Real-Time Syst
DOI 10.1007/s11241-006-9008-8

Experimental evaluation of linear time-invariant models for feedback performance control in real-time systems

M. Amirijoo · J. Hansson · S. H. Son · S. Gunnarsson

Abstract In recent years a new class of soft real-time applications operating in unpredictable environments has emerged. Typical for these applications is that neither the resource requirements nor the arrival rates of service requests are known or available a priori. It has been shown that feedback control is very effective to support the specified performance of dynamic systems that are both resource insufficient and exhibit unpredictable workloads. To efficiently use feedback control scheduling it is necessary to have a model that adequately describes the behavior of the system. In this paper we experimentally evaluate the accuracy of four linear time-invariant models used in the design of feedback controllers. We introduce a model (DYN) that captures additional system dynamics, which a previously published model (STA) fails to include. The accuracy of the models are evaluated by validating the models with regard to measured data from the controlled system and through a set of experiments where we evaluate the performance of a set of feedback control schedulers tuned using these models. From our evaluations we conclude that second order models (e.g., DYN) are more accurate

M. Amirijoo (✉) · J. Hansson
Department of Computer and Information Science, Linköping University, Sweden
e-mail: meham@ida.liu.se

J. Hansson
e-mail: jorha@ida.liu.se

J. Hansson
Software Engineering Institute, Carnegie Mellon University, USA
e-mail: hansson@sei.cmu.edu

S. H. Son
Department of Computer Science, University of Virginia, Charlottesville, Virginia, USA
e-mail: son@cs.virginia.edu

S. Gunnarsson
Department of Electrical Engineering, Linköping University, Sweden
e-mail: svante@isy.liu.se

than first order models (e.g. STA). Further we show that controllers tuned using second order models perform better than controllers tuned using first order models.

Keywords Feedback control scheduling · Modeling · Model validation · System identification

1 Introduction

In recent years a new class of soft real-time systems has emerged, e.g., web applications, e-commerce, agile manufacturing, and data intensive application. These applications run on resource-constrained platforms and typically operate in open and unpredictable environments, in which arrival patterns and the resource requirements of tasks are in general unknown. Feedback control scheduling has been introduced as a promising foundation for performance control of complex real-time systems (Lu et al., 2002a,c; Parekh et al., 2002; Cervin et al., 2002; Li and Nahrstedt, 1998; Amirijoo et al., 2006). It has been shown that feedback control is highly effective to support the specified performance of dynamic systems that are both resource insufficient and that exhibit unpredictable workloads.

To efficiently use feedback control scheduling it is necessary to have a model that adequately describes the behavior of the controlled system (Franklin et al., 1998; Glad and Ljung, 2000). The performance of a controller is a function of the accuracy of the model used and the design methodology and, hence, we need to use models that accurately describe the dynamics of the controlled system. For example, consider a service provider streaming video to a set of clients. The service provider sets the reference quality of the streams to a certain level, and also gives requirements on worst-case quality and how fast the quality should converge towards the reference quality in the case of a transient overload or a disturbance in the system (e.g., a server goes down). Now, using more accurate models when synthesizing controllers, enables the service provider to provide streams with more reliable quality and faster performance adaptation, as compared to using less accurate models. Hence, the quality is regulated much more efficiently even in the case of transient system overloads.

The information available to the designer for the purpose of modeling is typically of two kinds. First, there is knowledge about the system being controlled based on equations or laws of science describing the dynamics of the system. Using this approach, a system designer describes the system directly with mathematical equations based on the knowledge of the system dynamics. However, in the case the mathematical function of a system is unknown, or too complicated to derive, the use of models tuned using system profiling and statistical methods have shown to provide good results (Ljung, 1999). In these circumstances the designer turns to data taken from experiments directly conducted to excite the controlled system and measure its response (Ljung, 1999). Statistical methods are then used to tune the parameters of the model.

The contributions of this paper include a comparative study of several models used in designing feedback controllers for managing the performance of a real-time

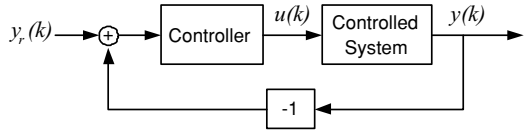


Fig. 1 Feedback loop structure

database (RTDB).[1] We introduce a model, called DYN, which generalizes a previously presented model (Lu et al., 2002a) by capturing additional system dynamics (hence the name DYN referring to dynamics). We compare the precision of DYN with the STA model, previously presented (Lu et al., 2002a), where some dynamic relations are approximated by static relations (hence the name STA referring to statics). In our evaluation we also include two models, auto regressive with extra signal (ARX) and output error (OE) (Ljung, 1999), tuned using experimental data and statistical methods. From our evaluations we have found that the dynamics of the controlled system under study can be adequately described using second order models as given by DYN, and that a first order model (e.g., STA) is not sufficient. It is found that DYN is significantly more accurate than STA and as accurate as the models tuned using experimental data and statistical methods. The tuning procedure of DYN is, however, simpler to use than the models tuned using experimental data and statistical methods as the tuning of these involve a considerable time-consuming and iterative search for a good model. Also, the tuning procedure of the statistical modeling approaches is complex and requires an in-depth understanding of the principles behind these techniques; a knowledge that is often lacking among practitioners in the area of real-time systems or computer science in general. Our findings show that by using more accurate models when tuning feedback controllers a significant improvement in performance adaptation and performance reliability is achieved. This results in faster QoS adaptation in the face of changes to required QoS or unexpected system behavior, e.g., failures or unknown system properties. Also, more accurate models result in increased QoS reliability as the actual QoS stays closer to the required QoS.

The remainder of this paper is organized as follows. The problem formulation is given in Section 2. The controlled system is defined and explained in Section 3. In Section 4 we present models describing the behavior of the controlled system, and in Section 5, we evaluate the quality of the presented models. In Section 6 we give an overview on related work, followed by Section 7, where conclusions and future work are discussed.

2 Problem formulation

We adopt the following notation of describing discrete variables in the time-domain. A sampled variable $a(k)$ refers to the value of the variable a at time kT, where T is the sampling period and k is the sampling instant. For the rest of this paper, we sometimes drop k where the notion of time is not of primary interest. A typical structure of a feedback control system is given in Fig. 1. The controller changes the behavior of the

[1] For an overview of RTDBs refer to, e.g., Ramamritham et al. (2004).

controlled system by adjusting the manipulated variable $u(k)$. Input to the controller is the difference between the reference $y_r(k)$, representing the desired behavior of the controlled system, and the actual system behavior given by the controlled variable $y(k)$. The control problem is how to compute the manipulated variable $u(k)$ such that the difference between the desired behavior and the actual behavior is minimized, i.e., we want to minimize for instance $(y_r(k) - y(k))^2$.

Given a performance specification and a linear time-invariant model of the controlled system, one can design a controller based on existing mathematical techniques, such as root locus, frequency response, and linear quadratic methods (Franklin et al., 1998; Glad and Ljung, 2000). Using these mathematical techniques is attractive as they enable us to derive analytic guarantees on the transient-state and steady-state behavior of the system. For some of the design techniques, such as root locus or state-space design, it is necessary to use linear time-invariant models that describe the relationship among the system variables in terms of mathematical expressions like differential (for continuous systems) or difference (for discrete systems) equations. Since computer systems are inherently discrete, below we only address models based on linear time-invariant difference equations.

In this work we are considering linear time-invariant models that describe the behavior of the class of real-time systems where the controlled variables, representing the performance of the system, are manipulated by varying the admitted workload. The load of admitted tasks may be varied by, e.g., changing the speed of the CPU, changing the quality level of tasks, or controlling admission. Given a model in terms of the difference equation, $a_1 y(k) + a_2 y(k-1) + \cdots + a_{m+1} y(k-m) = b_1 u(k) + b_2 u(k-1) + \cdots + b_{n+1} u(k-n)$, we say that the order of the model is the maximum of n and m. We aim at addressing the following problems in this paper. First we want to establish whether more accurate models have a significant impact on the performance of feedback control scheduling in real-time systems, with respect to minimizing $(y_r(k) - y(k))^2$. Second, we want to determine the difference in accuracy and the resulting control performance between first order and second order models.

Now, if second order models provide a significant increase in control performance, then it is preferred that a model based on the knowledge of the system is used rather than using statistical methods. The rationale of this approach is that it is easier to tune the model parameters as discussed in Section 1. Therefore the third goal of this work is to derive a simple second order model, which is based on the knowledge of the controlled system and that captures the dynamics of the system and is as general and simple as possible to facilitate applicability to other types of real-time systems. Hence, the model must be independent of particular actuation techniques, controlled systems, and measurement mechanisms and must be easily tuned by following a set of well-defined procedures.

3 The controlled system

As a case study we have chosen to apply STA and DYN to a RTDB system, since this type of system is by nature complex, including several types of transactions and shared resources resulting in transactions restarting and aborting. This presents a great challenge for a model to capture the behavior of a system. We start by defining the

controlled system and we refer to Table 5 for an overview of the variables used in this paper.

We consider a RTDB as the controlled system. The data and transaction models of the RTDB are presented below followed by an overview of the assumed feedback control scheduling architecture. Further, we identify a set of control-related variables, i.e., performance references, manipulated variables, and controlled variables.

3.1 Data and transaction model

We consider a real-time database, where there is one CPU as the main processing element. Transactions are classified either as update transactions or user transactions. Update transactions arrive periodically and may only write to data objects representing sensor values, while user transactions arrive aperiodically and may read or write to any data objects.

We apply imprecise computation (Liu et al., 1991) on user transactions, hence, we allow user transactions to deliver imprecise results in exchange for CPU resources. User transactions (T_i) are composed of one mandatory subtransaction m_i and $|O_i| \geq 1$ optional subtransactions $o_{i,j}$, where $o_{i,j}$ is the jth optional subtransaction of T_i. For the remainder of the paper, we let t_i denote a subtransaction of T_i. A mandatory subtransaction is completed when it completes in a traditional sense. The mandatory subtransaction gives an acceptable result and should be computed to completion before the transaction deadline. The optional subtransactions may be processed if there is enough time or resources available. While it is assumed that all subtransactions of a transaction T_i arrive at the same time, the first optional subtransaction $o_{i,1}$ becomes ready for execution when the mandatory subtransaction, m_i, is completed. In general, an optional subtransaction, $o_{i,j}$, becomes ready for execution when $o_{i,j-1}$ (where $2 \leq j \leq |O_i|$) completes. We set the deadline of every subtransaction t_i to the deadline of the transaction T_i. A subtransaction is terminated if it has completed or has missed its deadline. A transaction T_i is terminated when $o_{i,|O_i|}$ completes or one of its subtransactions misses its deadline. In the latter case, all remaining transactions of T_i are terminated as well. See Table 1 for a complete transaction model (see Table 5 for explanation of attributes). We use the following notation where attribute a_i refers to transaction T_i, and $a_i[t_i]$ is associated with subtransaction t_i of T_i. Upon arrival, a transaction presents the estimated average load $l_{E,i}$ and the relative deadline d_i to

Table 1 The assumed transaction model

Attribute	Update transactions	User transactions
$x_{E,i}$	–	$x_{E,i} = \sum_{\forall t_i} x_{E,i}[t_i]$
$x_{A,i}$	–	$x_{A,i} = \sum_{\forall t_i} x_{A,i}[t_i]$
e_i	NA	$e_i(\lvert COS_i \rvert) = (1 - \frac{\lvert COS_i \rvert}{\lvert O_i \rvert})^{n_i}$
p_i	–	NA
$i_{E,i}$	NA	$i_{E,i}[t_i] = i_{E,i}$
$i_{A,i}$	NA	$i_{A,i}[t_i] = i_{A,i}$
d_i	$d_i = p_i$	$d_i[t_i] = d_i = i_{A,i}$
$l_{E,i}$	$l_{E,i} = x_{E,i}/p_i$	$l_{E,i}[t_i] = x_{E,i}[t_i]/i_{E,i}$
		$l_{E,i} = x_{E,i}/i_{E,i}$
$l_{A,i}$	$l_{A,i} = x_{A,i}/p_i$	$l_{A,i}[t_i] = x_{A,i}[t_i]/i_{A,i}$
		$l_{A,i} = x_{A,i}/i_{A,i}$

the admission controller. The true load of the transaction is not known in advance due to variations in execution time, e.g., due to concurrency control causing restart of transactions.

For the remainder of this paper, we refer to transaction impreciseness as transaction error, where the transaction error of a user transaction T_i is denoted by e_i. A decrease of allocated resources for T_i results in an increase in e_i. Similarly, an increase of allocated resources for T_i results in a decrease in e_i. Transaction error is modeled as a function of completed optional subtransactions. For a transaction T_i, we use an error function (Chung and Liu, 1988) to approximate its corresponding transaction error given by, $e_i(|COS_i|) = (1 - \frac{|cos_i|}{|O_i|})^{n_i}$, where n_i is the order of the error function and $|COS_i|$ denotes the number of completed optional subtransactions of T_i. By choosing n_i we can model and support multiple classes of transactions showing different error characteristics. The choice of n_i depends on the application, e.g., it has been shown that anytime algorithms used in AI exhibit error characteristics where n_i is greater than one (Zilberstein and Russell, 1996).

3.2 Feedback control scheduling architecture

The general outline of the feedback control scheduling architecture is given in Fig. 2. Input to the controlled system, i.e., the RTDB is the set of arriving transactions and the change to the admitted estimated workload $\delta l_{ER}(k)$. Output from the real-time database (RTDB) is the set of terminated transactions and the controlled variable, namely the actual average transaction error,

$$e(k) = \frac{\sum_{T_i \in Terminated(k)} e_i}{|Terminated(k)|} \tag{1}$$

where $Terminated(k)$ is the set of admitted transactions terminated in the time interval $[(k-1)T, kT]$. Hence, rejected transactions do not contribute to $e(k)$. The goal of the Error Controller is to minimize the difference between the controlled variable and the reference, i.e., minimizing $(e_r(k) - e(k))^2$. This is done by changing the admitted workload through $\delta l_{ER}(k)$; increasing the admitted load results in an increase in $e(k)$ as each transaction receives less CPU resource, whereas decreasing the admitted load results in a decrease in $e(k)$.

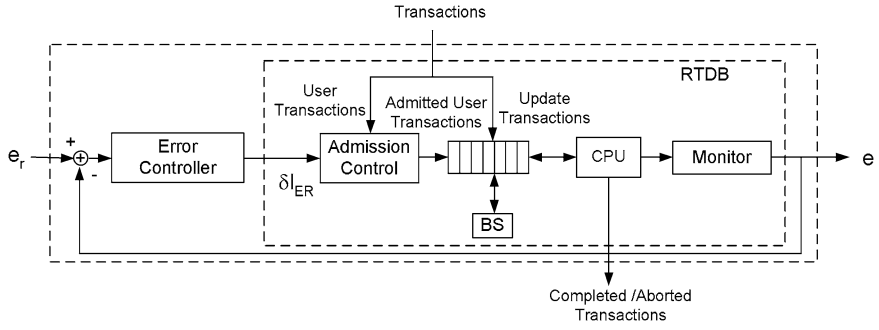


Fig. 2 Feedback control scheduling architecture

Update transactions and admitted user transactions are placed in the ready queue. The basic scheduler (BS) schedules the transactions in the ready queue. Update transactions have higher priorities than user transactions and, thus, the update transactions are scheduled before the user transactions. We consider two different scheduling algorithms as BSs: (i) Earliest Deadline First (EDF), where transactions are processed in the order determined by increasing absolute deadlines, and (ii) Highest Error First (HEF) (Amirijoo et al., 2003), where transactions are processed in the order determined by decreasing transaction error, which is dynamic and changes during the execution of the transactions. More specifically, the priority of a transaction decreases as its transaction error decreases. Thus, the priority of the transaction is dynamic and may change throughout the execution of the transactions. We include HEF in our comparison for the following reasons. First, in our earlier work we have shown that HEF has significantly better performance than EDF when minimizing the deviation in transaction error of terminated transactions (Amirijoo et al., 2003). Second, the characteristics of HEF are distinct to EDF, e.g., HEF is driven by transaction error which influences the way it inserts tasks and manages the ready queue. These characteristics, together with the characteristics of EDF, represent a bigger challenge for the model to capture the dynamics of the real-time system.

At each sampling instant k, the average of the transaction error of terminated transactions $e(k)$ is formed by the monitor. The performance error is computed by taking the difference between the performance reference $e_r(k)$ and $e(k)$. Based on the performance error the controller computes a change $\delta l_{ER}(k)$ to the estimated requested workload of user transactions $l_{ER}(k)$, such that $e(k)$ equals $e_r(k)$. We refer to $\delta l_{ER}(k)$ as the manipulated variable. Based on $\delta l_{ER}(k)$, the admission controller enforces the workload adjustment of user transactions. We consider the following admission model. Upon arrival to the system, an instance of a transaction is inserted in an arrival queue, which is sorted according to the arrival time. Transaction instances are removed from the front of the arrival queue and admitted if and only if the sum of the transaction instance workload and the workload of admitted tasks is less than the estimated requested workload of admitted tasks. Here, an increase in $\delta l_{ER}(k)$ results in more admitted user transactions and, hence, less CPU resources per user transaction, resulting in an increase in $e(k)$.

Note that although the length of the queue increases as the workload applied to RTDB increases, the queue length will not grow indefinitely. An increase in the queue length results in an increase in the response time of the transactions, causing the number of transactions missing their deadlines to increase as well. Stale transactions are aborted and removed from the queue and, therefore, an increase in the queue length results in more transactions to be aborted and removed from the queue. In the worst case, when all admitted transactions miss their deadlines, the rate of aborted and removed transactions is equal to the admission rate. As such, the length of the queue is bounded.

4 Modeling the controlled system

As described in Section 2, the thrust of this paper is on the experimental findings, namely we want to establish the differences in model accuracy and control performance

when using first order models and second order models. However, to fully understand the effects of the models on control performance, a complete description of STA and DYN with their structure and model parameters is essential. It is necessary to understand the discussions in Section 5, where the tuning procedure and validation of the models are given. In this section, a description of these two models are given.

Let $A(z)$ denote the Z-transform (Oppenheim and Willsky, 1996) of the time-domain variable $a(k)$. The goal of the modeling is to derive a transfer function (Franklin et al., 1998) describing the relation between the manipulated variable $\Delta L_{ER}(z)$, and the controlled variables $E(z)$ and $U(z)$. This is done by dividing the controlled system into subsystems and deriving the relationships describing the dynamics between the variables in the subsystems. Finally, the equations and relationships are put together forming the final transfer function.

The parameters of the models depend on a variety of factors, e.g., basic scheduler, temporal characteristics of the transaction set (e.g., relative deadlines), and admission controller. There are two approaches for deriving the parameters of the models. In the first approach the properties of the controlled system, e.g., transaction set and admission controller, are used to compute the model parameters. Here the relationships between the properties of the controlled system, e.g., a particular basic scheduler, and the model parameter must be derived. However, this approach yields on an overly complicated computing machinery, which cannot be practically used and adopted for different applications. We have therefore pursued a mixed approach of modeling from first principles and identification, where the model structure is derived using the knowledge of the system and the parameters of the model are tuned using experiments. This approach, which also was pursued by Lu et al. (2002a), greatly simplifies the modeling and tuning procedure. Below the models STA and DYN are presented along with the description of procedures for how to tune their parameters.

4.1 Model 1: STA

A block diagram showing the structure of STA (Lu et al., 2002a) is given in Fig. 3. Starting from the control input, the estimated requested workload of admitted user transactions $l_{ER}(k+1)$ in the next sampling period is changed through the manipulated variable $\delta l_{ER}(k)$, given by

$$l_{ER}(k+1) = l_{ER}(k) + \delta l_{ER}(k). \tag{2}$$

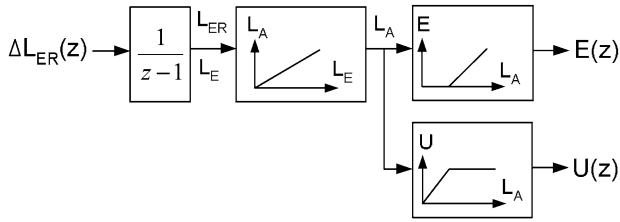


Fig. 3 Structure of STA

Hence, l_{ER} is the integration of the control input δl_{ER}. Now, the estimated admitted workload of user transactions $l_E(k)$ may differ from $l_{ER}(k)$, since the maximum workload that can be made available in the system may not be sufficient to satisfy $l_{ER}(k)$, or the admitted workload is decreased due to deadline misses and, consequently, early termination of transactions. In this model, however, we approximate the estimated load of admitted transactions by $l_{ER}(k)$, i.e., $l_E(k) = l_{ER}(k)$. In Section 4.2 we return to the discussion regarding the difference between $l_E(k)$ and $l_{ER}(k)$.

The actual workload of admitted user transactions, denoted $l_A(k)$, may differ from $l_E(k)$ due to incomplete knowledge about the controlled system, e.g. unknown execution times of the transactions and data conflicts. Therefore we get

$$g_A = \frac{l_A(k)}{l_E(k)}, \tag{3}$$

where the workload ratio g_A represents the workload variation in terms of actual total requested workload. For example, g_A equals two means that the actual workload is twice the estimated workload. It is obvious that g_A cannot be computed or derived from system profiling. However, g_A is included in the model for stability analysis of the closed-loop system.

We say that an output signal $y(k)$ is saturated when, in the steady-state, $y(k)$ remains unchanged even though the input signal $u(k)$ is altered. The relationship between the actual workload $l_A(k)$ and the utilization $u(k)$ is non-linear due to saturation. When $l_A(k)$ is less or equal to 100%, i.e. the CPU is underutilized, then $u(k)$ is not saturated and equals $l_A(k)$. However, when $u(k)$ is saturated, i.e., $l_A(k)$ is greater than 100%, then $u(k)$ remains at 100%, despite changes to $l_A(k)$, i.e.,

$$u(k) = \begin{cases} l_A(k), & l_A(k) \leq 100\% \\ 100\%, & l_A(k) > 100\%. \end{cases} \tag{4}$$

Continuing with $e(k)$, the relationship between the actual workload $l_A(k)$ and $e(k)$ is non-linear due to saturation. Let $l_{Th}(k)$ be the workload threshold of user transactions in the kth period for which admitted user transactions are precisely schedulable, i.e., transaction errors are zero. $e(k)$ is saturated when $l_A(k) < l_{Th}(k)$, and remains zero despite changes to $\delta l_{ER}(k)$, i.e.,

$$e(k) = 0, \quad l_A(k) < l_{Th}(k).$$

When not saturated, $e(k)$ increases non-linearly with $l_A(k)$. Since feedback control relies on linear systems, we linearize the relationship between $l_A(k)$ and $e(k)$ by deriving the derivative between $l_A(k)$ and $e(k)$ at the vicinity of the performance reference e_r, i.e.,

$$g_E = \frac{de(k)}{dl_A(k)}, \quad e(k) = e_r. \tag{5}$$

From (2) we see that the subsystem describing $l_{ER}(k)$ in terms of $\delta l_{ER}(k)$ is dynamic, as $l_{ER}(k)$ depends on $l_{ER}(k-1)$ and $\delta l_{ER}(k-1)$ and, hence, there is a

non-instantaneous relation between these variables. However, the subsystems from $l_{ER}(k)$ to $u(k)$ and $e(k)$ are static, as $e(k)$ and $u(k)$ depend on $l_{ER}(k)$ and, hence, there are direct and instantaneous relations between these variables. Hence, we assume that changes to $l_{ER}(k)$ result in immediate changes to $u(k)$ and $e(k)$. For this reason, we call this model STA, referring to the static relations between $l_{ER}(k)$, $u(k)$, and $e(k)$. Now, from (2)–(5), we obtain the following.

Under the condition $l_A(k) \leq 100\%$, there exists a transfer function,

$$G_{STAU}(z) = \frac{g_A}{z-1} \tag{6}$$

from the control input $\delta l_{ER}(k)$ to $u(k)$, where $l_{ER}(k)$ and $u(k)$ are statically related.

Under the condition $l_{Th} \leq l_A(k)$, there exists a transfer function,

$$G_{STAE}(z) = \frac{g_A g_E}{z-1} \tag{7}$$

from the control input $\delta l_{ER}(k)$ to $e(k)$, where $l_{ER}(k)$ and $e(k)$ are statically related.

Note that G_{STAU} and G_{STAE} must be linearized at different workloads if and only if their saturation zones are overlapping, which occurs if $l_{Th} > 100\%$. Contrary, if $l_{Th} \leq 100\%$, then for loads between l_{Th} and 100% both controlled variables $u(k)$ and $e(k)$ are unsaturated. This means that G_{STAU} and G_{STAE} may be linearized at the same workload if $l_{Th} \leq 100\%$ (as is shown in Fig. 6).

4.2 Model 2: DYN

We extend the model STA to include additional system dynamics, as shown in Fig. 4. We start by describing the requested load factor, followed by sources of dynamics and we finally present the model that captures the dynamics.

4.2.1 The requested load factor

Now, starting from the model input $\delta l_{ER}(k)$, we compute $l_{ER}(k)$ according to (2). Given a certain $l_{ER}(k)$, the estimated workload of admitted user transactions,

$$l_E(k) = \begin{cases} g_L l_{ER}(k), & l_{ER}(k) \leq \bar{l}_E(k) \\ g_L \bar{l}_E(k), & l_{ER}(k) > \bar{l}_E(k) \end{cases} \tag{8}$$

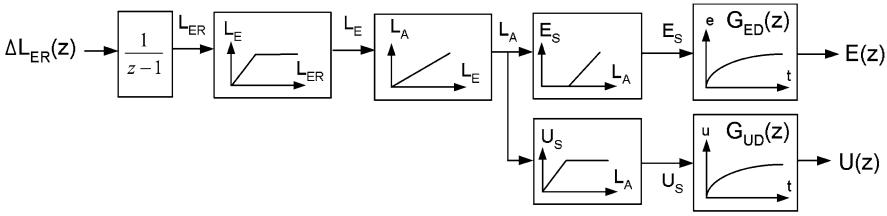


Fig. 4 Structure of DYN

is the product of the requested load factor g_L and the minimum of the estimated requested load and the maximum estimated load $\bar{l}_E(k)$ that can be made available in the system. The upper limit of the admitted load, given by $\bar{l}_E(k)$, makes the relationship between $l_{ER}(k)$ and $l_E(k)$ non-linear due to saturation. $l_E(k)$ becomes saturated when $l_{ER} > \bar{l}_E(k)$. When not saturated, $l_E(k)$ increases linearly with $l_{ER}(k)$. For example, we know that for real-time systems operating in highly unpredictable environments the arrival pattern of the tasks is hard or even impossible to predict. If admission control is used to enforce load adjustments, then $\bar{l}_E(k)$ is determined by the estimated external available workload, i.e., workload submitted to the system. Given a certain $l_{ER}(k)$, the estimated admitted load $l_E(k)$ may not equal $l_{ER}(k)$ since the available external workload may not be sufficient to fill up the system to the level given by $l_{ER}(k)$. To capture the difference between $l_E(k)$ and $l_{ER}(k)$, we derive the ratio between $l_{ER}(k)$ and $l_E(k)$ at the vicinity of $l_E(k)$ corresponding to e_r, and we obtain the requested load factor g_L.

4.2.2 Sources of dynamics in real-time systems

Now, under the condition $l_{ER}(k) < \bar{l}_E(k)$, it can be observed that given a certain increase in the estimated requested workload $l_{ER}(k)$, it may take some time before the estimated load $l_E(k)$ reaches $g_L l_{ER}(k)$. The time it takes for $l_E(k)$ to reach $g_L l_{ER}(k)$ depends on how the estimated admitted load is regulated. For example, when admission control is used to enforce load adjustments, the amount of workload submitted to the system $\bar{l}_E(k)$ determines how fast $l_E(k)$ converges toward $g_L l_{ER}(k)$. The greater $\bar{l}_E(k)$ is, the faster we can fill up the workload, reaching $g_L l_{ER}(k)$ earlier. Similarly, a decrease in $l_{ER}(k)$ does not result in an immediate decrease in $l_E(k)$, since currently running transactions have to terminate.

Furthermore, we argue that the relation between $l_A(k)$ and $e(k)$ is non-static. Given a certain $l_A(k)$, the time it takes for $e(k)$ to reach $g_E l_A(k)$, see (5), depends on the actual basic scheduler used. For example, under EDF scheduling (Liu and Layland, 1973), newly admitted transactions are more likely to be placed further down in the ready queue, since they are less likely to have earlier deadlines than transactions admitted earlier. This means that an actual change to $e(k)$ is not noticed until the newly admitted transactions are executing, which may take a while until the older ones have terminated.

4.2.3 Modeling the dynamics

Before describing DYN, we recall from Section 2 that the second order model that we construct has to be simple and general enough to be applicable to a wide spectrum of applications. The simplicity of the model facilitates easy and practical tuning of the model parameters. The dynamic relationship between l_{ER} and e is a product of the load adjustment mechanism and the basic scheduler. To form a model that is general enough, we introduce a simple difference equation that relates l_{ER} and e such that it takes some time for e to reach its steady-state once l_{ER} has changed. We show in Section 5.4.1 that the difference equation shows similar response time behavior as the true controlled variable. Under the assumption $l_{ER}(k) \leq \bar{l}_E(k)$, let

$$e_S(k) = g_L g_A g_E l_{ER}(k)$$

denote the final value of e in the steady-state given a certain l_{ER}, i.e., e_S is the value e converges to. The speed by which a controlled variable reaches its final values is determined by the time constant of the system and it depends on the load adjustment mechanism and the basic scheduler used. Let T_E denote the time constant of the subsystem with the input $e_S(k)$ and the output $e(k)$. The difference equation,

$$e(k+1) = \frac{T(e_S(k+1) - e(k))}{T_E} + e(k) \tag{9}$$

relates the input $e_S(k)$ and the output $e(k)$ such that it takes a number of samples for e to reach e_S. Initially, when the difference between $e_S(k)$ and $e(k)$ is large, $e(k)$ converges rapidly towards $e_S(k)$. However, the convergence speed decreases as the difference between $e_S(k)$ and $e(k)$ decreases. The convergence speed is determined by T_E, i.e., an increase in T_E results in a slower convergence. The Z-transform of (9) gives the transfer function,

$$G_{ED}(z) = \frac{E(z)}{E_S(z)} = \frac{TT_E^{-1}z}{z - 1 + TT_E^{-1}} \tag{10}$$

that models the dynamic relationship between the input l_{ER} and the output e, i.e., it captures the effects of the workload adjustment mechanism and the basic scheduler. Let,

$$\delta(k) = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}$$

denote the unit impulse function and define the unit step function as,

$$u(k) = \begin{cases} 1, & k \geq 0 \\ 0, & k < 0 \end{cases}$$

A unit step function applied on (10) gives the time domain solution,

$$e(k) = \left[\frac{T}{T_E}\delta(k) + \left(1 - \left(1 - \frac{T}{T_E}\right)^k\right)u(k-1)\right].$$

Here we can see that the greater T_E is, the more time it takes for e to reach its final value, i.e., the slower e reacts to changes in l_{ER} as e_S and l_{ER} are statically related.

Let us now apply a unit step on l_{ER}, i.e. $l_{ER}(k) = u(k)$, and examine the step response of the controlled system from l_{ER} to e. Let T_{EM} be the point in time at which $e(k)$ equals $(1 - e^{-1})e_S$, i.e., approximately 63% of the final value of $e(k)$. At time T_{EM} we have that

$$\begin{aligned} e\left(\frac{T_{EM}}{T}\right) &= (1 - e^{-1})e_S \\ &= e_S\left[\frac{T}{T_E}\delta\left(\frac{T_{EM}}{T}\right) + \left(1 - \left(1 - \frac{T}{T_E}\right)^{\frac{T_{EM}}{T}}\right)u\left(\frac{T_{EM}}{T} - 1\right)\right]. \end{aligned} \tag{11}$$

When designing controllers, the sampling period is set to be less than the time constant of the system, i.e., the control loop operates at a faster rate than the controlled system

(Franklin et al., 1998). This corresponds to $T < T_{EM}$, which is equivalent to $\frac{T_{EM}}{T} - 1 > 0$. Hence the impulse function in (11) is zero and the step function is one. This gives the simplified equation,

$$(1 - e^{-1})e_S = e_S \left[1 - \left(1 - \frac{T}{T_E} \right)^{\frac{T_{EM}}{T}} \right]$$

and solving for T_E gives

$$T_E = \frac{T}{1 - e^{-\frac{T}{T+T_{EM}}}}. \tag{12}$$

Hence, we compute T_E according to (12) where, given a step on l_{ER}, T_{EM} is the point in time at which $e(k)$ equals approximately 63% of the final value of $e(k)$.

Earlier we concluded that we have a dynamic relation between l_{ER} and l_A. According to (4), u is equal to l_A when u is not saturated and, hence we have a dynamic relation between l_{ER} and u. We show this in Section 5.4.1 where we measure the response of l_A given changes to l_{ER}. We use a transfer function $G_{UD}(z)$, similar to $G_{ED}(z)$, to represent the dynamics of u. Here, by choosing the time constant T_U of $G_{UD}(z)$ we can capture the time domain relationship between l_{ER} and u. We now give the following results, based on (8)–(10).

Under the conditions $l_{ER}(k) \leq \bar{l}_E(k)$ and $l_A(k) \leq 100\%$, there exists a transfer function,

$$G_{DYNU}(z) = \frac{g_L g_A T T_U^{-1} z}{(z-1)\left(z - 1 + T T_U^{-1}\right)} \tag{13}$$

from the control input $\delta l_{ER}(k)$ to $u(k)$, where $l_{ER}(k)$ and $u(k)$ are dynamically related.

Under the conditions $l_{ER}(k) \leq \bar{l}_E(k)$ and $l_{Th} < l_A(k)$, there exists a transfer function,

$$G_{DYNE}(z) = \frac{g_L g_A g_E T T_E^{-1} z}{(z-1)\left(z - 1 + T T_E^{-1}\right)} \tag{14}$$

from the control input $\delta l_{ER}(k)$ to $e(k)$, where $l_{ER}(k)$ and $e(k)$ are dynamically related.

We have extended the model STA, resulting in the new model DYN, which captures additional dynamics of the controlled system by giving more accurate time domain relations between the control input $\delta l_{ER}(k)$ and the outputs $u(k)$ and $e(k)$.

5 Model validation and control performance

The main objective of the model evaluation is to compare the accuracy of the models in terms of the degree of dynamics that they capture and to determine the performance of controllers tuned using the models. Also we want to establish whether a significant improvement in performance control is achieved when using second order models instead of first order models. In particular, we want to determine the accuracy by which DYN describes the dynamics of the average transaction error $e(k)$ with regard to the

control input $\delta l_{ER}(k)$. Furthermore, we want to establish whether a controller tuned using DYN provides better performance reliability and performance adaptation than a controller tuned using STA. We use an RTDB simulator to evaluate the performance of the controllers. The simulator can be configured to simulate different workload characteristics, thus, one can capture the behavior of a wide range of applications.

5.1 Simulator setup

The simulated workload consists of update and user transactions, which access data and perform virtual arithmetic/logical operations on the data. The workload of update transactions is approximately 50%, whereas the applied load of user transactions on the database, i.e., $\bar{l}_E(k)$, is set to 100%. We assume that there are no execution time estimation errors. As the basic scheduler we use EDF. The workload model of the update and user transactions are described as follows.

Data and update transactions. The DB holds 1000 temporal data objects where each data object is updated by a stream ($stream_i$, $1 \leq i \leq 1000$). Updates arrive periodically with the periods uniformly distributed in the range (100 ms, 50 s), i.e., U : (100 ms, 50 s), and with the estimated execution time $x_{e,i}$ given by U : (1 ms, 8 ms). Upon a periodic generation of an update, $stream_i$ gives the update an actual execution time given by the normal distribution $N : (x_{e,i}, \sqrt{x_{e,i}})$.

User transactions. Each $source_i$ generates a transaction T_i, consisting of one mandatory subtransaction, m_i, and $|O_i|$ ($1 \leq |O_i| \leq 10$) optional subtransaction(s), $o_{i,j}$ ($1 \leq j \leq |O_i|$). $|O_i|$ is uniformly distributed between 1 and 10. The estimated execution time of a subtransaction t_i, denoted $x_{e,i}[t_i]$, is given by U : (5 ms, 15 ms). Upon generation of a transaction, $source_i$ associates an actual execution time to each subtransaction t_i, which is given by $N : (x_{e,i}[t_i], \sqrt{x_{e,i}[t_i]})$. The deadline is set to $at_i + x_{e,i} \times slackfactor$, where at_i denotes the arrival time of T_i. The slack factor is uniformly distributed according to U : (20, 40). The inter-arrival time is exponentially distributed with the mean inter-arrival time set to $x_{e,i} \times slackfactor$.

5.2 QoS specification

We use the following time domain performance metrics to specify controller performance: (i) reference e_r, giving the desired level of the controlled variable $e(k)$, (ii) overshoot, denoted M_p, is the worst-case system performance in the transient system state (see Fig. 5) and it is given in percentage, and (iii) settling time, denoted T_s, is the time for the transient overshoot to decay and reach the steady state performance and, hence, it is a measure of system adaptability. The following QoS specification is assumed: $e_r = 20\%$, $T_s \leq 40$ s, and $M_p \leq 30\%$. This gives that $e \leq e_r \times (M_p + 100) = 26\%$. Further we require that the steady-state error E_{ss}, namely the difference between the reference and the controlled variable to be zero as k goes to infinity, i.e.,

$$E_{ss} = e_r - \lim_{k \to \infty} e(k) = 0.$$

The latter implies that the controlled variable must converge toward the reference.

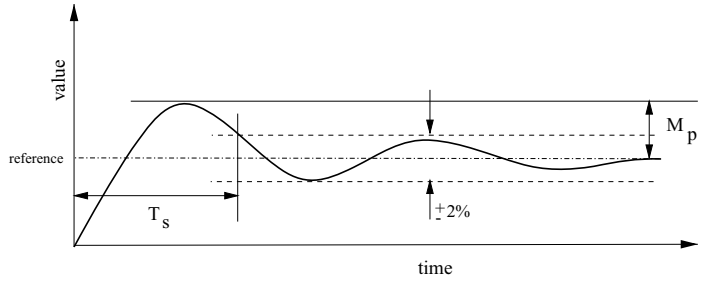


Fig. 5 Definition of settling time (T_s) and overshoot (M_p)

5.3 Sampling period selection

We have chosen the sampling period according to the following. As a rule of thumb the sampling period should be chosen to be about ten times the bandwidth of the controlled system (Ljung and Glad, 1994). This corresponds to a sampling period, where it takes about 5 to 8 samples for the controlled variable to increase to steady state given a step on the reference (Ljung and Glad, 1994). In other words the sampling period must be chosen such that 5 to 8 samples fit on the flank of the step response of the controlled variable. As we will see in Section 5.4.1, this corresponds to a sampling period of one second, i.e. $T = 1$s. Further, we have employed the method presented by Amirijoo et al. (2005), where the sampling period is chosen such that a given QoS specification is satisfied with respect to the settling time and the overshoot. The chosen sampling period effects the settling time depending on the model and the controller. For example the settling time increases as the sampling period increases when using STA and a P controller. Selecting a sampling period of one second (i.e., $T = 1$ s) satisfies the given QoS requirement (see Section 5.2) for all controllers as is shown in Section 5.6.3.

For comprehensiveness, we have also evaluated the performance of the controllers using sampling periods 0.5 s and 2.0 s. Due to space limitations we present the model tuning and validation (Sections 5.4 and 5.5) assuming a sampling period of 1.0 s. We revisit other sampling periods in Section 5.6.4, where the performance of the controllers are evaluated.

5.4 Model tuning

5.4.1 STA and DYN

The models STA and DYN are tuned by profiling the simulated real-time database, where l_{ER} is varied from 0 to 120% with 10% increase. We have monitored l_E and e, and the result is shown in Fig. 6. Assuming that execution time estimates are accurate, i.e., $g_A = 1$, we obtain $l_A = l_E$.

STA. At $e(k)$ equal to $e_r = 20\%$, the derivative g_E is estimated to be 0.84 by using (5) and Fig. 6. Hence, the tuned model is $G_{STAE} = \frac{0.84}{z-1}$.

DYN. As for STA, we derive g_E to be 0.84 at $e(k)$ equal to $e_r = 20\%$. From Fig. 6, we see that e equal to 20%, corresponds to $l_{ER} \approx 68\%$ and $l_E \approx 58\%$ and by applying

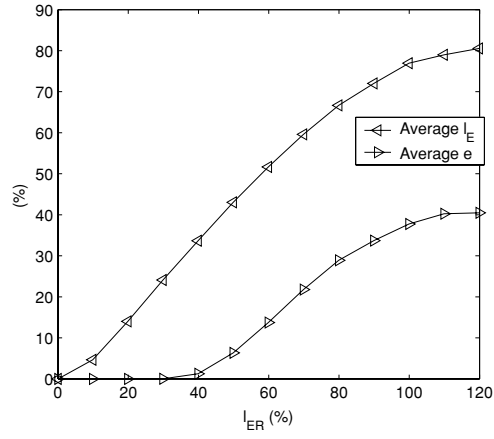
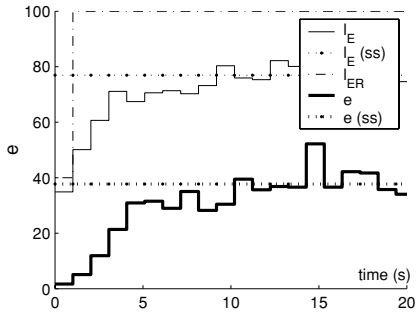
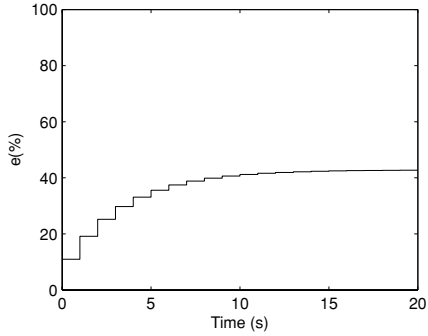


Fig. 6 Relation between l_{ER}, l_E, and e



a. Measured response time.



b. Simulated response time (DYN).

Fig. 7 Response time analysis

(8), we estimate g_L to be 0.85. From Fig. 6, we note that e starts increasing at $l_{ER} \approx 40\%$, which corresponds to $l_E \approx 35\%$ and, hence, the workload threshold $l_{Th} \approx 35\%$. Since for DYN we assume that $l_A > l_{Th}$, we apply a step from 40% to 100% on l_{ER} and we measure the time it takes for e to reach 63% of its final value, as illustrated in Fig. 7(a). The steady-state value of a variable is denoted with ss. Since the response time is discrete we interpolate between the samples to obtain a continuous response time. Thereby, we measure T_{EM} to be 2.38 s and, hence, $T_E \approx 3.91$ s according to (12). Putting everything together, the tuned model is $G_{DYNE} = \frac{0.1829z}{(z-1)(z-0.7439)}$. The simulated step response of the transfer function from l_{ER} to e is shown in Fig. 7(b). Note, in Fig. 7(a), the step is applied at time 1 s, where in Fig. 7(b) the step is applied at time 0 s.

5.4.2 ARX and OE

Regarding the statistical models ARX and OE (Ljung, 1999), we have profiled the system given by the input $l_{ER}(k)$ and the output $e(k)$, since the subsystem from $\delta l_{ER}(k)$ to $l_{ER}(k)$, i.e., the integration given by (2), is known. When the ARX and OE models

are tuned, we add $\frac{1}{z-1}$ to the derived models to form the complete model from δl_{ER} to e.

The fist step in building a model based on experimental data is to choose a suitable input signal. The input signal should excite the system such that the dynamics of the system is captured. It is therefore required that the input signal contains many frequencies. A good choice is a binary signal that shifts randomly between two levels, since such signal contains all frequencies. The two levels are chosen so that they correspond to maximum allowed variation, in order to capture as much of the system behavior as possible. The probability of the input signal to shift between levels is set such that it contains pulses that are constant over such long period that the output response more or less settles, but also some short ones that excite interesting fast modes in the system. Further, the input and measured output signals are preprocessed by removing the mean value of each signal before applying system identification algorithms.

According to the discussion above, we apply a binary signal that shifts randomly between two levels around the operation point e_r. Due to non-linearity of real-time systems we let $l_{ER}(k)$ to vary within an area around the operation point where the system is approximately linear, see Fig. 6. The operation point is $e_r = 20\%$ and the system is approximately linear for e between 1.4% and 39.5%, which corresponds to l_{ER} between 40% and 100%. Hence, we let l_{ER} vary between 40% and 100% and we set the probability of the input signal to shift between the levels to 0.5. We measure $e(k)$ during 600 s and divide the collected data into two parts, one used for estimating model parameters and one used for validation. This technique is referred to as cross validation, i.e., we validate the models based on new data sequence that are different from the ones used in model estimation. For the sake of identification, we remove 70% from l_{ER} and 20% (e_r) from e. The models are then tuned considering fit and residual analysis, i.e., we modified the parameters n_a, n_b, and n_f (Ljung, 1999) such that a great fit, as well as a low correlation between input and output residuals are obtained. Now that we have tuned models from l_{ER} to e, we add $\frac{1}{z-1}$ to the tuned models to form the complete model from δl_{ER} to e.

ARX. For the ARX model structure, a transfer function from δl_{ER} to e is tuned to $G_{ARXE} = \frac{0.1619z}{(z-1)(z-0.7638)}$.

OE. For the OE model structure, a transfer function from δl_{ER} to e is tuned to $G_{OEE} = \frac{0.2139z}{(z-1)(z-0.7268)}$.

5.5 Model validation

Once a model has been designed and its parameters computed, the crucial question is whether it is good enough for the intended purpose. Testing if a given model is good enough is known as model validation. This is carried out by checking whether the model agrees sufficiently well with observed data from the modeled system. Since the subsystem from $\delta l_{ER}(k)$ to $l_{ER}(k)$ is known, we validate the models considering the subsystem from $l_{ER}(k)$ to $e(k)$.

Fit and residual analysis are two well-known useful techniques for model validation, and below we give a brief overview of them. Let $\hat{e}(k)$ be the output predicted by the model. The percentage of the output variation explained by the model is given by the

metric fit (Ljung, 1999), and it is defined as

$$R^2 = 100 \times \left(1 - \frac{\frac{1}{N}\sum_{k=1}^{N}(e(k) - \hat{e}(k))^2}{\frac{1}{N}\sum_{k=1}^{N} e^2(k)}\right) \qquad (\%).$$

Hence, we normalize the squared sum of the prediction errors $\epsilon_p(k) = e(k) - \hat{e}(k)$, giving us a measure of the ability of the model to reproduce output data in terms of prediction. An increase in R^2 implies a decrease in prediction errors and, hence, the model produces outputs closer to measured data.

By validating models with residual analysis (Ljung, 1999), i.e., forming the correlation function describing the correlation between the input $l_{ER}(k)$ and model output residuals, i.e. $\epsilon_p(k) = e(k) - \hat{e}(k)$, we are able to measure how much of the dynamics of the controlled system is captured by the model. Ideally, the residuals should be independent on the input, i.e., the correlation between residuals and input should be zero. If this is not the case, then there are more system dynamics to describe than the model has picked up. More specifically, we form the correlation function,

$$\hat{R}_{\epsilon_p l_{ER}}(\tau) = \frac{1}{N}\sum_{k=1}^{N} \epsilon_p(k+\tau) l_{ER}(k)$$

where N is the number of samples, and test whether $\hat{R}_{\epsilon_p l_{ER}}(\tau)$ is close to zero, i.e., we check if ϵ_p and l_{ER} are uncorrelated. Usually, $\hat{R}_{\epsilon_p l_{ER}}$ is graphically displayed along with 99% confidence intervals. The rule is that if the correlation function goes significantly outside these confidence intervals for any τ, then $\epsilon_p(k+\tau)$ and $l_{ER}(k)$ probably are dependent for this value of τ and, hence, we do not accept the corresponding model as a good description of the system.

Now, let us validate the models with respect to fit and residual analysis. Given a system G from $\delta l_{ER}(k)$ to $e(k)$, we denote its subsystem from $l_{ER}(k)$ to $e(k)$ with G', e.g, the subsystem of G_{STAE} is denoted with $G'_{STAE} = g_A g_E$. We validate the models considering a binary signal $l_{ER}(k)$ as input, ranging from 40 to 100%. The result is presented in Fig. 8 (for clarity of presentation we consider a subset of the data). As we can see G'_{STAE} provides poor prediction of the output, whereas G'_{DYNE} produces a prediction much closer to the measured output. Analyzing input-output relations, the fit for each model is computed and gives the following: G'_{STAE}: −104.11%; G'_{DYNE}: 21.41%; G'_{ARXE}: 20.28%; G'_{OEE}: 21.60%. We observe that G'_{STAE}, compared to the

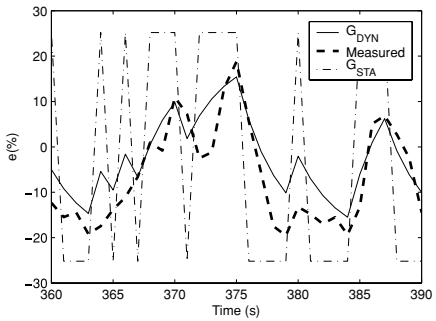
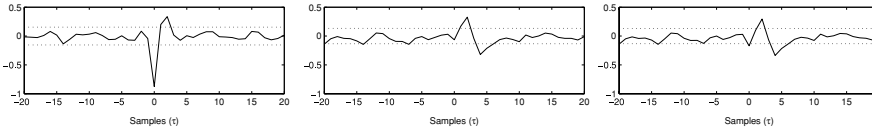


Fig. 8 The measured output and the output predicted by G'_{STA} and G'_{DYN}



a. G'_{STAE} b. G'_{DYNE} c. G'_{OEE}

Fig. 9 Residual analysis of G'_{STAE}, G'_{DYNE}, and G'_{OEE}

other models, describes the controlled system unsatisfactorily poorly. G'_{DYNE}, G'_{ARXE}, G'_{OEE} on the other hand give a good description of the system dynamics.

Figure 9 shows the correlation function $\hat{R}_{\epsilon_p l_{ER}}(\tau)$ for the models along with the 99% confidence intervals. The correlation function of G'_{STAE} goes significantly outside its confidence intervals. On the other hand, the correlation functions of G'_{ARXE}, G'_{OEE}, and G'_{DYNE} do not go significantly outside their confidence intervals. Here, we conclude that G'_{DYNE}, G'_{ARXE}, and G'_{OEE} describe to a great extent the dynamics of the system.

From the model validation procedure we conclude that G'_{DYNE}, G'_{ARXE}, and G'_{OEE} produce satisfactory predictions of the controlled signal, and they capture the dynamics of the controlled system to a greater extent compared to G'_{STAE}.

5.6 Control performance

We now evaluate the performance of a set of controllers tuned using the models, G_{STAE}, G_{DYNE}, G_{ARXE}, and G_{OEE}. We start by defining the performance metrics, continuing with the tuning procedure of the controllers, and finally we present the results obtained from the experiments.

5.6.1 Performance metrics

We distinguish the performance of controllers by how well they force the output $e(k)$ to follow or track a reference $e_r(k)$, despite presence of disturbances in the controlled system. It is therefore interesting to measure the difference between $e_r(k)$ and $e(k)$ over a period of time. For this reason, in our simulations we evaluate the controllers with respect to,

$$J_a = \frac{1}{N}\sum_{k=1}^{N}|e_r(k) - e(k)|, \quad J_s = \frac{1}{N}\sum_{k=1}^{N}(e_r(k) - e(k))^2 \tag{15}$$

where N is the number of samples taken. By J_s and J_a we can establish how well the controllers are able to keep $e(k)$ near $e_r(k)$, and also how well they react to changes in $e_r(k)$. The lower J_s and J_a are, the better a controller is able to keep $e(k)$ near $e_r(k)$, and also the faster $e(k)$ converges towards $e_r(k)$.

5.6.2 System stability

Stability refers to whether the input to the closed loop system (see Fig. 1), $G_c(z) = \frac{C(z)G(z)}{1+C(z)G(z)}$, with the input y_r and the output y, where $C(z)$ is the controller and $G(z)$ is the

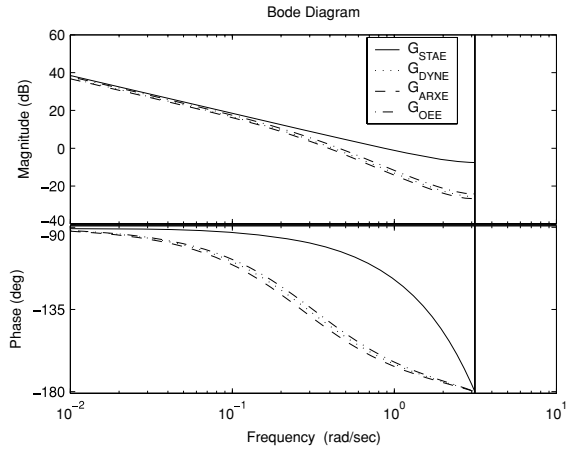


Fig. 10 Bode diagram for G_{STAE}, G_{DYNE}, G_{ARXE}, and G_{OEE}

controlled system, causes a bounded output. In this work, we consider stability in terms of the Bounded-Input-Bounded-Output (BIBO) relation (Franklin et al., 1998).[2] This means that if the input is bounded, then the output is bounded as well. By plotting the frequency response of $G(z)$, usually depicted as a Bode diagram (Franklin et al., 1998), the stability of the closed-loop system can be determined. A quantity that measures the stability margins of the system is gain margin (GM), which represents the condition under which the system is stable, and it is obtained by studying the magnitude and the phase plots of the Bode diagram of $G(z)$. GM gives the factor by which the magnitude of the controlled system $G(z)$ is less than one (i.e. zero dB) when the phase is -180°. Assuming that a P controller is used, where $u(k) = K_p(y_r(k) - y(k))$, then GM gives the upper bound of K_p for which the system is stable. Hence, setting K_p to a greater value than GM results in an unstable system. Figure 10 shows the Bode diagram of G_{STAE}, G_{DYNE}, G_{ARXE}, and G_{OEE}, where the upper plot shows the magnitude of the transfer functions, and the lower plot shows the phase of the transfer functions.

We observe that when the phase for G_{STAE} is -180°, the magnitude is -7.54 dB (corresponding to 0.42) and, hence, GM is $\frac{1}{0.42} = 2.38$. GM for the other models are computed similarly and are approximately 19.07 (G_{DYNE}), 21.79 (G_{ARXE}), and 16.15 (G_{DYNE}). We use GM of each model for testing the stability of the closed loop system when tuning the controllers.

5.6.3 Controller tuning

Having a QoS specification and a tuned model of the system, the next step is to tune the controllers based on the closed loop of the system. We use a proportional controller (P controller), where $\delta l_{ER}(k)$ is computed as $\delta l_{ER}(k) = K_p(e_r(k) - e(k))$, where K_p is the proportional gain. Depending on the dynamics of the controlled system, a P controller may produce non-zero steady-state error (Franklin et al., 1998), i.e., the difference between the controlled variable and its reference may be non-zero at steady-state. The steady-state error can be eliminated by adding a term proportional to the integral of the

[2] When the word stable is used without further qualification in this text, BIBO stability is considered.

Table 2 K_p and time domain performance

Model used	K_p	M_p	T_s	GM of the model used
G_{STAE}	≈1.1905	0	1 s	≈2.38
G_{DYNE}	≈0.3495	≈14%	≈21 s	≈19.07
G_{ARXE}	≈0.3558	≈14%	≈20 s	≈21.79
G_{OEE}	≈0.3491	≈14%	≈20 s	≈16.15

error, forming a proportional and integral controller (PI controller). However, we note that a P controller is sufficient since the controlled system has an integral part $\frac{1}{z-1}$ and, thus, an integral part for the controller is not needed to remove the steady-state error.

Therefore we use P controllers for all models and we tune K_p of the P controllers using root locus method in Matlab (1996), such that the QoS specification given in Section 5.2 is satisfied, i.e., the settling time is less than 40 s and at the same time the overshoot is less than 30%. The result is presented in Table 2. In Section 5.6.2 we derived the gain margins of each model that gives an upper bound of K_p that guarantees stability. The computed K_p of all controllers satisfy the gain margins and, hence, all closed-loop systems are stable. As shown in Table 2, the overshoot and the settling time for the controller tuned using G_{STAE} is zero and one, respectively. This is not surprising as given a model that statically relates l_{ER} and e, we can directly compute δl_{ER} and, hence, reach e_r within a sampling period without any overshoot. For simplicity we refer to the controller tuned with G_{STAE} as the STA controller (DYN, ARX, and OE controllers are defined similarly).

5.6.4 Evaluation of controller performance

In our experiments, one simulation run lasts for 10 minutes of simulated time. For all the performance data, we have taken the average of 10 simulation runs and derived 95% confidence intervals.

Experiment 1. To see how well the controllers react to changes to e_r, we set e_r to 20% at time 0, to 30% at time 200, and to 20% at time 400. The performance of the controllers is given in Table 3. As a consequence of the model, the control signal of the STA controller varies between −18.9 to 23.8%, whereas the control signal of the DYN controller varies between −4.95 to 7%. In other words, the STA controller controls the RTDB more aggressively, resulting in greater deviations between $e(k)$ and $e_r(k)$. Since the STA controller computes $\delta l_{ER}(k)$ according to a static relation between $l_{ER}(k)$ and $e(k)$, the controller does not consider the dynamics of the controlled system. Hence, it does not consider that given a certain $\delta l_{ER}(k)$, it may take a few samples

Table 3 J_s and J_a with 95% confidence intervals. The reference is varied

Model	J_s	J_a
G_{STAE}	284.19 ± 24.65	14.11 ± 0.65
G_{DYNE}	211.67 ± 14.51	12.02 ± 0.45
G_{ARXE}	209.98 ± 15.65	11.97 ± 0.48
G_{OEE}	212.50 ± 13.76	12.06 ± 0.41
$K_p = 0.20$	202.50 ± 12.97	11.78 ± 0.41

until the corresponding $e(k)$ is reached. Therefore, the STA controller persists with changing the load until the desired $e_r(k)$ is reached, at which point $e(k)$ overshoots due to aggressive control. This is handled more efficiently with the DYN controller, as it is tuned according to a dynamic model and, hence, the controller is more gentle when controlling the system. The performance of controllers tuned with G_{DYNE}, G_{ARXE}, and G_{OEE} does not differ considerably. However they all manage to provide a significantly lower J_s and J_a as compared to the controller tuned with STA. This means that the accuracy of the model used when designing a controller indeed affects the performance of the controller. Using DYN, which is more accurate than STA, results in a faster QoS adaptation in the face of changes to required QoS or unexpected system behavior, as compared to using STA. Further, a more accurate model results in an increased QoS reliability as the actual QoS stays closer to the required QoS.

Experiment 2. We have performed an additional experiment where we vary K_p from 0 to 2 by 0.1 step. The idea is to determine which K_p gives the best controller performance in terms of J_s and J_a. From our experiments we have observed that $K_p = 0.20$ gives the best performance with $J_s = 202.50 \pm 12.97\%$ and $J_a = 11.78 \pm 0.41\%$. Considering Table 3, we see that the performance of the controller with $K_p = 0.20$ does not differ considerably from the controllers tuned using G_{DYNE}, G_{ARXE}, and G_{OEE}. Although, the value of 0.20 was not estimated from the controller tuning, we see this as a substantial improvement considering the value of K_p suggested when using G_{STAE}.

Experiment 3. We notice from the measured J_s and J_a that $e(k)$ oscillates heavily around the reference. We know that there are various disturbances present in the controlled system. Random variations in arrival patterns of user transactions and restart or abort of transactions due to concurrency control lead to variations in user transaction load. Also, the update transactions have higher priority than user transactions and a change to the update transaction load also affects the user transactions. We pose the question whether the oscillations in $e(k)$ are caused by the controllers or the disturbances in the system. To characterize and measure the oscillations caused by the disturbances in the system, we run the system with open loop control, where no feedback information is used to control $e(k)$. When using feedback control, we can to some extent afford inaccuracies in the model of the controlled system, since the feedback information compensates for model inaccuracies. However, using open loop requires exact knowledge of the controlled system and for this reason we have profiled the system by varying the estimated admitted load l_{ER} to find the load that generates an e close to e_r. Note, the derived l_{ER} only holds if e_r and the workload characteristics, e.g. arrival patterns or execution time estimation error, do not vary. This assumption does not hold for most open real-time systems as the workload characteristics or desired QoS may vary, hence, the reason why feedback information is justified. For this particular experiment we keep e_r and the workload characteristics constant. We measure J_s and J_a as deviations around e_r, and the result is given in Table 4 along with the performance of the controllers where $e_r(k) = 20\%$. As we observe, the STA controller performs worse than open loop control. On the other hand, J_s and J_a of the DYN, ARX, and OE controllers are less to that of open loop control and, hence, we conclude that the oscillations are not induced by the controllers. The oscillations are rather caused by various disturbances present in the controlled system.

Table 4 J_s and J_a with 95% confidence intervals. The reference is constant

Model	J_s	J_a
G_{STAE}	266.40 ± 23.34	13.78 ± 0.69
G_{DYNE}	192.80 ± 13.79	11.53 ± 0.46
G_{ARXE}	192.30 ± 13.37	11.51 ± 0.45
G_{OEE}	193.06 ± 13.62	11.54 ± 0.45
Non-controlled system	211.92 ± 37.62	12.09 ± 1.01

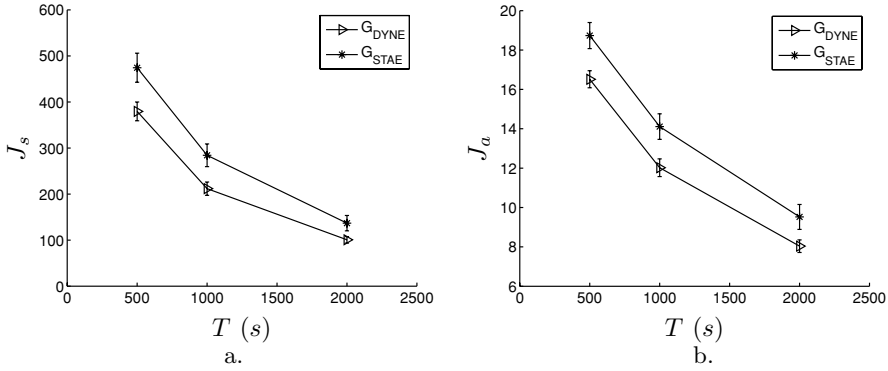


Fig. 11 J_s and J_a as a function of the sampling period T. The reference is varied

Experiment 4. In this experiment we study how J_s and J_a are affected by the sampling period T. We have for this reason designed controllers based on the sampling periods 0.5 s, 1.0 s, and 2.0 s. Note that $G_{STAE}(z)$, see (7), does not depend on T, hence, we have the same model when T varies. This results in that we get the same proportional gain K_P (controller parameter) for all T, i.e., $K_P = 1.1905$ for $T = 0.5$ s, ..., 2.0 s. On the other hand, the model $G_{DYNE}(z)$, see (14), is a function of T. This results in different proportional gains and we get that $K_P = 0.2962$ for $T = 0.5$ s, $K_P = 0.3495$ for $T = 1.0$ s, and $K_P = 0.4032$ for $T = 2.0$ s. Figures 11(a) and (b) show J_s and J_a as a function of T. As we can see J_s and J_a decrease as T increases, suggesting that the difference in the controlled variables and their references decreases as the sampling period increases. This is due to the decrease of the measurement disturbance variance as the sampling period increases (Amirijoo et al., 2005). For all cases the DYN controller provides better performance than the STA controller.

From the experiments outlined above, we conclude that using the first order model STA, which is an overly simplified model, yields in an even worse performance than open loop control. However adopting the simple and yet expressive second order model DYN, which captures additional system dynamics, results in a substantial performance enhancement compared to STA and open loop control, even in the presence of intense disturbances.

5.7 Summary of results and discussions

In our model evaluation we have considered methods used in modeling, such as fit and residual analysis, which are important tools for gaining confidence in a derived

model. We have shown that the first order model studied in this paper, i.e., STA, is less accurate than the presented second order models, i.e., DYN, ARX, and OE. Hence, the studies suggest that first order models are not adequate for describing the dynamics of the controlled system, and that second order models are more suitable for the purpose of control design. The results from the fit and residual analysis show that DYN is as accurate as the statistical models ARX and OE. As a final validation criterion we have evaluated the performance of controllers tuned based on the presented models. Here, we conclude that the controller tuned with DYN outperforms the controller tuned with STA with respect to performance adaptation, i.e., the controller tuned with DYN produces an $e(k)$ significantly closer to the reference than the controller tuned using STA. Moreover, the DYN controller performs as good as the ARX and OE controllers.

Although, there is a great body of knowledge within modeling of dynamic systems, there are really no well-established procedures for how to tune models using statistical methods as ARX and OE. Often, the model designer has to iteratively search for the best model, which may be quite time-consuming. The result of our work can be directly used to aid the model designer in finding a good model. By comparing the presented models using model validation and experiments, we have found that the dynamics of real-time systems can be adequately described using second order models as given by DYN. This result can be directly used to choose an appropriate ARX structure and hence, the model designer does not need to search for the best structure. The structure of the transfer function G'_{DYNE} corresponds to the structure of the ARX model with $n_a = 1$ and $n_b = 1$ (Ljung, 1999).

In this paper we applied DYN on a real-time database system. Our model can, however, easily be applied to other systems where for example the estimated requested load l_{ER} is the input and other performance metrics such as deadline miss ratio is the output of the controlled system. In the case when l_{ER} acts as input, we simply remove the integration given by $\frac{1}{z-1}$ from G_{DYNU} and G_{DYNE}. If deadline miss ratio is used to measure the performance of the system, then we replace $e(k)$ with deadline miss ratio, since $e(k)$ and deadline miss ratio have similar characteristics, as discussed in Section 4.2.

In this work we have tuned and linearized G_{STAE} and G_{DYNE} based on a QoS specification and given external load and execution time estimation errors. In reality, however, real-time systems are non-linear and external load may vary at run-time and/or execution time estimations may change. In control theory, one addresses this problem by (i) considering the average mode of operation, i.e., we tune models based on average external load or average execution time estimation error and rely on the inherent robustness of feedback control, or (ii) during run-time switch between linear controllers, which are tuned off-line, depending on some operating conditions, e.g. gain scheduling (Åström and Wittenmark, 1995; Franklin et al., 1998). This way we can utilize the design methods for linear systems, still conforming to non-linearities in the system. Other approaches include adaptive control (Åström and Wittenmark, 1995) where the behavior of the controlled system is monitored at run-time and controllers adapted accordingly. This way the controllers can continuously adapt to their environment and, hence, less emphasis is put on off-line modeling. However, there are several shortcomings with adaptive control applied on computing systems. First, adaptive control still cannot replace good models obtained off-line that are needed to choose performance specifications, structure of the controller, and design methods (Åström and Wittenmark, 1995). Second, since in adaptive control the

behavior of the controllers are constantly changing, issues related to safety become important, especially in safety-critical applications. Third, the memory storage and computing power that are available have a significant influence on the type of adaptive controller used. For example, in embedded systems with limited computing power, the use of adaptive control may not be feasible. Finally, another key difficulty is accurate system identification under closed-loop control. Remember that when tuning models based on profiling and statistical methods, we have to make sure to excite the system such that both the slow and the fast modes of the system are captured. This may not be possible for a system at run-time as the performance of the system may be critical.

6 Related work

Lu et al. (2002a) introduced a feedback control scheduling framework, including two models of real-time systems describing deadline miss ratio and utilization in terms of changes to estimated utilization. Their model, however, statically relates estimated utilization, deadline miss ratio, and utilization. In this paper, we have extended their model to capture the dynamic relation between these variables. Lu et al. (2001a) used system identification to model relative delays in web servers. They found that relative delays exhibit second order dynamics and that a first order model was not sufficient. In this paper we have argued that first order models are not accurate enough and we have shown that a second order model is sufficiently accurate for feedback control purposes. In contrast to Lu et al. (2001b) we use fit and residual analysis to determine the accuracy of the models. Parekh et al. (2002) use feedback control scheduling to control the length of a queue of remote procedure calls (RPCs) arriving at a server. They have used system identification for tuning their models. They show that they can achieve 97.6% fit on measured data, however, they do not validate their models using residual analysis, which is an important technique for evaluating model quality (Ljung, 1999). Li and Nahrstedt (1998) proposed a task model for QoS adaptations. In their model, there are dependencies among the tasks, expressing consumer-producer properties; tasks are characterized by input quality and output quality, and a model expressing the output quality of a target task is derived. The goal is to design controllers that force the target task to maintain the same output quality at a desired QoS reference. Their model does not extend to the case when several target tasks are available. In our model, we do not yet model precedence constrains among transactions, however, we consider the individual quality of a set of transactions. Lu et al. (2002c) introduced an architecture for differentiated caching services. The desired relation between the hit ratios of different content classes is enforced using per-class feedback control loops. The authors derive a model describing relative hit ratio in terms of adjusted cache space. Abdelzaher (2000) and Lu et al. (2002b) report some results on adaptive control applied on software systems. They show that using their approach the controllers are able to adapt to changes to the controlled system. The authors identify several open issues to be answered, such as how to choose model structures on-line, and also how to excite the system such that as much of the system dynamics can be captured. The authors suggest an automated identification of the model structure using a rule-based approach that uses a branch-and-bound search to arrive at the optimal model structure.

Table 5 Table of variables

Attribute	Description
d_i	relative deadline of T_i
δl_{ER}	change to the estimated requested workload of admitted user transactions
e_i	transaction error of T_i
e	average transaction error
e_S	steady-state value of e
g_A	workload ratio
$G_{DYNU}(z)$	model DYN relating $\delta l_{ER}(k)$ and $u(k)$
$G_{DYNE}(z)$	model DYN relating $\delta l_{ER}(k)$ and $e(k)$
g_E	relationship between l_A and e
$G_{ED}(z)$	subsystem with the input $e_S(k)$ and the output $e(k)$
g_L	requested load factor
$G_{STAE}(z)$	model STA relating $\delta l_{ER}(k)$ and $e(k)$
$G_{STAU}(z)$	model STA relating $\delta l_{ER}(k)$ and $u(k)$
$i_{A,i}$	average inter-arrival time of T_i
$i_{E,i}$	estimated inter-arrival time of T_i
J_a	average performance error
J_s	average squared performance error
l_A	actual workload of admitted user transactions
$l_{A,i}$	average load of T_i
l_E	estimated workload of admitted user transactions
$\bar{l}_E$	maximum estimated load that can be made available
$l_{E,i}$	estimated load of T_i
l_{ER}	estimated requested workload of admitted user transactions
l_{Th}	precisely schedulable workload threshold of user transactions
M_p	maximum overshoot in percentage
p_i	period of T_i
T_E	time constant of $G_{ED}(z)$
T_{EM}	the time it takes for $e(k)$ to reach approximately 63% of the final value
T_s	settling time
T	sampling period
u	utilization
$x_{A,i}$	average execution time of T_i
$x_{E,i}$	estimated (average) execution time of T_i

Several papers describe the application of imprecise computation on databases. Davidson and Watters (1988) proposed a method for generating monotonically improving answers in RTDBs and distributed RTDBs. A query processor, APPROXIMATE (Vrbsky and Liu, 1993), produces approximate answer if there is not enough time available. The accuracy of the improved answer increases monotonically as the computation time increases. The relational database system, called CASE-DB, can produce approximate answers to queries within certain deadlines (Ozsoyoglu et al., 1995). Neither of the approaches above have discussed performance and QoS management using feedback control. In previous work, we presented a set of algorithms for managing QoS based on feedback control scheduling and imprecise computation (Amirijoo et al., 2003, 2006), where QoS was defined in terms of transaction and data preciseness. We tuned the controllers using the model STA proposed by Lu et al. (2002a).

7 Conclusions

To efficiently use feedback control scheduling it is necessary to have a model that adequately describes the behavior of the system. In this paper we have experimentally evaluated the accuracy of a previously published model STA (Lu et al., 2002a), a model DYN presented in this paper, and two models tuned using statistical methods. This comparison is performed by validating the models in terms of fit and residual analysis. Using the results from model validation we have demonstrated that the second order model DYN is significantly more accurate than the first order model STA when describing, in terms of difference equations, the dynamics of a real-time system. From our studies we show that DYN is as accurate as the second order models ARX and OE tuned with experimental data and statistical methods. Further, we have carried out a set of experiments where we evaluate the performance of a set of controllers tuned using the presented models. We have shown that the controller tuned using the second order models, e.g., DYN, performs significantly better than the controller tuned using the first order model STA. The performance of the controller tuned using DYN is as good as the controllers tuned using the models derived from statistical methods. The model tuning procedure of DYN is, however, simpler to use than the tuning procedures used in statistical methods as the latter require time-consuming and iterative search for the best model.

For our future work, we will apply our proposed model to applications providing real-time MPEG services, and also extend the model to support service differentiation.

Acknowledgments This work was funded, in part by CUGS (the National Graduate School in Computer Science, Sweden), CENIIT (Center for Industrial Information Technology) under contract 01.07, NSF grants CCR-0098269 and IIS-0208758, and ISIS (Information Systems for Industrial Control and Supervision).

References

Abdelzaher TF (2000) An automated profiling subsystem for QoS-aware services. In: Proceedings of the real-time technology and applications symposium (RTAS)

Amirijoo M, Hansson J, Gunnarsson S, Son SH (2005) Enhancing feedback control scheduling performance by on-line quantification and suppression of measurement disturbance. In: Proceedings of the IEEE real-time and embedded technology and applications symposium (RTAS)

Amirijoo M, Hansson J, Son SH (2003) Error-driven QoS management in imprecise real-time databases. In: Proceedings of the IEEE Euromicro conference on real-time systems (ECRTS)

Amirijoo M, Hansson J, Son SH (2006) Specification and management of QoS in real-time databases supporting imprecise computations. IEEE Trans Comput 55(3):304–319

Boaghe OM, Billings SA, Li LM, Fleming PJ, Liu J (2002) Time and frequency domain identification and analysis of a gas turbine engine. Contr Eng Pract 10(12):1347–1356

Buttazzo GC (1997) Hard real-time computing systems. Kluwer

Cervin A, Eker J, Bernhardsson B, Årzén K. (2002) Feedback-feedforward scheduling of control tasks. Real-time Syst 23(1/2). Special Issue on Control-Theoretical Approaches to Real-Time Computing

Chung J, Liu JWS (1988) Algorithms for scheduling periodic jobs to minimize average error. In: Proceedings of the real-time systems symposium (RTSS)

Davidson S, Watters A (1988) Partial computation in real-time database systems. In: Proceedings of the workshop on real-time software and operating systems

Fang B, Kelkar AG, Joshi SM, Pota HR (2004) Modelling, system identification, and control of acoustic-structure dynamics in 3-D enclosures. Control Eng Pract 12(8):989–1004

Franklin GF, Powell JD, Workman M (1998) Digital control of dynamic systems, 3rd edn. Addison-Wesley

Glad T, Ljung L (2000) Control theory—Multivariable and nonlinear methods. Taylor and Francis

Li B, Nahrstedt K (1998) A control theoretical model for quality of service adaptations. In: Proceedings of the international workshop on quality of service
Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 20(1):46–61
Liu JWS, Lin K, Shin W, Yu AC-S (1991) Algorithms for scheduling imprecise computations. IEEE Comput 24(5).
Ljung L (1999) System Identification: Theory for the user, 2nd edn. Prentice-Hall
Ljung L, Glad T (1994) Modeling of dynamic systems. Prentice-Hall
Lu C, Abdelzaher TF, Stankovic JA, Son SH (2001a) A feedback control approach for guaranteeing relative delays in web servers. In: Proceedings of the IEEE real time technology and applications symposium (RTAS)
Lu C, Abdelzaher TF, Stankovic JA, Son SH (2001b) A feedback control architecture and design methodology for service delay guarantees in web servers. Technical Report CS-2001-06, Computer Science Department at University of Virginia
Lu C, Stankovic JA, Tao G, Son SH (2002a) Feedback control real-time scheduling: Framework, modeling and algorithms. Real-time Syst 23(1/2)
Lu Y, Abdelzaher TF, Lu C, Tao G (2002b) An adaptive control framework for QoS guarantees and its application to differentiated caching services. In: Proceedings of the international workshop on quality of service (IWQoS)
Lu Y, Saxena A, Abdelzaher TF (2001c) Differentiated caching services; a control-theoretical approach. In: Proceedings of the international conference on distributed computing systems (ICDCS).
Matlab (1996) Control systems toolbox user's guide. The Mathworks Inc.
Mourue G, Dochain D, Wertz V, Descamps P (2004) Identification and control of an industrial polymerisation reactor. Control Eng Pract 12(7):909–915
Oppenheim AV, Willsky AS (1996) Signals and Systems, 2nd edn. Prentice-Hall
Ozsoyoglu G, Guruswamy S, Du K, Hou W-C (1995) Time-constrained query processing in CASE-DB. IEEE Trans Knowledge Data Eng 7(6)
Parekh S, Gandhi N, Hellerstein J, Tilbury D, Jayram T, Bigus J (2002) Using control theory to achieve service level objectives in performance managment. Real-time Syst 23(1/2)
Ramamritham K, Son SH, DiPippo LC (2004) Real-time databases and data services. Real-Time Syst 28(2–3):179–215
Åström KJ, Wittenmark B (1995) Adaptive control. 2nd (edn), Addison-Wesley
Stankovic JA, Spuri M, Ramamritham K, Buttazzo GC (1998) Deadline scheduling for real-time systems. Kluwer
Suthasun T, Mareels I, Al-Mamun A (2004) System identification and controller design for dual actuated hard disk drive. Contr Eng Pract 12(6):665–676
Vrbsky SV, Liu JWS (1993) APPROXIMATE—a query processor that produces monotonically improving approximate answers. IEEE Trans Knowledge Data Eng 5(6):1056–1068
Zilberstein S, Russell SJ (1996) Optimal composition of real-time systems. Artif Intell 82(1–2):181–213

Mehdi Amirijoo is a Ph.D. student at the Department of Computer and Information Science in Linköping University, Sweden. He received his M.Sc. degree in computer science and engineering from Linköping University in 2002. His interests include real-time data services, scheduling, QoS management, automatic control, imprecise computation techniques, and sensor networks. He received the 2003 best M.Sc. thesis award issued by the Swedish National Real-Time Association (SNART).

Jörgen Hansson received the B.Sc. and M.Sc. degrees from the University of Skövde, Sweden, in 1992 and 1993, respectively. He received the Ph.D. degree in 1999 from Linköping University, Sweden, with which he is also affiliated as an associate professor. He is a senior member of the technical staff at the Software Engineering Institute at Carnegie Mellon University. His current research interests include real-time systems and real-time database systems and he has written 40 papers and edited two books in these areas. His research has focused on techniques and algorithms for ensuring robustness and timeliness in real-time applications that are prone to transient overloads, mechanisms and architectures for handling increasing amounts of data in real-time systems, and algorithms to ensure data quality in real-time systems. His current research interests include resource management, techniques and methodologies for data repositories functioning in realtime and embedded computing systems, adaptive overload management, and component-based software architectures for embedded and real-time systems. He is a member of the IEEE.

Sang Hyuk Son is a Professor at the Department of Computer Science of University of Virginia. He received the B.Sc. degree in electronics engineering from Seoul National University, M.Sc. degree from KAIST, and the Ph.D. in computer science from University of Maryland, College Park in 1986. He has been a Visiting Professor at KAIST, City University of Hong Kong, Ecole Centrale de Lille in France, Linköping University, and University of Skövde in Sweden. His current research interests include real-time computing, data services, QoS management, wireless sensor networks, and information security. He has served as an Associate Editor of IEEE Transactions on Parallel and Distributed Systems, and is currently serving as an Associate Editor for Real-Time Systems Journal and Journal of Business Performance Management. He has been on the executive board of the IEEE TC on Real-Time Systems, and served as the Program Chair or General Chair of several real-time and database conferences, including IEEE Real-Time Systems Symposium and IEEE Conference on Electronic Commerce. He received the Outstanding Contribution Award at the IEEE Conference on Embedded and Real-Time Computing Systems and Applications in 2004.

Svante Gunnarsson was born in Tranås, Sweden, 1959. He received the M.Sc. degree in Applied Physics and Electrical Engineering from Linköping University, Sweden, in 1983. He received his Ph.D. in Automatic Control from Linköping University, Sweden, in 1988, and is currently Professor in the Control group at the Department of E.E., Linköping University, Sweden. His research interests are in the areas of system identification, iterative learning control, and robot control.