpha = \alpha T^*$ varied from 0.01 to 1 while $\gamma_T = 4$. For γ_α less than 0.04 (which gives one change every 10 measurement steps) the RMS tracking error is again around 50, which is only 5% of the mean value of *Z*. Up to γ_α =0.1 (one change every 4 measurement steps) it is still less than 130, or 13% of the mean. Above this point, the error increases rapidly.

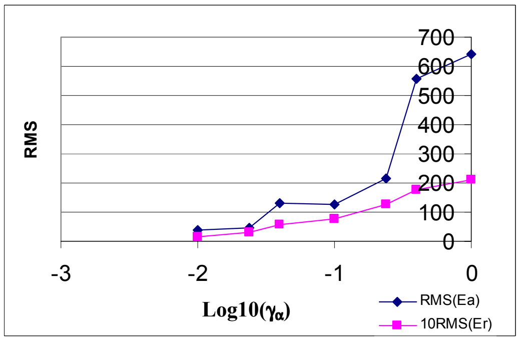

Figure 8 (Case 5) The RMS errors in *Z* as γ_α varies from 0.01 to 1

The error in predicting the mean user response time shows a similar trend. The filter gradually loses the capability to predict accurately, when the disturbance change rate is faster than one every tenth measurement step.

This confirms the evidence in Case 3 when the disturbance rate was held constant and the measurement step was varied.

Case 6: Disturbance to Service Demand

Disturbances to service demands can arise when the users of a system swing to using different functions or using functions differently, for example to requiring larger database searches for each operation.

The mean database server demand S_d changed randomly, using the same random disturbance process as for *Z* in Case 4 and 5 above. The average value is m_{Sd} = 10 ms., and $C_{Sd} = 1$.

As in Case 4, the normalized measurement time interval was varied from γ_T = 0.4 to 40, and the RMS tracking and prediction errors were determined.

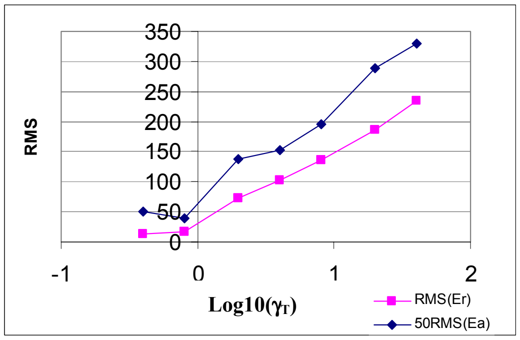

Figure 9 (Case 6). RMS errors in tracking random changes in S_d, as the measurement step (and γ_T) change.

Figure 9 shows quite small tracking errors for γ_T below 1 (Ea is about 1, or again about 10% of the mean parameter value m_{Sd} = 10 ms.). In the "good" range, the response time prediction error (Er about 50) is similar to previous cases.

Case 7: Different Disturbance Variances C

This final case considers different amplitudes of the random changes to the think time *Z*. The coefficient of variation C_Z of the disturbances to *Z* was varied over a range from 0.1 (quite small changes) to 2. The other parameters took the base values.

In Cases 4 and 5 the changes to *Z* around the mean of 1000 ms. had a standard deviation of 1000 also; here the standard deviation $\sigma_Z = m_Z C_Z$ varies from 100 to 2000. As expected, Figure 10 shows that the RMS prediction error is smaller when the disturbances are smaller.

For small values of C_Z ($C_Z < 0.3$) the tracking error approaches a constant value; this is because the filter responds to measurement errors even when disturbances are very small. Up to $C_Z = 1$, the errors are still moderate. At $C_Z = 2$, they explode to very large values.

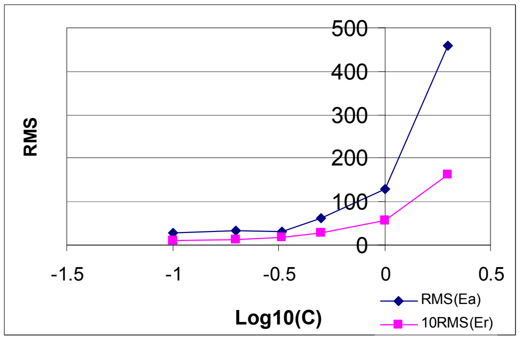

Figure 10. (Case 7) Tracking results for different values of C

5. Guidelines for Practical Implementation

One of the most important issues in implementing the filter is the choice of the size of measurement interval T. This interval is not only going to affect the performance of the filter, and hence the accuracy of the layered queuing model, but also it will dictate the timing for the actions of the Decision element (see Figure 1). Regarding the latter relationship, in classical control, the control and measurement intervals are the same: if the Decision element actuates the system every second, the decision is based on the measurements taken in a second-long time window. The experiments reported in the paper shows that the performance of the filter is dependent on the size of the measurement interval, with an optimum around a system constant T*.

T* is approximately 15.7s for the case study of this paper. The value of T* can be determined by experiment on the system to be controlled, by estimating the accuracy of averages over any period T and then applying Eq (8).

In practical implementations of the filter, what is important is not the length of the measurement in physical time units, but the number of events we observe, such as user interactions, which in turn triggers new measurable performance metrics such as response time or utilization. The system constant period of 15.7 s in our example translates to about 1500 measured response times, which suggests that the observation window should be set in the order of thousands measurable events. This result is also consistent with the results reported in [20]. However, this observation window might be too large for the Decision element and might allow the errors to accumulate. To compensate for that, the filter and the Decision elements might work as follows:

- A large observation window T could be chosen, with about 5000 events; the first control change can happen after the first 5000 events.
- A smaller control-step window Δ could be chosen, with about 100 or 500 events. The tracking and control calculations could be repeated every Δ seconds, using data from the past T seconds.

Other parameters needed for the implementation of the filter are the covariance matrices **Q** and **R** that characterize the errors of the model and of the measurement process, respectively. Both are diagonal matrices representing independent errors and disturbances. We found that it is best to update **R** at every filter step k, taking into account the measured values as in Eq. (9). Since **Q** represents expectations about the disturbance process, it also could be updated based on recent tracking experience. However, we have not experimented with this parameter.

The experiments of Case 3 showed that it is the ratio of **Q** to **R** that matters, not the values separately, and that **Q** can be overestimated (or **R** can be underestimated) by up to 2 orders of magnitude with only a slight effect on tracking quality.

The initial values for the estimated vector **a** as well as the structure of the layered queuing model **h(a)** must be chosen prior to starting the filter. This can be done by tracing or monitoring the software system under control. Experiments conducted in [20] showed that the convergence of the filter depends only slightly on the initial value $\mathbf{P_0}$; the filter has a good convergence when $\mathbf{P_0}$ is a diagonal matrix with the diagonal elements the square of the initial estimates **a**.

The sensitivity matrix $\mathbf{H}_k$, is computed at every step k as the numerical approximate derivatives of measured variables represented in **z,** computed at the estimated parameter values **a**. The numerical derivatives are calculated here by using the layered queuing model LQNS [4], with the results denoted **h(a)** in the equations (1)-(7). The elements of **a** were varied one at a time by an increment δ, and the model was solved to find the differences in the vector **z**. We found it was essential to use double precision arithmetic in the matrix operations of the filter.

A simplified version of the LQM calculations might provide adequate accuracy for the sensitivities, more quickly. However, for the system in Figure 2 the LQNS solution takes only a few milliseconds.

6. Conclusions

The most important conclusion from these results is that an Extended Kalman Filter is a practical tool for tracking the parameters of a time-varying layered queuing system. Its parameters can be set from information and assumptions about the system that are reasonable and not too difficult to make, and tracking performance is adequate over wide ranges of the parameters (not very sensitive).

Various filter modifications that have been found useful in other settings should still be investigated for the application to performance models.

Acknowledgements

This research was supported by the IBM Centre for Advanced Studies (CAS), Toronto, and by the Natural Sciences and Engineering Research Council of Canada.

About Authors

Tao Zheng is a PhD student from Carleton University. He received his Master degree from Carleton University in 2002. His research interests include: model building, tracking and optimizing for real time and distributed systems; performance modeling of distributed software, based on layered queuing network (LQN) models.

Jinmei Yang is a Research Assistant with the Real-Time and Distributed Systems Lab (RADS Lab) at Carleton University, Canada. She received her M.S. degree in Computer Engineering from Carleton University in 2005. Prior to joining RADS, she worked as wireless network engineer in China Mobile Communication Company. Her research interests include computer communication and networks, simulation and modeling, performance evaluation and QoS control.

Murray Woodside received the Ph.D. in Control Engineering from Cambridge University, England. He has taught and done research in stochastic control, optimization, queuing theory, performance modeling of communications and computer systems, and software performance. His current interests are software engineering and performance engineering of distributed systems and telecommunications software. In the period 1995 – 1999, he was Vice-Chair and Chair of SIGMetrics of the ACM. He is also Thrust Leader for the Software thrust of TRIO, the Telecommunications Research Institute of Ontario.

Marin Litoiu is Senior Research Staff Member with the Centre for Advanced Studies at the IBM Toronto Laboratory. He received his PhD degree from Carleton University, Ottawa, and holds a doctoral degree from University Politechnica of Bucharest(UPB). Prior to joining IBM (1997), he was a faculty member with the Department of Computers and Control Systems at the UPB and held research visiting positions with Polytechnic of Turin, Italy, (1994 and 1995) and Polytechnic University of Catalunia (Spain), and the European Center for Parallelism (1995).

Gabriel Iszlai is a Senior Developer with the IBM Tivoli On Demand group in Toronto, Canada. He received his B.S. degree in 1992. Prior to joining IBM he worked as a Network Architect for ThinkDynamics, a company acquired by IBM in May 2003. He was one of the initial designers of the former ThinkControl application, known today as IBM Tivoli Intelligent Orchestrator. Prior to that, he worked for over 8 years in the IT industry for different European telecom companies.

References

[1] Tarek Abdelzaher, Kang G. Shin, Nina Bhatti, ``Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 1, Jan 2002.

[2] E. Brookner, *Tracking and Kalman Filtering Made Easy*, Wiley Interscience, 1998.

[3] Yixin Diao, Xue Lui, Steve Froehlich, Joseph L. Hellerstein, Sujay Parekh, Lui Sha, "On-Line Response Time Optimization of An Apache Web Server," *International Workshop on Quality of Service*, 2003.

[4] R.G. Franks, S. Majumdar, J.E. Neilson, D.C. Petriu, J.A. Rolia and C.M. Woodside, "Performance Analysis of Distributed Server Systems," *Proc. Sixth International Conference on Software Quality*, Ottawa, Canada, October 28-30, 1996, pp. 15-26.

[5] Greg Franks, "Performance Analysis of Distributed Server Systems," PhD. thesis, Carleton University, Jan. 2000.

[6] Neha Gandhi, Joseph L. Hellerstein, Sujay Parekh, and Dawn M Tilbury, "Managing the Performance of Lotus Notes: A Control Theoretic Approach," *Proceedings of the Computer Measurement Group*, 2001.

[7] Hellerstein J., Diao Y., Parech S., Tilbury D., *Feedback Control of Computing Systems*, John Wiley &Sons, Inc., 2004.

[8] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons Inc., 1991.

[9] S. Julier, J. Uhlmann, H.F. Durant-Whyte, "A new method for approximating nonlinear transformations of means and covariances in filters and estimators," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 477-482, March 2000.

[10] R.E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of ASME, Journal of Basic Engineering*, vol. 82, pp 34-45, March 1960.

[11] T. Lefebvre, H. Bruyninckx, and J. De Schutter, "Kalman filters for nonlinear systems: a comparison of performance," *Internal Report 01R033,* KU Leuven, 2001, http://people.mech.kuleuven.ac.be/~tlefebvr/publicatie.htm.

[12] Litoiu M., Woodside M., Zheng T., "Hierarchical model based autonomic control of software systems," *Proceedings of Design and Evolution of Autonomic Software (DEAS'05) Workshop*, St. Louis, USA, May 2005.

[13] Ying Lu, Tarek Abdelzaher, Chenyang Lu, Lui Sha, Xue Liu, ``Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers," *Real-Time and Embedded Technology and Applications Symposium,* Toronto, Canada, May 2003.

[14] D. A. Menasce, M. Bennani, "On the Use of Performance Models to Design Self-Managing Computer Systems," *Proc. 2003 Computer Measurement Group Conference*, Dallas, TX, Dec. 7-12, 2003.

[15] D. A. Menasce, "QoS-aware software components," *IEEE Internet Computing*, March/April 2004, Vol. 8, No. 2.

[16] M. Norgaard, N.K Poulsen, and O. Ravn, " New developments in state estimations for nonlinear systems," *Automatica*, vol 36, no. 11, pp. 1627-1638, November 2000.

[17] J. R. Rolia and Kenneth Sevcik, "The method of layers," *IEEE Transactions on Software Engineering*, Vol. 21, No. 8, pp. 689-700, 1995.

[18] L. Stojanovic, J. Schneider, A. Maedche, S. Libischer, R. Studer, Th. Lumpp, A. Abecker, G. Breiter, and J. Dinger, "The role of ontologies in autonomic computing systems," *IBM Systems Journal*, v. 43, n. 3, 2004.

[19] H Tanizaki, *"Nonlinear Filters: Estimation and Applications- Second, Revised and Enlarged Edition,"* Springer-Verlag, Berlin-Heilderberg, 1996.

[20] Murray Woodside, Tao Zheng, Marin Litoiu , "The use of optimal filters to track parameters of performance models," *Proc. 2nd Int. Conf. on Quantitative Evaluation of Systems (QEST05)*, Torino, Sept. 2005.