# Toward Adaptive Control of QoS-Importance Decoupled Real-Time Systems

Mehdi Amirijoo*, Per Brännström*, Jörgen Hansson†, Svante Gunnarsson‡, Sang H. Son§
*Department of Computer and Information Science, Linköping University, Sweden, {meham,x06perbr}@ida.liu.se
†Software Engineering Institute, Carnegie Mellon University, USA, hansson@sei.cmu.edu
‡Department of Electrical Engineering, Linköping University, Sweden, svante@isy.liu.se
§Department of Computer Science, University of Virginia, USA, son@cs.virginia.edu

***Abstract*—This paper deals with differentiated services in real-time systems. Tasks submitted to a real-time system are differentiated with respect to importance and QoS requirements. We use feedback control to enforce the requirements in QoS and ensure a hierarchical admission policy based on the importance of the tasks. The results show that the requirements are met during steady state when the workload is constant. The feedback control approach does not satisfactorily manage QoS when there is a sudden and significant workload change (transient state) due to the time-variant nature of the system. To address this, we present preliminary and promising results using adaptive control, and report on some challenges we are facing when applying the theory.**

***Index Terms*—Automated resource allocation, workload management, adaptive control, differentiated services**

## I. Introduction

In this paper we study real-time systems delivering differentiated services. There is a deadline associated with each task, which marks the point in time at which the task must finish its execution. Tasks in the system are classified into classes, where each class has (i) a level of importance and (ii) a quality of service (QoS) requirement in terms of the ratio of tasks missing their deadlines, i.e., deadline miss ratio. The QoS decreases as the deadline miss ratio increases. During system overload the execution of some tasks has to be rejected or their execution postponed for some time. Deciding which task to admit is based on the importance level of the task; least important tasks are rejected in favor of more important tasks during overloads. We assume that accurate execution time estimates of tasks are not available, hence, actual execution times may deviate significantly from estimates. Further, we assume that the energy supply to the system is infinite (i.e., no battery is used).

Systems with the above mentioned characteristics are found in client/server applications where clients submit requests that must comply with deadlines. For example, the client may request a live TV feed, resulting in the server to periodically encode and transmit frames to the client. The clients may choose from subscriptions that differ with respect to video quality, i.e., deadline miss ratio of the video encoding task. Further some clients may be willing to pay more for subscriptions with higher availability. This corresponds to a high level of importance of the video encoding task.

The first contribution of this paper is a performance specification model that enables the tasks to be classified according to their importance and QoS requirement. The expressive power of the performance specification model allows a system operator to specify the desired nominal system performance and system adaptability in the face of unexpected failures or load variation. The second contribution is an architecture and an algorithm, based on feedback control [1], for managing the workload. Performance studies show that the suggested approach fulfills the QoS and importance requirements when the workload is constant. Rapid changes in workload, which puts the system in transient state, are handled less satisfactorily due to the time-varying nature of the system. For this we suggest an adaptive control framework that is able to react to changes in system parameters.

The rest of this paper is organized as follows. In section II a model for specifying the QoS and importance requirements is described. In section III, we present the approach and in section IV, the results of performance evaluations are presented. We are currently in the process of extending the approach presented in this paper with adaptive control [2] and in section V we report on initial findings and some experiences that we are facing at the moment of the writing. Related work is described in section VI and in section VII we give the conclusion.

## II. System Specification

Tasks are classified into service classes based on their importance. There are $V$ service classes and a service class is denoted with $svc^v$, where $1 \leq v \leq V$. Within a service class, tasks are further divided into subclasses, where each subclass represents a unique QoS requirement. In other words, subclasses of a service class hold tasks that are equally important but that have different QoS requirements. A subclasses of a service class $svc^v$ is denoted by $sbc^{v,b_v}$ ($1 \leq b_v \leq B_v$), where $B_v$ is the number of subclasses in $svc^v$. Tasks in $sbc^{v,b_v}$ are more important than tasks in $sbc^{v',b_{v'}}$ if and only if $v < v'$, for any $b_v$ and $b_{v'}$. For any $b'_v$ where $1 \leq b'_v \leq B_v$ and $b'_v \neq b_v$, tasks in $sbc^{v,b'_v}$ and $sbc^{v,b_v}$ are equally important, however, they have different QoS requirements.

The QoS requirement of each subclass is specified as follows. Let $T$ be the sampling period. If not specified otherwise, let $var^{v,b_v}(k)$ denote the variable $var$ of $sbc^{v,b_v}$ during the

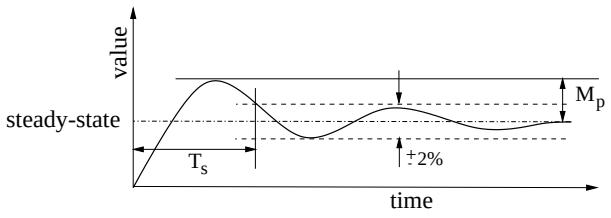


Figure 1. Definition of settling time ($T_s$) and overshoot ($M_p$)

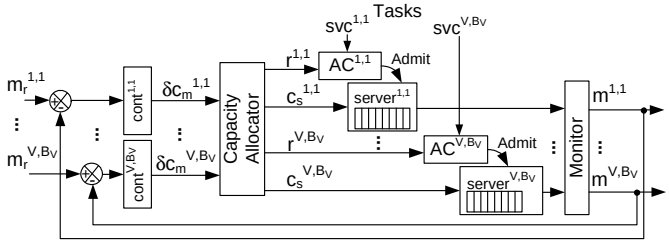


Figure 2. QoS management architecture using feedback control

time interval $(k-1)T < t \leq kT$. The number of tasks missing their deadlines is given by $n_{Miss}^{v,b_v}(k)$. A task is terminated if it has completed before its deadline or has missed its deadline. The number of terminated tasks is denoted with $n_{Term}^{v,b_v}(k)$. QoS is expressed in terms of the reference, overshoot, and settling time [1] of deadline miss ratio

$$m^{v,b_v}(k) = \frac{n_{Miss}^{v,b_v}(k)}{n_{Term}^{v,b_v}(k)}.$$

The desired miss ratio during during nominal system operation is called the reference. Overshoot $M_p^{v,b_v}$ is the worst-case system performance during the transient phase (see Figure 1) and it is given in percentage. Settling time $T_s^{v,b_v}$ is the time for the overshoot to decay and stay within $2\%$ of the reference (see Figure 1).

The following example shows a specification of QoS requirements: $\{m_r^{1,1} = 0.20, m_r^{1,2} = 0.10, m_r^{2,1} = 0.05, m_r^{2,2} = 0.15, m_r^{3,1} = 0.15\}$. For all subclasses $T_s^{v,v_b} \leq 85s$ and $M_p^{v,v_b} \leq 35\%$. We see that tasks in $sbc^{1,1}$ are more important than tasks in $sbc^{2,1}$, however, the QoS requirement for $sbc^{1,1}$ is weaker than the QoS requirement for $sbc^{2,1}$, i.e., $m_r^{1,1} > m_r^{2,1}$. This shows that the specification of importance is decoupled from the specification of QoS.

## III. Approach

### A. Architecture

The architecture of our QoS management scheme is given in Figure 2. To provide individual QoS guarantees for each task subclass we have to enforce isolation among the subclasses by bounding the resources given to the tasks in each subclass. This is achieved by using servers [3]. Let $server^{v,b_v}$ denote the server for $sbc^{v,b_v}$ and $c_s^{v,b_v}$ denote the capacity (in terms of execution time) of $server^{v,b_v}$. We assign priorities to the servers according to their importance, i.e., $server^{v,b_v}$ has higher priority than $server^{v+1,b_{v+1}}$. Servers within a service class have the same priority, i.e., $server^{v,1}, \ldots, server^{v,B_v}$ have the same priority. In general, $server^{v,b_v}$ serves any pending tasks in its ready queue within the limit of $c_s^{v,b_v}$ or until no more tasks are waiting, at which point $server^{v,b_v}$ becomes suspended and the next server in $svc^v$, i.e., $server^{v,b_v+1}$ becomes active. If $server^{v,b_v}$ is the last server of $svc^v$ (i.e., $b_v = B_v$), then $server^{v+1,1}$ becomes active. The server $server^{v',b_{v'}}$ is suspended and $server^{v,b_v}$ is reactivated if and only if $v < v'$, new tasks in $sbc^{v,b_v}$ arrive, and $c_s^{v,b_v} > 0$. The capacity is replenished periodically with the sampling period $T$. Earliest deadline first (EDF) (see e.g., [3]) is used to schedule the tasks within each server.

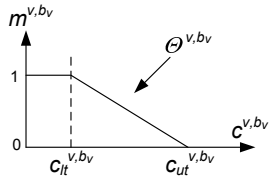


Figure 3. The relation between capacity and deadline miss ratio

At a sampling instant $k$, the difference between each controlled variable $m^{v,b_v}$ and its reference $m_r^{v,b_v}$ is formed and fed into the respective deadline miss ratio controller $cont^{v,b_v}$. Based on this each deadline miss ratio controller computes a requested change $\delta c_m^{v,b_v}$ to $c_s^{v,b_v}$. If $m^{v,b_v}$ is higher than $m_r^{v,b_v}$, then a positive $\delta c_m^{v,b_v}$ is returned, requesting an increase in the capacity so that $m^{v,b_v}$ converges to its reference. The requested change in capacity of all subclasses is given to the capacity allocator, which distributes the capacities according to the service class level. During overloads it may not be possible to accommodate all requested capacities. Instead, the amount of capacity $r^{v,b_v}$ that is not accommodated is returned to the admission controller, which rejects tasks with a total execution time of $r^{v,b_v}$.

### B. Modeling and Controller Design

For the purpose of the controller design we have modeled the controlled system using $\mathcal{Z}$-transform theory [1]. Starting with the manipulated variable, the total capacity during period $(k-1)T < t \leq kT$ is,

$$c^{v,b_v}(k) = c^{v,b_v}(k-1) + \delta c_m^{v,b_v}(k-1). \quad (1)$$

Note, the total capacity $c^{v,b_v}$ includes the capacity given to the server $c_s^{v,b_v}$ and $r^{v,b_v}$. Now, there exists a nonlinear relation between $c^{v,b_v}$ and $m^{v,b_v}$, as shown in Figure 3. For capacities less than the lower threshold $c_{lt}^{v,b_v}$ all tasks miss their deadlines, hence, $m^{v,b_v} = 1$. The miss ratio $m^{v,b_v}$ decreases as $c^{v,b_v}$ increases, since more CPU time is allocated to tasks. There exists an upper threshold $c_{ut}^{v,b_v}$ at which $m^{v,b_v}$ becomes zero. We linearize the relationship between $m^{v,b_v}$ and $c^{v,b_v}$ at the vicinity of $m_r^{v,b_v}$, i.e.,

$$m^{v,b_v}(k) = \Theta^{v,b_v}(k) c^{v,b_v}(k) \quad (2)$$

where $\Theta^{v,b_v}(k)$ is the time-varying miss ratio gain, see Figure 3, which among other factors depends on the incoming workload in $sbc^{v,b_v}$. Equations (1) and (2) give that,

$$m^{v,b_v}(k) = m^{v,b_v}(k-1) + \Theta^{v,b_v}(k) \times \delta c_m^{v,b_v}(k-1). \quad (3)$$

For now we assume that $\Theta^{v,b_v}(k)$ is constant and consider the time-varying case in section V where we discuss extensions

with adaptive control [2]. By taking the $\mathcal{Z}$-transform of (3), where $\Theta^{v,b_v}(k)$ is constant, we obtain the transfer function,

$$P^{v,b_v}(z) = \frac{M^{v,b_v}(z)}{\Delta C_m^{v,b_v}(z)} = \frac{\Theta^{v,b_v}}{z-1}. \quad (4)$$

We now investigate whether a proportional (P) controller [1] is enough in terms providing a zero steady state error, i.e., a zero difference between $m_r^{v,b_v}(k)$ and $m^{v,b_v}(k)$ during steady state. The P controller has the transfer function $K_P$, where $K_P$ is a tunable parameter. The closed loop function from $M_r^{v,b_v}(z)$ to $M^{v,b_v}(z)$ is given by

$$G_c^{v,b_v}(z) = \frac{K_P^{v,b_v}\Theta^{v,b_v}}{z-(1-K_P^{v,b_v}\Theta^{v,b_v})}.$$

For the closed-loop system to be stable the pole must be within the unit circle, i.e., $0 < K_P^{v,b_v}\Theta^{v,b_v} < 2$. Under the assumption that the closed-loop system is stable, then the steady state error is zero since $G_c^{v,b_v}(1) = 1$ [1]. Hence, a P controller is sufficient.

The simplicity of the model (4) facilitates the derivation of the settling time. Assume that $m_r^{v,b_v}(k)$ is a step function, i.e., $m_r^{v,b_v}(k) = 1$ for $k \geq 0$ and $m_r^{v,b_v}(k) = 0$ for $k < 0$. Then, $M^{v,b_v}(z) = G_c^{v,b_v}(z)\frac{z}{z-1}$ and for $k \geq 0$ we have that

$$m^{v,b_v}(k) = 1 - (1 - K_P^{v,b_v}\Theta^{v,b_v})^k. \quad (5)$$

Using (5) we obtain that the settling time is,

$$T_s^{v,v_b} = \left\lceil \frac{\ln 0.02}{\ln(1-K_P^{v,b_v}\Theta^{v,b_v})} \right\rceil T.$$

For example, by setting $K_P^{v,b_v} = \frac{0.21}{\Theta^{v,b_v}}$ we obtain that $T_s^{v,v_b} = 85s$ given that $T = 5$s. We have carried out an experiment where we have set the load to 200% (for details see section IV-A) and found that $\Theta^{1,1} \approx -0.00077$. This gives that $K_P^{1,1} \approx -273$ according to above (note that $\Theta^{1,1}$ is negative). We have for simplicity used the same $K_P$ for all subclasses, i.e., $K_P^{v,b_v} = -273$ for all $v$ and $b_v$.

### C. Capacity Allocation

Figure 4 shows how $c_s^{v,b_v}$ and $r^{v,b_v}$ are computed. This algorithm implements the capacity allocator in Figure 2. We start allocating capacities with respect to the service classes, starting with subclasses in $svc^1$. The requested capacity $q_{req}^{v,b_v}(k+1)$ of subclass $sbc^{v,b_v}$ is the sum of the previously computed capacity $q^{v,b_v}(k)$ and the requested change in capacity $\delta c_m^{v,b_v}(k)$ that is given by the controller (line 7). Then we compute the sum $q_{req}^v(k+1)$ of the requested capacities of all subclasses of service class $svc^v$ (line 8). The ratio $ratio^v$ of requested capacity $q_{req}^v(k+1)$ that can be allocated is derived (lines 10-14). The assigned capacities are computed by taking the product of the requested capacities and the ratio (line 18).

If the entire requested capacity cannot be accommodated, i.e., $ratio^v < 1$, we enforce the capacity adjustment by rejecting more tasks, i.e., increasing $r^{v,b_v}(k+1)$ (lines 19 and 20). However, if the requested capacity is accommodated then we try to reduce the number of rejected tasks (lines 21-24). The rejected capacity $r^{v,b_v}(k+1)$ is lowered and additional capacity is allocated to compensate for the decrease in $r^{v,b_v}(k+1)$ (lines 22 and 23). If the requested capacity is accommodated and no tasks were rejected during the previous sampling interval then we do not reject any tasks during the next sampling interval, i.e., we set $r^{v,b_v}(k+1)$ to zero (lines 25 and 26). Finally, after the capacity allocation we check to see whether there is any spare capacity $q_s(k+1)$, which is evenly distributed among the servers (lines 31-35). $B$ denotes the total number of subclasses, i.e., $B = \sum_{v=1}^{V} B_v$.

```
ComputeCapacity(δc_m^{1,1}(k+1), ..., δc_m^{V,B_V}(k+1))
 1: q(k+1) ← 0
 2: for v = 1 to V do
 3:     q_req^v(k+1) ← 0
 4:     r^v(k+1) ← 0
 5:     for b_v = 1 to B_v do
 6:         r^v(k) ← r^v(k) + r^{v,b_v}(k)
 7:         q_req^{v,b_v}(k+1) ← max(0, q^{v,b_v}(k) + δc_m^{v,b_v}(k))
 8:         q_req^v(k+1) ← q_req^v(k+1) + q_req^{v,b_v}(k+1)
 9:     end for
10:     if 0 ≤ q_req^v(k+1) < T − q(k+1) then
11:         ratio^v = 1
12:     else
13:         ratio^v = (T − q(k+1)) / q_req^v(k+1)
14:     end if
15:     q(k+1) ← q(k+1) + q_req^v(k+1) × ratio^v
16:     w^v ← T − q(k+1)
17:     for b_v = 1 to B_v do
18:         q^{v,b_v}(k+1) ← q_req^{v,b_v}(k+1) × ratio^v
19:         if ratio^v < 1 then
20:             r^{v,b_v}(k+1) ← r^{v,b_v}(k) + q_req^{v,b_v}(k+1) − q^{v,b_v}(k+1)
21:         else if r^v(k) > 0 then
22:             r^{v,b_v}(k+1) ← r^{v,b_v}(k) − (r^{v,b_v}(k) / r^v(k)) w^v
23:             q^{v,b_v}(k+1) ← q^{v,b_v}(k+1) + r^{v,b_v}(k) − r^{v,b_v}(k+1)
24:             q(k+1) ← q(k+1) + r^{v,b_v}(k) − r^{v,b_v}(k+1)
25:         else
26:             r^{v,b_v}(k+1) ← 0
27:         end if
28:     end for
29: end for
30: q_s(k+1) ← T − q(k+1)
31: for v = 1 to V do
32:     for b_v = 1 to B_v do
33:         c_s^{v,b_v}(k+1) ← q^{v,b_v}(k+1) + (q_s(k+1)) / B
34:     end for
35: end for
```

Figure 4. The capacity allocation algorithm

The algorithm in Figure 4 runs in the worst-case in $\mathcal{O}(VB_{max})$, where $B_{max} = \max_{1 \leq v \leq V} B_v$. Hence, the time complexity is pseudo-polynomial [4], showing that the algorithm scales well with the number of service classes and number of subclasses.

## IV. Performance Evaluation

### A. Experiment Setup

One simulation run lasts for 2000s of simulated time. For all the performance data, we have taken the average of 10 simulation runs and derived 95% confidence intervals. We consider aperiodic tasks where the average execution time $aet_i$ of a task $\tau_i$ is uniformly distributed between 1ms and 10ms, i.e., $U : (1ms, 10ms)$. The actual execution time of an instance of $\tau_i$ is given by the normal distribution $N : (aet_i, \sqrt{aet_i})$. The average inter-arrival time and the relative deadline of $\tau_i$ are

set to $aet_i \times slackfactor_i$, where the slack factor is uniformly distributed according to $U : (50, 100)$. The inter-arrival times are exponentially distributed. The workload of $\tau_i$ is given by $\frac{1}{slackfactor_i}$.

To assess whether the importance requirement of the tasks is satisfied we measure the admission ratio,

$$ar^{v,b_v}(k) = \frac{n^{v,b_v}_{Admit}(k)}{n^{v,b_v}_{Submit}(k)}$$

where $n^{v,b_v}_{Admit}(k)$ is the number of admitted tasks and $n^{v,b_v}_{Submit}(k)$ is the number of submitted tasks in $sbc^{v,b_v}$ during the time interval $(k-1)T < t \leq kT$.

In the experiment presented here, we consider five subclasses $sbc^{1,1}$, $sbc^{1,2}$, $sbc^{2,1}$, $sbc^{2,2}$, and $sbc^{3,1}$. The QoS specification given in section II is used. The workload submitted to the real-time systems is distributed among the subclasses according to 25%, 10%, 25%, 25%, and 15%. The workload distribution captures the cases when the workload is equally divided among the subclasses in a service class (subclasses in $svc^2$) and the case when the workload is not equally divided among the subclasses in a service class (subclasses in $svc^1$). No workload is submitted before time $0s$, hence, the critical instant occurs at $0s$ which produces the worst-case workload change. This puts the system in a transient state, where $m$ and $ar$ vary significantly in response to the change in workload. The transient state is followed by the steady state where $m$ and $ar$ have settled.

### B. Experiment 1: Steady State

The goal of this experiment is to see how the approach reacts to increasing submitted load. We show that a given system specification is satisfied. We have set $K_P^{v,b_v} = -273$ for all subclasses. We measure $m$ and $ar$ and apply loads from 20% to 500%. The applied workload during a simulation run is constant and we increase the workload between the runs. Recall from section IV-A that the workload increases stepwise at time 0s. This causes a transient state followed by a steady state where $m$ and $ar$ have converged. We measure the system performance during steady state (the transient state performance is examined in detail in section IV-C) and, as such, we start measuring $m$ and $ar$ at $700s$. Figure 5 shows $m$ and $ar$. The dashed lines denote the references.

Starting with the admission ratio given in Figure 5, we note that as the load increases the admission ratio $ar^{3,1}$ of subclass $sbc^{3,1}$, representing the least important tasks, decreases. When most of the tasks in $sbc^{3,1}$ are rejected, the admission ratio of $sbc^{2,1}$ and $sbc^{2,2}$, i.e., $ar^{2,1}$ and $ar^{2,2}$ starts decreasing. For loads over 300% the admission ratio of $sbc^{1,1}$ and $sbc^{1,2}$ starts decreasing. Hence, the strict hierarchical admission policy, where the least important tasks are rejected in favor of the most important tasks, is enforced. Turning to deadline miss ratio, we note that $m^{v,b_v}$ increases as the load increases, reaching the references at 100% load. The miss ratio is close to the reference for loads greater than 100%, hence, the QoS requirement is satisfied. Note that $m^{v,b_v}(k)$ is computed over admitted tasks, hence, $m^{v,b_v}(k) = 0$ when $ar^{v,b_v}(k) = 0$.

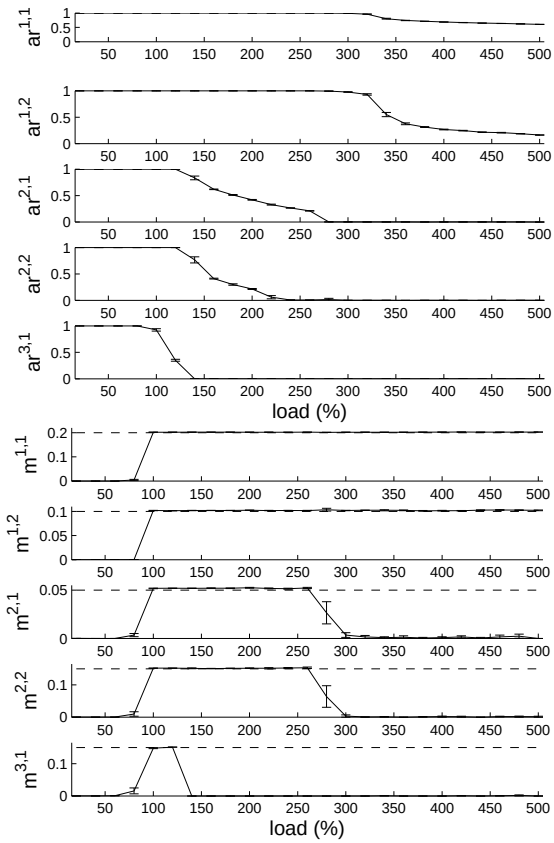

Figure 5. Experiment 1: Varying load

In summary we have shown that the approach provides reliable performance that is consistent with the system specification. More specifically, the admission mechanism enforces the strict hierarchical admission policy, where the least important tasks are rejected and the most important tasks are admitted and executed. Also, the experiments show that during nominal system operation (steady state) the QoS requirement is satisfied with respect to the references for the deadline miss ratio, i.e., the deadline miss ratio equals the desired miss ratio. This is a key step toward performance management for systems where importance and QoS are decoupled.

### C. Experiment 2: Transient State

Studying the average performance is often not enough when dealing with dynamic systems and, therefore, we study the transient performance. In section III-B we mentioned that the same control parameter $K_P^{v,b_v}$ is used for all subclasses. The goal of these experiment is twofold. First, we establish whether it is sufficient to use the same $K_P^{v,b_v}$ for all subclasses. Second, we show if a constant $K_P^{v,b_v}$ results in similar settling times for different loads.

We first investigate the response of the deadline miss ratio for $m^{1,1}(k)$ and $m^{2,1}(k)$ when the load is set to 200% and the same $K_P$ is used for $sbc^{1,1}(k)$ and $sbc^{2,1}(k)$. As we can see in Figure 6(a), there is a significant difference in settling times between $m^{1,1}(k)$ and $m^{2,1}(k)$, thus, using the same $K_P$ is not sufficient. By decreasing $K_P^{2,1}$, i.e., increasing $|K_P^{2,1}|$, we obtain shorter settling times as shown in Figure 6(b).

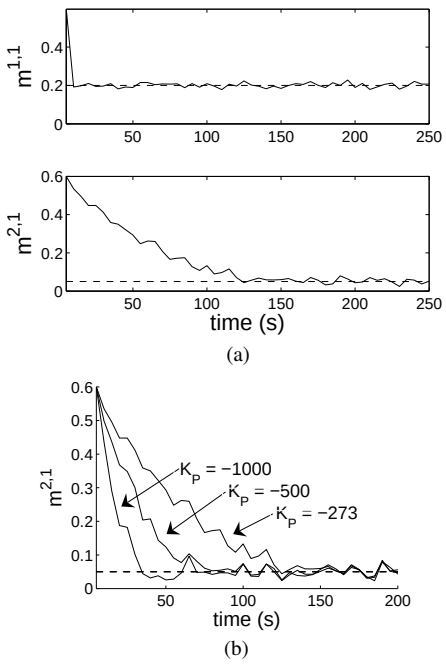


Figure 6. Transient performance when 200% load is applied.

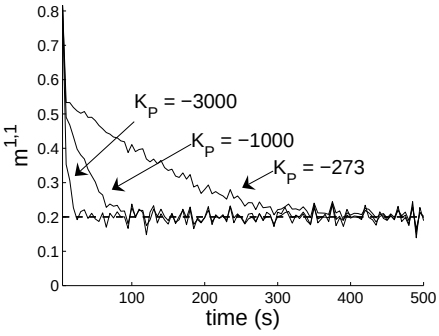


Figure 7. Transient performance when 500% load is applied.

Consequently, to provide similar QoS guarantees with respect to settling time there is a need to use different $K_P$ for the subclasses.

Next, we investigate the response of $m^{1,1}(k)$ when the load is 500% and $K_P^{1,1} = -273$. Recall, that we computed this value for $K_P^{1,1}$ based on 200% load. As we can see from Figure 7 the settling time increases significantly when the load is 500% (compare to the settling time in Figure 6(a) where 200% load is applied). By reducing $K_P^{1,1}$ we can decrease the settling time as shown in Figure 7. We conclude that there is a need to alter $K_P$ as a function of the load applied on the system. The control parameter $K_P$ must decrease as the load increases.

From the experiments in this section we draw two conclusions. First, we showed that the settling time of the controlled variable deadline miss ratio varies significantly between the subclasses. Although, the deadline miss ratio converges to the reference, as predicted by the theory in section III-B, there is a difference in the convergence rate. As such, it is insufficient to design controllers based on experimental data from one single subclass. Second, we showed that the control parameters must vary with the load in order to provide settling time guarantees. These observations suggest that we need to pursue an approach allowing the control parameters to adapt according to prevailing conditions. We are currently investigating this approach and we report on some initial findings and experiences in the following section.

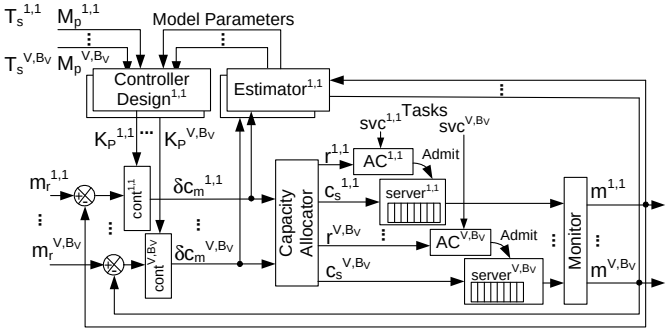


Figure 8. QoS management architecture using feedback control

## V. Extending with Adaptive Control

As we noted in section IV-C significantly different settling times were observed among the subclasses and for varying workload. There is a rich body of knowledge within the signal processing and control community regarding time-varying systems, e.g., gain scheduling and self-tuning regulators [2]. Gain scheduling is less attractive in our case since we have to build controllers for each subclass and different workloads. Instead, we pursue the approach where self-tuning regulators are used. Here we estimate the system parameters in real-time and update the control parameters accordingly, as shown in Figure 8.

In the following we drop the superscript $v, v_b$ when the notion of subclass is not of primary interest. The signals $\delta c_m$ and $m$ of a subclass are forwarded to the respective estimator, which estimates the parameters of the controlled system. Given the QoS specification in terms of the overshoot $M_p$ and settling time $T_s$, the control parameter $K_P$ is updated once a new model parameter estimate is available. The design of $K_P$ is carried out through pole placement where the position of the poles is constant, i.e., the settling time of $m$ is preserved from one update to another.

We assume the following model structure $m(k) = \varphi^T(k)\Theta^0(k) + e(k)$, where $\varphi^T(k) = (m(k-1), \ldots, \delta c_m(k-1), \ldots)$ is the regression vector of the lagged variables $m(k)$ and $\delta c_m(k)$, $\Theta^0(k) = (\Theta_1^0(k), \Theta_2^0(k), \ldots)^T$ is the true system parameters that we wish to estimate, and $e(k)$ is the measurement disturbance [5]. When estimating system parameters there is a significant advantage in incorporating available prior information. For example, we could assume the structure given by (3), i.e., $m(k) = \Theta_1(k)m(k-1) + \Theta_2(k)\delta c_m(k-1)$. However, $\Theta_1 = 1$ according to (3), hence, we only estimate $\Theta_2(k)$. We use (3) and introduce the model

$$\delta m(k) = m(k) - m(k-1) = \Theta_1(k)\delta c_m(k-1).$$

Now that we have arrived at a model the next question is which algorithm to choose for estimating $\Theta_1(k)$. Let $\hat{\Theta}(k)$ be the estimate of $\Theta^0(k)$. There is a vast choice of different algorithms in the literature, e.g., recursive least squares (RLS), RLS with exponential forgetting, least mean squares (LMS) and normalized LMS (NLMS) [2]. There is also the possibility to model $\Theta^0(k)$ as a time-varying process,

$$\Theta^0(k+1) = \Theta^0(k) + w(k) \tag{6}$$

and use a Kalman observer to estimate $\Theta^0(k)$ [6]. These algorithms have a common structure,

$$\hat{\Theta}(k) = \hat{\Theta}(k-1) + P(k)\varphi(k)\left(m(t) - \varphi^T(k)\hat{\Theta}(k-1)\right)$$

and they differ the criterion they are minimizing, e.g., least squares or least mean squares. The choice of criterion determines $P(k)$. The RLS with exponential forgetting and the Kalman observer approach are useful when handling time-variant systems, i.e., when (6) applies. However, one of the problems with exponential forgetting is that $P(k)$ diverges when $\delta c_m(k) = 0$. The main obstacle with using a Kalman observer is that the variance of $w(k)$ must be known; this knowledge is often lacking [6]. We have therefore in our work chosen to use a more pragmatic approach, namely, the LMS algorithm $P(t) = \gamma$, which is commonly used for estimating systems [6].

Our initial results show that the estimates $\hat{\Theta}(k)$ converge toward $\Theta^0(k)$ as long as the controlled system is excited. In our simulations we have assumed that the workload is constant. This represents the worst-case scenario from a system identification perspective as we do not have any disturbances, causing $\delta c_m(k)$ to fall into a steady state. Also we have observed that the settling time of the estimate is significantly longer than the settling time of $m(k)$. As such, $\delta c_m(k) \approx 0$ before the estimate has converged. Once $\delta c_m(k) \approx 0$, no additional information about the controlled system is gained and this causes the estimate to drift.

To allow the estimate to converge when the workload is constant (representing the worst-case) we are considering employing an approach where we periodically add a disturbance to $\delta c_m(k)$. The estimation is turned on while the disturbance exists and turned off when the disturbance is removed to avoid the estimate to drift. The disturbance should be active long enough for the estimate to converge. This way we can track the system and adapt the control parameters at the expense of variations in $m(k)$, which causes a degradation in QoS. We expect, however, $\delta c_m(k)$ to vary naturally due to alterations in workload and, as such, the addition of a disturbance to $\delta c_m(k)$ will not be necessary for the majority of time.

## VI. Related Work

Due to space limitation we mainly discuss adaptive control. For references to computer performance control we refer to [1]. Abdelzaher report on some results on workload parameter estimation using the RLS method [7]. Lu et al. applied adaptive control in Web servers with the goal of controlling the relative hit ratio of different classes [8]. A self-tuning regulator using the NLMS algorithm [2] and pole placement, similar to the one presented in section V, was used. They show that adaptive control results in better control than a feedback loop with no adaptation. This work was extended with a stochastic adaptive control algorithm for handling parameter uncertainty and disturbances in the system [9]. However, as the authors argue, the statistical characterization of such disturbances are difficult to obtain. This work is an extension to our previous work [10], where we addressed differentiated QoS management of real-time databases using P controllers. In this paper we consider general real-time systems rather than real-time databases. We motivate through experiments why there is a need in adaptive control and discuss extensions with adaptive control.

## VII. Conclusion

In this paper we have considered real-time systems where tasks have importance and QoS requirements. Important tasks must be admitted and executed and less important tasks must be rejected during overloads. The QoS, expressed in terms of deadline miss ratio of admitted tasks, must comply with a given requirement. We show that the presented approach satisfies these requirements during steady state when the workload is constant. However, the results are less satisfactory for cases when the workload changes abruptly, which represents a worst-case condition in real-time systems. We showed why this is the case and proposed solutions using adaptive control. We are currently in the process of implementing the proposed approach. Initial results are promising and we expect our complete approach to fully satisfy a given system specification.

## References


[1] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[2] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed. Addison-Wesley, 1995.

[3] G. C. Buttazzo, *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, 1997.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[5] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. H. Son, "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)*, 2005.

[6] L. Ljung and S. Gunnarsson, "Adaptation and tracking in system identification - a survey," *Automatica*, vol. 26, no. 1, pp. 7–21, 1990.

[7] T. F. Abdelzaher, "An automated profiling subsystem for QoS-aware services," in *Proceedings of the Real-Time Technology and Applications Symposium (RTAS)*, 2000.

[8] Y. Lu, T. F. Abdelzaher, C. Lu, and G. Tao, "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)*, 2002.

[9] Y. Lu, T. Abdelzaher, and G. Tao, "Direct adaptive control of a web cache system," in *Proceedings of the American Control Conference*, 2003.

[10] M. Amirijoo, J. Hansson, S. H. Son, and S. Gunnarsson, "Generalized performance management of multi class real-time imprecise data services," in *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS)*, 2005.