Tracking Adaptive Performance Models Using Dynamic Clustering of User Classes

Hamoun Ghanbari, Cornel Barna
Dept. of Computer Science
York University
Toronto, ON, Canada
{hamoun, cornel}@cse.yorku.ca

Marin Litoiu
Dept. of Computer Science
York University
Toronto, ON, Canada
mlitoiu@yorku.ca

Murray Woodside
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, ON, Canada
cmw@sce.carleton.ca

Tao Zheng, Johnny Wong
School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
{t3zheng, jwwong}@uwaterloo.ca

Gabriel Iszlai
Centre for Advanced Studies
IBM Toronto Lab
Toronto, ON, Canada
giszlai@ca.ibm.com

ABSTRACT

Estimation techniques have been largely applied to track hidden performance parameters (e.g. service demands) of web based software systems. In this paper we investigate dynamic multiclass modeling of such systems, with variable classes of service, aiming at finding a low complexity model yet with enough accuracy. We propose a combination of clustering algorithm and tracking filter for effective grouping of classes of services. The tracking estimator is based on a layered queuing model with parameters for CPU demands and the user load intensity of each class of service. Clustering uses the K-means algorithm. The target application is autonomic control of web clusters, where changes occur at different rates and amplitudes and at random time instants. Experiments show that the tracking is effective, and reveal good filter settings for different variations.

Categories and Subject Descriptors

D.4.8 [**Software Engineering**]: Performance —*modeling and prediction, queueing theory*

General Terms

Performance

Keywords

Performance models, service demand estimation, tracking filter, URL clustering

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

1. INTRODUCTION

In web based systems, in order to maintain the desired quality of service when the service workload varies dynamically (e.g. [2, 7, 8, 28, 1, 11, 14, 19]) one can rapidly explore multiple allocation decisions and find near-optimal service provisioning [14, 28] based on a pre-constructed performance model. A problem in this model-based approach is the estimation of hidden or difficult-to-determine parameters of user triggered scenarios [20, 21, 4, 18, 31, 30, 16, 6, 5]. For example, measuring service times or demands for each class of service, even if technologically possible, is not practical because of the overhead introduced.

Some authors of this paper have previously investigated the use of estimators to track these hidden performance model parameters when all user triggered services are treated as a single class [27, 32, 29]. The estimation was performed by a Kalman filter using system measurement, such as response times, throughput and server utilization.

In this paper, we consider multi-class models to represent software systems. When user services have different performance behavior, then a single class model, which averages all services, is not an accurate description of the system. On the other hand, in web applications, services are identified by different URLs and, if each URL is treated as a class, the performance model solution cost (time) becomes excessive. Thus it seems natural to try to group the URLs with similar resource usage into a smaller number of classes and to control the modeling error. Since the resource demands associated with each URL may change with time (i.e. due to changes in the URL parameters) we propose an adaptive clustering that regroups the URLs depending on the time-varying demands associated with the requests. The estimation filters track these changes, and the performance model predicts their effect on quality of service (e.g., on response time of different services).

A use case of the proposed estimation technique is the feedback based adaptation architecture presented in Figure 1. In such architecture, the performance of an application that offers user services is controlled by a feedback loop. A

decision module makes resource *allocation decisions* based on the *performance model*, *quality of service* (QoS) targets and other system goals. A *workload classifier* and a *state estimator* group execution paths into runtime classes resulting into a new model with smaller set of classes. This model exploits the trade-off between *complexity* (lessened by having a smaller set of modeled classes) and *accuracy* (grows with the number of modeled classes). The role of *state estimator* is to update and track estimates of the model parameters (e.g., service demands).

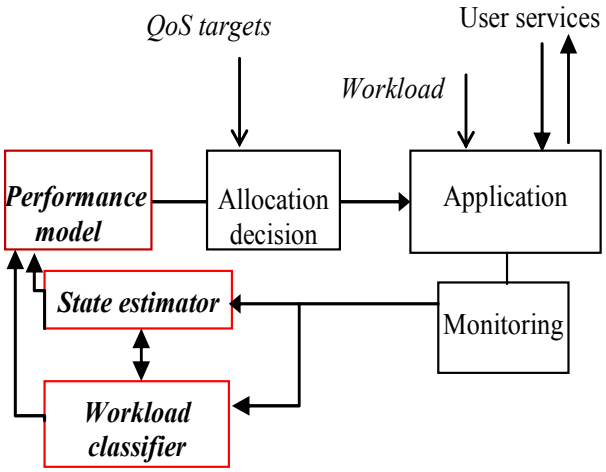


Figure 1: Architecture of an autonomic loop based control.

An online tuned multi-class performance model will provide better control of application performance since it can estimate and predict the effect of decisions for each service offered by the application (as opposed to an average across all services).

The original contributions presented in this paper are:

1. Estimation of hidden per class performance states for multi-class models
2. Dynamic workload classification
3. Evaluation of the effectiveness of approximation methods needed to make the classification and the multi-class estimator practical
4. The interaction of the rate of system change, in determining the accuracy of classification and tracking

The remainder of this paper is organized as follows. Section 2 describes our approach for parameter estimation and tracking. The applicability of the proposed approach is demonstrated by the experiments in Section 3. Related work, conclusions, and future works are discussed in Sections 4 and 5 respectively.

2. DYNAMIC CLASSIFICATION AND ESTIMATION

This section will present our classification and estimation methodology.

2.1 Classes

In our investigation, a class is distinguished by its behavior, namely, its demand for CPU, Disk, and I/O operations across layers of clustered servers.

The smallest granularity of behavior we consider is the response to a request for a given URL. Figure 2 shows as an example, scatter plot of the CPU demands for 100 different URLs on a 2-tier environment composed of a web and a database server. The demands by the i'th URL, $S_i = (S_{w,i}, S_{d,i})$, is shown as a demand tuple in the plot.

In terms of resource management, it may not be possible or effective to allocate resources to individual URLs because the granularity of resource allocation may be too fine. An obvious approach to reduce the granularity is to form groups of URLs and allocate resource at the group level. This can be done by aggregating together URLs with similar $(S_{w,i}, S_{d,i})$ demand pairs into a single cluster (or class). In Figure 2(a) to 2(f), 100 URLs are aggregated into 1 to 6 aggregate classes, respectively, with the value of the aggregate demand pair for each class shown as a heavy dot.

In our autonomic control architecture (see Figure 1), analysis of a LQM is also more manageable (or solution can be obtained more quickly) if the number of classes is small. In this context, the best aggregation should therefore be based on the tradeoff between fidelity of modeling (best with 100 classes) and economy in the model (best with one class).

2.2 Model

In order to capture the behavior of multi-tier web services deployed in a data center environment we used Layered Queuing Model (LQM) and APERA [13] which is a LQM solver. In LQM, a web application is viewed as layers of software introduced by the different tiers (e.g. web and database servers) and hardware layers of the underlying infrastructure. Each layer is represented by a Queuing Network Model (QNM) which can be solved by the mean value analysis.

Model inputs for LQMs include 1) the structure of the model including the services and their interactions for representative scenarios, and a topology of the underlying middleware and hardware and 2) the quantitative performance metrics such as workload component (W_i), identified with the number of users (N_c) and mean think time (Z_c) for close models or by the external arrival rate (λ_i), and service times or demands (S_c) for each class of service c. Outputs of LQM include response times (R_c), throughput (X_c) for each class and server utilization (U_j) for each server.

Figure 3 shows a typical Layered Queuing Model (LQM) of a web-based application which we will also use through this paper. We assume that there are C classes of requests; the "User" block in Figure 3 represents N_c users in class c at their browsers. We use Z_c to denote the mean think time of class c users. N_c and Z_c are the workload parameters for class c; they correspond to the W_i's defined earlier.

Variation in load arriving at the system can be modeled by variations in Z_c or in N_c; a larger N_c or a smaller Z_c leads to a higher level of offered load.

The "WebServer" block represents the server software with M threads, running on processor WSProc (this is indicated by the "host" relationship). The box labeled "webOp" represents the operation done for the users, and requires a mean CPU demand of $S_{w,c}$ for requests in class c and on average 1 database operations. The database is running on its own

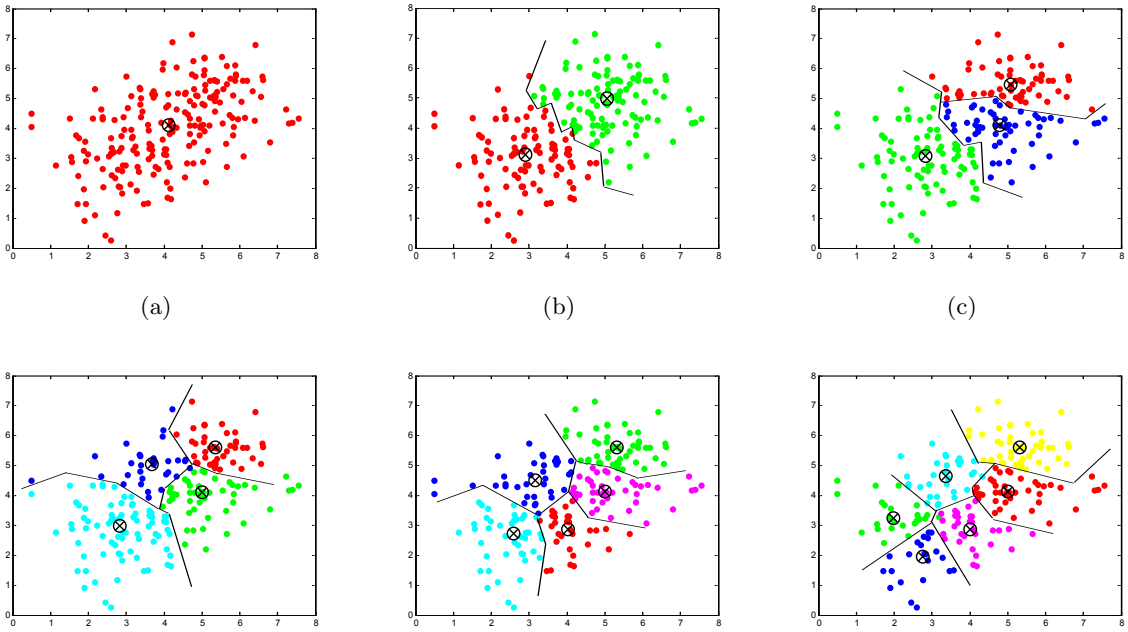


Figure 2: CPU demand of 100 URLs on a web server ($S_{w,i}$) on vertical axis vs. database server ($S_{d,i}$) on horizontal axis.

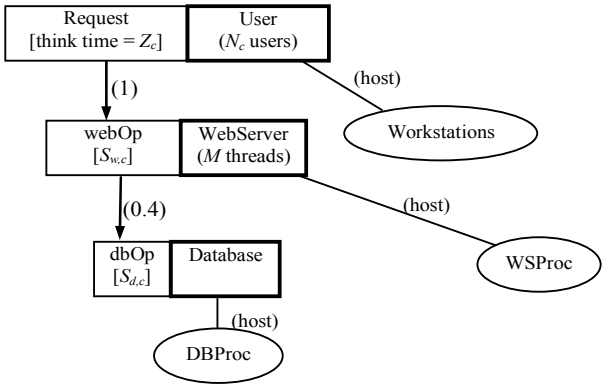


Figure 3: Multiclass Layered Queuing Model of a Web application.

device DBProc. "dbOp" represents the mean CPU demands of $S_{d,c}$ at DBProc for a class c request.

Outputs of LQM of Figure 3 include:

- R_c – mean response times to class c users $(R_1, \ldots, R_C)$
- R_c^d – mean response times to class c users at database server $(R_1^d, \ldots, R_C^d)$
- X_c – throughput of class c $(X_1, \ldots, X_C)$
- U_w – utilization of web server processor
- U_d – utilization of database processor

Usually, the directly measurable performance data are those given by model outputs, averaged over a measurement time interval of length T:

$$z = [R_1, \ldots, R_C, R_1^d, \ldots, R_C^d, X_1, \ldots, X_C, U_w, U_b]$$

Values for the think time Z_c and the CPU demands $S_{w,c}$ and $S_{d,c}$ 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 based on the available measured data (See next section).

2.3 Estimation with Extended Kalman Filter

Similar to [29, 32] we use Kalman filter [26] in the "State Estimator" (see Figure 1) to estimate model states and parameters based on the available measured data. Figure 4 depicts the architecture of this extended Kalman filter. Let x and z be vectors representing the parameters and measured performance of the various classes, respectively. The LQM maps x to an output vector y (i.e., $y = h(x)$) which represents the predicted performance. The filter then estimates x (which is not directly measurable) based on the observed performance z and modeled performance y.

The filter computations are recursive, beginning from an initial estimate x_0. In each recursive step k the filter finds the current parameter estimate x_{k-1}, based on the most recent parameter estimate x_{k-1} and the current observation vector z_k, in a way that it minimizes prediction error vector e_k (i.e. the difference between predicted performance $h(x_{k-1})$ and the observed one z_k):

$$e_k = z_k - h(x_{k-1}) \quad (1)$$

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

$$x_k = x_{k-1} + K_k \; e_k \quad (2)$$

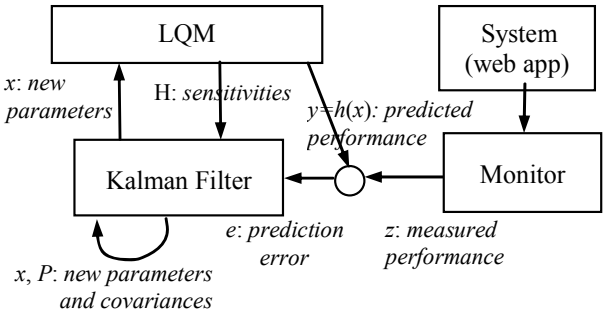


Figure 4: Kalman filter architecture.

where matrix K_k or "Kalman Gain" is the optimal gain matrix (in fact it is only suboptimal when h is a nonlinear function, as it is here).

An important issue in using Kalman filters is convergence. The necessary and sufficient condition is given by the identifiability condition [23, 25, 26] which requires that, at minimum, we have more measured parameters than estimated state parameters: i.e., $\dim(x) \leq \dim(y)$.

For the case of multi-class models, $\dim(x)$ and $\dim(y)$ depend on the number of classes and the size of the deployment topology. For example, consider an application with C URLs which are deployed over J servers. The measured parameters are most likely the mean response time and throughput of each class and the utilization of each server. If we also measure the response time of each class at an intermediary server, then we have other C measurements. We thus have $\dim(y) = 3C + J$. If we wish to estimate the demand of each class at each server and the mean think time for each class, then $\dim(x) = C \times (J + 1)$. For this example, when the application is deployed on 2 tiers ($J = 2$), $\dim(x) < \dim(y)$ regardless of the value of C. However, when $J = 3$, $\dim(x) > \dim(y)$ for $C > 3$; this implies that with more than 3 classes, it will be difficult to estimate the hidden parameters and we need more measurement points. To minimize the measurement overhead we should therefore keep the number of classes small, if possible.

2.4 Dynamic Clustering Algorithm

In this section, we first discuss the modeling error due to clustering and then present an algorithm to determine the best choice of the number of clusters C and the grouping of URLs into these clusters.

2.4.1 Modeling Error

Suppose there are C clusters (i.e. $c = 1, \ldots, C$), and L URLs (i.e. $i = 1, \ldots, L$) and $c(i)$ denotes the cluster (or class) which contains URL i. Let $R_{c(i)}$ be the predicted mean response time of class $c(i)$ requests. For the case of no clustering (i.e., each URL is treated as a separate class), let $R(L)_i$ be the mean measured response time of requests for URL i. Then a modeling error measure $E(C)$ for C clusters is given by:

$$E(C) = \sqrt{\frac{1}{L}\sum_{i=1}^{L}\left(\frac{R(L)_i - R_{c(i)}}{R(L)_i}\right)^2} \qquad (3)$$

The hypothesis is that the error $E(C)$ tends to decrease when the number of clusters is increased. However, finding the clusters and the multi-class performance model associated with the clusters is complex as E is a measure of how well the results from the filter and the model fit the measured data.

2.4.2 Dynamic Clustering Algorithm

Our classification algorithm is shown in Algorithm 1. Input to this algorithm are the Layered Queuing Model (LQM), a measurement vector z and an error threshold A. The vector z can include workload elements λ_i, the measured response time R_i^m for URL i ($i = 1, 2, \ldots, L$) and the total utilization of the servers U_j ($j = 1, 2, \ldots, J$). The algorithm outputs the best values of the number of clusters C and the grouping of URL's into these clusters (or classes) in terms of modeling error and modeling complexity, and the service time estimates at the various servers for each of the C classes.

Algorithm 1: Estimation and Classification Algorithm.

input : LQM, z, A
output : The best choice of the number of clusters C, aggregated demand ($S_{j,c}$) and URL members of clusters

1 Estimate $S_{j,i}$, the service demand of URL i at server j, from measurement data ($\forall j \in \{1, \ldots, J\}$, $\forall i \in \{1, \ldots, L\}$) using the model with no clustering; and set $C = 1$.

2 Cluster the URLs based on the $S_{j,i}$'s into C clusters.

3 Estimate the parameters $S_{j,c}$, the service demand of class c at server j; solve the LQM with C classes and obtain results for R_c, the mean response time of class c ($\forall j \in \{1, \ldots, J\}$, $\forall c \in \{1, \ldots, C\}$).

4 Calculate the modeling error $E(C)$

5 If $E(C) > A$, then increase C by 1 and go to Step 2.

6 Return C, the URLs in cluster c, and $S_{j,c}$ ($\forall j \in \{1, , J\}$, $\forall c \in \{1, ..., C\}$)

In our investigation, the workload W_i is time-varying. The autonomic control loop shown in Figure 1 is executed at regular intervals. If the modeling error of the existing clustering configuration is less than A, only the regular estimation is performed using the Kalman filter. On the other hand, if modeling error is greater than A, our classification algorithm is invoked to obtain a new clustering of URLs such that the modeling error becomes less than A.

The classification algorithm starts by estimating the service demands of all the URLs at the various servers from the measured response times using a model with no clustering (Step 1). The service demand of URL i at server j is denoted by $S_{j,i}$ ($\forall j \in \{1, \ldots, J\}$, $\forall i \in \{1, \ldots, L\}$). The estimation is done by using the Extended Kalman filter described in subsection 2.3 to derive the hidden state parameters (the service demands in this case) from the measures that are available. We also use LQM, explained in subsection 2.2, as the model component in the Extended Kalman filter structure (see Figure 2). This model maps the state parameter vector x, composed of service demands $S_{j,i}$, and the workload parameters W_i to output parameters y such as mean response time of each URL and utilization of each server, as given by the relationship $z = h(x)$. It is worth noting that h is a non-linear function for open models and an iterative algorithm for closed models. For complex models, especially for closed models, deriving a symbolic representation

of derivative matrix (H) is very difficult and our approach is to approximate the partial derivatives.

In Step 2, the K-means clustering algorithm [9, 12] is used to perform unsupervised grouping of service demands. K-means has a low complexity and is adaptable to a continuous nature of workload classification problem. Moreover, it is able to detect clusters in an efficient way which does not require computing the distance of all points in space to each other. K-means takes as input the number of distinct clusters to generate (C) and will determine the size of the clusters based on the structure of the data.

The modeling error $E(C)$ is calculated at Step 4 using analytic results for R_c obtained in Step 3. If $E(C) \leq A$, the algorithm terminates and returns C, the URL assigned to each of the C clusters, and the estimated service parameters for the different classes (see Step 6).

Step 5 corresponds to the situation where the modeling error $E(C)$ is larger than the acceptable error A. This is usually a result of gradual changes in service demands over time. Here the algorithm re-calculates the clusters; this may lead to a larger number of clusters (more estimation precision) or reshaping with the same number of clusters. Both approaches can improve the accuracy of estimation and decrease the error. Steps 3 and 4 are then repeated to compute $E(C)$ for the clusters obtained from re-calculation. Note that a larger number of clusters would increase the computational cost of estimation, but it will decrease $E(C)$.

3. EXPERIMENTS

In this section we validate the method with a set of experiments. The first set of experiments were done on real systems using TPC-W benchmark [17]. The second set of experiments used a simulated system in order to further show the strengths and limitations of the method.

3.1 TPC-W benchmark and FIFA98 workload

TPC-W is a web application composed of 14 URLs, each having different service demands on web and database servers, and a workload defined in terms of percentage of total number of users. The benchmark has three workload mixes, buying, ordering, and browsing. Based on the workload mix the distribution of workload amongst URLs varies. This workload is generated using emulated browsers (EBs) whose behavior and navigation is controlled by a Markov chain with certain probabilities to match the desired distribution of load between URLs. As a result, there is a Markov chain per user mix.

We deployed the Java implementation of TPC-W [24] with some modification, on a cluster of four Tomcat web servers and one single MySQL database server, with Linux as the operating system. In order to be realistic, we used FIFA98's [3] workload instead of TPC-W's original workload. The TPC-W's original workload is just to test scalability but FIFA98 reflects variations that servers might experience at runtime. We picked a portion of the day 21st's workload (see Figure 5), extracted the web pages, lowered the number of requests with the factor of 2 (to factor in our smaller scale deployment topology), and finally used the Little's law [15] (i.e. $N = X(R + Z)$) to convert the obtained throughput (X) to the number of users (N) and think time (Z) used by emulated browsers of TPC-W. In this conversion we assumed that FIFA98 website had maintained the same response time over the sampling period. We then used the

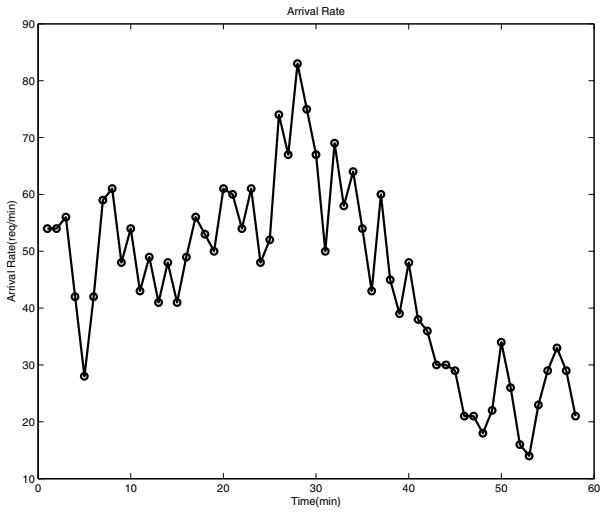


Figure 5: FIFA98 workload, day 21, over an hour.

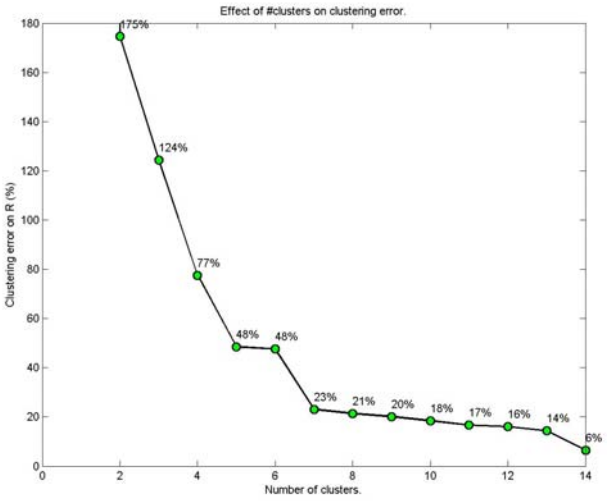


Figure 6: Modeling error decreases with the number of clusters.

obtained number of users to the TPC-W benchmark using equal mix of buying, browsing, and ordering scenarios.

For obtaining data, we monitored one of the web servers and the database and logged response times, throughputs, utilizations plus the workload parameters. Each sample represented a minute of work and the total length of experiment was 1 hour resulting into 60 samples.

We noticed that under this workload, the demands do not change frequently, and suspected that this is mainly because 1) the workload is unable to fully saturate the system or 2) the fact that combination of workload mixes are the same.

We performed three main analyses on this experiment. First we ran the algorithm statically with different number of clusters and measured the modeling error. In our evaluation, we used an expanded definition of the error metric given by its average value over the duration of the experiment (from t_1 to t_2):

$$E = \frac{\sum_{step=1}^{M} E(C)_{step}}{M}$$

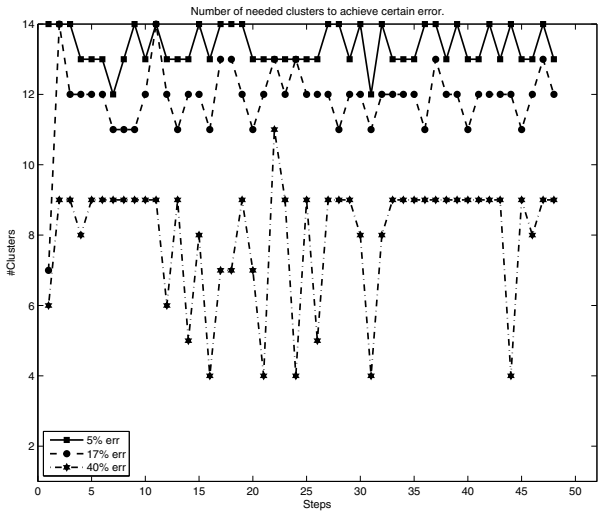


Figure 7: The minimal number of needed clusters to reach certain modeling error. We can reach 40% and 17% error, consecutively using 9 and 12 clusters on average. In order to reach 5% error we have to perform full clustering.

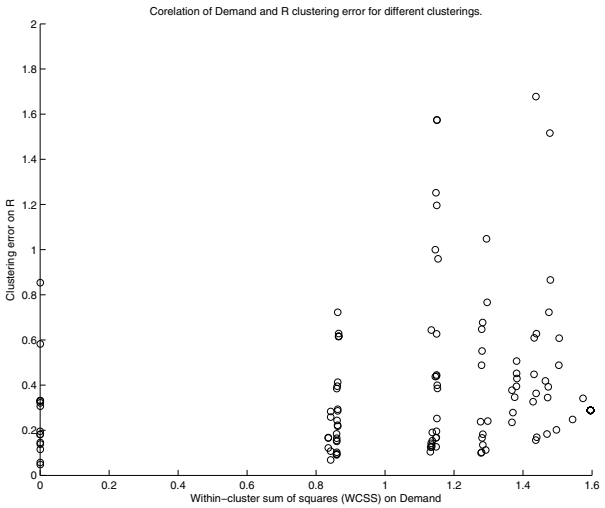


Figure 8: Correlation between WCSS for demands and the error achieved on response times estimation.

where *step* is a variable which ranges over the estimation steps. $E(C)$ is the modeling error defined in subsection 2.4.1 and M is the number of estimation steps.

As we expected, the error decreased while we increased the number of clusters (see Figure 6). The experiment also shows that modeling the system with one or even two classes introduces a large modeling error and that modeling with an intermediate number of classes (8, for example) might give us an acceptable modeling error.

In the second analysis, we applied our estimation and clustering algorithm to find the minimal number of needed clusters to reach certain modeling error. The algorithm is applied at each sampling period and, as a result, the clusters change dynamically. As Figure 7 shows, we can reach 17% error using between 7 and 12 clusters for the duration of the experiment. For a modeling error less than 40% we need 9 clusters on average. In order to reach 5%, there are sampling periods in which we need maximum number of classes.

The third analysis observed the correlation between the within cluster sum-of-squares (WCSS) for demands and the modeling error achieved using the Estimation and Classification Algorithm. We chose 140 different groupings (for each number of clusters we generated 10 random combinations). This let us navigate all possible WCSS's that could result from different groupings. Figure 8 shows that, on average we experienced a larger modeling error for the clusters with higher WCSS error. In other words, the modeling error is minimized, whenever the WCSS is minimized. As a result, our assumption is validated since K-means is exactly the algorithm to minimize the WCSS.

3.2 Estimation and Clustering for highly variable demands

In TPC-W experiment, the service demands did not vary in short run. As a result, it was unlikely that a URL moved from one cluster to another very often. However, in a long run (e.g. a month in real system measures) URLs might change place due to variation in the database records they access or in change of URL parameters. In short run, URL parameters variations may change the demands as well.

We performed a simulated experiment for such a case to investigate the effectiveness of the algorithm under non-uniform variations in demands. The simulator software used was CSim discrete event-based simulator.

To keep the presentation simple but also to highlight the merits of the proposed method, in the simulation we varied Z_c and the CPU demands, and kept N_c and the request frequencies constant.

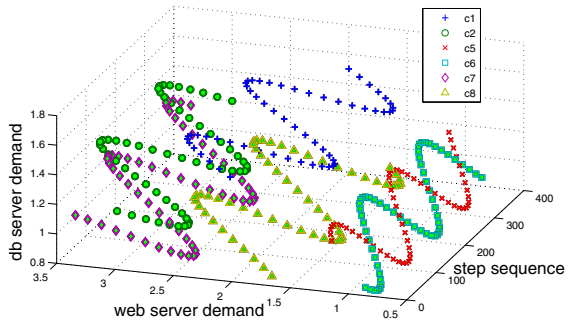
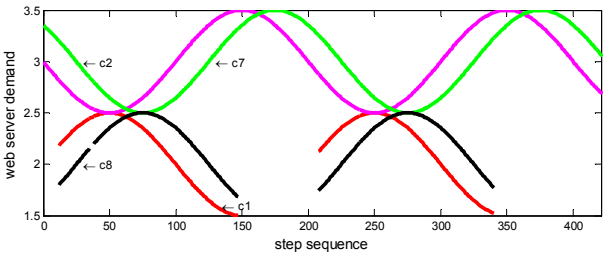
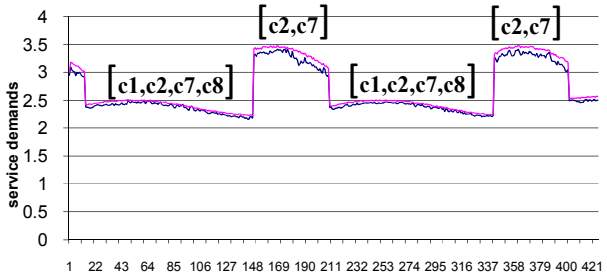


Figure 9: Combined Web-DB service demands for a specific set of classes [c1,c2,c5,c6,c7,c8].

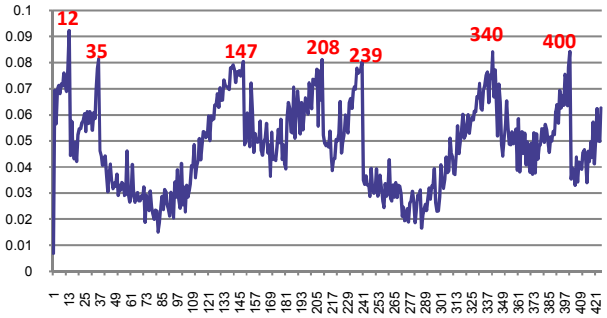
This experiment shows the efficiency of clustering and estimation for a web based application when the estimated parameters change at different rates and phases (small change in service demands). This web based application is an e-commerce site with 8 URLs (browse, buy, checkout, admin, login, logout, add, and remove). The URLs have the same number of users (N_c), mean user think time (Z_c), and time variant service demands $S_{w,c}$ and $S_{d,c}$. Service demands follow the sine curve with the same period but different phases (see Figure 9). Because of demand variations, URLs migrate from one cluster to another and clusters evolve over time. In real applications, the service demands are not going to change that dramatically, we consider those variations as a



(a)



(b)



(c)

Figure 10: Variation of service demands in (a), real and and tracked service demands for 2 clusters [c2,c7] and [c2,c7,c1,c8] in (b), and modeling error with dynamic clustering applied in (c).

stress load on the algorithm. Because of the variation of the service demands, we expect that the different classes will be re-clustered periodically.

Table 1 shows the clusters suggested by K-means algorithm. The acceptable modeling error A is set to 8% over 421 simulation steps. The column 'Action Nature' indicates the kind of change that has occurred when the past and current clustering are compared; Ag, Br, Re, and Mv stand for aggregation, breaking, re-structuring, and movement respectively. Notice that in each re-clustering, the clusters are re-computed from scratch; meaning that the algorithm is oblivious to 'Action Nature'.

Figure 10(a) shows the variation of service demands in a changing cluster ([c2,c7]+[c1,c8]). Real and tracked service demands for the cluster are shown in Figure 10(b) while the modeling error is depicted in Figure 10(c).

According to Figure 10(a) and demand formulas, at step 50, $(S_{w,1}, S_{d,1})$ and $(S_{w,2}, S_{d,2})$ get close. This suggests that c1 and c2 should be in the same cluster near that step. At step 150, the service demands of c1 and c2 are quite different,

Step	Action Nature	Grouping (Pre-event)	Grouping (Post-event)
12	Ag, Mv	[c1,c5,c6,c8] [c2,c7] [c3] [c4]	[c1,c2,c7,c8] [c3,c4] [c5,c6]
35	Br	[c1,c2,c7,c8] [c3,c4] [c5,c6]	[c1,c2,c7,c8] [c3] [c4] [c5,c6]
147	Mv	[c1,c2,c7,c8] [c3] [c4] [c5,c6]	[c1,c5,c6,c8] [c2,c7][c3][c4]
208	Ag, Mv	[c1,c5,c6,c8] [c2,c7] [c3] [c4]	[c1,c2,c7,c8] [c3,c4] [c5,c6]
239	Br	[c1,c2,c7,c8] [c3,c4] [c5,c6]	[c1,c2,c7,c8] [c3] [c4] [c5,c6]
340	Mv	[c1,c2,c7,c8] [c3] [c4] [c5,c6]	[c1,c5,c6,c8] [c2,c7][c3][c4]
400	Br, Mv	[c1,c5,c6,c8] [c2,c7] [c3] [c4]	[c1,c2,c7] [c5,c6,c8] [c3,c4]

Table 1: Changes in clustering structure during simulation.

indicating that these two classes are more likely in different clusters. These observations are consistent with our results where the clustering at step 50 is [c1,c2,c7,c8][c3][c4][c5,c6] and [c1,c5,c6,c8][c2,c7] [c3][c4] at step 150. Notice that c1 and c8 are shown in Figure 10(a) only when they are part of the cluster [c2,c7,...]. This explains the step-function-like behavior of the estimated demand for the changing cluster in Figure 10(b).

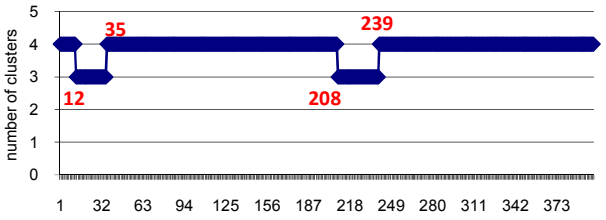


Figure 11: Number of clusters versus simulation steps.

Based on Figure 10(c) during the simulation, the modeling error exceeds the threshold A, 7 times. This means the classification algorithm described in subsection 2.4 is activated 7 times among the 421 steps. One could see that not every re-clustering necessarily results into a different number of clusters. For example, at step 147, the clustering changes from [c1,c2,c7,c8][c3][c4][c5,c6] to [c1,c5,c6,c8][c3][c4][c2,c7]. The number of clusters remains the same, but c1 and c8 have been moved to a different cluster. Figure 11 shows that the total number of clusters changes only 4 times over the 400 steps.

We can conclude that our estimation and classification algorithm works quite well, since it is able to keep the error below $A = 0.08$ with the smallest number of clusters and acceptable frequency of re-clustering.

The algorithm also satisfies our claim in Section 1: being able to exploit the complexity-accuracy trade-off and keeping both the error and the estimation cost low. In terms of cost, our algorithm yielded half the required clusters (see Figure 11) compared to full URLs estimation, which from the theory reduces the estimation cost by a factor 2^3 ($\frac{1}{2^3}$ of the original cost). This is due to the fact that, the cost of Kalman estimation in each step is dominated by Kalman

gain calculation, which is $O(l^3)$ while l is the number of measured variables. The reason is that the dominating term during gain calculation is a inversion of a matrix of size $l \times l$:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (4)$$

In this equation, R_k is measurement noise covariance matrix with size $l \times l$.

In terms of accuracy, it was able to keep the error near zero compared to fully aggregated URLs case. See Figure 12 for the error comparison between our algorithm, one cluster case, and full URLs estimation.

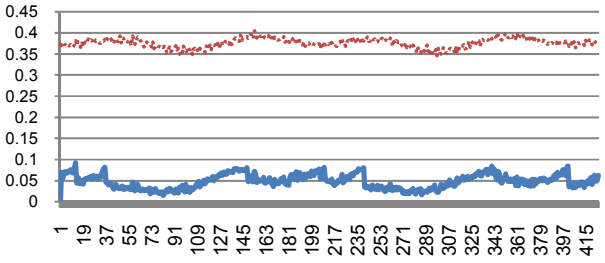


Figure 12: Comparison of the modeling error for one cluster case (dotted line), dynamic clustering case (solid line), and full URLs estimation (solid horizontal line at $y = 0$).

4. RELATED WORK

Some of the early works in CPU demand estimation are done in [20] and [21]. They used regression analysis to predict demands of certain program in a distributed system. The reference [4] uses regression splines instead of linear and polynomial regression functions to better capture the irregularities of demand functions. References [18, 31] employ the multivariate linear regression technique for a one-tier network. However, [31] focuses more on modeling the inter-request dependencies of session-based systems and [18] focuses on mechanisms to deal with issues such as insignificant flows, collinear flows, space and temporal variations, and background noise. In [30] and [16], multi-class queuing models were used to infer the per-class service times at different servers of a two-tier web cluster using throughput, utilization, and per-class response time measurements. They try to minimize the sum of predicted response time mean square errors using a non-linear optimization solvers and quadratic minimization programs. References [6, 5] claim to perform demand prediction of enterprise workloads by discovering patterns, but the referred demand is per-application not per-request; thus they do not tackle the same problem as ours. Finally reference [10] contributed by using Maximum Likelihood Estimation together with queuing model to estimate resource consumption in an ERP system.

Our tracking filter based approach is closer to what we previously presented in [27, 29, 32]: it uses LQM which is adaptable to open and closed workload types, and is designed to be used in online estimation and control. However, theoretically our approach is applicable to regression based techniques as well. One can apply regression to track changes in service demands by continually reapplying the technique on new measurement samples and feed the result into our clustering algorithm. In this case regression can be tuned by adjusting the number of past samples used.

The major contribution of this paper is the dynamic estimation of multi-class model parameters and the dynamic clustering of user requests. To the best of our knowledge, the combination of dynamic clustering and cluster parameter estimation has never been investigated before. Only reference [22] is close to our work in terms of final goal. They try to categorize requests based on their resource usage characteristics without performing server instrumentation. Thus they ignore the prior knowledge of which URL a request is accessing and only uses aggregate metrics such as total CPU usage over time. Moreover, their utilized approach is also different, as they use a machine learning technique called independent component analysis (ICA). Of course, their technique will erase the need for server instrumentation but might affect the accuracy due to not using the extra available information.

5. CONCLUSIONS AND FUTURE WORK

We took another step towards correct estimation of performance parameters by incorporating multi-class Layered Queuing Models into autonomic computing loops. Since the targeted applications are web applications, we considered the application URLs as first class entities: we started by considering each URL request as a class. However, from practical point of view, this creates models that are too complex to manage at runtime and requires detailed instrumentation to get per class service times, per class visit ratios, etc. To mitigate these two factors, we investigated a tracking approach which identifies performance parameters of groups of URLs (we call them clusters) instead of individual URLs. We proposed an algorithm that finds the appropriate the number of clusters with a pre-defined clustering accuracy.

We applied the clustering and tracking algorithm to two scenarios:

1) In the first experiment, we deployed our technique on TPC-W benchmark deployed on a cluster of web servers. The workload was obtained from well-known FIFA98 archives. In this experiment, first, we observed that modeling error is reduced as the number of clusters increases. Second, we tested 140 different random ways to cluster the classes and computed their average error values (E) and observed that the clusterings with the smaller average distance of URLs to centroids (i.e. with less cluster sum-of-squares) have less error; thus showing the usefulness of the K-means algorithm. Moreover, we dynamically computed the number of needed clusters that keep the modeling error under a threshold. For example, if one can accept 17% error, the number of needed clusters for estimation would be dropped from 14 to 9 on average.

2) In the second experiment we deployed our algorithm on a system with 8 URLs. It was shown that our Extended Kalman filter could track hidden states successfully and the correctness of the filter rose as it tried more classes and re-estimated service demands.

5.1 Future Work

In this paper, we used the maximum acceptable error value (8%) obtained from experience to control the balance between the introduced error and complexity of the model (noting that our goal was to minimize both). As future work, one can model the relationship between computation complexity and error explicitly and try to find an optimal number of clusters which minimizes both.

Another future challenge is the integration of multiclass model with the decision component of the adaptive loop. It is foreseeable that the decisions on provisioning to be made per class rather than per overall system, although the structure of the loop is still centralized. Another possible approach is to consider each class being controlled by one feedback loop and model separately the class interferences. This latter approach will lead to a decentralized adaptive system.

6. ACKNOWLEDGMENTS

This research was supported by the IBM Toronto Centre for Advanced Studies, and by OCE, the Ontario Centre of Excellence, as part of the program of the Centre for Research in Adaptive Systems (CERAS).

7. REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, pages 80–96, 2002.

[2] T. F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, June 2003.

[3] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, 2000.

[4] M. Courtois and M. Woodside. Using regression splines for software performance analysis. In *Proceedings of the 2nd international workshop on Software and performance*, pages 105–114, Ottawa, Ontario, Canada, 2000. ACM New York, NY, USA.

[5] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Capacity management and demand prediction for next generation data centers. In *Proceedings of the International IEEE Conference on Web Services*, pages 43–50, Salt Lake City, Utah, USA, July 2007. IEEE Computer Society.

[6] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proceedings of the IEEE 10th International Symposium on Workload Characterization (IISWC'07)*, volume 0, pages 171–180, Boston, MA, 2007. IEEE Communications Society.

[7] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing (ICAC '09)*, pages 117–126, Barcelona, Spain, June 2009. ACM.

[8] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-optimization in computer systems via online control: Application to power management. In *First International Conference on Autonomic Computing (ICAC'04)*, pages 54–61, New York, New York, May 2007. IEEE Computer Society.

[9] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, NY, USA, 1990.

[10] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 1–10, Pisa, Italy, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[11] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 15–22. IEEE Computer Society, 2009.

[12] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition Society*, 36(2):451–461, Feb. 2003.

[13] M. Litoiu. APERA (Application performance evaluator and resource allocation tool). www.alphaworks.ibm.com/tech/apera.

[14] M. Litoiu, M. Woodside, and T. Zheng. Hierarchical model-based autonomic control of software systems. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[15] J. D. C. Little. A proof for the queuing formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, June 1961. ArticleType: research-article / Full publication date: May - Jun., 1961 / Copyright l' 1961 INFORMS.

[16] Z. Liu, L. Wynter, C. H. Xia, and F. Zhang. Parameter inference of queueing models for it systems using end-to-end measurements. *Performance Evaluation, Elsevier*, 63(1):36–60, 2006.

[17] D. Menasce. TPC-W: a benchmark for e-commerce. *Internet Computing, IEEE*, 6(3):83–87, 2002.

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

[19] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. Technical, IBM T.J. Watson Research Center, Yorktown Heights, NY, May 2003.

[20] J. Rolia and V. Vetland. Parameter estimation for performance models of distributed application systems. In *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, page 54. IBM Press, 1995.

[21] J. Rolia and V. Vetland. Correlating resource demand information with ARM data for application services. In *Proceedings of the 1st international workshop on Software and performance*, pages 219–230. ACM New York, NY, USA, 1998.

[22] A. B. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker. Automatic request categorization in internet services. *SIGMETRICS Perform. Eval. Rev.*, 36(2):16–25, 2008.

[23] H. Tanizaki. *Nonlinear filters: estimation and applications*. Springer Verlag, 1996.

[24] V. Turau, A. K. Manjhi, T. Cain, T. Heil, E. Weglarz, T. Bezenek, and J. Kiefer. TPC-W java implementation. http://mitglied.multimania.de/jankiefer/tpcw/.

[25] E. A. Wan and R. V. D. Merwe. The unscented kalman filter for nonlinear estimation. In *The IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC*, pages 153–158, Alberta, Canada, 2000.

[26] G. Welch and G. Bishop. An introduction to the kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[27] M. Woodside, T. Zheng, and M. Litoiu. The use of optimal filters to track parameters of performance models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 74–84, Torino, Italy, 2005. IEEE Computer Society.

[28] M. Woodside, T. Zheng, and M. Litoiu. Service system resource management based on a tracked layered performance model. In *IEEE 3rd International Conference on Autonomic Computing, 2006 (ICAC'06)*, pages 175–184, Dublin, Ireland, June 2006. IEEE Computer Society.

[29] J. Xu, A. Oufimtsev, M. Woodside, and L. Murphy. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. In *Proceedings of the 2005 conference on Specification and verification of component-based systems (SAVCBS '05)*, Lisbon, Portugal, 2005. ACM.

[30] L. Zhang, C. Xia, M. Squillante, and W. N. M. III. Workload service requirements analysis: A queueing network optimization approach. In *Proceedings of 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems*, Washington, DC, 2002. IEEE Computer Society.

[31] Q. Zhang, L. Cherkasova, and E. Smirni. A Regression-Based analytic model for dynamic resource provisioning of Multi-Tier applications. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing*, page 27, Jacksonville, Florida, 2007. IEEE Computer Society.

[32] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai. Tracking time-varying parameters in software systems with extended kalman filters. In *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 334–345. IBM Press, 2005.