Application of Control Theory to a Commercial Mobile Service Support System

Payam Amani[a, 1], Bertil Aspernäs[c], Karl Johan Åström[b,] Manfred Dellkrantz[b], Maria Kihl[a], Gabriela Radu[a],
Anders Robertsson[b], and Andreas Torstensson[c]
[a]Dept. of Electrical and Information Technology, Lund University, Sweden
[b]Dept. of Automatic Control, Lund University, Sweden
[c]Ericsson AB, Karlskrona, Sweden
{payam.amani, maria.kihl}@eit.lth.se, {manfred.dellkrantz, andersro, kja}@control.lth.se,
luminita.radu@gmail.com, bertil.aspernas@telia.com, andreas.torstensson@ericsson.com

Abstract—**The Mobile Service Support system (MSS), which Ericsson AB develops, handles the setup of new subscribers and services into a mobile network. Experience from deployed systems show that traffic monitoring and control of the system will be crucial for handling overload situations that may occur at sudden traffic surges. In this paper we identify and explore some important control challenges for this type of systems. Further, we present analysis and experiments showing some advantages of proposed solutions. First, we develop a load-dependent server model for the system, which is validated in testbed experiments. Further, we propose a control design based on the model, and a method for estimation of response times and arrival rates. The main contribution of this paper is that we show how control theory methods and analysis can be used for commercial telecom systems. Parts of our results have been implemented in commercial products, validating the strength of our work.**

Keywords— Performance management; telecommunication systems; queuing theory; control theory; database servers; admission control; Kalman filters

I. INTRODUCTION

Resource management of computer systems, which has gained increased attention during recent years, was explored already in the late 60's [1][2]. It is an essential mechanism to handle load disturbances such as traffic surges and changes in user behavior. Poorly managed resources can severely degrade the performance of a system with potentially large financial consequences.

The work presented in this paper is motivated by a commercial Mobile Service Support System (MSS), developed and produced by Ericsson AB. Mobile Service Support Systems are used by the network operators for all processing regarding new subscribers and services in the network. Each new subscriber or service requires processing and data storage in several network nodes. The systems are in general multi-tier systems, implemented as distributed server clusters, where web and application servers process the incoming requests and database servers are used for data storage. The resource management of these systems, based on measurements of the system states such as actual utilization and response times, is crucial for the optimization of operation cost and the guarantee of service level agreements during load surges, for example during marketing campaigns or various events.

Therefore, the challenge is how to control system performance while providing guarantees on convergence and disturbance rejection. The solution is based on *dynamic control schemes*, which monitors the systems and provides actions when needed. Several types of resource–management mechanisms have been proposed and evaluated in the literature. In larger computer systems, *load balancing* is performed in order to distribute the demand for resources uniformly over a number of resource units (computers, CPUs, memory, etc.), thus avoiding the case that among the nodes with similar functionalities some are under-utilized while others are overloaded [3][4]. During overload periods, when more resources are requested than are available, *admission control* mechanisms reduce the load to the system by blocking or delaying some of the requests [5][6][7][8]. For Internet applications, virtualized server systems can be used to divide physical resources into a number of separated platforms where different web applications are allowed to operate without affecting one another. *Dynamic resource allocation* between the virtualized platforms serves as a new and easy way to perform resource optimization on web server systems [9][10][11]. In the last years, the field of *power and energy management* has become important. Large software systems have high energy consumption, which means that dynamic resource optimization of these systems may considerably lower the operating costs for the network operators [12][13][14][15].

However, all optimization techniques require accurate performance models of the involved computing systems. The operation region is mainly high traffic load scenarios, which means that the computing systems show non-linear dynamics that needs to be characterized accurately [16]. A software system is basically a network of queues, as examples, the CPU ready queue, semaphore queues, socket queues, and I/O device queues, which store requests in waiting of service in the processors. Therefore, queuing models can be used when

[1] This work represents the outcome of a long-term collaboration between Lund University and Ericsson AB and the contributors are listed in alphabetic order.

describing the dynamic behavior of server systems [1][17][18][19].

The concept of Load-Dependent Server (LDS) models, in which the response time of the jobs in the system is a function of the service time of the jobs and current number of jobs waiting to be served has, to the best of our knowledge, firstly been introduced in [20]. In [20][21][22], standard benchmarks were used for workload generation and also regression models to capture the system dynamics. In [23], a queuing network model which represents the load dependent behavior of the LDS was presented and validated with simulations. In [24], a theoretical analysis of the D/G/1 and M/G/1 models with load dependency assumptions was presented.

In this paper, we investigate solutions to some important control challenges identified for the commercial MSS developed by Ericsson AB. We present a load-dependent server model, which is validated in experiments. The model has been previously published in [25]. Further, we extend [25] by proposing and validating an admission control mechanism based on a load-adaptive controller. A modified version of the controller has been implemented in the Ericsson product. Finally, we show how extended Kalman filters can be used for estimating the response times and arrival rates in the system.

The paper is organized as follows. In Section II, the Ericsson product is described and the control challenges identified for the system are presented. In Section III, the testbed used for some of the experiments is described. In Section IV, the load-dependent server model is presented and validated. In Section V, the load-adaptive controller is presented and experiments validating its performance are described. In Section VI, our work on response time estimation based on extended Kalman filters is presented. Finally, in Section VII, some conclusions are presented.

II. System and Problem description

The Mobile Service Support system (MSS), which Ericsson AB develops, handles the setup of new subscribers and services into a mobile network. It presents to the operator and its business support systems a unified middleware where complex functions, such as setting up a new subscriber or modifying services for an existing subscriber, can be easily invoked. The software architecture is complex with several layers and distributed infrastructures, which means that specific parts of the system will not have complete knowledge of the interactions among other parts of the system.

A. System architecture

The system architecture is illustrated in Figure 1. One request to the MSS from an upstream system normally results in a number of requests downstream out on the mobile network to several different network elements (NEs). A network element is usually a database storing subscriber and service data, for example, the Home Location Register (HLR). A user id, which needs to be fetched from one database, needs to be supplied in a query to another database to get the system consistent.

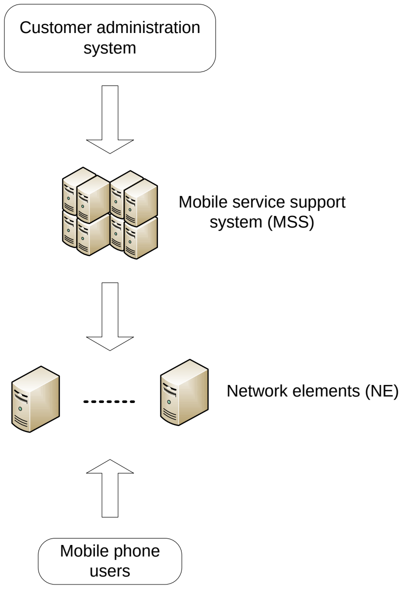
Customer administration system
Mobile service support system (MSS)
Network elements (NE)
Mobile phone users

Figure 1. Mobile service support system (MSS)

In parallel to the changes and setups that the MSS performs, the network is also used by the end users. Services being set up by the MSS are queried by base stations and other systems requiring that information. In respect to the MSS, this traffic can be considered as unknown background traffic, in contrast to the known traffic flowing through the MSS.

B. Control challenges

The experience from deployed Ericsson systems shows that there can be problems with overload in the NEs. The measurable load arriving from the MSS and the unknown (not directly measurable) load arriving from mobile users may interfere with each other, creating a race for resources that may lead to overload in a NE. When one NE becomes overloaded and unresponsive, this may result in the entire transaction requiring rollback to avoid in-consistencies in the network. Such a rollback may require manual work which is of course costly for the operator.

To protect against such situations, traffic monitoring and control are crucial. In cooperation with Ericsson AB, some important control challenges have been identified for this type of system. These challenges are described below. In the following sections our collaborative work on these challenges will be presented. The models and control designs are based on response times, as this metric is rather easily measurable in the real system and because the response times can be mapped to the load status of the controlled system using the proposed model.

1) Performance models

The first challenge is to design a performance model for the NEs, since good control designs are based on sufficiently accurate system models. The model should capture the dominant load dynamics of the NEs. Most service performance metrics such as response times and service rates depend on queue state dynamics, which means that queue models are suitable for these systems.

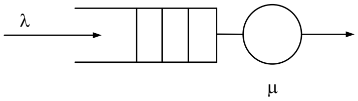

Figure 2. M/M/1 model

For the objective of performance control, simple models, such as single server queues, are often preferred. The model should only capture the dominating load dynamics of the system, since a well-designed control system can handle many model uncertainties [26].

The classical M/M/1 model, where a single-server queue processes requests that arrive according to a Poisson process with exponential distributed service times, see Figure 2, has been shown to accurately capture the response time dynamics of a web server system [27]. However, experience from deployed systems and lab measurements have shown that databases may not have M/M/1 dynamics [28]. Therefore, other models are required that more accurately captures the dynamics of database servers.

2) Admission control in MSS

The NEs are loaded by two traffic sources, the measurable traffic coming to the MSS and the unknown (unmeasurable) traffic coming from the mobile users, as illustrated in Figure 3. The average arrival rates can be denoted as λ for the measurable traffic and λ_u for the unknown traffic. Overload in the NEs can be detected by monitoring the response time of requests sent to each node. When the average requests' response times exceed some threshold, the MSS can classify the involved NE as overloaded and thereby start actions to lower the arrival rate to that particular NE, in order to achieve an acceptable arrival rate, denoted as λ_c. Therefore, the second control challenge is to design an admission control scheme that can handle the unknown traffic at the NEs and further can handle the time varying mean measured traffic rates experienced in the systems.

3) Monitoring and estimation

One of the problems when designing control mechanisms in these types of systems is the lack of performance information. The designed protocols basically provide no means of control communication between the MSS and the NEs that can be used by a control system. Therefore, the third control challenge that has been identified is the design of monitoring and estimation mechanisms that could help in the design of, for example, an admission control scheme. The estimation scheme can be used as feed-forward control in the control system, and thereby improving the performance of the control system compared to when only using feedback control. In collaboration with Ericsson AB, some preliminary work on the application of extended Kalman filters for load estimation have been started for systems as in Figure 3.

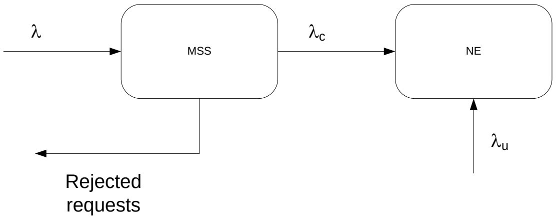

Figure 3. Load at the NEs

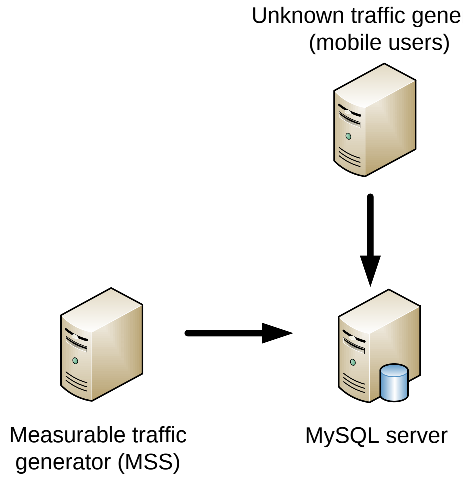
Unknown traffic generator (mobile users)
Measurable traffic generator (MSS)
MySQL server

Figure 4. Testbed for the experiment.

III. TESTBED

To validate some of the proposed solutions, we have performed a series of experiments in our server lab. We developed a MSS testbed with two traffic generators, one for the measurable traffic and one for the unknown traffic, and a MySQL 5.1.41 database server as depicted in Figure 4. The computers were connected to a local 100 Mbit/s Ethernet network.

The traffic generators were implemented in Java, using the JDBC MySQL connector, and they were executed on computers with an AMD Phenom II X6 1055T Processor at 2.8 GHz and 4 GB main memory. The operating system was Ubuntu 10.04.2 LTS. The traffic generators use 200 working threads and generate MySQL queries according to a Poisson process with average rate λ and λ_u queries per second. Both traffic generators were validated in order to guarantee that they were not a bottleneck in the experiments.

The database server has several relations with the same structure but with different number of tuples. The maximum number of allowed concurrent connections is set to 100. The structure of the relations comes from the Scalable Wisconsin Benchmark [29] with 10 million tuples. Two basic types of queries are used, `SELECT` (read) and `UPDATE` (write).

The queries look like this:

```
 SELECT * FROM <relation> WHERE unique1=?;
 UPDATE <relation> SET unique2=? WHERE
unique1=?;
```

The question marks are replaced with uniformly distributed random numbers from zero to ten million.

IV. PERFORMANCE MODELS

In this section, we focus on the modeling aspects of database servers. The objective is to develop a performance model for the database server that captures the dynamics during high loads. The performance model can be used in resource optimization schemes, as admission control systems, in order to maximize the throughput of the database server, while keeping some latency constraints. One of the challenges for these database servers is that they have a write-heavy workload, which means that the CPU is not the bottleneck during high loads. This means that previous work on performance modeling of server systems may not be applicable since they assume CPU-intensive workload.

A. M/M/m model with load dependency (M/M/m-LDS)

We propose to add load-dependency to an M/M/m system. In all load-dependent server models, the service time for a request will be dependent on the number of concurrent requests in the system. This load-dependency will model effects of the operating system, memory use, etc., which may cause service degradation when there are many concurrent jobs in a computing system [22]. In the experiment section, we will show that the M/M/m-LDS model accurately captures the behavior of various database workload.

The properties of the load dependent M/M/m model (*M/M/m-LDS*) are set by an exponential distributed base processing time, x_{base} =1/μ, and a dependency factor, *f*. When a request enters the system, it gets the base processing time x_{base} assigned to it. A single request in the system will always have a processing time of x_{base}. Each additional request inside the system increases the residual work for all requests inside the system (including itself) by a percentage equal to the dependency factor *f*. When a request leaves the system all other requests have their residual work decreased by *f* percent again. This means that if *n* concurrent requests enter the system at the same point, they will all have a processing time of

$$x_s(n) = x_{base} \cdot (1+f)^{n-1} \qquad (1)$$

A special case is when *f* = 0. It means that there is no load dependency, and all requests will have processing time x_{base}.

The system can process a maximum of *m* concurrent requests at each time instance. Any additional request will have to wait in the queue. New requests arrive according to a Poisson process with average rate λ.

Therefore, the system can be modeled as a Markov chain as illustrated in Figure 5.

The average service rate of the system depends on the number of concurrent requests in the system, *k*, derived as follows:

$$\mu_k = \begin{cases} \dfrac{k\mu}{(1+f)^{k-1}} & if \quad 0 < k < m \\ \\ \dfrac{m\mu}{(1+f)^{m-1}} & if\ k \ge m \end{cases} \qquad (2)$$

By solving the balance equations, stationary probability distribution of existence of *k* concurrent requests in the system is calculated as below:

$$\pi_k = \begin{cases} \dfrac{\left(\dfrac{\lambda}{\mu}\right)^k}{k!}(1+f)^{\frac{k(k-1)}{2}}\pi_0 & if \quad 0 < k < m \\ \\ \dfrac{\left(\dfrac{\lambda}{\mu}\right)^k}{m^{k-m}\cdot m!}(1+f)^{(m-1)(k-\frac{m}{2})}\pi_0 & if\ k \ge m \end{cases} \qquad (3)$$

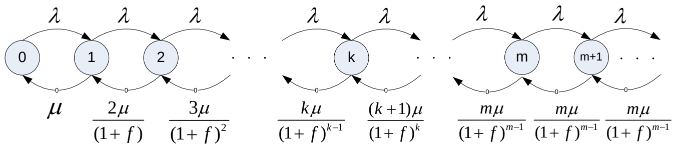
λ
λ
λ
λ
λ
λ
λ
λ
0
1
2
. . .
k
. . .
m
m+1
. . .
μ
$\frac{2\mu}{(1+f)}$
$\frac{3\mu}{(1+f)^2}$
$\frac{k\mu}{(1+f)^{k-1}}$
$\frac{(k+1)\mu}{(1+f)^k}$
$\frac{m\mu}{(1+f)^{m-1}}$
$\frac{m\mu}{(1+f)^{m-1}}$
$\frac{m\mu}{(1+f)^{m-1}}$

Figure 5. Illustration of M/M/m-LDS model as a Markov chain.

As the sum of the probabilities of all possible states equals to one, π_0 can be derived as follows:

$$\sum_{k=0}^{\infty}\pi_k = 1 \rightarrow$$

$$\pi_0 = \frac{1}{1+\sum_{k=1}^{m-1}\dfrac{\left(\dfrac{\lambda}{\mu}\right)^k}{k!}(1+f)^{\frac{k(k-1)}{2}} + \dfrac{\mu\left(\dfrac{\lambda}{\mu}\right)^m(1+f)^{\frac{m(m-1)}{2}}}{(m-1)!(\mu m-\lambda(1+f)^{m-1})}} \qquad (4)$$

The stability condition in this case is

$$\frac{\lambda}{\mu m}(1+f)^{m-1} < 1 \qquad (5)$$

The average number of requests in the system, *N*, can be calculated as below:

$$N = \sum_{k=1}^{\infty} k \cdot \pi_k = N_1 + N_2$$

$$N_1 = \sum_{k=0}^{m-1}\frac{\left(\dfrac{\lambda}{\mu}\right)^k(1+f)^{\frac{k(k-1)}{2}}}{(k-1)!}\pi_0 \qquad (6)$$

$$N_2 = \frac{\left(\dfrac{\lambda}{\mu}\right)^m(1+f)^{\frac{m(m-1)}{2}}(\mu m^2-\lambda(m-1)(1+f)^{m-1})\mu}{(m-1)!(m\mu-\lambda(1+f)^{m-1})^2}\pi_0$$

Finally by means of Little's theorem [30], the average time each request spends in the system, *T*, can be derived as follows.

$$T = \frac{N}{\lambda} \qquad (7)$$

B. M/M/m/n model with load dependency (M/M/m/n-LDS)

In case that the queue is limited to *n* positions, the probability for an empty system, π_0, can be determined as follows. This queuing system is named as *M/M/m/n-LDS*.

$$\pi_0 = \frac{1}{I+II+III}$$

$$I = 1+\sum_{k=1}^{m-1}\frac{\left(\dfrac{\lambda}{\mu}\right)^k(1+f)^{\frac{1}{2}k(k-1)}}{k!} \qquad (8)$$

$$II = \frac{(1+f)^{\frac{1}{2}m^2+\frac{1}{2}m+mn-n-1}\lambda^{n+m+1}}{m^n\mu^{n+m}m!(\lambda(1+f)^{m-1}-\mu m)}$$

$$III = -\frac{(1+f)^{\frac{1}{2}m(m-1)}\lambda^m}{\mu^{m-1}(m-1)!(\lambda(1+f)^{m-1}-\mu m)}$$

Further, the average number of requests in the system is as follows:

$$N = N_1 - N_2$$

$$N_1 = \sum_{k=0}^{m-1} \frac{k\left(\frac{\lambda}{\mu}\right)^k (1+f)^{\frac{1}{2}k(k-1)} \cdot \pi_0}{k!}$$

$$N_2 = \frac{\mu(1+f)^{\frac{1}{2}m^2-\frac{1}{2}m-1}}{m^{m-1}\left(-\lambda(1+f)^{m-1}+\mu m\right)} \cdot \frac{N_{2_{n1}} + N_{2_{n2}} - N_{2_{n3}}}{N_{2_{D1}} + N_{2_{D2}} + N_{2_{D3}}}$$

$$N_{2_{n1}} = -\lambda(n+m)(1+f)^{\left(\frac{1}{2}`m^2+\frac{3}{2}m+mn-n-1\right)}\left(\frac{\lambda}{\mu}\right)^{n+m+1}\left(\frac{1}{m}\right)^{n+1}$$

$$N_{2_{n2}} = \left(m(1+f)^m \mu(n+m+1)(1+f)^{\left(\frac{1}{2}`m^2+\frac{1}{2}m+mn-n\right)}\right)\left(\frac{\lambda}{\mu}\right)^{n+m+1}\left(\frac{1}{m}\right)^{n+1}$$

$$N_{2_{n3}} = \left(-\lambda(1+f)^m \mu(m-1) + (1+f)\mu m^2\right)\left(\frac{\lambda}{\mu}\right)^m (1+f)^{\frac{1}{2}m(m-1)}$$

$$N_{2_{D1}} = \left(\frac{1}{m}\right)^m (1+f)^{\frac{1}{2}m(m-1)} m!\left(-\lambda(1+f)^{m-1}+\mu m\right)\left(\sum_{k=1}^{m-1}\frac{\left(\frac{\lambda}{\mu}\right)^k (1+f)^{\frac{1}{2}k(k-1)}}{k!}\right) \quad (9)$$

$$N_{2_{D2}} = \left(\frac{-\lambda(1+f)^{m^2+mn-n-1}}{(\mu m)^{n+m}}\right) + \left(\frac{1}{m}\right)^m (1+f)^{\frac{1}{2}m(m-1)} m!\left(-\lambda(1+f)^{m-1}+\mu m\right)$$

$$N_{2_{D3}} = \mu m\left(\frac{\lambda(1+f)^{m-1}}{\mu m}\right)^m$$

Finally, the average response time for a request can be derived using Little’s theorem.

C. Parameter tuning

In a telecom system with latency constraints, the dominant dynamic of the system is often characterized by the average response time, *T*, when varying the average arrival rate, *λ*. Tuning of the parameters of the LDS model in a way that it fits the measured data from the actual server system is a necessary step in modeling of such systems. Assuming that λ and *T* are measureable, there are three main parameters for the M/M/m-LDS model, *m*, *f* and μ, to tune in order to fit the model on the measured data. Further, for the M/M/m/n-LDS there is an extra parameter, *n*, to tune.

Therefore, in Figures 6-10, the effects of changing model parameters on dynamics of average response time versus mean arrival rate of queries are illustrated. In the rest of the paper, this graph will be called the *λ/T* graph. In each figure, it is assumed that two (three) of the parameters are fixed and the one that is mentioned is the variable. As the equations for calculating the mean response times are rather complex and the parameters are interdependent, more than one set of parameters can be fit on the measured data. Thus using these figures, a heuristic rule for tuning the parameters of the LDS model can be achieved.

In the cases where the M/M/m-LDS model is used, the first parameter to be tuned is the number of servers, *m*. As it can be seen in Figure 6, by increasing the maximum number of concurrent requests that can be processed in the system, the linear part of the λ/T graph will be shorter and the exponential rising rate of the graph is increased. In this case it is assumed that (*f*, μ) = (0.6, 22).

The second parameter to be tuned is the dependency factor, *f*. As shown in Figure 7, by decreasing the dependency factor, the linear part of the λ/T graph is increased, however, the change is slower than in the case where *m* is decreased. On the other hand the exponential rising rate of the graph is increased in comparison with the case where *m* is decreased. Here, it is assumed that (*m*, μ) = (3, 22).

The effects of changing μ on the λ/T graph while fixing the two other parameters is illustrated in Figure 8. As shown in the figure, by increasing μ in equal steps, the λ/T graph will be shifted to the right in equal steps. In this case, where (*m*, *f*) = (3, 0.6), the rate of rising of the graph is decreased.

In cases where the M/M/m/n-LDS model is used, there will be a saturation of the response times when the load is high enough to overload the queue. Here, it is assumed that the default values are (*m*, *n*, *f*, μ) = (4, 15, 0.6, 22). Figure 9 and Figure 10 show the effects when varying *m* and *f* respectively. In each case, the values of the other three parameters are constant. The general effect of changing the parameters is similar as for the case with the infinite queue, with the difference that the response times saturate when the load is high.

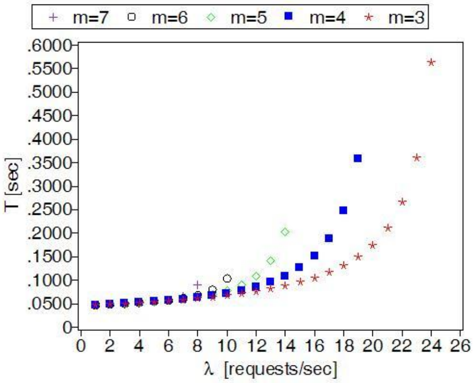

Figure 6. Variations of the λ/T graph for a special scenario with *m* as variable when (*f*,μ) = (0.7, 22).

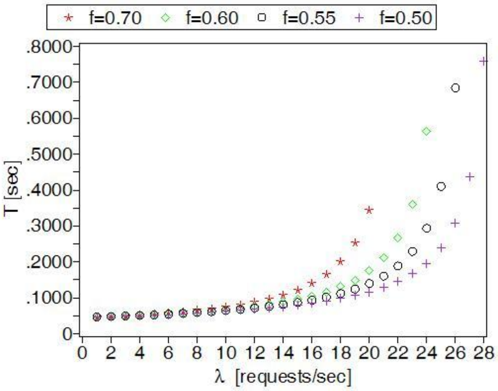

Figure 7. Variations of λ/T graph for a special scenario with *f* as variable when (*m*,μ) = (3, 22).

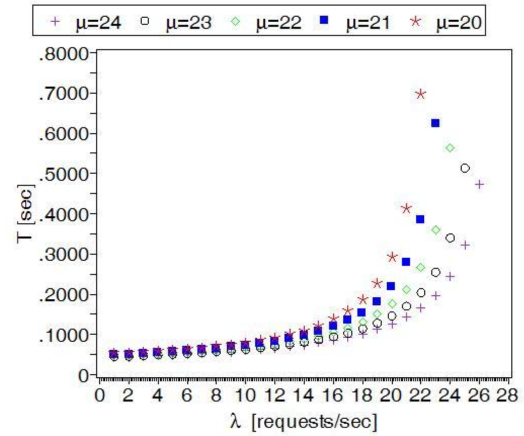

Figure 8. Variations of λ/T graph for a special scenario with μ as variable when (*m*, *f*) = (3, 0.6).

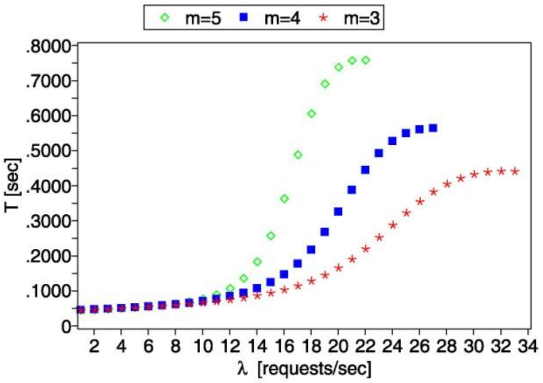

Figure 9. Variations of λ/T graph for a special scenario with *m* as variable when (*n*, *f*, μ) = (15, 0.6, 22).

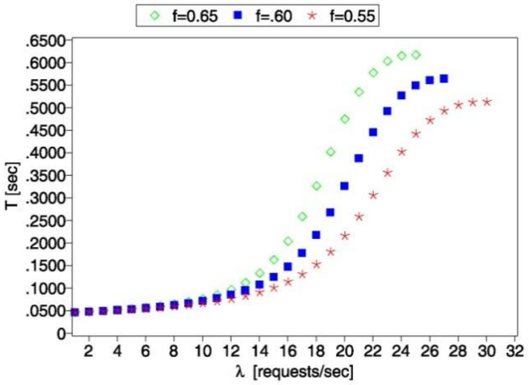

Figure 10. Variations of λ/T graph for a special scenario with *f* as variable when (*m*, *n*, μ) = (4, 15, 22).

D.Experiments

In order to validate the model, we have performed a series of experiments in our testbed, as described in Section III. In this case, the arrival rate of the unknown traffic was set to zero. The dynamics of the database server highly depends on the mix of requests, since SELECT and UPDATE queries require different amount of server capacity. Therefore, experiments with varying workload mix have been performed.

Figure 11, Figure 12, and Figure 13 show the results from experiments where the arrival rate is varied from low load to high load. The graphs show the average response times of queries as a function of the arrival rate. We have fitted M/M/m/n-LDS models for the data using the tuning steps described in the previous section. In both scenarios, the CPU utilization was very low, also for high loads. The maximum CPU load was about 5%.

In order to model the network delays, we have added a bias of 0.023 seconds in the average response times of the proposed models.

In Figure 11, the workload is based on 100% UPDATE queries. The fitted model in this case has the following parameters (*m*, *n*, *f*, μ) = (3, 81, 0.75, 37.1). Figure 12 depicts the same experiment setup when using a mix of 25% SELECT queries and 75% UPDATE queries. The fitted M/M/m/n-LDS model in this case has the following parameters (*m*, *n*, *f*, μ) = (6, 73, 0.44, 35.2). In Figure 13 only SELECT queries are used. In this case the model parameters are (*m*, *n*, *f*, *μ*) = (6, 240, 1.39, 38).

The results verify that the proposed model can represent the average dynamics of a database server with various workloads very well

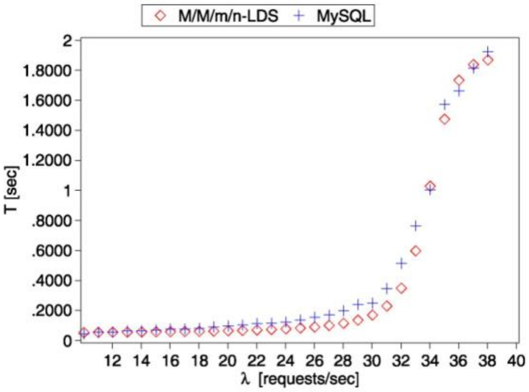

Figure 11. Performance of the M/M/m/n-LDS queuing model in modeling steady state dynamics of a MySQL database server using UPDATE queries.

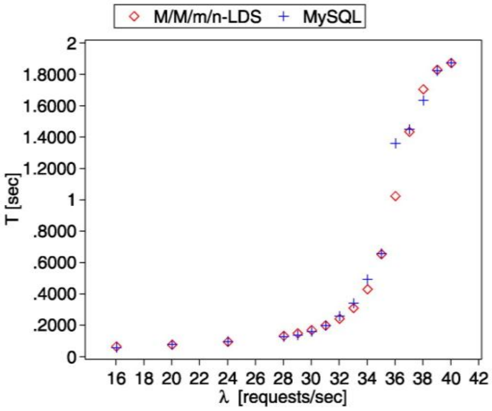

Figure 12. Performance of the M/M/m/n-LDS queuing model in modeling steady state dynamics of a MySQL database server using mixed queries.

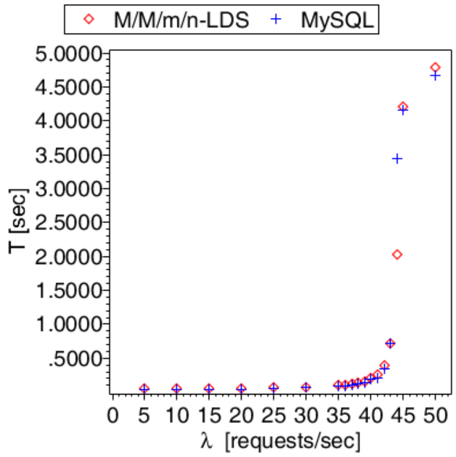

Figure 13. Performance of the M/M/m/n-LDS queuing model in modeling steady state dynamics of a MySQL database server using SELECT queries.

V. Admission Control

As part of the collaboration with Ericsson AB, we have designed an admission control mechanism for the measurable traffic to the NEs, as illustrated previously in Figure 3. As a direct effect of this work, a modified version of the control mechanism has been implemented in the Ericsson product. In this section, the controller design and its validation are described.

A. Control structure

The MSS includes a control system, as illustrated in Figure 14, which should ensure that the load on a specific NE is kept at an acceptable level. The control objective is to keep the mean response times of the NE queries below a desired value while maximizing the throughput. The control actions must be based on a limited amount of control information, due to the standardized protocols and the layered software architecture. The control system includes a controller and a gate.

The controller uses a response time reference value, T_{ref}, and measurements to determine an acceptable workload to the database server. The acceptable workload is defined by the normalized rate of admitted queries, λ_A, which corresponds to the ratio of the average arrival rate of the admitted requests over the higher bound of the average arrival rate of the requests. It is desired that the control system performs robustly in presence of fluctuations in the average arrival rate of the queries sent to the database. Therefore, the controller design is crucial for guaranteeing the control objectives.

The gate ensures the ratio λ_A of arriving queries is admitted to the database. In the experiments, the gate rejects requests that cannot be admitted. However, in the real product, this is not feasible. Instead, the real product has a traffic shaping mechanism that adds delays to the responses to the customer administration system. Since the communication with the customer administration system is synchronous, adding delays to the responses will lower the arrival rate of requests.

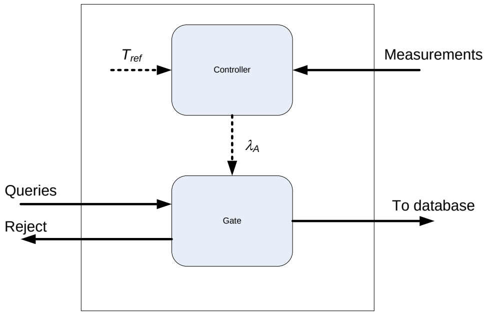

Figure 14. Control system

In this paper, we focus on the controller performance. Therefore, the implementation of the gate is not the main focus as long as it can be assumed that the gate actuates the control signal accurately.

B. Controller design

We have designed a controller that can guarantee the control objectives for the system. The controller, called the Load-Adaptive Controller (LAC), only uses measurements of the query response times. A classical PID controller [26] includes one Proportional part (P), one Integral part (I), and one Derivative part (D) that determines the control signal based on the deviation of the input signal from the reference value. For stochastic systems, the derivative part will amplify the effect of high frequency noise in the response time error and thus deteriorate the overall performance of the system.

Therefore, the LAC is based on a modified PI controller with anti-windup. The LAC adapts its proportional gain with the variations in the mean arrival rates of queries sent to the database. The structure of the modified PI controller is illustrated in Figure 15.

The total load of the NE is determined by the aggregated arrival rates of the measurable and the unknown traffic streams. However, assuming that the unknown traffic is stationary during a limited time period and that the database server behaves as a conservative queuing system [30], a specific admitted ratio of the traffic will correspond to a specific mean response time, as illustrated in Figure 16.

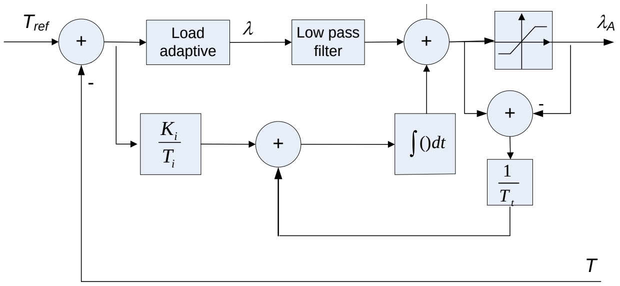

Figure 15. Load-adaptive controller (LAC)

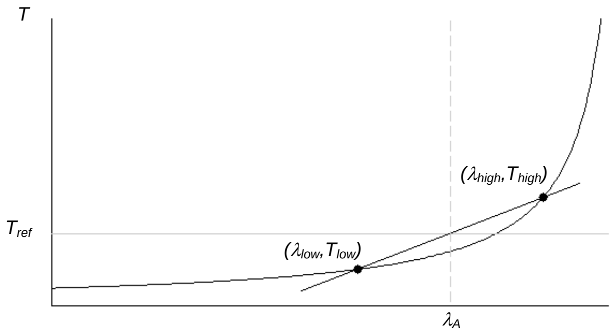

Figure 16. An illustration of the LAC calculations.

The controller continuously keeps track of two points in this graph, one low point, (λ_{low}, T_{low}), which is situated below the reference response time, T_{ref}, and one high point, $(\lambda_{high}, T_{high})$, which is situated above T_{ref}. As the control system operates only based on measured response times of NE queries, λ_{low} guarantees that those measurements exist for all sampling intervals. The upper limit for mean arrival rates of the queries processed by the NE while not overloading the database is represented by λ_{high}. The starting values for λ_{low} and λ_{high} are set to 5% and 100% respectively.

The admittance rate of the incoming queries is iteratively updated so that its corresponding response time meets the desired value. Every sampling time, the controller calculates the average response time, T, over the last period. If the average response time during sampling period k, T_k, is too high, ($T_k>T_{ref}$), the high point is updated as $(\lambda_{high}, T_{high}) = (\lambda_k, T_k)$ where λ_k is the normalized admitted arrival rate during interval k. If the average response time during interval k is too low, ($T_k<T_{ref}$), the low point is updated as $(\lambda_{low}, T_{low}) = (\lambda_k, T_k)$. It is now assumed that the optimal normalized arrival rate, λ_o, which gives a response time of exactly T_{ref} is in the interval $[\lambda_{low}, \lambda_{high}]$. Therefore, the next normalized admitted arrival rate, λ_{k+1}, can be interpolated from these points using classic geometry:

$$\lambda_{k+1} = \lambda_k + \frac{\lambda_{high} - \lambda_{low}}{T_{high} - T_{low}}(T_{ref} - T_k) \tag{10}$$

Therefore, the quotient $(\lambda_{high}-\lambda_{low})/(T_{high}-T_{low})$ is used as proportional gain in the P-part of the controller. The algorithm will converge to the desired response time value assuming that the arrival process is stationary or slowly changing. It is obvious that the control gate cannot admit more queries than the incoming ones. This upper limit will be noted in the calculations and treated as a saturation limit of the control signal.

The integral I-part of the controller is used when the P-part is not enough for keeping the steady state error to zero. The integral part uses a controller parameter, K_i, which in conventional PI controllers are equal to the proportional gain. However, in this case, as the proportional gain changes drastically due to the load-adaptive algorithm, using the conventional PI structure will lead to a reduced phase margin which will drive the system to unstable region. Therefore, K_i is chosen as a static gain and its suitable value is determined in tuning phase of the controller.

Further, the parameter T_i is the integration time constant and T_t is the integrator‘s reset time constant in the anti-windup mechanism. Anti-windup is added to avoid building up of the integration part when the control gate is saturated or completely open. It is desired to choose small values for T_t so that the integrator resets quickly. Generally, T_t is chosen to be less than T_i.

A low pass filter is added after the proportional gain to smoothen the response time error signal as it is very noisy. The bandwidth of this filter should be suitably chosen so that its effect on the in-band characteristics of the response time errors is minor while attenuating high frequency components of that signal.

C. Experiments

To investigate the controller performance, a Java implementation of the controller was deployed as a web application to a Glassfish application server, placed on the server acting as traffic generator in Figure 4. The web application also included the traffic generator that generated requests for the web application. For each request, the admission control decides whether to allow the request to be sent to the database or rejected. The traffic generator for unknown traffic did not have an admission control, and was set to a specific average arrival rate that could be altered during run time. All requests sent to the database server were SELECT queries (according to the query structure described earlier). The λ/T graph for this particular scenario setting is shown in Figure 17. The saturation of the system is not shown in the graph for clarity reasons, since the operation region is around the “knee”.

To test the performance of the controller, a scenario was chosen where the load changed from slight overload to high overload. The reference response time, T_{ref}, was set to 0.2 seconds. According to the λ/T graph in Figure 17, this corresponds to a total arrival rate of approximately 40 queries per second.

In this paper, two experiments are shown, one with a step in the unknown traffic and one with a step in the measurable traffic. The controller parameters were set to T_i=4, K_i=0.5, T_r=1, and the sampling time h=0.5 seconds. T_i was determined as a multiple of the sampling time, chosen so that the controller was able to maximize the throughput while keeping the mean response times below T_{ref}. K_i was set equal to the sampling time. To give the controller time to settle this state was kept for 100 seconds after which a step in the traffic was performed. The resulting graphs are shown in Figure 18 and Figure 19. The graphs show the average dynamics from 100 runs.

In the first experiment, shown in Figure 18, the starting arrival rate was set to 23 requests per second for the measured traffic and 22 requests per second for the unknown traffic. The step increased the arrival rate of the unknown traffic by 10 (to 32) requests per second, resulting in a more severe overload situation.

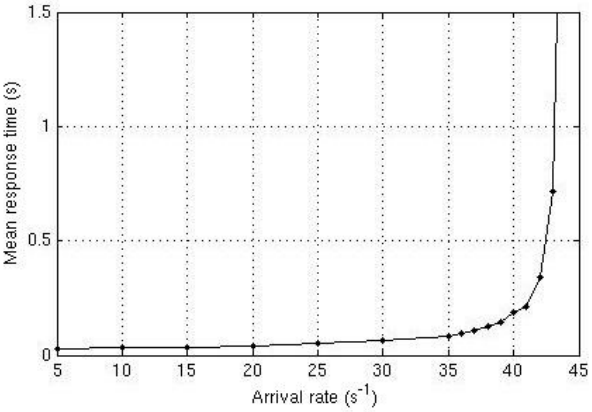

Figure 17. λ/T graph for the admission control experiments.

The second experiment, shown in Figure 19, was similar to the first experiment. However, the arrival rate step was in the measurable traffic instead. To obtain a similar control signal response as in the first experiment, the step in the controllable traffic had to be larger. Therefore, the observable arrival rate was increased from 23 requests per second to 51 requests per second.

Both experiments show a well-behaved controller, with a reasonable settling time and smooth dynamics after the step.

VI. Monitoring and Estimation

The system in Figure 1 is complicated with many different queues, caches and databases. Attempting to capture all details gives models that are too complex for on-line control. Extensive experience in the field of control has clearly demonstrated that simple models that capture essential behavior can be very beneficial [31]. One aspect of the collaboration with Ericsson has been to explore if benefits can also be obtained for monitoring and control of the MSS. A crucial issue is what complexity of the models is required for estimation and control of the MSS.

Response time and arrival rates are variables of prime concern. The variables have strong variations, which can be reduced by averaging. A more effective way is to construct estimators that exploit the dynamic behavior of the system. Exploration of such estimators has been one of the goals of the project.

A key feature of the system shown in Figure 1 is that there are two traffic streams. The measured traffic, generated by the customer administration system has a known arrival rate $\lambda_{c,}$ can be controlled. The unknown stream, which is created by the mobile phone users, has an arrival rate λ_u that cannot be controlled. Monitoring and control of the system can be improved if good estimates of the average service time are available.

An abstraction of the system in Figure 1 is shown in Figure 20, where an estimator and the controller have been included. In this section, we will focus on the estimator, which only has access to measurements of the measured arrival stream λ and the response time T. All actions by the NEs and the MSS have been represented by one queue that represents the aggregated behaviors.

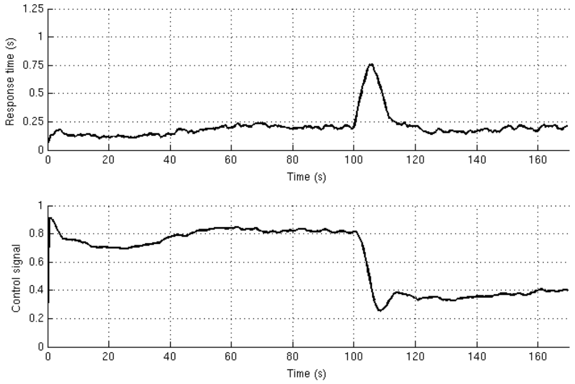

Figure 18. Performance of the LAC with step in unknown traffic.

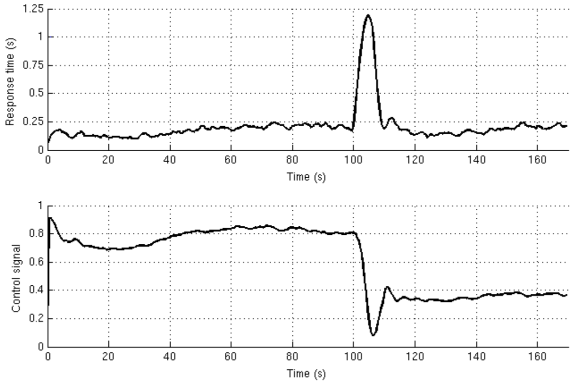

Figure 19. Performance of the LAC with step in observable traffic.

The queue length is represented by the variable *x*, which captures the aggregated behavior of many different queues in the real system. The variable *x* can be interpreted as a virtual queue length. The queue length cannot be measured. The actual response time *T* and the actual arrival times can, however, be measured. Variations in *x* reflect changes in the system‘s load.

A. Flow Model

To model the system, we will make an additional abstraction by assuming that the variables *x* and *T* are continuous and that they vary continuously in time. The behavior of the system can then be captured by the simple flow model:

$$\frac{dx}{dt} = \lambda - \mu_{\max} f(x) \tag{11}$$

where *x* is the virtual queue length, λ_c is the known arrival rate, λ_u is the unknown arrival rate, μ_{max} is the maximum service rate and *f* is a monotone function with the range [0, 1]. The response time is given by

$$T = t_0(1+x) = t_0(1+f^{-}(\rho)) \tag{12}$$

where $t_0 = 1/\mu_{max}$ is the average time to serve one job when the queue is empty and ρ is the normalized service rate or the utility $\rho = \lambda/\mu_{max}$.

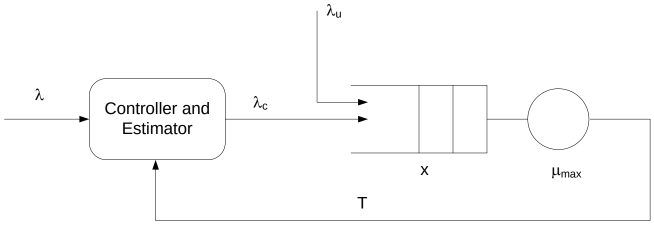

Figure 20. Schematic diagram of an abstraction of the MSS in Figure 1 with a controller and estimator.

The response time goes to infinity as λ approaches μ_{max} if the range of the function f is [0, 1]. The function f gives significant freedom in adjusting the behavior to real queue behavior.

The model (11), (12) has been used extensively to model queuing systems [33]. The simple M/M/1 queue can be represented by (12) with $f = x/(x + 1)$ [32].

Even if the model (11), (12) is simple it captures some important features of real queuing systems, for example the fact that response time increases with queue length. The model also captures the behavior that the rate of change of the response time increases with increasing arrival rate. The behavior of the system can be shaped by the function f.

In the project, we have investigated simulated models with servers and we have demonstrated that it is possible to find functions f which matches the steady state behavior of simulated systems. An illustration is given in Figure 21.

B. Estimation algorithm

There are significant variations in the arrival and response times due to their discrete nature. To monitor and control the system it is necessary to smooth these variations. For example, the average arrival rate of the controlled stream can be estimated the simple exponential smoother

$$\begin{aligned} \hat{t}_i^+ &= \hat{t}_i + k_3(h_a - \hat{t}_i) \\ \hat{\lambda}_c^+ &= 1/\hat{t}_i^+ \end{aligned} \qquad (13)$$

where t_i is the arrival time and h_a is the time since the last arrival update.

One advantage with the model (11), (12) is that it is possible to use Kalman filtering [31] to combine the model, which captures the gross behavior of the queuing system, with measured data.

If continuous data was available, an extended Kalman filter for the service time is given by

$$\begin{aligned} \frac{d\hat{x}}{dt} &= \lambda_c + \lambda_u - \mu_{\max} f(\hat{x}) + k_1(T - t_0(1+\hat{x})) \\ \frac{d\lambda_u}{dt} &= k_2(T - t_0(1+\hat{x})) \end{aligned} \qquad (14)$$

This filter will capture the behavior that response time increases with increasing queue length and arrival rate. The detailed behavior can be shaped by the function f.

It must be considered that the real measurements are events that represent arrival of a request or a completed response. To deal with this, we have developed an event-based Kalman filter.

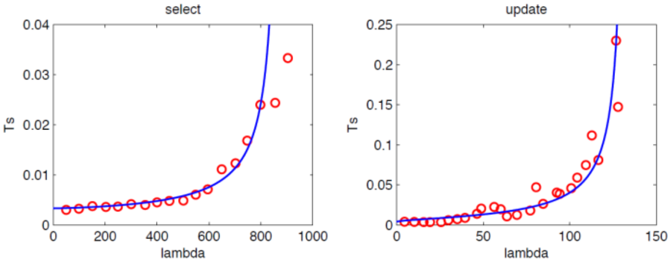

Figure 21. Service times for the operations SELECT (left) and UPDATE on an SQL server and predictions based on the model (12) with $f(x)=(1/(1+x))^n$, n = 1.5 and μ_{max} = 880 for SELECT and n = 0.15 and μ_{max} = 132 for UPDATE.

At arrivals, the queue length is updated according to the flow model:

$$\hat{x}^+ = \hat{x} + h_a(\hat{\lambda}_c + \hat{\lambda}_u - \mu_{\max} f(\hat{x})) \qquad (15)$$

This difference equation is simply a forward Euler approximation of (11). Equation (15) is simply a prediction of x based on the model (11). Information about x is obtained when a service is completed. The queue length and the unknown arrival rate are then updated as

$$\begin{aligned} \hat{x}^+ &= \hat{x} + h_d(\lambda_c + \lambda_u - \mu_{\max} f(\hat{x}) + k_1(T - \hat{T})) \\ \hat{\lambda}_u^+ &= \hat{\lambda}_u + h_d k_2(T - \hat{T}) \end{aligned} \qquad (16)$$

where h_d is the time since the last departure update. The arrival rate can be estimated because it results from the model (11) and (12) that the arrival rate is observable from a measurement of service time [31].

C. Experiment

The Kalman filter estimator was evaluated using a discrete-event simulation program written in Java, The program simulates a single server queue with exponentially distributed service times with mean μ_{max}=100 requests per second. The queue has two arrival processes, representing the measurable and unknown traffic. The Kalman filter has been evaluated for a number of scenarios validating its performance. However, in this paper we show the results of one specific scenario.

In this scenario, the unknown arrival process was a stationary Poisson process with mean 42.5 requests per second. The measurable arrival process was basically a Poisson process with changing average rate. The arrival rate, λ, was the sum of one constant part and one part represented by a sine function as given by

$$\lambda(t) = C + a \cdot \sin(kt) \qquad (18)$$

The parameters were chosen so that the system can handle the workload over long time but with periodic overloads, hence

$$\mu_{\max} - a < C < \mu_{\max} \qquad (19)$$

Therefore, the numerical values used in the simulations are C=42.5 and a = 20 requests per second.

The differential equations describing the behavior of the estimates between events were approximated using first order forward Euler discretization.

Figure 22 shows the response times and the arrival rate, both real values and estimates for a time period of 20 seconds during the simulation. The estimate error is shown in Figure 23. It can be seen how the Kalman filter manages to follow the real system during the quick rises in response time around time 424 and 427. Here the mean square error is $\sigma = 7.4 \cdot 10^{-4}$ for the period 415 < t < 420 and $\sigma = 1.1 \cdot 10^{-2}$ for the period 425 < t < 430. The mean square error for the entire experiment is $\sigma = 1.9 \cdot 10^{-2}$.

VII. Conclusions

Accurate control designs using control theory are essential for resource management in computer systems. In this paper we have presented work performed in collaboration with Ericsson AB, investigating how control theory can improve the performance of a commercial mobile service support system. Together with Ericsson AB, we have identified three major control challenges, and investigated solutions. The first challenge is to find accurate performance models for the system, with the objective to capture the system dynamics. The second challenge is to develop an admission control scheme that can handle unknown traffic and load surges. The final challenge is to develop estimation methods for accurate prediction of response times and arrival rates in systems with unknown traffic.

In this paper, the challenges have been treated rather independent of each other. However, the future goal is to be able to use all solutions together, in order to improve the system performance and speed up the development process. The performance model could be tuned using real data and then used for validating control designs, which is much easier than implementing the designs in testbeds or the real system. Also, in the future, the estimation algorithms should be incorporated in the control system, improving the control decisions.

Acknowledgment

The authors at Lund University are members of the Lund Center for Control of Complex Engineering Systems (LCCC). Maria Kihl and Anders Robertsson are members of the Excellence Center at Linköping-Lund in Information Technology (eLLIIT). The work is partly funded by the Swedish Research Council, grant VR 2010-5864.

References

[1] B. Brawn and F. Gustavson. “Program behavior in a paging environment”, AFIPS Fall Joint Computer Conference, pages 1019-1032, 1968.

[2] Crocus, “Systemes d'Exploitation des Ordinateurs”, Dunod, Paris, 1975.

[3] Y. Diao, C. Wu, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco, “Comparative studies of load balancing with control and optimization techniques," American Control Conference, 2005.

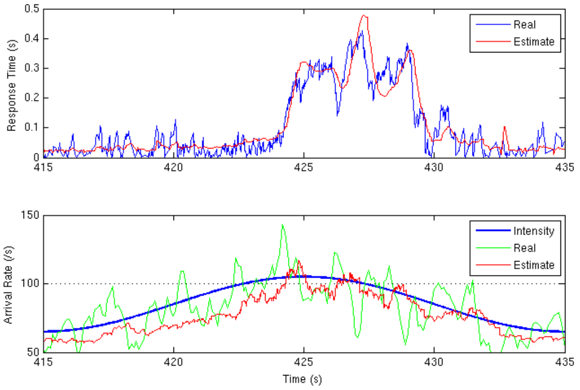

Figure 22. Kalman filter estimates of response times and estimation of arrival rate.

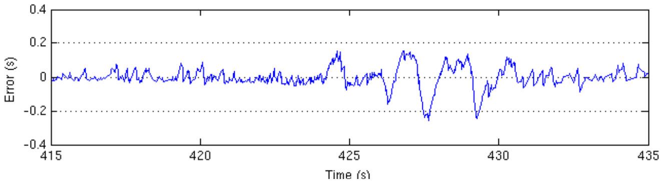

Figure 23. Proposed Kalman filter’s response time prediction error.

[4] Y. Fu, H. Wang, C. Lu, and R. Chandra, “Distributed utilization control for real-time clusters with load balancing," IEEE International Real–Time Systems Symposium, 2006.

[5] M. Kihl, A. Robertsson, M. Andersson, and B. Wittenmark, “Control theoretic analysis of admission control mechanisms for web server systems,” *The World Wide Web Journal*, Springer, vol. 11, no. 1, 2008.

[6] X. Chen, H. Chen, and P. Mohapatra, “Aces: an efficient admission control scheme for QoS-aware web servers," *Computer Communication*, vol. 26, no. 14, 2003.

[7] X. Liu, J. Heo, L. Sha, and X. Zhu, “Adaptive control of multi–tiered web applications using queuing predictor," 10th IEEE/IFIP Network Operation Mangement Symposium, 2006.

[8] T. Voigt and P. Gunningberg, “Adaptive resource based web server admission control”, 7th International Symposium on Computers and Communications, 2002.

[9] M. Kjaer, M. Kihl, and A. Robertsson, “Resource Allocation and Disturbance Rejection in Web Servers using SLAs and Virtualized Servers”, *IEEE Transaction on Network and Service Management*, Vol. 6, No. 4, 2009.

[10] W. Xu, X. Zhu, S. Singhal, and Z.Wang, “Predictive control for dynamic resource allocation in enterprise data centers," 10th IEEE/IFIP Network Operation Mangement Symposium, 2006.

[11] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal, “AutoParam: automated control of application-level performance in virtualized server environments", 2nd IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, 2007.

[12] R. Bianchini and R. Rajamony, “Power and energy management for server systems,” *IEEE Computer*, vol. 37, no. 11, 2004.

[13] H. Claussen, L.T.W Ho, and F. Pivit, “Leveraging advances in mobile broadband technology to improve environmental sustainability”, *Telecommunications Journal of Australia*, Vol. 59, No. 1, 2009.

[14] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, “Dynamic voltage scaling in multitier web servers with end-to-end delay control," *IEEE Transactions on Computers*, vol. 56, no. 4, 2007.

[15] E. Elnozahy, M. Kistler, and R. Rajamony, “Energy-efficient server clusters,”, *Lecture Notes in Computer Science 2325*. Springer-Verlag Berlin Heidelberg, 2003.

[16] M. Kihl, A. Robertsson, and B. Wittenmark, “Performance Modelling and Control of Server Systems using Non-linear Control Theory”, 18th International Teletraffic Congress, 2003.

[17] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, “Web server performance measurement and modeling techniques”, *Performance Evaluation*, Vol. 33, No. 1, 1998.

[18] D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Services*, Prentice Hall, 2002.

[19] R. D. van der Mei, R. Hariharan, and P. K. Reeser, “Web server performance modeling”, *Telecommunication Systems*, Vol. 16, No. 3, 2001.

[20] H. Perros, Y. Dallery, and G. Pujolle, “Analysis of a queueing network model with class dependent window flow control,” 11th Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE, pp. 968–977 vol.2, May 1992.

[21] A. Rak, and A. Sgueglia, "Instantaneous Load Dependent Servers (iLDS) Model for Web Services," International Conference on Complex, Intelligent and Software Intensive Systems, 2010.

[22] Curiel, M. and Puigjaner, R., “Using load dependent servers to reduce the complexity of large client-server simulation models”, *Performance Engineering, LNCS 2047*, Springer-Verlag Berlin Heidelberg, 2001.

[23] V. Mathur and V. Apte, “A computational complexity-aware model for performance analysis of software servers”, IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2004.

[24] K. K. Leung, Load-dependent service queues with application to congestion control in broadband networks, *Performance Evaluation*, Vol. 50, Issue 1, October 2002, pp 27-40.

[25] M. Kihl, P. Amani, A. Robertsson, G. Radu, M. Dellkrantz, and B. Aspernäs, “Performance modeling of databases in Telecommunication service management systems”, IARIA 7th International Conference on Digital Communications (ICDT), 2012.

[26] K J. Åström and B. Wittenmark, *Computer–Controlled Systems*. Upper Saddle River, NJ: Prentice Hall, 1997. Dover reprint 2011.

[27] J. Cao, M. Andersson, C. Nyberg and M. Kihl, “Web Server Performance Modeling using an M/G/1/K*PS Queue”, International Conference on Telecommunication, 2003.

[28] M. Kihl, G. Cedersjö, A. Robertsson, B. Aspernäs, “Performance measurements and modeling of database servers”, Sixth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, 2011.

[29] D.J. DeWitt. The Wisconsin benchmark: Past, present, and future, The Benchmark Handbook for Database and Transaction Processing Systems, 1, 1991.

[30] L. Kleinrock, *Queueing Systems, Volume I: Theory*, Wiley Interscience, New York, 1975.

[31] K. J. Åström and R. Murray. *Feedback Systems - An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[32] D. Tipper and M.K. Sundareshan, “Numerical methods for modeling computer networks under nonstationary conditions”, *IEEE Journal on Selected Areas in Communications*, 8(9), pp 1682-1695, Dec 1990.

[33] Carson E. Agnew, “Dynamic modeling and control of congestion-prone systems”, *Operations Research*, 24(3), pp. 400-419, 1976.