The Use of Optimal Filters to Track Parameters of Performance Models

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

Marin Litoiu
Centre for Advanced Studies,
IBM Toronto Lab
Canada
marin@ca.ibm.com

Abstract

Autonomic computer systems react to changes in the system, including failures, load changes, and changed user behaviour. Autonomic control may be based on a performance model of the system and the software, which implies that the model should track changes in the system. A substantial theory of optimal tracking filters has a successful history of application to track parameters while integrating data from a variety of sources, an issue which is also relevant in performance modeling. This work applies Extended Kalman Filtering to track the parameters of a simple queueing network model, in response to a step change in the parameters. The response of the filter is affected by the way performance measurements are taken, and by the observability of the parameters.

Keywords

Autonomic systems, Software performance, Parameter Tracking Performance Modeling, Layered Queuing, Model Building

1. Introduction

Autonomic self-optimizing systems have two kinds of goals: they must maintain adequate quality of service (QoS), typically defined by service-level agreements (SLAs), and they may also seek to provide efficient operation, using a minimum of resources. Typically, priority is given to maintaining the SLA as performance constraints at all times, while seeking the minimum cost.

Basic autonomic capability is provided by a controller with these functions:

- *monitoring* performance variables,
- *deciding* to change system management variables,
- *executing* the decision.

The changes in the system are based on a description or model of the system. Dynamic regression models have been described by several researchers: by Hellerstein and co-workers [3] [5] to control the memory used by an IBM® Lotus® Notes® application, and by Abdelzaher and co-workers [1] [11] to adjust the number of threads in Web servers. Queuing network models were described by Menasce [12] as predictive models for self-optimization, with a corresponding architecture [13].

In [10] the present authors have proposed a hierarchical autonomic architecture to control QoS for a Web services system, in which the workload may change with time. It combines dynamic models for workload change, layered queuing models (LQM) for performance prediction and threshold-based control to seek self-optimization. Each level in the hierarchy uses performance monitoring to provide a tracking model, and model-based decision-making for control. One level of the hierarchy is illustrated in Figure 1.

The model follows the changes in the workload imposed by users, and the Decision element searches the model to determine changes that will maintain QoS contracts in an economical way. This provides self-tuning, self-balancing, and self-provisioning.

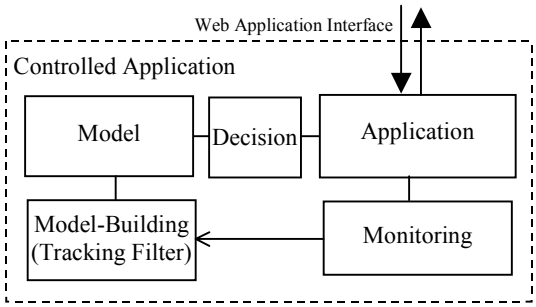

Figure 1. Use of a tracking filter in autonomic control of performance

The goal of this paper is to show how we can keep the performance model up-to-date as the software system changes. We will show how an Extended Kalman Filter can be constructed for a performance model, investigate its usefulness on an example, and

identify potential pitfalls to its application. As far as we know, this is the first use of Kalman filtering to track changes in a performance model. Novel aspects include modeling the dynamics of parameter change, the linearization of the observation function (which is the model itself), and the choice of covariance values for the filter, which affect the gain matrix.

The types of systems that can benefit from our approach are *information and transaction-based software* applications, for management, e-commerce, insurance, banking, brokerage, etc. In these systems, users log in, alternate requests with *think times,* and then log out. In terms of performance modeling, these systems are best described by closed models.

2. The Extended Kalman Filter for Parameter Tracking

The problem of tracking the changing value of states and parameters, based on measurements of related quantities, is a common one in engineering systems. Kalman Filters are widely used; one example is in tracking vehicle positions (see [2]).

In general, the theory assumes there is a state-and-parameter vector **x**, which changes over time by a known law $\mathbf{x}_{new} = \mathbf{f}(\mathbf{x}_{old})$ that includes random disturbances. From an observation vector $\mathbf{y} = \mathbf{h(x)}$ (which may include errors), an estimate $\hat{\mathbf{x}}$ for **x** is obtained, with the following *tracking filter* structure:

$\hat{\mathbf{x}}_{old}$ = previous estimate of x

$\hat{\mathbf{y}}$ = prediction of observation **y** based on $\hat{\mathbf{x}}_{old}$ and the model

z = new observation vector

$\hat{\mathbf{x}}_{new} = \hat{\mathbf{x}}_{old} + \mathbf{K}(\mathbf{z} - \hat{\mathbf{y}})$

The filter gain matrix **K** can be determined in a variety of ways, such as on Bayesian grounds to give the conditional expectation of **x**, conditioned on the stream of observations, the distribution of observation errors, and some initial distribution of $\hat{\mathbf{x}}$.

For cases with linear functions **f()** and **h()** and normally distributed independent disturbances and errors, Kalman's classic paper [8] derived a filter with this structure that minimizes a quadratic norm on the estimation errors.

2.1 The Extended Kalman Filter

Extensions cover nonlinear functions **f()** and **h()**, and non-normal distributions. We will adopt the most-used variant, the Extended Kalman Filter (EKF), summarized as follows (see, for example, [2] [16]).

The state or parameter vector **x** to be estimated evolves over time by a dynamic law of the form:

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1} \quad (1)$$

in which k denotes an integer time variable, **u** is a vector of known inputs, and **w** is a random disturbance or drift. **w** may represent actual random and unobserved forces acting on the system, or may represent modeling uncertainty; it is a random vector, "white" (independent over time), with mean 0 and a known *disturbance error covariance matrix* **Q**.

Observations are made of a vector **z,** which is a function of these parameters, with added measurement error. At step k, it is defined by:

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

where $\mathbf{v}_k$ is a white measurement noise, with mean 0 and a known *measurement error covariance matrix* **R**.

The estimate $\hat{\mathbf{x}}$ of **x** is computed recursively in two steps:

Step 1. Prediction:

1.1 project the state ahead:

$$\hat{\mathbf{x}}_k^- = \mathbf{f}(\hat{\mathbf{x}}_{k-1}^-, \mathbf{u}_{k-1}) \quad (3)$$

1.2 project $\mathbf{P}_k$, the estimated covariance matrix for the estimates of **x**:

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q} \quad (4)$$

Step 2. Feedback:

2.1. compute the Kalman gain **K** as:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R})^{-1} \quad (5)$$

2.2 correct the state vector:

$$\hat{\mathbf{x}}_k^- = \hat{\mathbf{x}}_{k-1}^- + \mathbf{K}_k(\mathbf{z}_k - \mathrm{h}(\hat{\mathbf{x}}_{k-1}^-)) \quad (6)$$

2.3 correct the error covariance $\mathbf{P}_k$:

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

with the notation:

$\mathbf{z}_k$ = the measured value for **z** at time k.

$\mathbf{h}(\hat{\mathbf{x}}_k^-) = \hat{\mathbf{y}}_k$ = predicted value for **z**.

A = linear term in the Taylor expansion of **f(x)** = $\partial \mathbf{f}/\partial \mathbf{x}$. In our case, **A** is the identity matrix.

H = linear term in the Taylor expansion of **h(x)** = sensitivity of observations to parameters = $\partial \mathbf{h}/\partial \mathbf{x}$.

R = measurement error covariance.

Q = disturbance error covariance.

The recursive filter has to be initialized with the estimated state $\hat{\mathbf{x}}_0^-$ (for Step 1.1) and an initial error covariance matrix $\mathbf{P}_0$ (for Step 1.2). We'll see in the next sections how their selections can affect the filter performance.

Where **f(x)** or **h(x)** are nearly linear, the filter is expected to have near-optimal properties. The optimality and convergence properties depend on the way the functions **f(x)** or **h(x)** are linearized around an operating point [9]. The Extended Kalman Filter presented above linearizes **f(x)** and **h(x)** by a first-

order Taylor series around the state estimate $\hat{\mathbf{x}}_{k-1}^{-}$ and does not take linearization errors into account. A variant called the Iterative Extended Kalman Filter (IEKF) linearizes **h(x)** around the predicted state estimate $\hat{\mathbf{x}}_{k}^{-}$. Other variants of the filter, such as the Unscented Kalman Filter [7] or the Divided Difference Filter [14], capture the linearization errors in the covariance matrices. They were shown in [9] to provide better estimates when dealing with nonlinear **f(x)** functions, while EKF and IEKF provide better performance when dealing with nonlinear **h(x)**, which is the case here. This motivates our use of EKF instead of other types of filter.

2.2 Tracking for a Performance Model

For a performance model, we model **x** as a vector of n parameters that drift randomly. Thus:

$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{w}_k$

The drift $\mathbf{w}_k$ is assumed to be an independent normally distributed n-vector with covariance matrix $\mathbf{Q} = \text{diag}(q_1, q_2, \ldots q_n)$.

Also, the measurement m-vector **z** is modeled by the results **y** of the performance model calculation, with added errors of estimation:

$\mathbf{y_k} = \mathbf{h}(\mathbf{x}_k)$

$\mathbf{z}_k = \mathbf{y}_k + \mathbf{v}_k$

For a large step duration S, the sampling error $\mathbf{v}_k$ will be approximately independent and normally distributed. Its covariance matrix is defined as $\mathbf{R} = \text{diag}(r_1, r_2, \ldots r_m) = \text{diag}(\mathbf{r})$, where $r_i = \text{Var}(z_i)$.

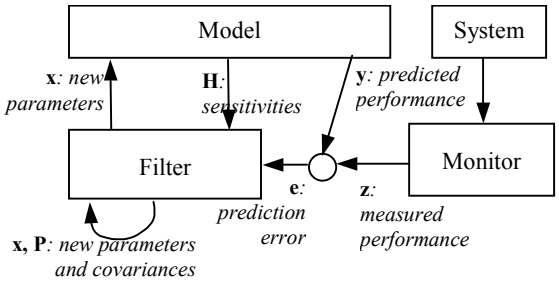

Figure 2. Logical architecture of a tracking filter for model parameters

The functional architecture of a system with a Kalman Filter to track its performance model is shown in Figure 2. The filter has a feedback structure, in that the parameter estimates at each step are used to predict the performance for the next step, and the errors of prediction are used to correct the estimates.

2.3 The Example Performance Model

To simplify the presentation, a very basic queueing network model will be considered as the performance model in this paper, as shown in Figure 3. It represents a small Web server with its disk (node 2) and a separate node for CGI application service (node 3). A response includes all the work done between visits to the "Users" node in the Figure, which represents the operation in which a user responds to one system output and generates the next request to the system. Users have a characteristic "think" time for this operation, which will be set to zero here. Service times are assumed to be exponential. Then the queueing model has four parameters:

N = the number of active jobs, assumed to be constant (so this is a "closed" model), with default value 4,

D = [D(1), D(2), D(3)] = the total average demands for service by nodes 1, 2 and 3, with default values [2, 3, 4] sec/response.

The model is also assumed to satisfy the separability conditions, which means that it can be solved by Mean Value Analysis (MVA) [6].

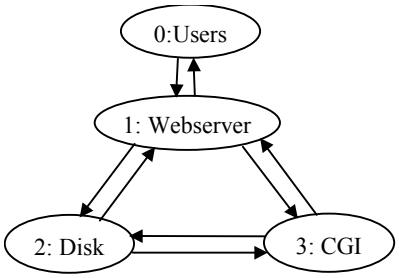

Figure 3. A simple queueing model

Some performance measures that at least potentially could be measured in the system are:

f = the throughput of requests from Users, per sec.

T = the mean response time = N/f,

T(i) = the part of each response time that is spent queueing and in service at node i,

U(i) = the mean utilization of node i,

N(i) = the mean number in system at node i.

In practice, we may use only some of these. The first case to be considered has measurement vector:

$\mathbf{z} = [T(1), T(2), T(3), f]$

The prediction vector $\hat{\mathbf{y}}$ has the same structure, and the error vector is $\mathbf{e} = \mathbf{z} - \hat{\mathbf{y}}$.

The vector of parameters to be estimated is

$\mathbf{x} = \mathbf{D} = [D(1), D(2), D(3)]$

Later, the question of tracking N will be considered as well. The filter operation is illustrated in Figure 4.

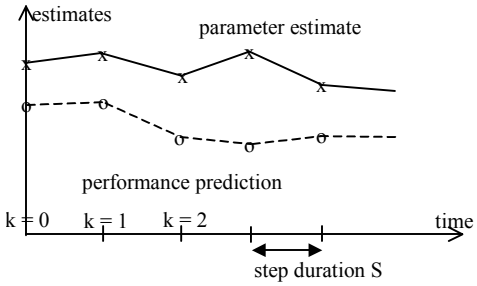

Figure 4. Operation of the estimating filter over time

2.4 Measurement Errors and Drift

The errors **v** are statistical sampling errors, which depend on the system and on the sampling period S. They were estimated with N = 4, **D** = [2, 3, 4], and sampling period S1 = 100000 sec, by simulating the model with batched means. S1 was chosen arbitrarily, and includes about 20,000 responses. The variances for T(i), U(i) and f were as follows:

- for T(1..3): 0.0109, 0.0374, 0.0745
- for U(1..3): 0.0000737, 0.0001531, 0.0000872
- for f: 0.0000154

For a given vector **z** of measurements, there is a vector **r** of variances, which are expressed in the error covariance matrix **R** = diag (**r**). Thus, for **z** = [T(1), T(2), T(3), f], and step size S1,

R = **R**(S1)
= diag(0.0109, 0.0374, 0.0745, 0.0000154) (8)

For other sampling step sizes S, the variances are multiplied by (S1/S):

$$\mathbf{R} = (S1/S)\,\mathbf{R}(S1) \qquad (9)$$

The factor (S1/S) in equation (9) follows from S and S1 being long enough, so that error variances are inversely proportional to the number of samples, and that the number of samples of each step is proportional to S. It was tested by simulating for S = 400000.

The variances of the drifts in the parameters are typically unknown, but **Q** should represent typical drift magnitudes to be tracked. If the drift has independent increments, the variances will add, making the variance proportional to S. For this work, a base case was taken of unit drift variances for S = S1 = 100000, giving:

$$\mathbf{Q} = \mathbf{I}_3 * (S/S1) \qquad (10)$$

where $\mathbf{I}_3$ is the 3x3 identity matrix.

3. Experiment: A Step Change

We will consider a situation where the system parameters make a step change from values $\mathbf{D}_{\mathbf{old}}$, which are constant, to new (constant) values **D** at time zero. At the moment of the change, the tracking filter has a correct estimate $\hat{\mathbf{x}}_0 = \mathbf{D}_{\mathbf{old}}$. The estimated parameters are suddenly far from correct, the prediction error **e** becomes much larger and the filter executes a transient response, to acquire the new parameter values.

The *step response* of the tracking filter can provide insight into many properties of the filter, and how it should be applied. We consider:

- How quickly does the filter settle to the neighbourhood of the new parameter values **D**?
- What is the steady-state tracking error over time, after settling to the new values?
- What is the influence of the step (sampling) time?
- What is the influence of the matrices **R**, **Q**, and $\mathbf{P}_0$?
- What is the influence of using an incorrect model structure?

Since **R** captures the accuracy of the measured data, a larger **R** will lead the filter to adjust more slowly to prediction errors. Conversely, a larger **Q** will lead the filter to depend more on the measurements. Too small a **Q** causes the filter to assume there is little or no change in the system; the filter will gradually close down and stop tracking. $\mathbf{P}_0$ only affects the initial conditions of the filter.

The role of **H** is to connect the prediction errors back to sources in model parameter errors. If the model has a different structure, **H** will cause the filter to interpret the measurements according to the model.

Base Case:

- The server demands change from **D** = [4, 5, 6] to **D** = [2, 3, 4] at time zero and remains constant.
- The performance model has the same structure and the filter begins at $\hat{\mathbf{x}}_0^- = [4, 5, 6]$.
- $\mathbf{P}_0$, the estimated covariance of the initial estimate $\hat{\mathbf{x}}_0^-$, was chosen as the squares of the initial estimates, so $\mathbf{P}_0$ = diag(16, 25, 36).
- The base-case sampling step time is S1 = 100000 time units, which gives **R** as in equations (8), (9) above.
- From equation (10), $\mathbf{Q} = \mathbf{I}_3$.

The output sensitivity matrix **H** is the matrix of derivatives of output values to demand parameters **D**. Given a set of estimated demands, the derivatives were calculated exactly by differentiating through the mean-value-analysis steps for solving the queueing network (see Appendix). **H** could also be found numerically.

3.1 The Filter Simulation for a Step Response

To study the filter behaviour, the system was replaced by a simplified representation, which returns a measurement vector **z,** which is the sum of the exact

result vector **y** for **D** = (2, 3, 4) and a simulated sampling error **v**. **v** is a generated, independent, normally distributed random vector with component variances given by the diagonal of **R**.

At each step, the model equations were also solved with the estimated parameters $\hat{\mathbf{x}}_{k-1}^{-}$ to find the new predicted measures **y** = h($\hat{\mathbf{x}}_{k-1}^{-}$), and the prediction error **e = z - y**. Then the filter was used to update to $\hat{\mathbf{x}}_{k}^{-}$, and the step was advanced.

3.2 The Filter Step Response, Base Case

A sample transient response of the tracking filter, equations (1)-(7), is shown in Figure 5. The upper three curves show the response of the parameter tracking, and the lower curve plots the throughput prediction error. As we shall see later, parameter estimates with large errors can give a correct throughput prediction.

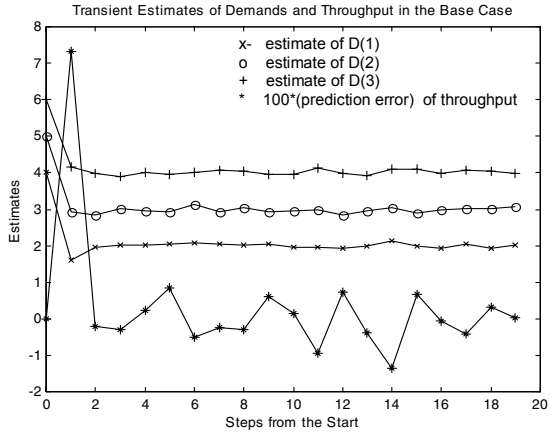

Figure 5. A sample transient

Figure 5 shows that, at the beginning of the transient, the estimates change almost at once to the neighbourhood of the new values of **D**, and stay in that neighbourhood with a small random variation. The bottom curve shows that the filter predicts the throughput very well. The error (multiplied by 100 so it can be seen more clearly) settles quickly to near zero. The actual mean throughput is about 0.21/sec, and the error settles to within roughly ± 0.015, or about 7%.

An average step response was found across 1000 repetitions of the transient, and Figure 6 shows the average values, plus or minus one standard deviation. The variations are very small. The throughput prediction errors are also shown, again multiplied by 100. Their mean goes quickly to near zero, and their standard deviation appears to be about 0.005. This is about 2.5% of the mean system throughput of about 0.21 transactions/sec.

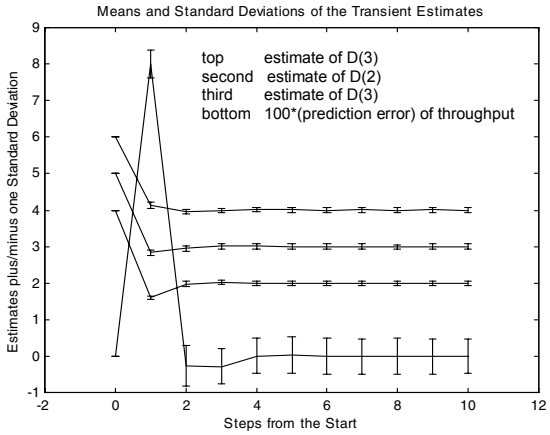

Figure 6. Estimates during a transient, with error bars for one standard deviation.

The impact of the size of the sampling step size S, and the matrices **P**(0), **Q**, and **R,** was investigated by varying them by factors ranging from 0.01 to 1000.

Figure 7 shows the effect of scaling **R**. The Y axis is the observed standard deviation of the parameter estimates in the steady-state part of the response (which, after observing Figure 6, was taken to begin at the fifth sampling step). The throughput prediction error was multiplied by 10 in Figure 7, to bring it into the scale of the other variables.

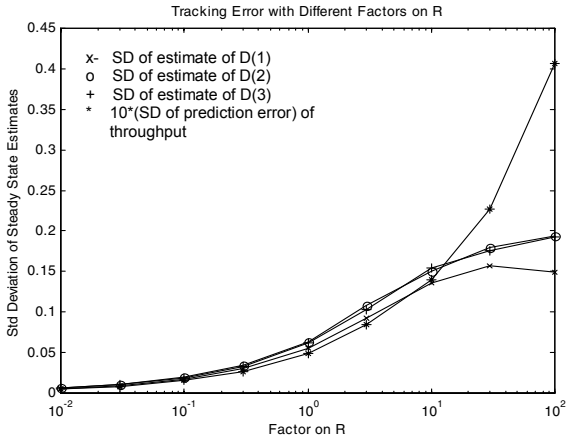

Figure 7. Effect of scaling the error variance matrix R on the accuracy of estimates

As **R** is increased, the filter tends to reduce its adaptation to the measurements, and its prediction quality suffers.

P(0) and **Q** were found to have very little effect on the long-run tracking accuracy, so the results are not shown here. As the scale factors varied from 0.01 to 1000, the standard deviation values stayed within

10% of the values shown in Figure 7 for a scale factor of 1.

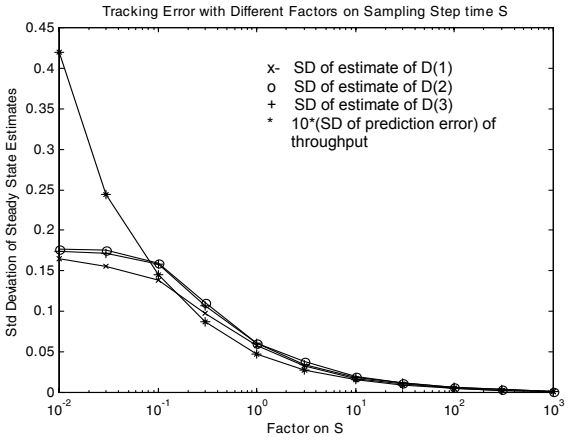

Figure 8. Effect of scaling the sampling step time S on the accuracy of estimates

However, the length of the sampling step S had a significant effect, as shown in Figure 8. S affects the measurement error (since statistical sampling error variance declines as 1/S) and also the parameter drift. (A longer step permits greater drift, which is assumed here to have a variance proportional to S.)

Despite the assumed greater drift, the filter gave more accurate estimates with larger S.

4. Other Measurement Vectors

The measurements T(1) – T(3) used above have a close relationship to the node service demands D(1) – D(3). Often we have less information, or more indirect information. A tracking filter can combine different kinds of data, as long as a corresponding prediction can be made and the sensitivity **H** can be calculated. This "data fusion" capability of tracking filters is important in many applications (e.g. [2]).

Suppose we have a mixture of measurements of different types, such as:

$$\mathbf{y} = [T(1), U(2), f]$$

This also has only three components instead of four. Then the tracking filter should be able to combine these disparate data to estimate **D**.

4.1 Case 2: Fusion of Diverse Measures

A filter was created to work with the measurement set **y** = [T(1), U(2), f], by incorporating the appropriate sensitivities in the matrix **H**. Figure 9 shows that it succeeds just as well as the previous case in acquiring the new parameter values, settling almost at once to near the correct demand values [2, 3, 4]. The throughput prediction has greater errors, and over-estimates the throughput at step 2. However, in general, the filter behaves very well.

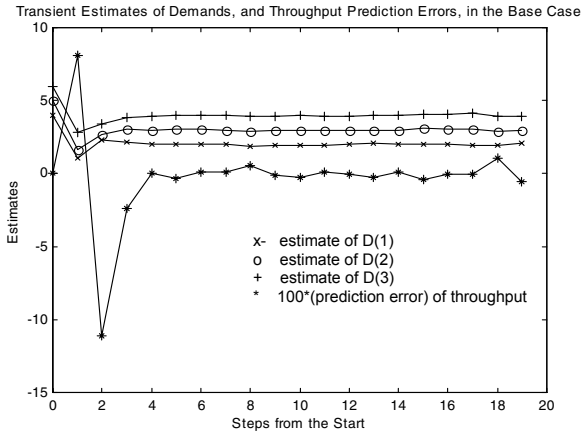

Figure 9. A transient response in Case 2, with different measurements: y = [T(1), U(2), f]

Over the range of values of the sampling step size S shown in Figure 10, the variation in the estimated parameters is only a little larger than in Figure 8. The standard deviations are about 10% larger at a factor of 10^{-2} and also at a factor of 1. The throughput prediction error is smaller at 10^{-2} but a little larger at a factor of 1. Overall, we can say the filter performance is very similar for this measurement vector.

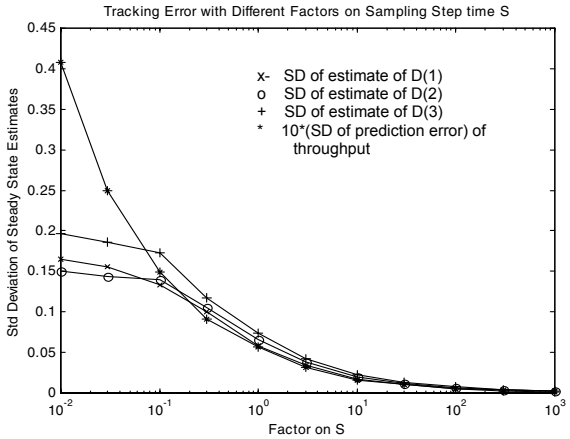

Figure 10. Variability of estimates and predicted throughputs in Case 2

4.2 Case 3: Inadequate Information

Suppose that T(1) used in Case 2 were dropped and a measurement vector **y** = [U(2), f] were used.

The transient response recorded in Figure 11 looks quite promising. It settles into a small range around the correct parameter values (2, 3, 4) within two steps, and the throughput prediction error is small ($< \pm 0.02$).

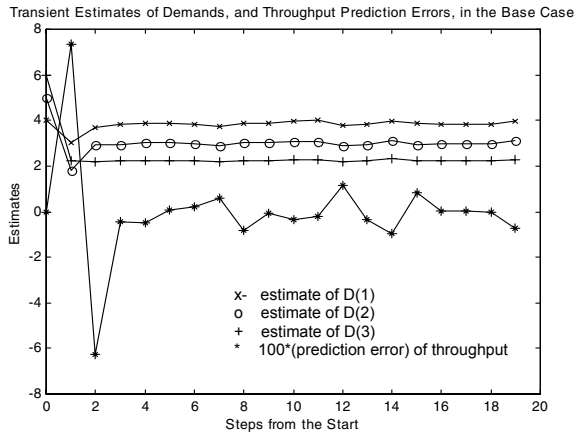

Figure 11. Transient with measurements only for y = [U2, f]

However, the statistical analysis of the steady-state convergence is not so positive. Figure 12 shows the standard deviation of parameter estimates, from 20000 steps of the simulated algorithm in steady state at each step size S = 100000*(scale factor). The errors for D(1) and D(3) are much larger than in Case 2, while the errors for D(2) are much the same. The small errors for D(2) reflect the better quality of information about the behaviour of node 2, provided by measuring U(2).

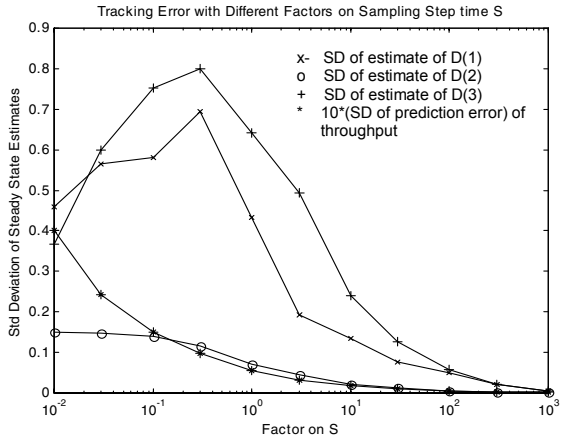

Figure 12. Variability of steady-state estimates in Case 3 (measurement y = [U2, f])

For an extreme example, a filter was constructed driven only by throughput measurements. A typical transient response is shown in Figure 13. The filter appears to converge, but to the wrong values. However, the prediction error is no worse because many models can predict the same congestion delay. Here it seems that D(3) has been set to zero, compensated by larger values of D(1) and D(2).

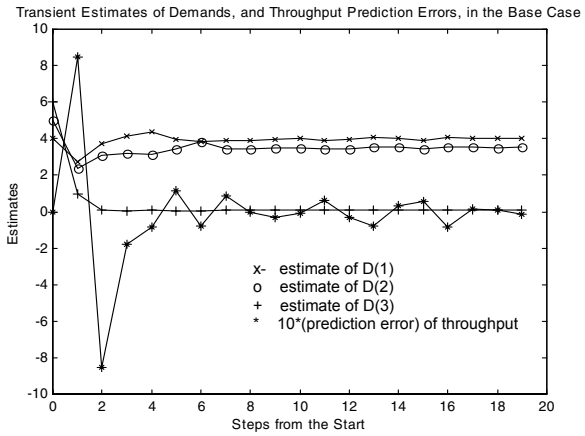

Figure 13. Tracking with only one measured value, the throughput

4.3 Case 4: Bottlenecked Estimated Model

If the estimated parameters make the model heavily bottlenecked at one server, it is well-known that its performance predictions become insensitive to parameters of the other servers. This affects the **H** matrix used by the filter, and might affect the ability of the filter to track to the new parameter values.

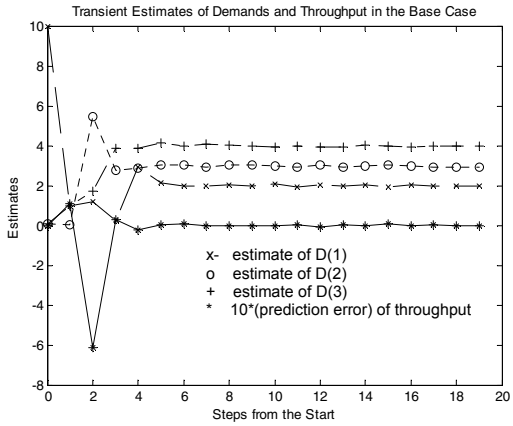

Figure 14. Heavily bottlenecked initial model

Case 4 was like Case 1, but with starting demand estimates [10, 0.1, 0.1] that imply a severe bottleneck at server 1. The transient results in Figure 14 show that it takes a little longer to reach the steady state at the new correct values: about four steps, instead of 1. However, it in no sense is "stuck" in the bottleneck mode, as the large parameter is quickly reduced.

5. Use of an Approximate Model

Since performance models are always approximations, the impact of a structural mismatch between the system and model must be considered.

5.1 Case 5: Wrong Model Structure

We consider first a model with only two nodes, (while the actual system has three as before), and **y** = [T(1), T(2), f]. Figure 15 shows the transient estimates starting from **D** = [4, 5]. They show stable behaviour, and throughput prediction errors that are only slightly larger than in the base case of Figure 5.

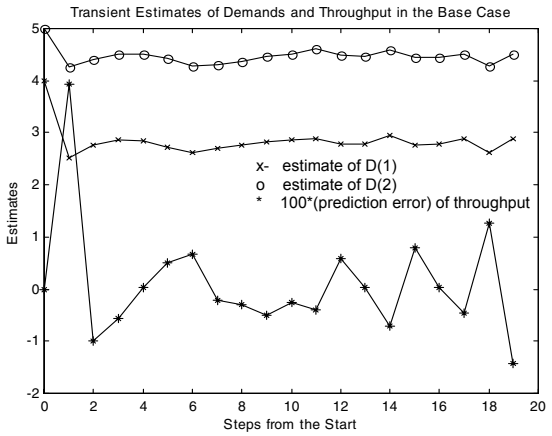

Figure 15. Typical transient of the estimates for a two-node model

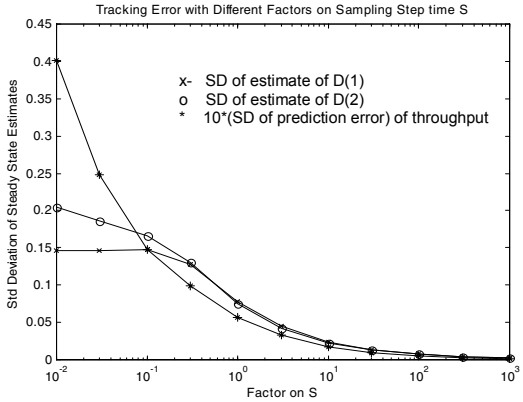

Figure 16. Standard deviations of estimates using a two-node model

These comments are borne out by the results in Figure 16 for the variation of the estimates. The filter settled to values for D(1), D(2) of about 2.78, 4.44 for every case. The errors are about the same size as in Figure 8. Thus the simpler model seems to predict as well, and to track as well, as the three-node model. Simplified models are important in practice.

5.2 Case 6: Incorrect Parameter Value N

The model was set up for 7 users, while the actual system (as in Case 1) has N = 4. N is not tracked and thus is not corrected. The measurements were **y** = [T(1), T(2), T(3), f].

The filter was unable to cope with this situation; it settled to a predictor with an error of about 25% in throughput (Figure 17). It was seeking a compromise between errors in predicting the per-node response times T(i) and the throughput f. Rather than display details of this bad situation, we will address the question of tracking the population N for this model.

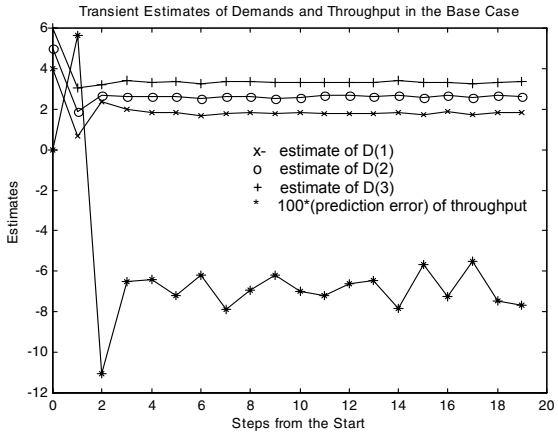

Figure 17 Transient response with N = 7

6. Estimation of the User Population N

The user population N is not a continuous variable and cannot be tracked by the usual filter based on derivatives. However, the well-known Schweitzer approximation for queueing models treats N as real, giving the calculations shown in the Appendix. Notice that we do not use the Schweitzer approximation to calculate performance, but only for the derivatives with respect to N. The nearest integer will be assigned to N if N is not an integer during the iteration steps.

Results similar to those reported above for tracking two or three parameters are shown in Figures 18 and 19 with four parameters **x** = [D(1), D(2), D(3), N]. We considered a step change from [4, 5, 6, 7] to [2, 3, 4, 4]. The transient in Figure 18 shows good convergence within two steps for both the demands and user population. The standard deviations of tracking in Figure 19, over changes in the sampling step sizes, are also very similar in magnitude to the earlier results shown in Figure 8 for the base case. The tracking errors decrease with the size of the sampling step S, and the filter converges faster for N than for the demands.

The derivative calculation by the Schweitzer approximation appears to be successful. Thus, even a discrete variable can be tracked by the Kalman filter approximately.

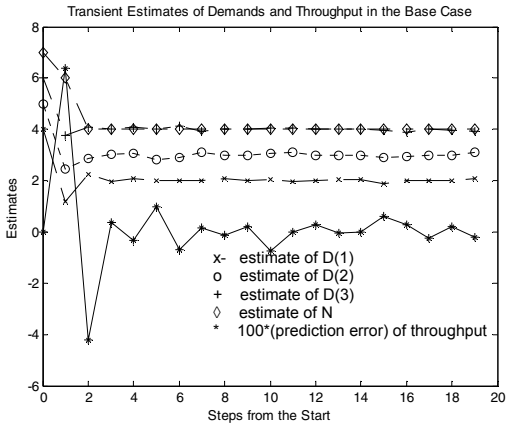

Figure 18. Transient Including Estimates of N

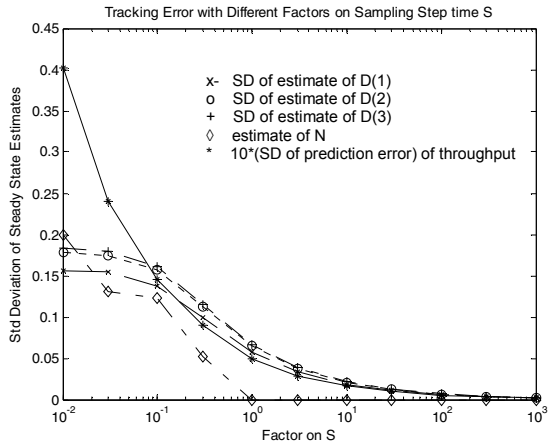

Figure 19. Errors Including Estimates of N

The results also show that as long as we can include the important model parameters (N in this case) into the tracking parameters, and get the respective sensitivity data, those model parameters could be tracked correctly.

7. Conclusions

Even though the performance model investigated here is a small and simple one, it is clear from the experience reported above that the Extended Kalman Filter can be applied to track changes in parameters of queueing models. In any case, practical models are often simple.

No problems were experienced with stability or with slow convergence, over wide ranges of parameter values. When parameters $\mathbf{P}_0$ and **Q,** which must often be guessed, were varied over four orders of magnitude, the behaviour of the estimator was very little affected.

The tracking filter operates in discrete steps, and the step time must be quite long. The basic step time used here, of 100000 time units, would contain about 21000 responses (at a mean throughput of 0.21), and this gave acceptable accuracy; longer step times were better.

The filter converged remarkably quickly following a step change in the system parameters, which was the condition investigated here. We may expect similarly good tracking on steadily drifting or randomly drifting parameter values.

A convenient aspect of the Kalman Filter is its capability to *fuse* data of different types, such as delay, throughput, and utilization, as demonstrated in Section 4.1. It is not necessary to have direct data on every server in the system, but more measured variables naturally give smaller estimation errors.

The filter was quite robust to the use of an approximate model, such as a model with fewer servers than the actual system. However it did not function satisfactorily with an incorrect user population. It will be necessary to estimate the customer population N as well, although as N is a discrete variable it does not fit the Kalman filter framework perfectly well. However, we showed that for separable queuing networks, we can find approximate derivatives by using the Schweitzer formula. The results showed that, using those approximations, the filter tracks the user population accurately. For more general models, such as layered queueing models, finding the derivatives of N as well as well as other sensitivities, may be solved by numerical differentiation (by multiple solutions at slightly perturbed parameter values).

Since the filter depends on sensitivities, and bottlenecked systems have low sensitivity to parameters away from the bottleneck, it was feared that a bottlenecked system or model might get "stuck" away from the correct values of parameters. However, this did not happen.

This work concentrated on investigating feasibility and is not exhaustive. However, the investigation was overwhelmingly positive about the potential of these estimators to track parameter changes.

Acknowledgements

This research was supported by grants from IBM Corporation and from CITO (Communications and Information Technology Ontario).

References

[1] Tarek Abdelzaher, Kang G. Shin, Nina Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach,"' *IEEE Trans. on Parallel and Distributed Systems* , V. 13, n. 1, 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, and Lui Sha. "On-Line Response Time Optimization of An Apache Web Server," *Int. Workshop on Quality of Service*, 2003.
[4] Neha Gandhi, Joseph L. Hellerstein, Sujay Parekh, and Dawn M Tilbury, "Managing the Performance of Lotus Notes: A Control Theoretic Approach", *Proc. of the Computer Measurement Group*, 2001.
[5] Hellerstein J., Diao Y., Parech S., Tilbury D., *Feedback Control of Computing Systems*, Wiley, 2004.
[6] R. Jain, *The Art of Computer Systems Performance Analysis*. Wiley, 1991.
[7] S. Julier, J. Uhlmann, H.F. Durant-Whyte, "A new method for approximating nonlinear transformations of means and covariances in filters and estimators," *IEEE Trans. on Automatic Control*, v. 45, pp. 477-482, 2000.
[8] 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.
[9] 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.
[10] Litoiu M., Woodside M., Zheng T., "Hierarchical model based autonomic control of software systems," *Proc. of Design and Evolution of Autonomic Software* (DEAS'05) Workshop, St. Louis, USA, May 2005.
[11] Ying Lu, Tarek Abdelzaher, Chenyang Lu, Lui Sha, Xue Liu, "Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers," *Proc. Real-Time and Embedded Technology and Applications Symposium,* Toronto, May 2003.
[12] D. A. Menasce, M. Bennani, "On the Use of Performance Models to Design Self-Managing Computer Systems," *Proc. 2003 Computer Measurement Group Conference*, Dallas, Dec, 2003.
[13] D. A. Menasce, "QoS-aware software components," *IEEE Internet Computing*, Vol. 8, No. 2, 2004.
[14] M. Norgaard, N.K Poulsen, and O. Ravn, " New developments in state estimations for nonlinear systems," *Automatica*, vol 36, pp. 1627-1638, 2000.
[15] 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 J.*, v. 43, 2004.
[16] H Tanizaki, *"Nonlinear Filters: Estimation and Applications- Second, Revised and Enlarged Edition,"* Springer-Verlag, Berlin-Heilderberg, 1996.
[17] Hai Wang , K.C. Sevcik, "Experiments with improved approximate mean value analysis algorithms", *Performance Evaluation*, v.39, p.189-206, Feb. 2000.

Appendix

The exact recursive mean value analysis equations for a separable queueing network are [6]:

$$T(i)^N = (N(i)^{N-1} + 1)\, D(i)\ ,\ i = 1,..,\ n \quad \text{(A1)}$$
$$f^N = N / \Sigma_i\, T(i)^N$$
$$N(i)^N = f^N\, T(i)^N,\quad i = 1,...,\ n$$

where:
N = the population of jobs or customers in the model,
$N(i)^N$ = mean jobs at node i, at population N,
$T(i)^N$ = residence time at node 1 per system response, at population N
f^N = system throughput at population N,
D(i) = demand at node i per system response.

The MVA equations are applied with initial conditions $T(i)^1 = D(i)$, and are applied for each value of N up to the desired value. To find the derivatives, one can simply differentiate these equations. Thus for differentiation with respect to D(j), we obtain:

$$\partial\, T(i)^N / \partial\, D(j) = [\,\partial\, N(i)^{N-1} / \partial\, D(j)]\, D(i)\ ,\ i = 1,..,\ n$$
$$\partial\, f^N / \partial\, D(j) = -\,[\,N / (\Sigma_i\, T(i)^N)^2\,]\, \Sigma_i\, \partial\, T(i)^N / \partial\, D(j)$$
$$= -\,(1/N)\,(f^N)^2\, \Sigma_i\, \partial\, T(i)^N / \partial\, D(j)$$
$$\partial\, N(i)^N / \partial\, D(j) = T(i)^N\, \partial\, f^N / \partial\, D(j) + f^N\, \partial\, T(i)^N / \partial\, D(j),$$

with initial conditions $\partial\, T(i)^1 / \partial\, D(j) = \delta_{ij}$.

The derivatives of U(i), the utilization of node i, are found from

$$U(i) = f(i)\, D(i) \text{ to be } \partial\, U(i)^N / \partial\, D(j)$$
$$= D(i)\ \partial\, f^N / \partial\, D(j) + f(i)\, \delta_{ij}\ .$$

Derivatives with respect to N

The well-known Schweitzer approximation for Equation (A1) above is:

$$T(i)^N \approx [N(i)^N\,(1 - (1/N)) + 1]\, D(i),$$

(The Schweitzer approximation is discussed in a recent paper [17] which also proposes an improvement on it.) This can be differentiated with respect to N to give:

$$\partial T(i)^N / \partial N \approx [\partial N(i)^N / \partial N\,(1-(1/N)) + N(i)^N (1/N^2)]D(i)$$

In evaluating this derivative we can use the values calculated by the exact MVA. We also need the following derivatives (found by differentiating the exact MVA equations:

$$\partial f^N / \partial N = 1 / \Sigma_i\, T(i)^N - (1/N)\,(f^N)^2\, \Sigma_i\, (\partial T(i)^N / \partial N)$$
$$\partial\, N(i)^N / \partial\, N = \partial\, f^N / \partial\, N\ D(i)\ ,\quad i = 1,...,\ n$$

The derivatives with respect to N require solving these three simultaneous nonlinear equations, which was done by a fixed-point iteration starting from:

$$\partial\, N(i)^N / \partial\, N = 1/K$$

Trademarks