Real-Time Systems (Special Issue on Control-Theoretical Approaches to Real-Time Computing). To appear in 2002.

Feedback-Feedforward Scheduling of Control Tasks

Anton Cervin, Johan Eker, Bo Bernhardsson, Karl-Erik Årzén

Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
anton@control.lth.se

Abstract

A scheduling architecture for real-time control tasks is proposed. The scheduler uses feedback from execution-time measurements and feedforward from workload changes to adjust the sampling periods of the control tasks so that the combined performance of the controllers is optimized. The performance of each controller is described by a cost function. Based on the solution to the optimal resource allocation problem, explicit solutions are derived for linear and quadratic approximations of the cost functions. It is shown that a linear rescaling of the nominal sampling frequencies is optimal for both of these approximations. An extensive inverted pendulum example is presented, where the performance obtained with open-loop, feedback, combined feedback and feedforward scheduling, and earliest-deadline first scheduling are compared. The performance under earliest-deadline first scheduling is explained by studying the behavior of periodic tasks under overload conditions. It is shown that the average values of the sampling periods equal the nominal periods, rescaled by the processor utilization.

1. Introduction

There is currently a trend towards more flexible real-time control systems. Combining scheduling theory and control theory opens up the possibility to achieve higher CPU-resource utilization and better control performance. To obtain the best results, co-design of the scheduler and the controllers is necessary.

Control tasks are generally viewed by the scheduling community as *hard* real-time tasks with *fixed* sampling periods and *known* worst-case execution times (WCETs). Upon closer inspection, neither of these assumptions need necessarily be true. For instance, many control algorithms are quite robust against variations in sampling period and input-output latency. In some situations it is also possible to let the controller actively compensate for the variations by, e.g., recalculating the controller parameters. Hence, the timing nondeterminism can be viewed as a disturbance or uncertainty affecting the control loop. Controllers can also be designed to switch between different modes with different execution times and/or different sampling periods. An on-line scheduler that uses feedback to dynamically adjust the control task attributes in order to optimize the global control performance can be interpreted as a controller in itself, i.e., as a feedback scheduler. The control performance, or *Quality-of-Control* (QoC), can, hence, be interpreted as a quality-of-service measure.

In this work, we present a feedback scheduler for control tasks that attempts to keep the CPU utilization at a high level, avoid overload, and distribute the computing resources among the control tasks so that the total control performance is optimized. While we want to keep the number of missed deadlines as low as possible, control performance is our primary objective. Thus, control tasks, in our view, fall in a category somewhere between hard and soft real-time tasks. The known-WCET assumption is relaxed by the use of feedback from execution-time measurements. We also introduce feedforward to further improve the regulation of the utilization. The feedforward action can be interpreted as an admission controller. The article is based on the feedback scheduling approach in [Cervin and Eker, 2000]. There the problem of scheduling a set of hybrid control tasks was studied. The hybrid controllers switched between different modes with different execution-time characteristics. The feedback scheduler adjusted the controller task periods through a simple linear rescaling without any direct connection to control performance optimization. In this paper we instead study the problem of scheduling a set of linear controller tasks. The performance of each controller is described by a cost function. By starting and stopping the different control tasks the CPU load is varied. The sampling periods are adjusted so that the total cost of the controllers is minimized.

In [Eker et al., 2000] a feedback scheduler for the special case of linear-quadratic (LQ) control tasks

was presented. An LQ-controller is a linear state-feedback controller that is designed to minimize a quadratic cost criterion. Formulas for the LQ cost function and its derivative with respect to the sampling period were given. The scheduling problem was formulated as a recursive optimization problem based on the exact formulas for the derivatives of the cost functions. Due to the high computational costs involved, an approximate version of the scheduler was also developed. The cost functions were approximated by quadratic functions of the sampling periods, and explicit expressions for the optimal sampling periods were derived. That result is utilized in this paper. Furthermore, we allow for a set of general linear controllers and also study the results obtained when the cost functions are approximated by linear functions of the sampling periods. On the implementation side, both priority-based and earliest-deadline-first (EDF) based approaches are studied. A new result concerning the properties of EDF scheduling for periodic control tasks during overload conditions is presented.

1.1 Outline of the paper

Additional motivation and a survey of related work is contained in Section 2. The CPU-resource allocation problem for a set of control tasks is treated in Section 3. The structure of the proposed scheduling architecture is described in Section 4. The feedback scheduling example is presented in Section 5. A comparison is made between open-loop scheduling, feedback-based scheduling and combined feedback and feedforward scheduling. Also, ordinary earliest-deadline-first scheduling is studied. A new result concerning the properties of EDF schedules for periodic tasks during overload conditions is presented in Section 6. The conclusions are given in Section 7.

2. Background

Most control systems are embedded systems where the computer is a component in a larger engineering system. The control system is often implemented on a microprocessor using a real-time kernel or a real-time operating system. The real-time kernel or the operating system uses multiprogramming to multiplex the execution of the tasks on the CPU. The CPU time, hence, constitutes a shared resource which the tasks compete for. To guarantee that the time requirements and time constraints of the individual tasks are all met, it is necessary to schedule the usage of the shared resource.

In traditional hard real-time computing models, it is assumed that control tasks fit the simple task model, i.e., the tasks are periodic, or can be transformed to periodic tasks, with a fixed period, a known worst-case bound on the execution time (WCET), and a hard deadline. The fixed-period assumption of the simple task model has also been widely adopted by the control community and has, e.g., resulted in the development of the sampled computer-control theory with its assumption on deterministic, equi-distant sampling.

Upon closer inspection it is quite clear that many of the assumptions of the simple model are too restrictive. First, the assumptions do not allow us to use low-cost general purpose hardware and off-the-shelf operating systems, which in general are not able to give any guarantees on determinism. These systems are, typically, designed to achieve good average performance rather than guaranteed worst-case performance. They often introduce significant non-determinism in task scheduling. For computation-intensive high-end applications, the large variability in execution time caused by modern hardware architecture also becomes visible. The effect of this on the control loop is jitter in sampling period and control delay (input-output latency). In order to maintain good control performance it is important to compensate on-line for the variations. A requirement for this is that the necessary timing information is provided by the real-time kernel.

The assumptions of the simple model are also overly restrictive with respect to the characteristics of many control loops. Many control loops are not periodic, or they may switch between a number of different fixed sampling periods. Control loop deadlines are not always hard. On the contrary, many controllers are quite robust towards variations in sampling period and response time. Hence, it is questionable whether it is necessary to model them as hard deadline tasks. It is also in many cases possible to compensate on-line for the variations by, e.g., recomputing the controller parameters. Obtaining an accurate value for the WCET is generally a difficult problem. Measuring WCET always implies the risk of underestimation, whereas analytical execution time analysis tools are still rare. An alternative way may be to instead measure the actual execution time every task invocation and to adjust the task parameters accordingly. Finally, it is also possible to consider control systems that are able to do a tradeoff between the available computation time, i.e., how long the controller may spend calculating the new control signal, and the control loop performance.

On-line task attribute adjustment mechanisms can in certain situations be viewed as controllers in themselves. Using feedback from on-line measurements of, e.g., actual execution times, the resource utilization of the different tasks is changed. If the tasks are control tasks, feedback from the control performance can also be used to, e.g., optimize control performance under given resource utilization

constraints. The feedback scheduler then solves an optimization problem to perform the task attribute adjustment. The use of on-line optimization as a way of realizing feedback control is quite common in advanced control systems. One example is model predictive control (MPC) where a convex optimization problem is solved in each sample.

2.1 Related Work

The related work falls into three categories: integrated control and real-time system design, quality-of-service approaches in real-time systems, and flexible and adaptive real-time system algorithms and architectures.

In [Seto et al., 1996], sampling period selection for a set of control tasks is considered. The performance of a task is given as function of its sampling frequency, and an optimization problem is solved to find a set of optimal task periods. Co-design of real-time control systems is also considered in [Ryu et al., 1997], where the performance parameters are expressed as functions of the sampling periods and the input-output latencies. [Shin and Meissner, 1999] deals with on-line rescaling and relocation of control tasks in a multi-processor system.

The second area of related work is on quality-of-service (QoS) aware real-time software, where a system's resource allocation is adjusted on-line in order to maximize the performance in some respect. In [Li and Nahrstedt, 1998] a general framework is proposed for controlling the application requests for system resources using the amount of allocated resources for feedback. It is shown that a PID (proportional-integral-derivative) controller can be used to bound the resource usage in a stable and fair way. In [Abeni and Buttazzo, 1999] task models suitable for multimedia applications are defined. Two of these use PI control feedback to adjust the reserved fraction of CPU bandwidth. The resource allocation scheme Q-RAM is presented in [Rajkumar et al., 1997]. Several tasks are competing for finite resources, and each task is associated with a utility value, which is a function of the assigned resources. The system distributes the resources between the tasks to maximize the total utility of the system. In [Abdelzaher et al., 1997] a QoS renegotiation scheme is proposed as a way to allow graceful degradation in cases of overload, failures or violation of pre-run-time assumptions. The mechanism permits clients to express, in their service requests, a range of QoS levels they can accept from the provider, and the perceived utility of receiving service at each of these levels. The approach is demonstrated on an automated flight-control system. Control-theoretical approaches for QoS adaptation are also presented in [Abdelzaher and Shin, 1999].

The third area relates to the wealth of flexible scheduling algorithms available. An interesting alternative to linear task rescaling is given in [Buttazzo et al., 1998], where an elastic task model for periodic tasks is presented. The relative sensitivity of tasks to rescaling are expressed in terms of elasticity coefficients. Schedulability analysis of the system under EDF scheduling is given. Task attribute adjustment strategies are also presented in [Nakajima, 1998; Kuo and Mok, 1991; Kosugi et al., 1994; Nakajima and Tezuka, 1994; Lee et al., 1996].

The idea of using feedback in scheduling has to some extent been used previously in general purpose operating systems in the form of multi-level feedback queue scheduling [Kleinrock, 1970; Blevins and Ramamoorthy, 1976; Potier et al., 1976]. However, this has mostly been done in an ad-hoc way. Related to our work, [Stankovic et al., 1999; Lu et al., 1999] present a scheduling algorithm, the FC-EDF, that explicitly uses feedback in combination with EDF scheduling. A PID controller regulates the deadline miss-ratio for a set of soft real-time tasks with varying execution times, by adjusting their requested CPU utilization. It is assumed that tasks can change their CPU consumption by executing different versions of the same algorithm. An admission controller is used to accommodate larger changes in the workload. In [Lu et al., 2000] the same approach is extended. An additional PID controller is added that instead controls the CPU utilization. The two controllers are combined using a *min*-approach. The resulting hybrid controller scheme, named FC-EDF^2, gives good performance both during steady-state and under transient conditions. Although related to the work presented here, there are important differences. In our approach the tasks that are scheduled are controllers, controlling some physical plants. The performance, or *Quality-of-Control (QoC)* is explicitly used by the feedback scheduler to optimize the total performance of all the controller tasks.

In order to evaluate how the scheduling of controller tasks influence the control performance it is necessary to have simulators that allow joint simulation of continuous-time plant dynamics, discrete-time controllers, and the real-time scheduling of the corresponding controller tasks. The simulations in the current paper are based on the Matlab/Simulink toolbox presented in [Eker and Cervin, 1999]. A similar tool is presented in [Palopoli et al., 2000].

An extensive survey with additional references to other related work in the area of control and CPU scheduling can be found in [Årzén et al., 1999].

3. Control Loop Resource Allocation

In this section, we take a higher-level, idealized view of a feedback scheduling system and study the problem of distributing limited computing resources to a number of control loops. For now, it is assumed that lower-level mechanisms in the real-time system can provide the feedback information needed, and that the available computing resources can be divided in an exact way.

We study the problem of scheduling several independent control tasks on a shared processing unit. The plants are described by linear continuous-time systems, the controllers are described by linear discrete-time systems, and the performance of the control loops are measured using quadratic performance criteria. The goal is to assign sampling periods to the control loops such that the overall control performance is optimized subject to the schedulability constraint. The effects of control delay and jitter on the control performance are neglected at this stage.

The next section introduces a standard mathematical model of a computer-controlled system. More details can be found in textbooks on sampled-data control theory, e.g. [Åström and Wittenmark, 1997].

3.1 Control System Description

Each physical process (or *plant*) to be controlled is assumed to be described by a set of linear differential equations, which give the relationship between the input (the control signal) and the output (the measurement signal). A standard mathematical representation of such a system is

$$\begin{aligned} \frac{dx(t)}{dt} &= Ax(t) + Bu(t) + Gw(t) \\ y(t) &= Cx(t) + v(t) \end{aligned} \tag{1}$$

Here, x is a vector of state variables (the *state* of the plant), u is the control signal, y is the measurement signal, and w and v are uncorrelated zero-mean Gaussian white noises which disturb the states and the measurements. A, B, C, and G are constant matrices that describe the dynamics of the plant.

The goal of the controller is to keep the plant state as close as possible to zero (which is assumed to be the desired state) in the presence of noise and other disturbances. As the plant exists in the real world, it is described in continuous time. On the other side, the controller is implemented in a computer and is therefore inherently a discrete-time system. A general linear controller can be described by a set of linear difference equations,

$$\begin{aligned} x^c_{k+1} &= \Phi^c x^c_k + \Gamma^c y_k \\ u_k &= C^c x^c_k + D^c y_k \end{aligned} \tag{2}$$

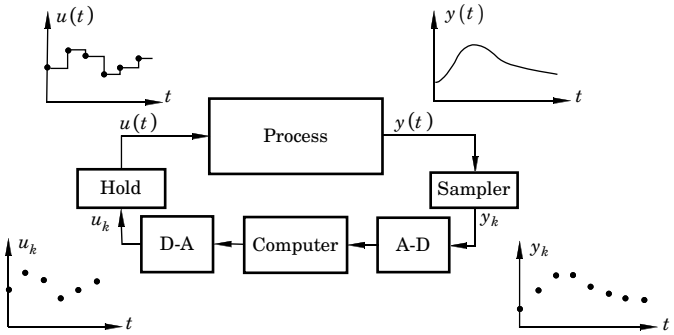

Figure 1 Relationships among the continuous and the discrete signals in the control loop.

Here, x^c is the state vector of the controller, and Φ^c, Γ^c, C^c, and D^c are matrices which describe the dynamics of the controller. Depending on how the controller is designed, these matrices may or may not be functions of the sampling period. In the interconnection between the continuous-time and the discrete-time system, y is sampled and u is held at a constant value until the next update, see Figure 1.

The controller could be designed using a large number of techniques, for instance loop shaping, pole placement, or using optimal control theory. There is also the choice between direct discrete design or translation from continuous design to discrete implementation. The linear controller description (2) covers many common control principles, including PID control, state feedback control from an observer, lead-lag filtering, etc.

3.2 Evaluating Control Performance

A good control design should fulfill a large number of criteria, including fast disturbance rejection, good robustness towards plant variations, low noise sensitivity, etc. In this work, however, we focus on sampling period selection, and we choose a simple performance criterion that can be used to directly compare the performance of the control loops under different sampling periods. The performance of a controller is measured using a quadratic cost criterion,

$$J = \lim_{T\to\infty} \frac{1}{T} \mathop{\mathrm{E}}_{v,w} \left\{ \int_0^T \left(x^T(t) Q_1 x(t) + u^T(t) Q_2 u(t) \right) dt \right\} \tag{3}$$

The cost J can be interpreted as a weighted sum of the stationary variance of the plant state and the control signal. The constant matrices Q_1 and Q_2 are weights that are chosen for each plant. A large value of J indicates large deviations from the desired state or large control signals, and is thus worse than a lower value of J. An infinite cost implies that the control system is unstable.

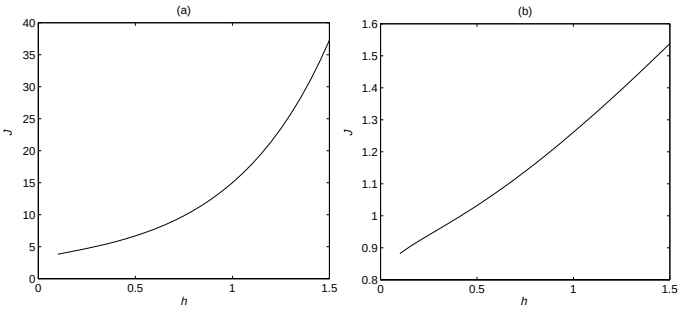

Figure 2 Cost functions for two different plants, (a) Inverted pendulum, $G(s) = \frac{1}{s^2-1}$, (b) DC servo, $G(s) = \frac{1}{s(s+1)}$.

Evaluation of the cost (3) for a given plant, a given controller, and a given sampling period h is a straight-forward task that involves three computational steps. First, the plant description and the cost function are discretized. Second, the closed-loop system is formulated and the stationary variance of the states in the system is computed. Third, the cost J is obtained as a linear weighting of the state variance. More details about the sampling of a plant and a cost function can be found in [Åström and Wittenmark, 1997].

3.3 LQG-Control

In the examples in this paper we use LQG (linear-quadratic-Gaussian) controllers. An LQG-controller is an optimal controller, explicitly designed to minimize the criterion (3). The optimal control parameters depend on the sampling period, and they are obtained using direct discrete design. Standard control design software (e.g. MATLAB) is used to compute the LQG-controller parameters.

3.4 Cost Functions

A controller can normally give satisfactory performance within a range of sampling periods. Computing the cost J for such a range we obtain a *cost function*. The cost function $J(h)$ will have different shapes for different plants and different controllers, but in general it is an increasing function. In Figure 2, LQG-cost functions have been calculated for two different plants, an inverted pendulum and a DC servo. For small sampling periods, the cost increases linearly in both cases.

It should be pointed out that cost functions are not necessarily increasing functions. If the controller has fixed parameters and is designed for a nominal sampling period, the controller might perform worse with both shorter and longer sampling periods. Cost functions are not necessarily convex functions either. An example of a very irregular cost function is given in [Eker et al., 2000].

3.5 Performance Optimization

The feedback scheduler should control the workload of the processor by adjusting the sampling periods of the controllers. At the same time, it should optimize the overall control performance. This is formulated as an optimization problem. Given n control tasks with average execution times $\mathbf{C} = [\, C_1 \quad \dots \quad C_n \,]^T$ and sampling periods $\mathbf{h} = [\, h_1 \quad \dots \quad h_n \,]^T$, the feedback scheduler should solve the problem

$$\begin{aligned} \min_{\mathbf{h}} J &= \sum_{i=1}^{n} J_i(h_i) \\ \text{subject to} \quad & \sum_{i=1}^{n} C_i/h_i \le U_{sp} \end{aligned} \tag{4}$$

where U_{sp} is the desired processor utilization level. This problem has nonlinear constraints. To get linear constraints, the costs are recast as functions of the sampling frequencies, $\mathbf{f} = [\, f_1 \quad \dots \quad f_n \,]^T$,

$$V_i(f) = J_i(1/h) \tag{5}$$

The problem is now written

$$\begin{aligned} \min_{\mathbf{f}} V &= \sum_{i=1}^{n} V_i(f_i) \\ \text{subject to} \quad & \mathbf{C}^T\mathbf{f} \le U_{sp} \end{aligned} \tag{6}$$

If the functions $\mathbf{V}(\mathbf{f}) = [\, V_1(f_1) \quad \dots \quad V_n(f_n) \,]^T$ are decreasing and convex, the optimal solution $\mathbf{f} = [\, \bar{f}_1 \quad \dots \quad \bar{f}_n \,]^T$ fulfills the Kuhn-Tucker conditions

$$\begin{aligned} \mathbf{V_f}(\mathbf{f}) + \lambda\mathbf{C} &= 0 \\ \lambda(U_{sp} - \mathbf{C}^T\mathbf{f}) &= 0 \\ \lambda &\ge 0 \end{aligned} \tag{7}$$

where $\mathbf{V_f}$ is the gradient and λ is the Lagrange multiplier.

Solving the optimization problem exactly can be very time-consuming, especially if the cost functions $V_i(f)$ are non-convex. Just evaluating a cost function in a single point involves a large amount of computations. If the resource allocation problem is to be solved by an on-line optimizer, the cost functions for the plants must be computed off-line and then approximated by simpler functions. A quadratic approximation was suggested in [Eker et al., 2000]. Here, we also present a linear approximation. The solution to the simplified optimization problem can in both cases be interpreted as a simple linear rescaling of a set of nominal sampling periods.

Quadratic Approximation Assume that the cost functions can be approximated by

$$J_i(h) = \alpha_i + \beta_i h^2 \tag{8}$$

or, equivalently,

$$V_i(f) = \alpha_i + \beta_i (1/f)^2 \tag{9}$$

Applying the Kuhn-Tucker conditions (7) yields the explicit solution

$$f_i = \left(\frac{\beta_i}{C_i}\right)^{1/3} \frac{U_{sp}}{\sum_{j=1}^{n} C_j^{2/3} \beta_j^{1/3}} \tag{10}$$

Notice that the constants α_i can be disregarded, i.e., it is sufficient to estimate the curvature of the cost functions.

Linear Approximation Assume that the cost functions can be approximated by

$$J_i(h) = \alpha_i + \gamma_i h \tag{11}$$

or, equivalently,

$$V_i(f) = \alpha_i + \gamma_i / f \tag{12}$$

This often seems to be a better approximation than (9), especially for open-loop stable plants, or any plants sampled reasonably fast. Applying the Kuhn-Tucker conditions (7) yields the explicit solution

$$f_i = \left(\frac{\gamma_i}{C_i}\right)^{1/2} \frac{U_{sp}}{\sum_{j=1}^{n} (C_j \gamma_j)^{1/2}} \tag{13}$$

Notice that the constants α_i can be disregarded, i.e., it is sufficient to estimate the slope of the cost functions.

Interpretation as Simple Rescaling Both the quadratic and the linear cost function approximations yield quite simple, explicit formulas for optimal task frequency assignment, that could be used on-line in a feedback scheduler. If a new task arrives, or if the execution time of a controller suddenly changes, new sampling periods could be calculated using (10) or (13).

However, not even that amount of calculations is really needed. In both cases, it can be noted that each task receives a share of the CPU that is proportional to a task constant. In the quadratic case, the proportionality constant is $(\beta_i/C_i)^{1/3}$, and in the linear case it is $(\gamma_i/C_i)^{1/2}$. The ratios between the optimal sampling periods are thus constant and do not depend on the available resources or the number of current tasks in the systems. This implies that, if the nomial sampling periods have been chosen wisely, optimal feedback scheduling can be performed by simple rescaling of the task periods. This is formulated in the following theorem:

THEOREM 1
If the cost functions of the current tasks in the system can be described by either a) quadratic functions of the sampling period, Eq. (8), or by b) linear functions of the sampling period, Eq. (11), and if nominal sampling frequencies $\mathbf{f_0} = [\, f_{01} \quad \dots \quad f_{0n} \,]$ are chosen in proportion to a) $(\beta_i/C_i)^{1/3}$ or b) $(\gamma_i/C_i)^{1/2}$, then simple rescaling of $\mathbf{f_0}$ to meet the utilization constraint is optimal with respect to the total control performance J.

Proof: Follows from the proportionality argument above. □

Additional Constraints It is possible to add more constraints to the approximate optimization problem and still retain a simple solution.

First, one can let the nominal sampling periods $\mathbf{f_0}$ be minimal sampling periods. If $\mathbf{C}^T\mathbf{f_0} \leq U_{sp}$, then the nominal periods are used, otherwise they are rescaled. This constraint prevents the CPU from being fully loaded when it is not necessary from a control performance point of view.

Second, one can impose maximum sampling periods to some tasks. This leads to an iterative solution (linear programming), where the remaining tasks are rescaled until all constraints are met.

4. A Feedback-Feedforward Scheduling Architecture

To implement the feedback scheduling principle given in the previous section, a feedback-feedforward scheduling architecture is developed. It is assumed that the control tasks can switch between different modes, with possibly very different execution-time demands. Hybrid controllers that switch between different control algorithms depending on the state of the process, external signals, etc., are a large class of controllers that exhibit such behavior. The execution times of the algorithms in the different modes are not known exactly, but must be estimated. The cost functions may also change between the different modes.

The structure of the feedback scheduler is shown in Figure 3. A set of control tasks generate jobs that are fed to a run-time dispatcher. The scheduler receives feedback information about the actual execution time, c_i, of the jobs. It also receives feedforward information from control tasks when they switch mode. In this way, the scheduler can pro-act rather than react to sudden changes in the workload. Also, the scheduler may change the parameters in the optimization routine according to the current modes of the controllers. The scheduler attempts to keep the CPU utilization, U, as close as possible to a utiliza-

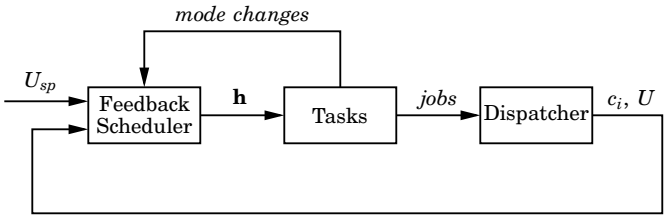

Figure 3 Block diagram of the feedback-feedforward scheduling structure.

tion set-point, U_{sp}. This is done by manipulating the sampling periods, **h**.

4.1 Design Considerations

The feedback scheduler can be implemented in one of two main ways. The first option is to design it as an inherent mechanism, alongside the task dispatcher, in the real-time operating system. The other way, which we discuss here, is to implement it as an application task, executing in parallel with the control tasks. The advantage of this approach is that it could be implemented on top of existing real-time operating systems.

A number of other additional design considerations exist:

- The utilization set-point, U_{sp}, must be chosen. The choice will depend on the scheduling policy of the dispatcher and on the sensitivity of the controllers to missed deadlines. A too low set-point will give low resource utilization and poor control performance. A too high set-point, on the other hand, may cause tasks to suffer from temporary overruns. Notice that the well-known, guaranteed utilization bounds of 100 % for deadline-driven scheduling and 69 % for priority-driven scheduling [Liu and Layland, 1973] are not valid in this context, since the assumptions about known, fixed WCETs and fixed periods are violated. Also, recent research, e.g. [Palopoli et al., 2000], shows that operation close to or even above the level $U = 1$ may give very good control performance in some cases, despite a large number of missed deadlines.
- The feedback scheduler will execute as a periodic task in the system, and its period, h_{FBS}, must be chosen. A short period will give good control of the utilization but also consume much of the available resources. A longer period will consume less resources, but make the scheduler respond slower to load disturbances.
- If the controller parameters depend on the sampling period, the controller must be aware of its current period and adjust its parameters accordingly. On-line recalculations are often too costly, so parameters for a range of sampling periods must be calculated off-line and stored in a table.

4.2 Controlling the Utilization

The feedback scheduler controls the processor utilization by assigning task periods that optimize the overall control performance. However, this requires that estimates of the execution times of the tasks, $\hat{C}_i$, are available. It is assumed that the real-time operating system can monitor the execution-time of individual jobs. An estimate $\hat{C}_i$ is obtained from filtered job execution-time measurements,

$$\hat{C}_i(k) = \lambda \hat{C}_i(k-1) + (1-\lambda)c_i \tag{14}$$

where λ is a forgetting factor. Setting λ close to 1 results in a smooth, but slow estimate. A λ close to 0 gives a faster estimate, but it will also capture more of the high-frequency execution-time noise. This is a typical trade-off in control design.

As detailed in Section 3.5, near-optimal task periods can be calculated by rescaling of a set of nominal sampling periods. First, the estimated nominal requested utilization is computed by

$$\hat{U}_0 = \sum_{i=1}^{n} \frac{\hat{C}_i}{h_{0i}} \tag{15}$$

Here, h_{0i} can be a function of the mode of the controller, if the different modes should have different cost functions. Second, new task periods are calculated by the rescaling

$$h_i = h_{0i} \frac{U_{sp}}{\hat{U}_0} \tag{16}$$

Additional constraints (minimum and maximum sampling periods) can also be dealt with, see Section 3.5.

4.3 Feedforward Information

The role of the feedforward information from the controllers to the feedback scheduler is three-fold. First, the scheduler can react to sudden changes in the workload by executing an extra time in connection with a mode change. The controller, which typically executes more frequently than the feedback scheduler, is responsible for signaling the scheduler as soon as a mode switch condition has been detected. If the mode change is likely to increase the workload, and if the switching time itself is not critical, the controller could delay the switch one or several sampling periods while the scheduler recalculates the periods.

Such a non-critical mode switch could for instance be the result of an operator entering a new setpoint for the controller.

Second, the scheduler needs to keep track of the modes of the controllers in order to compute optimal task periods according to the cost functions. The controller can communicate the new nominal sampling period at the mode switch, or the scheduler can keep a table of the nominal sampling periods for all the controller modes.

Third, the feedforward information allows the scheduler to run separate execution-time estimators in the different modes. The forgetting factor λ can then be chosen according to the execution-time variability within each mode. At a mode change, the scheduler can immediately switch to the current estimate in the new mode. This further improves the regulation of the utilization at the mode changes.

5. A Feedback Scheduling Example

As an example of feedback scheduling, we study the problem of simultaneously stabilizing four inverted pendulums. The pendulum is a common benchmark process because of its open-loop instability, which makes it very sensitive to disturbances, including timing faults. Different scheduling approaches are evaluated by co-simulation of the scheduler, the control tasks, and the pendulums. By simulating the execution of the tasks, the effects of control delay and jitter (due to varying execution times and scheduling) on the control performance are also captured in the results.

5.1 Plants and Controllers

Each pendulum is described by the system (see Section 3.1)

$$\begin{aligned} \frac{dx(t)}{dt} &= \begin{bmatrix} 0 & 1 \\ \omega_0^2 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \omega_0^2 \end{bmatrix} u(t) + v(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + e(t) \end{aligned} \tag{17}$$

where ω_0 is the natural frequency of the pendulum, and the noise variances are given as $\mathbf{E}v^2 = 1/\omega_0$ and $\mathbf{E}e^2 = 10^{-4}$. The four pendulums have different lengths, which correspond to different frequencies: $\boldsymbol{\omega_0} = [\,10 \quad 13.3 \quad 16.6 \quad 20\,]$.

LQG-controllers for the plants are calculated according to the design weights

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad Q_2 = 1 \tag{18}$$

This corresponds to minimization of the cost criterion

$$J = \lim_{T\to\infty} \frac{1}{T}\mathbf{E}\left\{ \int_0^T (y^2(t) + u^2(t))dt \right\} \tag{19}$$

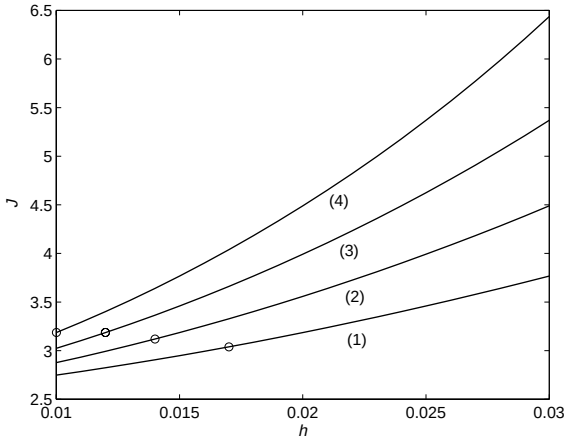

Figure 4 The cost functions for the four inverted pendulums, (1)–(4). The circles indicate the nominal sampling periods.

i.e., the controller should attempt to minimize the sum of the variance of the output signal and the control signal.

The costs functions for the four pendulums shown in Figure 4. It is seen that the cost functions can be reasonably well approximated by linear functions (see Section 3.5),

$$J_i(h) = \alpha_i + \gamma_i h. \tag{20}$$

The estimated slopes are

$$\gamma = [\,43 \quad 67 \quad 95 \quad 127\,]. \tag{21}$$

It can be noted that, the higher the natural frequency of the pendulum is, the more sensitive the controller is towards an increase in the sampling period. This is quite intuitive.

Nominal sampling periods are chosen according to the solution of the performance optimization problem and according to the rule of thumb [Åström and Wittenmark, 1997] that states that the sampling period should be chosen such that $0.2 < \omega_0 h < 0.6$. The resulting nominal periods are

$$\mathbf{h_0} = [\,17 \quad 14 \quad 12 \quad 10\,]\,\text{ms} \tag{22}$$

These periods have been indicated in Figure 4. The optimal costs associated with these sampling periods are

$$\mathbf{J_0} = [\,3.04 \quad 3.12 \quad 3.19 \quad 3.19\,] \tag{23}$$

These are the expected costs if the controllers could really execute at their nominal sampling periods, with zero delay and zero jitter. Implemented in a computer, the controllers will suffer from various amounts of sampling jitter, input-output delay, and output jitter, and the resulting cost will be higher.

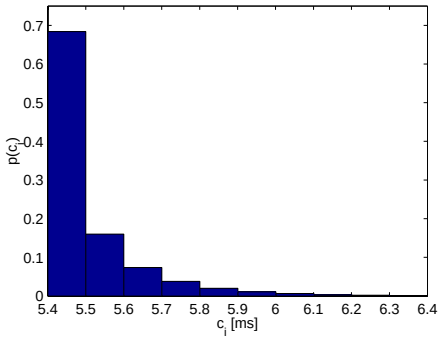

Figure 5 Probability distribution of the control task execution time.

To allow for fast changes between different sampling periods during run-time, the controller parameters Φ^c, Γ^c, C^c, and D^c are calculated off-line for a range of different sampling periods for each pendulum controller and stored in a table.

5.2 The Experiments

Each of the controllers has two different modes: on and off. When on, the average task execution time is $C = 5.5$ ms. The controller code consists of three parts. First, A-D conversion is requested, i.e. the output of the plant is sampled. Second, a new control signal is computed according to the control algorithm. Third, D-A conversion is requested, i.e. the new control signal is sent out to the plant. The total execution time of the task is assumed to vary according to the probability distribution shown in Figure 5. When the controller is turned off, the execution time is zero.

At the start of the experiment, $t = 0$, Controllers 1 and 2 are on, while Controllers 3 and 4 are off. At $t = 2$, Controller 3 switches on, and at $t = 4$, Controller 4 also switches on. The four controllers run in parallel until $t = 6$.

It is initially assumed that the feedback scheduler and the tasks are implemented in a priority-preemptive real-time system. The feedback scheduler is given the highest priority while the control tasks are assigned rate-monotonic priorities. The reason for studying the priority-preemptive setting is clarity—under rate-monotonic scheduling it is easy to predict the effects of overloads. In this case, Controller 1 will be given the lowest priority and will thus suffer the most during an overload.

It is assumed that the execution time of the feedback scheduling task is $C_{FBS} = 2$ ms. Its period is chosen as $h_{FBS} = 200$ ms and the utilization set-point is chosen as $U_{sp} = 0.85$ to yield good control performance and not too many missed deadlines. The execution-time estimation forgetting factor is chosen as $\lambda = 0.99$. This gives smooth estimates but will cause the scheduler to react slowly to mode switches if the feedforward action is not used.

The experiment is repeated for different scheduling approaches. First, open-loop scheduling is attempted. Then, feedback scheduling without the feedforward mechanism is tried. Then, feedback-feedforward scheduling is studied. In the end, open-loop EDF scheduling is also studied.

It is important to specify the behavior of the periodic tasks in the case of missed deadlines. No matter when a task finishes, the next release time is set to be the current release time plus the assigned period of the task. Thus, a task which has missed many deadlines may have a release time which is far back in time compared to the actual starting time of the task. This kind of implementation of periodic tasks penalizes especially the low-priority tasks in the case of overload under fixed-priority dispatching.

To measure the performance of a controller, the accumulated cost is recorded,

$$J_i(t) = \int_0^t (y^2(\tau) + u^2(\tau))d\tau. \tag{24}$$

Compared to the criterion in Eq. (19), this quantity is not rescaled by the time horizon. If the pendulum falls down, the accumulated cost is set to infinity. The pendulums are subjected to identical sequences of process noise and measurement noise in all simulations. The execution times also consist of identical random sequences in all cases.

During each experiment, the schedule, i.e. the execution trace, is also recorded, together with an estimate of the current requested utilization. This quantity is computed as

$$U_{req} = \sum_{i=1}^{n} \frac{c_i}{h_i}, \tag{25}$$

where c_i is the execution time of the latest invocation of task i and h_i is the sampling period currently assigned to task i. Proper regulation of the utilization should keep this quantity approximately equal to or less than U_{sp}.

5.3 The Simulation Environment

The complete set-up, including the scheduler, the tasks, and the pendulums are simulated using the Matlab/Simulink-based tool described in [Eker and Cervin, 1999]. The top level view of the simulation model is shown in Figure 6. Here, the performance of the pendulums can be studied. Opening the Computer block, details about the execution may be studied, see Figure 7.

The Real-Time Kernel block simulates a tick-based, preemptive kernel with an arbitrary, user-defined

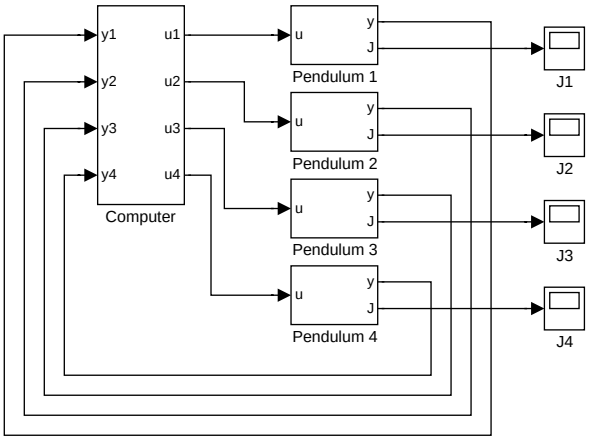

Figure 6 Top-level of the simulation model of the feedback scheduling system.

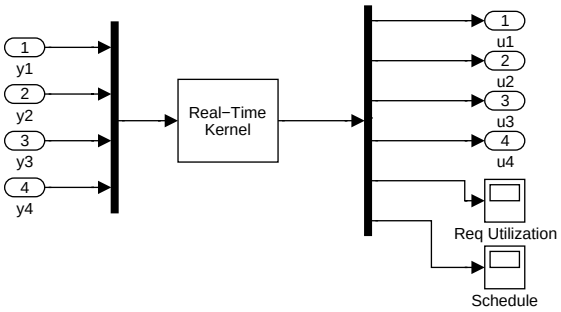

Figure 7 Opening up the Computer block in Figure 6, details about the execution may be studied.

dispatch policy. In the experiments, the tick-size is set to 0.5 ms. The controllers and the feedback scheduler are implemented as functions that the kernel executes repeatedly during a simulation. The scheduler and the controllers communicate information using message-passing primitives in the simulated kernel.

5.4 Simulation Results

The simulation results in the four different scheduling cases are presented and discussed below.

Open-Loop Scheduling Under open-loop scheduling, the controllers attempt to execute at their nominal sampling periods. The accumulated costs for the pendulums are shown in Figure 8, the requested processor utilization is shown in Figure 9, and a close-up of schedule at $t = 4$ is shown in Figure 10.

From time $t = 0$, Tasks 1 and 2 consume (in average) $C/h_{01} + C/h_{02} = 0.72$ CPU and control performance is good. The accumulated costs are expected to increase by about 3 per second according to the optimal costs in (23), and this holds in practice as well.

At $t = 2$, Task 3 starts to execute. Together with Task 2, the two tasks consume $C/h_{01} + C/h_{02} = 0.85$ CPU, which leaves only 0.15 CPU to Task 1, which is the lowest-priority task. The resulting average period is $h_1 = C/0.15 = 37$ ms, which is actually sufficient to stabilize the pendulum, although the cost increases more rapidly.

At $t = 4$, Task 4 is turned on. Together with Task 3, the two tasks request $C/h_{03} + C/h_{04} = 1.01$ CPU, and this blocks Tasks 1 and 2 completely. The result is that both pendulums fall down.

Feedback Scheduling Under feedback scheduling, the tasks starts to execute at their nominal sampling periods when turned on. The feedback scheduler then adjusts the periods every 200 ms. The accumulated costs for the pendulums are shown in Figure 11, the requested processor utilization is shown in Figure 12, and a close-up of schedule at $t = 4$ is shown in Figure 13.

At $t = 2$, Task 3 is turned on and the requested utilization raises above the utilization setpoint of 0.85. At $t = 2.1$, the feedback scheduler adjusts the periods, and the overload condition is eventually removed. The same thing is repeated at $t = 4$ when Task 4 is turned on. The transient overload causes the performance of Controller 1 to degrade, but the situation is soon stabilized. The slow response is due to the forgetting factor $\lambda = 0.99$. A lower value would correct the overload situation faster, but it would also increase the variance of the utilization in stationarity.

Feedback-Feedforward Scheduling Under feedback-feedforward scheduling, tasks that switch modes also immediately activate the feedback scheduler. The accumulated costs for the pendulums are shown in Figure 14, the requested processor utilization is shown in Figure 15, and a close-up of schedule at $t = 4$ is shown in Figure 16.

The results are similar to those under feedback scheduling, except at the mode changes. Here, the overloads are avoided thanks to immediate rescaling of the task periods at $t = 2$ and $t = 4$. The transients are avoided, and the performance of all the controllers is good throughout.

Open-Loop EDF Scheduling Under open-loop EDF scheduling, all tasks attempt to execute at their nominal periods. The accumulated costs for the pendulums are shown in Figure 17 and a close-up of schedule at $t = 4$ is shown in Figure 18. The requested processor utilization is identical to the one under open-loop rate-monotonic scheduling, shown in Figure 9.

Strategy	J_1	J_2	J_3	J_4	$\sum J_i$
Ideal (computed expected value)	18	19	13	6	56
Open-loop scheduling	∞	∞	13	5	∞
Feedback scheduling	32	22	16	7	77
Feedback-feedforward scheduling	23	22	16	7	68
Open-loop EDF scheduling	23	25	19	∞	∞

Table 1 Final accumulated costs for the four pendulums under different scheduling strategies. The symbol ∞ indicates that the pendulum has fallen down.

Although the system is overloaded from $t = 2$, the performance of Controllers 1–3 is good throughout. The reason is that ordinary EDF scheduling acts as a natural period rescaling mechanism in overload situations. This property is discussed further in Section 6.

Task 4 experiences some problems, however. When it is released at $t = 4$, the system has been overloaded for two seconds. This means that the absolute deadlines of Tasks 1–3 lie somewhere backwards in time, and they will have priority over Task 4, which initially has the absolute deadline $4 + h_{04} = 4.010$. The result is that Task 4 is blocked until around $t = 4.35$, before which the pendulum has fallen down.

Summary of Results and Discussion The final accumulated costs for the four pendulums in the different cases are summarized in Table 1.

Feedback-feedforward scheduling gives the best overall control performance, although open-loop EDF scheduling does a quite good job at scheduling Tasks 1–3. The reason that the control performance is slightly better under feedback-feedforward scheduling than under open-loop EDF scheduling is that, in the EDF case, the controllers do not adjust their parameters according to the actual sampling periods.

Also indicated in the table are the ideal expected accumulated costs of the controllers. They are trivially computed from knowing the optimal costs (23) and the duration that each controller is running.

The true expected cost, in stationarity, of the controller in the real-time system can also, with some restrictions, be numerically calculated using analysis techniques from the theory of control systems with random delays [Nilsson, 1998]. The transient behavior of the controllers at the mode changes is much harder to analyze, and there we must rely on simulations.

6. EDF as a Feedback Scheduling Mechanism?

As seen in the simulations in the previous section, the performance of the controllers under open-loop EDF scheduling was quite good, despite the system being permanently overloaded and all deadlines being missed. This can be explained by the following theorem, which to the best of the authors' knowledge has not been formulated before:

THEOREM 2
Assume a set of n periodic tasks, where each task i is described by a fixed period, T_i, a fixed execution time, C_i, a relative deadline, D_i, and a release offset, O_i. The jobs of the tasks are scheduled according to their absolute deadlines (i.e., EDF scheduling). If $U = \sum_{j=1}^{n} C_j/T_j > 1$, then the average actual resulting task period of task i in stationarity, $\bar{T}_i$, will be given by $\bar{T}_i = T_i U$.

Proof: See appendix A. □

Theorem 1 and Theorem 2 taken together gives the following:

COROLLARY 1
An ordinary EDF scheduler can be interpreted, in stationarity, as an optimal feedback scheduler for control tasks that have cost functions that can be described by quadratic or linear functions of the sampling period. □

However, this result rests upon several assumptions, and there are pitfalls. First, it is assumed that the controller samples the plant when it starts to execute, and not when it is released. This way, the control delay will be bounded, even though the task latency (the finish time minus the release time) might approach infinity.

Second, it is assumed that jitter has only negligible impact on the control performance. This may not be true during a permanent overload situation where the tasks start to execute non-preemptively. While the average period of a task is given by Theorem 2, the jitter may be unbounded because of the non-preemptive execution pattern. In the feedback scheduling example in the previous section this was not a problem, since all the tasks had execution times and periods of the same magnitude.

As seen in the example, problems may occur when tasks switch mode (and this is when feedback scheduling is really needed). Since tasks are scheduled using old deadlines, it will take time for a resource redistribution to have effect. One solution would be to reset the release time of all tasks to the current time immediately following a mode change.

Another problem with the open-loop EDF approach is that the period information is not communicated to the controllers. Thus, they cannot use the correct control parameters, and this degrades the performance to some degree. This could be corrected by letting the control tasks measure their own actual periods.

7. Conclusions

A scheduler architecture has been proposed that combines feedback and feedforward action in order to optimize control performance while maintaining high resource utilization. The feedback part relaxes the requirement on known execution-time bounds for multitasking control systems. The feedforward part allows for rapid adaptation to changing load conditions.

The control performance, or Quality-of-Control (QoC), is considered as a quality-of-service measure that should be maximized. Optimal adjustment strategies for the controller task periods have been derived for the cases when the cost function is a quadratic function of the sampling period and when it is a linear function of the sampling period. The adjustment strategy uses linear rescaling, making it computationally efficient, and hence, possible to use on-line.

The different strategies have been evaluated on a realistic simulation example. The proposed approach gives substantially better results than what is achieved using classical open-loop scheduling methods. A new result for periodic tasks with EDF scheduling under overload conditions makes it possible to, in certain situations, to interpret a plain EDF dispatcher as a feedback scheduler for control tasks.

Acknowledgments

This work has been funded by ARTES (the Swedish real-time systems research initiative), by LUCAS (the Lund University center for applied software research), and by TFR.

References

Abdelzaher, T., E. Atkins, and K. Shin. 1997. QoS negotiation in real-time systems, and its application to flight control. In *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium*, pp. 228–238.

Abdelzaher, T. and K. Shin. 1999. QoS provisioning with qContracts in web and multimedia servers. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pp. 44–53.

Abeni, L. and G. Buttazzo. 1999. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, pp. 70–77.

Årzén, K.-E., B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha. 1999. Integrated control and scheduling. Technical Report TFRT-7586, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Åström, K. J. and B. Wittenmark. 1997. *Computer-Controlled Systems*, 3rd edn. Prentice Hall.

Blevins, P. and C. Ramamoorthy. 1976. Aspects of a dynamically adaptive operating system. *IEEE Transactions on Computers* 25(7): 713–725.

Buttazzo, G., G. Lipari, and L. Abeni. 1998. Elastic task model for adaptive rate control. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 286–295.

Cervin, A. and J. Eker. 2000. Feedback scheduling of contol tasks. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 4871–4876.

Eker, J. and A. Cervin. 1999. A Matlab toolbox for real-time and control systems co-design. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, pp. 320–327.

Eker, J., P. Hagander, and K. Erik Årzén. 2000. A feedback scheduler for real-time control tasks. *Control Engineering Practice* 8(12): 1369–1378.

Kleinrock, L.. 1970. A continuum of time-sharing scheduling algorithms. In *AFIPS Conference Proceedings, Spring Joint Computer Conference*, pp. 453–458.

Kosugi, N., K. Takashio, and M. Tokoro. 1994. Modification and adjustment of real-time tasks with rate monotonic scheduling algorithm. In *Proceedings of the 2nd Workshop on Parallel and Distributed Systems*, pp. 98–103.

Kuo, T.-W. and A. Mok. 1991. Load adjustment in adaptive real-time systems. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pp. 160–170.

Lee, C., R. Rajkumar, and C. Mercer. 1996. Experiences with processor reservation and dynamic QoS in Real-Time Mach. In *Proceedings of Multimedia Japan 96*.

Li, B. and K. Nahrstedt. 1998. A control theoretical model for quality of service adaptations. In *Proceedings of 6th International Workshop on Quality of Service*, pp. 145–153.

Liu, C. L. and J. W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20(1): 40–61.

Lu, C., J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley. 2000. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pp. 13–23.

Lu, C., J. Stankovic, G. Tao, and S. H. Son. 1999. Design and evaluation of a feedback control EDF scheduling algorithm. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pp. 56–67.

Nakajima, T.. 1998. Resource reservation for adaptive QoS mapping in Real-Time Mach. In *Proceedings of the 6th International Workshop on Parallel and Distributed Real-Time Systems*, pp. 1047–1056.

Nakajima, T. and H. Tezuka. 1994. A continuous media application supporting dynamic QoS control on Real-Time Mach. In *Proceedings of ACM Multimedia'94*, pp. 289–297.

Nilsson, J.. 1998. Real-time control systems with delays. Ph.D. Thesis TFRT-1049, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Palopoli, L., L. Abeni, and G. Buttazzo. 2000. Real-time control system analysis: An integrated approach. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pp. 131–140.

Potier, D., E. Gelenbe, and J. Lenfant. 1976. Adaptive allocation of central processing unit quanta. *Journal of the ACM* 23(1): 97–102.

Rajkumar, R., C. Lee, J. Lehoczky, and D. Siewiorek. 1997. A resources allocation model for QoS management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pp. 298–307.

Ryu, M., S. Hong, and M. Saksena. 1997. Streamlining real-time controller design: From performance specifications to end-to-end timing constraints. In *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium*, pp. 91–99.

Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin. 1996. On task schedulability in real-time control systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, Washington, DC, pp. 13–21.

Shin, K. and C. Meissner. 1999. Adaptation of control system performance by task reallocation and period modification. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 29–36.

Stankovic, J. A., C. Lu, S. H. Son, and G. Tao. 1999. The case for feedback control real-time scheduling. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.

A. Proof of Theorem 2

The release time of job number k of task i is kT_i+O_i. The deadline of job number k of task i is $kT_i+O_i+D_i$. Let $k_i(t)$ be the number of finished jobs of task i at time t. After a while, CPU is never idle, therefore,

$$I_{idle}(t) + \sum_i (k_i(t)C_i + e_i(t)) = t \tag{26}$$

where $I_{idle}(t)$ is the accumulated idle time, e_i is how long the current invocation of task i has executed. Both $I_{idle}(t)$ and e_i are bounded and $0 \le e_i \le C_i$. Furthermore, due to the overload situation, tasks are finished in the order of their deadlines. Therefore,

$$k_l(t)T_l + D_l + O_l \le (k_i(t)+1)T_i + D_i + O_i \tag{27}$$

(Otherwise task l would not have finished job number $k_l(t)$ before task i finished job number $k_i(t)+1$). Symmetrically,

$$k_i(t)T_i + D_i + O_i \le (k_l(t)+1)T_l + D_l + O_l \tag{28}$$

The two equations above give

$$\begin{aligned} k_l(t)T_l + D_l + O_l - T_i \le k_i(t)T_i + D_i + O_i \\ \le (k_l(t)+1)T_l + D_l + O_l \end{aligned} \tag{29}$$

Hence,

$$\begin{aligned} \frac{k_l(t)T_l + D_l - T_i + O_l}{k_l(t)T_l} \le \frac{k_i(t)T_i + D_i + O_i}{k_l(t)T_l} \\ \le \frac{(k_l(t)+1)T_l + D_l + O_l}{k_l(t)T_l} \end{aligned} \tag{30}$$

Here, the limit of the left-hand side and the right-hand side are both equal to one, so,

$$\lim_{t\to\infty} \frac{k_j(t)T_j}{k_l(t)T_l} = 1 \tag{31}$$

Rearranging the terms in Eq. (26) and letting $t \to \infty$,

we have

$$
\begin{aligned}
1 &= \lim_{t\to\infty} \frac{1}{t} \sum_i k_i(t) C_i \\
&= \lim_{t\to\infty} \frac{1}{t} \sum_i k_i(t) T_i \frac{C_i}{T_i} \\
&= \lim_{t\to\infty} \frac{\sum_j k_j(t) T_j}{t} \sum_i \frac{k_i(t) T_i}{\sum_j k_j(t) T_j} \frac{C_i}{T_i} \\
&= \lim_{t\to\infty} \frac{k_i(t) T_i \sum_j \frac{k_j(t) T_j}{k_i(t) T_i}}{t} \sum_i \frac{1}{\sum_j \frac{k_j(t) T_j}{k_i(t) T_i}} \frac{C_i}{T_i} \\
&= \lim_{t\to\infty} \frac{k_i(t) T_i n}{t} \sum_i \frac{1}{n} \frac{C_i}{T_i} \\
&= \lim_{t\to\infty} \frac{k_i(t) T_i}{t} \sum_i \frac{C_i}{T_i}
\end{aligned}
\tag{32}
$$

Hence,

$$
\bar{T}_i = \lim_{t\to\infty} \frac{t}{k_i(t)} = T_i \sum_j \frac{C_j}{T_j} = T_i U \tag{33}
$$

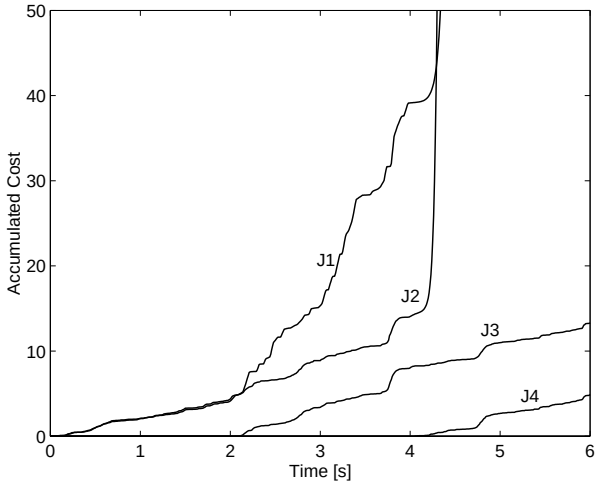

Figure 8 Accumulated costs under open-loop scheduling.

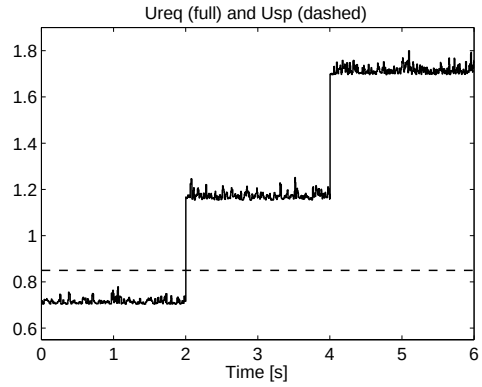

Figure 9 Requested utilization under open-loop scheduling.

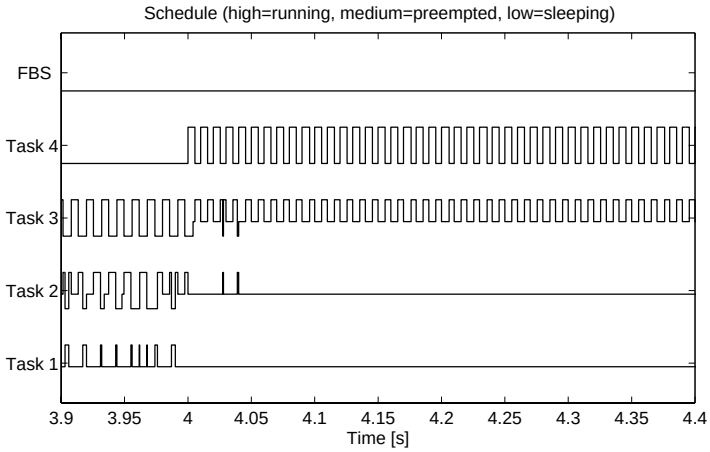

Figure 10 Close-up of schedule at $t = 4$ under open-loop scheduling.

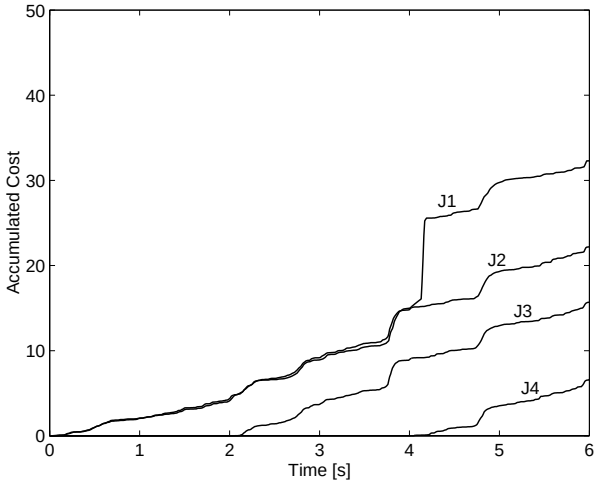

Figure 11 Accumulated costs under feedback scheduling.

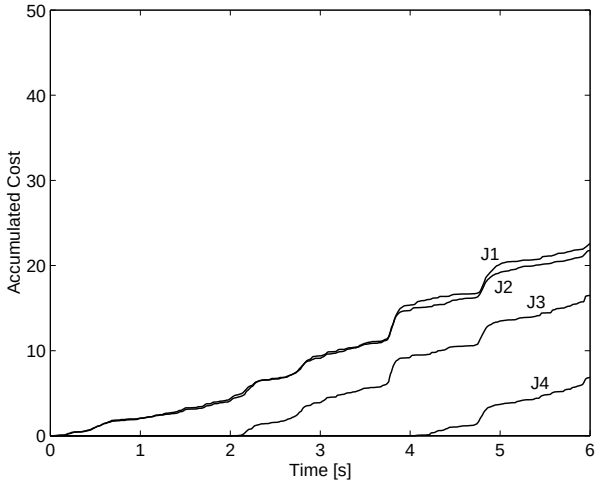

Figure 14 Accumulated costs under feedback-feedforward scheduling.

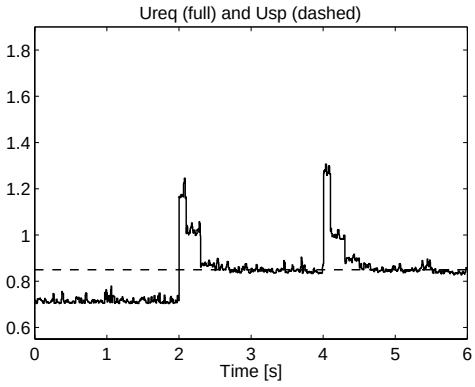

Figure 12 Requested utilization under feedback scheduling.

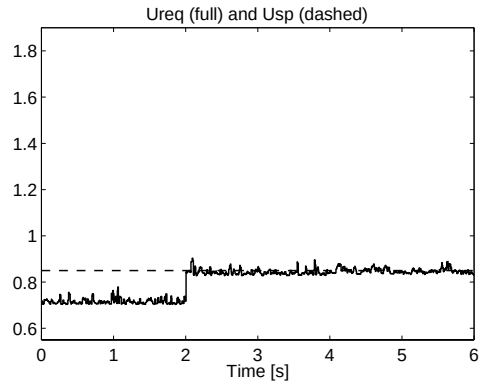

Figure 15 Requested utilization under feedback-feedforward scheduling.

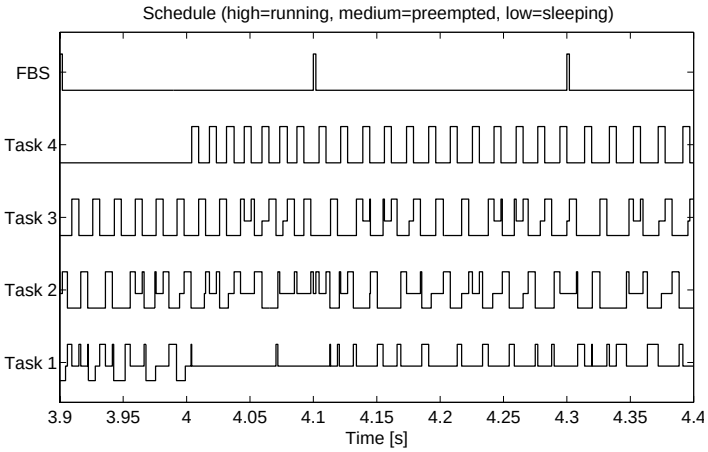

Figure 13 Close-up of schedule at $t = 4$ under feedback scheduling.

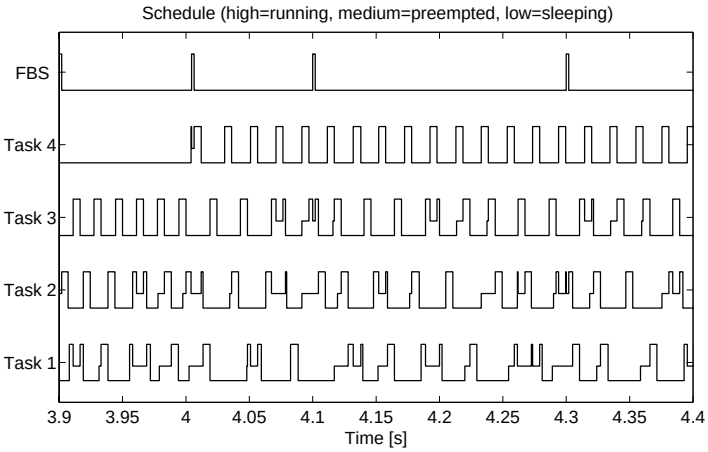

Figure 16 Close-up of schedule at $t = 4$ under feedback-feedforward scheduling.

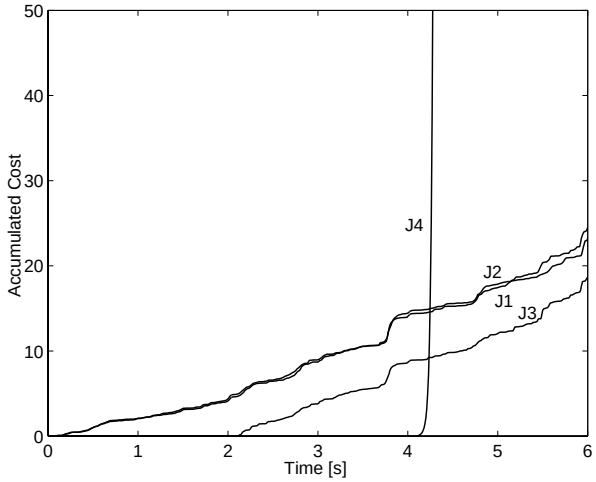

Figure 17 Accumulated costs under open-loop EDF scheduling.

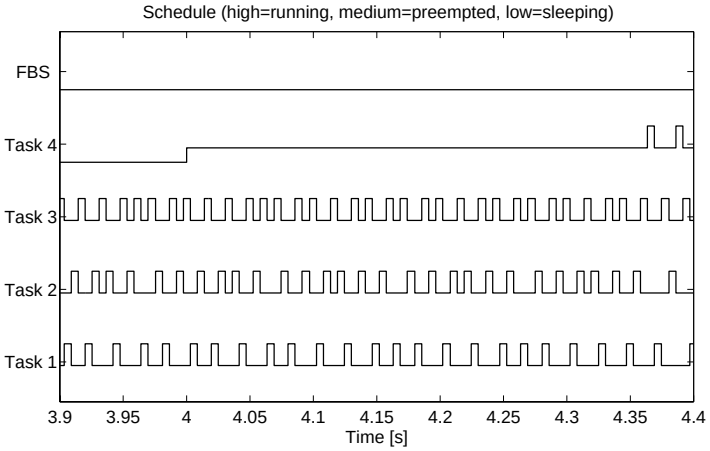

Figure 18 Close-up of schedule at $t = 4$ under open-loop EDF scheduling.