Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing

Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms[*]

Chenyang Lu John A. Stankovic Gang Tao[†] Sang H. Son
Department of Computer Science [†]*Department of Electrical and Computer Engineering*
University of Virginia, Charlottesville, VA 22903
e-mail: {chenyang, stankovic, son}@cs.virginia.edu gt9s@ee.virginia.edu[†]

Abstract

This paper presents a Feedback Control real-time Scheduling (FCS) framework for adaptive real-time systems. An advantage of the FCS framework is its use of feedback control theory (rather than ad hoc solutions) as a scientific underpinning. We apply a control theory based methodology to systematically design FCS algorithms to satisfy the transient and steady state performance specifications of real-time systems. In particular, we establish dynamic models of real-time systems and develop performance analyses of FCS algorithms, which are major challenges and key steps for the design of control theory based adaptive real-time systems. We also present a FCS architecture that allows plug-ins of different real-time scheduling policies and QoS optimization algorithms. Based on our framework, we identify different categories of real-time applications where different FCS algorithms should be applied. Performance evaluation results demonstrate that our analytically tuned FCS algorithms provide robust transient and steady state performance guarantees for periodic and aperiodic tasks even when the task execution times vary by as much as 100% from the initial estimate.

1. Motivation and Introduction

Real-time scheduling algorithms fall into two categories: *static* and *dynamic* scheduling. In static scheduling, the scheduling algorithm has complete knowledge of the task set and its constraints, such as deadlines, computation times, precedence constraints, and future release times. The Rate Monotonic (RM) algorithm and its extensions [15][19] are static scheduling algorithms and represent one major paradigm of real-time scheduling. In dynamic scheduling, however, the scheduling algorithm does not have complete knowledge of the task set or its timing constraints. For example, new task activations, not known to the algorithm when it is scheduling the current task set, may arrive at a future unknown time. Dynamic scheduling can be further divided into two categories: scheduling algorithms that work in *resource sufficient* environments and those that work in *resource insufficient* environments. Resource sufficient environments are systems where the system resources are sufficient to *a priori* guarantee that, even though tasks arrive dynamically, at any given time all the tasks are schedulable. Under certain conditions, Earliest Deadline First (EDF) [19][31] is an optimal dynamic scheduling algorithm in resource sufficient environments. EDF is a second major paradigm for real-time scheduling. While real-time system designers try to design the system with sufficient resources, because of cost and unpredictable environments, it is sometimes impossible to guarantee that the system resources are sufficient. In this case, EDF's performance degrades rapidly in overload situations. The Spring scheduling algorithm [34] can dynamically guarantee incoming tasks via on-line admission control and planning and thus is applicable in resource insufficient environments. Many other algorithms [31] have also been developed to operate in this way. These admission-control-based algorithms represent the third major paradigm for real-time scheduling. However, despite the significant body of results in these three paradigms of real-time scheduling, many real world problems are not easily supported. While algorithms such as EDF, RM and the Spring scheduling algorithm can support sophisticated task set characteristics,

[*] Supported in part by NSF grants CCR-9901706, CCR-0098269, and EIA-9900895, and DARPA grants F33615-01-C-1905 and N00014-01-1-0576. This paper is an extension to previous papers published in the Proceedings of IEEE Real-Time Systems Symposium [22][23].

they are all "open loop" scheduling algorithms. Open loop refers to the fact that once schedules are created they are not "adjusted" based on continuous feedback. While open-loop scheduling algorithms can perform well in predictable environments in which the workloads can be accurately modeled (e.g., traditional process control systems), they can perform poorly in *unpredictable* environments, i.e., systems whose workloads cannot be accurately modeled. For example, systems with open-loop schedulers such as the Spring scheduling algorithm are usually designed based on *worst-case* workload parameters. When accurate system workload models are not available, such an approach can result in a highly underutilized system based on an extremely pessimistic estimation of workload.

In recent years, a new category of soft real-time applications executing in open and unpredictable environments is rapidly growing [32]. Examples include open systems on the Internet such as online trading and e-business servers, and data-driven systems such as smart spaces, agile manufacturing, and defense applications such as C4I. For example, in an e-business server, neither the resource requirements nor the arrival rate of service requests are known *a priori*. However, performance guarantees are required in these applications. Failure to meet performance guarantees may result in loss of customers, financial damage, liability violations, or even mission failures. For these applications, a system design based on open loop scheduling can result in an extremely expensive and underutilized system.

As a cost-effective approach to achieve performance guarantees in unpredictable environments, adaptive scheduling algorithms have been recently developed. While early research on real-time scheduling was concerned with guaranteeing complete avoidance of undesirable effects such as overload and deadline misses, adaptive real-time systems are designed to handle such effects dynamically. There remain many open research questions in adaptive real-time scheduling. In particular, how can a system designer specify the performance requirement of an adaptive real-time system? And how can a designer systematically design a scheduling algorithm to satisfy system performance specifications? The design methodology for automatic adaptive systems has been developed in feedback control theory [12]. However, feedback control theory has been mostly applied in mechanical and electrical systems. In trying to apply feedback control theory to a computer systems domain, the modeling and implementation of adaptive real-time systems face significant research challenges. Some of those challenges are answered in this paper.

Recently, several works have applied control theory to computing systems. For example, several papers [3][9][10][11] presented flexible scheduling techniques to improve digital control system performance. These techniques are tailored to the specific characteristics of digital control systems instead of general adaptive real-time computing systems. Li and Nahrstedt [18] utilized control theory to develop a feedback control loop to guarantee desired network packet rate in a distributed visual tracking system. Hollot et. al. [13] apply control theory to analyze a congestion control algorithm on IP routers. A control-theoretical approach has also been applied to provide QoS guarantees in web servers [2][4][21] and e-mail servers [25]. While these works use control theory analysis on computing systems, they do not directly address timing constraints, which is the key requirement of real-time systems.

Transient state performance of adaptive real-time systems has received special attention in recent years. For example, Brandt et. al. [8] evaluated a dynamic QoS manager by measuring the transient performance of applications in response to QoS adaptations. Rosu et. al. [27] proposed a set of performance metrics to capture the transient responsiveness of adaptations and its impact on applications. The paper proposed metrics that are similar to settling time and steady-state error metrics found in control theory.

However, to the authors' best knowledge, no unified framework exists to date for designing an adaptive real-time computing system from performance specifications of desired dynamic response. In this paper we present *Feedback Control real-time Scheduling* (*FCS*), an analytical framework that maps QoS control in adaptive real-time systems to feedback control theory. Our control theoretical framework includes the following elements:

- A scheduling architecture that maps the feedback control structure to adaptive resource scheduling in real-time systems,

- A set of performance specifications and metrics to characterize both transient and steady state performance of adaptive real-time systems, and
- A control theory based design methodology for resource scheduling algorithms to satisfy system performance specifications.

In contrast to *ad hoc* approaches that rely on laborious design/tuning/testing iterations, FCS enables system designers to systematically design adaptive real-time systems with established analytical methods to achieve desired performance guarantees in unpredictable environments.

The feedback control real-time scheduling architecture is described in Section 2. Performance specifications and metrics for adaptive real-time systems are presented in Section 3. The control theory based design methodology is presented in Section 4. An analytical model for generic CPU bound real-time systems is established in Section 5. Based on this model, the design and analysis of a set of FCS algorithms are given in Section 6. Performance evaluation results of these scheduling algorithms are presented in Section 7. Finally, we conclude this paper in Section 8.

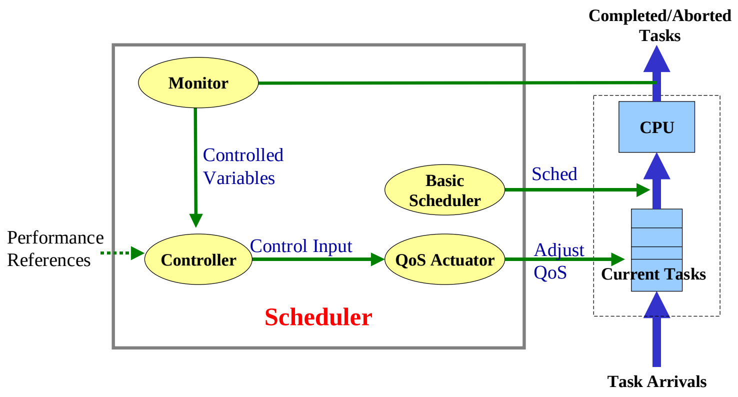


Figure 1: Feedback Control Real-Time Scheduling Architecture

2. Feedback Control Real-Time Scheduling Architecture

Our feedback control real-time scheduling (FCS) architecture (Figure 1) is composed of a feedback control loop composed of a Monitor, a Controller, a QoS Actuator, and a Basic Scheduler. Before we describe the components of the FCS architecture, we define our task model and a set of control related variables.

2.1. Task Model

Each task has several *QoS levels.* In this task model, each task T_i has N QoS levels ($N \geq 2$). Each QoS level j ($0 \leq j \leq N$-1) of T_i is characterized by the following attributes:

$D_i[j]$:	the relative *deadline*
$EE_i[j]$:	the *estimated execution time*
$AE_i[j]$:	the (actual) *execution time* that can vary considerably from instance to instance and is unknown to the scheduler
$V_i[j]$:	the *value* task T_i contributes if it is completed at QoS level j before its deadline $D_i[j]$. The lowest QoS level 0 represents the rejection of the task and $V_i[0] \leq 0$ (when $V_i[0] < 0$, it is called the *rejection penalty* [6]). Every QoS level contributes a value of $V_i[0]$ if it misses its deadline.

For periodic tasks:

$P_i[j]$:	the invocation *period*

$B_i[j]$: the *estimated CPU utilization* $B_i[j] = EE_i[j] / P_i[j]$
$A_i[j]$: the (actual) *CPU utilization* $A_i[j] = AE_i[j] / P_i[j]$

For aperiodic tasks:
$EI_i[j]$: the *estimated inter-arrival-time* between subsequent invocations
$AI_i[j]$: the *average inter-arrival-time* that is unknown to the scheduler
$B_i[j]$: the *estimated CPU utilization* $B_i[j] = EE_i[j] / EI_i[j]$
$A_i[j]$: the (actual) *CPU utilization* $A_i[j] = AE_i[j] / AI_i[j]$

In this model, a higher QoS level of a task has a higher (both estimated and actual) CPU utilization and contributes a higher value if it meets its deadline, i.e., $B_i[j+1] > B_i[j]$, $A_i[j+1] > A_i[j]$, and $V_i[j+1] > V_i[j]$. In the simplest form, each task only has two QoS levels (corresponding to the admission and the rejection of the task, respectively). In many applications including web services [4], multimedia [8], embedded digital control [11], and systems that support imprecise computation [20] or flexible security [30], each task has more than two QoS levels and the scheduler can trade-off the CPU utilization of a task with the value it contributes to the system at a finer granularity. The QoS levels can differ in term of execution time and/or period/inter-arrival-time. For example, a web server can dynamically change the execution time of a HTTP session by changing the complexity of the requested web page [4]. For another example, several papers have shown that the deadlines and periods of tasks in embedded digital control systems and multimedia players can be adjusted on-line [8][10]. A key feature of our task model is that it characterizes systems in unpredictable environments where task's *actual* CPU utilization is *time varying* and *unknown* to the scheduler. Such systems are amenable to the use of feedback control loops to dynamically correct the scheduling errors to adapt to load variations at run-time.

2.2. Control Related Variables

An important step in designing the FCS architecture is to decide the following variables of a real-time system in terms of control theory.

- *Controlled variables* are the performance metrics controlled by the scheduler. Controlled variables of a real-time system may include the *deadline miss ratio* $M(k)$ and the *CPU utilization* $U(k)$ (also called miss ratio and utilization, respectively), both defined over a time window ($(k-1)W, kW$), where W is the *sampling period* and k is called the *sampling instant*.
  - The miss ratio $M(k)$ at the k^{th} sampling instant is defined as the number of deadline misses divided by the total number of completed and aborted tasks in a sampling window $((k-1)W, kW)$. Miss ratio is usually the most important performance metric in a real-time system.
  - The utilization $U(k)$ at the k^{th} sampling instant is the percentage of CPU busy time in a sampling window $((k-1)W, kW)$. CPU utilization is regarded as a controlled variable for real-time systems due to cost and throughput considerations. CPU utilization is important because of its direct linkage with the deadline miss ratio (see Section 5).
  - Another controlled variable might be the total value $V(k)$ delivered by the system in the k^{th} sampling period. In the remainder of this paper, we do not directly use the total value as a controlled variable, but rather address the value imparted by tasks via the QoS Actuator (see Figure 1 and Section 7.1)

- *Performance references* represent the desired system performance in terms of the controlled variables, i.e., the desired miss ratio M_S and/or the desired CPU utilization U_S. For example, a particular system may require deadline miss ratio $M_S = 0$ and CPU utilization $U_S = 90\%$. The difference between a performance reference and the current value of the corresponding controlled variable is called an *error*, i.e., the miss ratio error $E_M = M_S - M(k)$ and the utilization error $E_U = U_S - U(k)$.

- *Manipulated variables* are system attributes that can be dynamically changed by the scheduler to affect the values of the controlled variables. In our architecture, the manipulated variable is the *total estimated utilization* $B(k) = \sum_i U_i[l_i(k)]$ of all tasks in the system, where T_i is a task with a QoS level of $l_i(k)$ in the k^{th} sampling window. The rationale for choosing the total estimated utilization as a manipulated variable is that real-time scheduling policies such as EDF and Rate/Deadline Monotonic can guarantee no deadline misses when the CPU is not overloaded, and in normal situations, the miss ratio increases as the system load increases. The other controlled variable, the utilization $U(k)$, also usually increases as the total estimated utilization increases. However, the utilization is often different from the total estimated utilization $B(k)$, which is due to the estimation error of execution times when workload is unpredictable and time varying. Another difference between $U(k)$ and $B(k)$ is that $U(k)$ can never exceed 100% while $B(k)$ does not have this limit.

2.3. Feedback Control Loop

The FCS architecture features a feedback control loop that is invoked at every sampling instant k. It is composed of a Monitor, a Controller, and a QoS Actuator (Figure 1).

1) The *Monitor* measures the controlled variables ($M(k)$ and/or $U(k)$) and feeds the samples back to the Controller.

2) The *Controller* compares the performance references with corresponding controlled variables to get the current errors, and computes a change $D_B(k)$ (called the *control input*) to the total estimated requested utilization based on the errors. The Controller uses a control function to compute the correct manipulated variable value to compensate for the load variations and keep the controlled variables close to the references. The detailed design of the Controller is presented in Section 6.

3) The *QoS Actuator* dynamically changes the total estimated requested utilization at each sampling instant k according to the control input $D(k+1)$ by adjusting the QoS levels of tasks. The goal of the QoS Actuator is to enforce the new total estimated requested utilization $B(k+1) = B(k) + D_B(k)$. Under the utilization constraint of $B(k+1)$, the QoS Actuator calls a QoS optimization algorithm (see Section 7.1) to maximize the system value. In the simplest form, each task has only two QoS levels and the QoS Actuator is essentially an admission controller. In this paper, we assume the system has *arriving-time QoS control*, i.e., the QoS Actuator is also invoked upon the arrival of each task. The arriving-time admission control isolates disturbances caused by variations in task arrival rates (see Section 5). Feedback control scheduling in systems without arriving-time QoS control was previously studied in [21].

2.4. Basic Scheduler

The FCS architecture has a Basic Scheduler that schedules admitted tasks with a scheduling policy (e.g., EDF or Rate/Deadline Monotonic). The properties of the scheduling policy can have significant impact on the design of the feedback control loop. Our FCS architecture permits plugging in different policies for this Basic Scheduler and then designing the entire feedback control scheduling system around this choice.

A key difference between our work and the previous work is that while previous work often assumes the CPU utilization of each task is known *a priori*, we focus on systems in unpredictable environments where tasks' actual CPU utilizations are unknown and time varying. This more challenging problem necessitates the feedback control loop to dynamically correct the scheduling errors at run-time.

3. Performance Specifications and Metrics

We now describe the second element of the FCS framework, the performance specifications and metrics for adaptive real-time systems. Traditional metrics such as the average miss-ratio cannot capture the transient behavior of the system in response to load variations. Recently, a set of metrics [21][27] was proposed to characterize both transient and steady state behavior of an adaptive system. In this section, we extend and map the metrics to dynamic responses of control systems. The performance specifications consist of a set of *performance profiles*[1] in terms of the controlled variables, utilization $U(k)$, and miss ratio $M(k)$. We also present a set of representative *load profile*s adapted from control theory [12].

3.1. Performance Profile

The performance profile characterizes important transient and steady state performance of a real-time system. $M(k)$ and $U(k)$ characterize the system performance in the sampling window $((k\text{-}1)W, kW)$. In contrast, traditional metrics for real-time systems such as average miss-ratio and average utilization are defined based on a much larger time window than the sampling period W. The average metrics are often inadequate metric in characterizing the dynamics of the system performance in response to overload conditions [22]. The performance profile of a real-time system includes the following:

- *Stability*: A real-time system is stable if its miss ratio $M(k)$ and utilization $U(k)$ are always bounded for bounded references. Since both miss ratio and utilization are naturally bounded in the range [0, 100%], stability is a necessary condition to prevent miss ratio and utilization from staying at the undesirable 100% limit.

- *Transient-state response* represents the real-time system's responsiveness and efficiency of QoS adaptation in reacting to changes in run-time conditions.
    - *Overshoot* M_o and U_o: For a real-time system, we define *overshoot* as the maximum amount that the system overshoots its miss ratio or utilization reference divided by its miss ratio or utilization reference, i.e., $M_o = (M_{max} - M_S) / M_S$, $U_o = (U_{max} - U_S) / U_S$, respectively. The maximum miss ratio M_o and utilization U_o in the transient state is called the *absolute overshoot.* Overshoot is important to a real-time system because a high transient miss-ratio or utilization can cause system failure in many systems such as robots and media streaming [8].
    - S*ettling time* T_s*:* The time it takes the system to enter a steady state in response to a load profile. The settling time represents how fast the system can settle down to steady state with desired miss ratio and/or utilization.

- *Steady-state error* E_{SM} and E_{SU}: The difference between the average values of miss ratio $M(k)$ and/or utilization $U(k)$ in steady state and its corresponding reference. The steady state error characterizes how precisely the system can enforce the desired miss ratio and/or utilization in steady state.

- *Sensitivity* S_p: Relative change of a controlled variable in steady state with respect to the relative change of a system parameter p. For example, sensitivity of miss ratio with respect to the task execution time S_{AE} represents how significantly the change in the task execution time affects the system miss-ratio. Sensitivity describes the robustness of the system with regard to workload or system variations.

3.2. Load Profile

According to control theory, the performance profile of an adaptive system may be specified assuming representative *load profiles* including *step load* and *ramp load*. The step load represents the *worst case* of

[1] The performance profile has been called the *miss-ratio profile* in [22]. The performance profile can be generalized to other metrics such as response time, throughput, and value-cognizant metrics.

load variation that overloads the system instantaneously, while the ramp load represents a nominal form of load variation. The load profiles are defined as follows.

- *Step-load SL*(L_n, L_m): a load profile that instantaneously jumps from a nominal load L_n to a higher load $L_m > L_n$ and stays constant after the jump. Instantaneous load change such as the step load is more difficult to handle than gradual load change.

- *Ramp-load RL*(L_n, L_m, T_R): a load profile that increases linearly from the nominal load L_n to a higher load $L_m > L_n$ during a time interval of T_R sec. Compared with the step load, the ramp signal represents a less severe load variation scenario.

One key advantage of using the above load profiles for performance specification is that they are amenable to well-established design and analysis methods in control theory and, therefore, fits well with our control theoretical framework. This means that a system designer can use control theory method to analytically design the system to satisfy a performance profile in response to a load profile as defined above. Specifically, a load profile can be modeled as disturbance signals in the form of a step or ramp signal. Based on control theory, a linear system's dynamic properties can be determined by its dynamic response to a step signal or a ramp load regardless of its parameters including the magnitude of load variation (L_m-L_n) and the ramp duration T_R. If a real-time system can be approximated with a linear model in its operation conditions, its performance profile can be determined by stressing the system with a step load, i.e., the system can achieve satisfactory performance under any combinations of step and ramp load if its performance profile in response to a step load or ramp load satisfies its specifications.

However, if a real-time system is non-linear, the dynamic response of a system in response of any load variations cannot be determined by its response to a single step load or a single ramp load because the system performance depends on the specific parameters of the load profiles. In this case, the performance profiles in response to specific load profiles are only "indications" of the system performance in general. In this case, the load profiles are application-specific based on a set of expected load characteristics and system requirements.

The load profiles are *abstractions* of the workload, and there can be many possible instantiations of the same load profile. The instantiation of a load profile should incorporate the knowledge of the workload, and, therefore, the load profile should be viewed as an enhancement to existing benchmarks. For example, the system load can be interpreted as the total requested CPU utilization in the system where CPU is the bottleneck resource. For another example, the load of an Internet server may be interpreted as the number of concurrent users.

4. Control Theory Based Design Methodology

The third element of our FCS framework is the control theory based design methodology. Based on the scheduling architecture and the performance specifications, a system designer can systematically design an adaptive resource scheduler to satisfy the system's performance specifications with established analytical methods in control theory. This methodology is in contrast to existing *ad hoc* approaches that depend on laborious design/tuning/testing iterations. The control design methodology is as follows.

1) The system designer specifies the desired dynamic behavior with transient and steady state performance metrics. This step maps the performance requirements of an adaptive real-time system to the dynamic response specification of a control system.

2) The system designer establishes a dynamic model of the real-time system for the purposes of performance control. A dynamic model describes the mathematical relationship between the control input and the controlled variables of a system with differential/difference equations or state matrices. Modeling is important because it provides a basis for the analytical design of the

controller. However, modeling has been a major challenge for applying control theory to real-time systems due to the lack of established differential/difference equations to describe real-time systems.

3) Based on the performance specs and system model from step 1) and 2), the system designer applies established mathematical techniques (i.e., the Root Locus method, frequency design, or state based design) of feedback control theory [12] to design FCS algorithms that *analytically* guarantee the specified transient and steady-state behavior at run-time.

Compared with existing *ad hoc* approaches, this analytical design approach significantly reduces design time of adaptive systems. The resultant system's parameters can be easily tuned with existing control theory methods and the resultant system can be proved to satisfy its performance specifications. In contrast, the tuning adaptive systems designed with *ad hoc* methods often depends on repeated testing, guessing, or rule-of-thumb without performance guarantees at run-time.

5. Modeling the Controlled Real-Time System

A key step of using the control theory methodology is to establish an analytical model to approximate the *controlled system* in the FCS architecture.

The controlled system includes the QoS Actuator, the scheduled tasks, the CPU, the Basic Scheduler, and the Monitor. The control input to the controlled system is the change in the total estimated utilization $D_B(k)$. The output of the controlled system includes the controlled variables, miss ratio $M(k)$ and utilization $U(k)$. Although it is difficult to precisely model a nonlinear and time varying system such as a real-time system, we can approximate such a system with a linear model for the purpose of control design because of the robustness of feedback control with regard to system variations. We now derive the model from the control input to the output.

Starting from the control input, the QoS actuator changes the total estimated utilization $B(k+1)$ in the next sampling period according the control input $D_B(k)$ at every sampling instant k:

$$B(k+1) = B(k) + D_B(k) \tag{1}$$

Since the precise execution time of each task is unknown and time varying, the total (actual) requested utilization $A(k)$ may differ from the total estimated requested utilization $B(k)$:

$$A(k) = G_a(k)B(k) \tag{2}$$

where $G_a(k)$, called the *utilization ratio*, is a time-variant variable that represents the extent of workload variation in terms of total requested utilization. For example, $G_a(k) = 2$ means that the actual total requested utilization is twice of the estimated total utilization. Since $G_a(k)$ is time variant, we use the maximum possible value $G_A = \max\{G_a(k)\}$, called the *worst-case utilization ratio*, in control design to guarantee stability in all cases. Hence Equation 2 can be simplified to the following formula for the purpose of control design:

$$A(k) = G_A B(k) \tag{3}$$

The relationship between the total requested utilization $A(k)$ and the controlled variables are *nonlinear* due to *saturation*, i.e., the controlled variables remain constant when the control input $D_B(k) \neq 0$. When the CPU is underutilized ($A(k) \leq 1$), the utilization $U(k)$ is outside its saturation zone and equals $A(k)$; when the CPU is *overloaded*, however, $U(k)$ saturates at 1 because it can never exceed 100%.

Based on Equations 1-3, the analytical model for the utilization output is as following:

$$U(k) = U(k\text{-}1) + G_A D_B(k\text{-}1) \qquad (A(k) \le 1) \tag{4}$$

$$U(k) = 1 \qquad (A(k) > 1) \tag{5}$$

Now we derive the model for the miss ratio output $M(k)$. In contrast with $U(k)$ that saturates in overload conditions, $M(k)$ saturates at 0 when the CPU is *underutilized*, i.e., the total requested utilization is below a *utilization threshold* $A_{th}(k)$. In real-time scheduling theory, *schedulable utilization bounds* have been derived for various real-time scheduling policies under different workload assumptions [19]. A utilization bound A_b is typically defined as a fixed *lower bound* for all possible workloads under certain assumptions, while we define the utilization threshold $A_{th}(k)$ as the time varying *actual* threshold for the system's particular workload in the k^{th} sampling period (and hence $A_b \le A_{th}(k)$).

When $A(k) > A_{th}(k)$, $M(k)$ usually increases nonlinearly with the total requested utilization $A(k)$. The relationship between $M(k)$ and $A(k)$ needs to be linearized by taking the derivative at the vicinity of the performance reference M_S as the miss ratio factor G_m:

$$G_m = \frac{dM(k)}{dA(k)}$$

In practice, the *miss ratio factor* G_m can be estimated experimentally by plotting a miss ratio curve as a function of the total utilization based on experimental data. We use the maximum slope G_M at the vicinity of M_S in control design to guarantee stability. Given the miss ratio factor, we have the following linearized formula for the purpose of control design:

$$M(k) = M(k\text{-}1) + G_M(A(k) - A(k\text{-}1)) \tag{6}$$

Based on Equations 1-3 and 6, the analytical model for the miss ratio output is as following:

$$M(k) = 0 \qquad (A(k) \le A_{th}(k)) \tag{7}$$

$$M(k) = M(k\text{-}1) + G_M G_A D_B(k\text{-}1) \qquad (A(k) > A_{th}(k)) \tag{8}$$

Note that different scheduling policies in the Basic Scheduler usually lead to a different utilization threshold $A_{th}(k)$. For example, if EDF is plugged into the FCS architecture and the workload is composed of independent and periodic tasks, the utilization threshold A_{th} = 100%. In comparison, the utilization threshold is usually lower than 100% if RM is plugged into the architecture. Therefore, the scheduling policy and the workload characteristics affect the choices on the controlled variable and its performance reference.

We now convert the models to z-domain transfer functions that are amenable to control theory methods. Let $Y(z)$ be the z-transform of a output variable $y(k)$ (e.g., $U(k)$ or $M(k)$) and $X(z)$ be the z-transform of a input variable $x(k)$. A linear system (or a linearized system) can be represented by a transfer function $P(z)$ such that $Y(z) = P(z)X(z)$. For our controlled real-time system, its linearized models in Equations 4 and 8 are equivalent to the following transfer functions outside their own unsaturated zones, respectively.

$$\textit{Utilization}: \qquad P_U(z) = G_A / (z\text{-}1) \tag{9}$$

$$\textit{Miss ratio}: \qquad P_M(z) = G_A G_M / (z\text{-}1) \tag{10}$$

Since the model for miss-ratio and utilization are the same except for the extra miss-ratio factor G_M in Equation 10, for simplicity of discussion we use a same formula $P(z)$ to represent the transfer functions of both controlled variables:

$$P(z) = G / (z\text{-}1) \tag{11}$$

where G is called the process gain. $G = G_A$ for the utilization output and $G = G_A G_M$ for the miss ratio output.

The saturation of $U(k)$ and $M(k)$ renders special challenges in the controller design because the output becomes unresponsive to the control input in the saturation zones. However, since the utilization threshold is lower than 100% (i.e., $A_{th}(k) \leq 1$) for any scheduling policy [19], the saturation zones of $U(k)$ and that of $M(k)$ are always mutually exclusive. This leads to the following important property of real-time systems:

> **Property 1:** At any instant of time, at least one of the controlled variables ($U(k)$ and $M(k)$) does not saturate in a real-time system.

We design a scheduling algorithm, FC-UM, to solve the saturation problem by taking advantage of this property in Section 6.4.3.

6. Design of Feedback Control Real-Time Scheduling Algorithms

In this section, we apply control theory methods to the Controller, the key component of FCS algorithms. We first present the control algorithm and the model of the feedback control loop for each controlled variable. Based on the analytical models, we apply control theory to tune the Controller and develop mathematical analyses on the performance profiles of the resultant Controller. We then present several FCS algorithms to handle different types of real-time systems.

6.1. Design of the Controller

At each sampling instant k, the Controller computes a control input $D_B(k)$, the change in the total estimated requested utilization, based on the miss ratio error $E_M(k) = M_S - M(k)$ and/or the CPU utilization error $E_U(k) = U_S - U(k)$. In this section, we focus on a Controller for a single controlled variable. The goal of the Controller includes (1) guaranteed stability, (2) zero steady state error, (3) zero sensitivity to workload variations, and (4) satisfactory settling time and overshoot. Since the same control function can be used for both controlled variables, we use the same symbol $E(k)$ to denote the miss ratio error $E_M(k)$ and the utilization error $E_U(k)$. Similarly we use S to denote the miss ratio reference M_S and utilization reference U_S, and the symbol $y(k)$ to denote the miss ratio reference $M(k)$ and utilization reference $U(k)$.

For the FCS architecture, we choose to use a simple P (Proportional) control function [12] to compute the control input. The P control function is in Equation 12 where K_P is a tunable parameter.

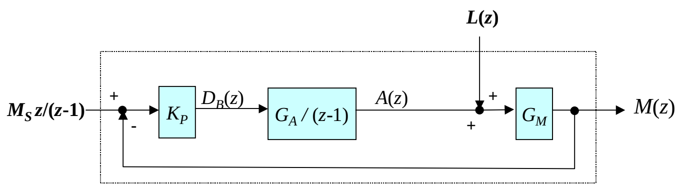
$$D_B(k) = K_P E(k) \tag{12}$$

The rationale for using a P Controller instead of a more sophisticated Controller such as PID (Proportional-Integral-Derivative) Controller is that the controlled system includes an integrator in the QoS Actuator (Equation 1) such that zero steady state error can be achieved without an I (Integral) term in the Controller (see detailed analysis in Section 6.2). The D (Derivative) term is not used in this case because Derivative control may amplify the noise in miss ratio and utilization due to frequent workload variations in unpredictable environments.

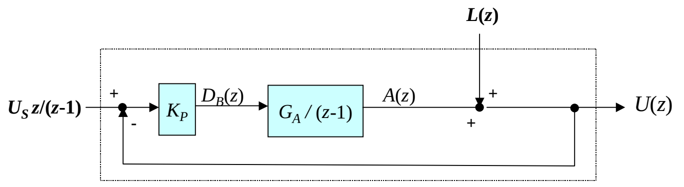
The performance of the real-time system depends on the Controller parameter K_P. An *ad hoc* approach to design the Controller is to repeat numerous experiments on different parameter values. In our work, we apply established control theory methods to tune the parameters analytically to guarantee the performance specifications. In Section 6.2 we first tune the Controller for each of the controlled variables based on the linear models of the controlled system (Equation 11). Due to the saturation properties, the performance of the closed loop system may deviate from the linear case.

6.2. Closed-Loop System Model

The system output is miss ratio $M(k)$ or utilization $U(k)$. There are two input signals to a closed loop system with a single (miss ratio or utilization) Controller.



(a) Miss Ratio Control



(b) Utilization Control

Figure 2: Models of Feedback Control Loops

6.2.1. Reference Input and Arrival Overload

The first input is the performance reference S (i.e., M_S or U_S) modeled as a step signal, $Sz/(z\text{-}1)$ in the z domain. Note that with the arrival-time QoS control mechanism in our FCS architecture, the particular form of load profiles does not affect the system's response because the actual tasks admitted into the system are always determined by the QoS Actuator. Therefore, the system response to the reference input corresponds to the system performance in response to arrival overload. Given the model of the controlled system $P(z)$ (Equation 11) and the Controller $C(z)$ (Equation 12), we can establish a same closed-loop transfer function of both miss ratio and utilization control in response to the reference input:

$$\begin{aligned} &H_S(z) = \frac{C(z)P(z)}{1+C(z)P(z)} = \frac{K_P G}{z-(1-K_P G)} \qquad (a) \\ &Y(z) = H_S(z)\frac{z}{z-1}S \qquad (b) \end{aligned} \tag{13}$$

where $G = G_A$ for utilization control, and $G = G_A G_M$ for miss ratio control.

6.2.2. Disturbance Input: Internal Overload

The second input to the closed-loop system is the internal overload when admitted tasks' CPU utilizations vary. The internal overload can be modeled as a *disturbance* that adds to the total requested utilization $A(k)$ (see Figure 2ab). In particular, a step load $SL(L_n, L_m)$ is modeled as a step signal $L(k)$ that jumps instantaneously from 0 to $(L_m - L_n)$, or $L(z) = (L_m - L_n)z/(z\text{-}1)$ in the z domain; a ramp load $RL(L_n, L_m, T_R)$ is modeled as a ramp signal $L(k)$ that linearly increases from 0 to $(L_m - L_n)$ in a duration of T_R sec. Note that in the case of internal overload input, the specific load profile decides the input signal and therefore has a direct impact on the system performance. In this paper, we focus our analysis on the step load profile because it represents more severe load variations than the ramp load with a finite duration. Regarding the disturbance input, the transfer function for utilization control and the system output in response to the internal overload as follows.

$$\begin{aligned} &H_D(z) = \frac{C(z)P(z)}{1+C(z)P(z)} = \frac{z-1}{z-(1-K_P G_A)} \qquad (a) \\ &Y(z) = H_S(z)\frac{z}{z-1}S + H_D(z)L(z) \qquad (b) \end{aligned} \tag{14}$$

The above transfer function is also applicable to miss ratio control except the disturbance input would be $G_M L(k)$ or $G_M L(z)$ to account for the extra G_M term in Figure 2a.

6.3. Control Tuning and Analysis

We now present the tuning and analysis of the utilization Controller and the miss ratio Controller based on the analytical models described in Equations 13a and 14a. According to control theory, the performance profile of a system depends on the poles of its closed loop transfer function. Based on Equations 13a and 14a, we can place the closed loop pole $p = 1 - K_P G$ at the desired location by choosing the right value for the control parameter K_P. We now use control theory to derive K_P to achieve the desired performance profile.

- **Stability Condition:** *The sufficient and necessary condition for the utilization and the miss ratio control to guarantee stability is*

$$0 < K_P < 2/G \tag{15}$$

Proof: According to control theory, a system is stability if and only if all the poles $\{p_j \mid 0 \le j \le n\}$ (n is the total number of poles) of its transfer function are in the unit circle of z-plane [12], i.e., $|p_j| < 1$ $(0 \le j \le n)$.

From Equations 13a and 14a, the only pole of the utilization and the miss ratio control system in response to the arrival overload and the internal overload is

$$p_0 = 1 - K_P G$$

Hence, the utilization control and the miss ratio control guarantee stability if and only if $|1 - K_P G| < 1 \Leftrightarrow 0 < K_P < 2/G$. Therefore, the sufficient and necessary condition of stability is Equation 15.

We derive the steady state performance of the utilization and the miss ratio control system by applying the Final Value Theorem [12] to the system output in Equations 13b and 14b. The following steady state analysis assumes that the stability condition in Equation 15 is satisfied.

- **Steady state error (arrival overload):** *under the stability condition in Equation 15, in response to an arrival overload, the miss ratio and the utilization control guarantee zero steady state error.*

Proof: Let $y(k)$ be the output of a stable system, the *Final Value Theorem* of digital control theory states that the system output converges to a final value

$$\lim_{k \to \infty} y(k) = y(\infty) = \lim_{z \to 1} (z-1) Y(z)$$

From Equation 13b, the output of the utilization and the miss ratio control in response to an arrival overload is

$$Y(z) = \frac{K_P G}{z - (1 - K_P G)} \frac{Sz}{z-1}$$

where S represents the constant performance reference. Hence it follows that

$$y(\infty) = \lim_{z \to 1} (z-1) Y(z) = \lim_{z \to 1} (z-1) \frac{K_P G}{z - (1 - K_P G)} \frac{Sz}{z-1} = S \tag{16}$$

that is, the steady state error $E_{SC} = S - y(\infty) = 0$.

- ❑ **Steady state error (internal overload):** *under the stability condition in Equation 15, the miss ratio and the utilization control achieve zero steady state error despite the presence of an internal overload***.**

Proof: From Equation 14b, the system output of the utilization and miss ratio control in response to an internal overload $SL(L_m, L_n)$ is

$$Y(z) = \frac{K_P G}{z-(1-K_P G)} \frac{Sz}{z-1} + \frac{z-1}{z-(1-K_P G)} \frac{\Delta L z}{z-1}$$

where $\Delta L = L_m - L_n$ for the utilization control, and $\Delta L = G_M(L_m - L_n)$ for the miss ratio control.

Applying the Final Value Theorem to the above equation, the final value of the utilization control and the miss ratio control is

$$y(\infty) = \lim_{z\to 1}(z-1)Y(z) = \lim_{z\to 1}((z-1)(\frac{K_P G}{z-(1-K_P G)} \frac{Sz}{z-1} + \frac{z-1}{z-(1-K_P G)} \frac{\Delta L z}{z-1})) = S \qquad (17)$$

It follows that the steady state error $E_{SC} = S - y(\infty) = 0$.

- ❑ **Sensitivity:** *under the stability condition in Equation 15, the steady-state performance of the utilization control and the miss ratio control has zero sensitivity with regard to task execution times, inter-arrival-times, and miss ratio factor.*

Proof: In Equation 11, $G = G_a(k)$ for the utilization control, and $G = G_a(k)G_m(k)$ for the miss ratio control. The variation in $G_a(k)$ represents the variation in the task execution times and/or inter-arrival-times, and the variation in $G_m(k)$ represents the variation in the miss ratio factor.

From Equations 16 and 17, the final output of the utilization and miss ratio control system in response to the arrival overload and the internal overload always equals the performance reference S for any value of G if it satisfies the stability condition (Equation 15), that is, the related sensitivity is zero.

In summary of our steady state analysis, we have proven that, under the stability condition in Equation 15, the utilization control and the miss ratio control always achieve the performance reference in steady state in response to arrival and internal overload. Furthermore, we have also shown that this guarantee is robust with regard to task execution times, inter-arrival-times, and the miss ratio factor.

According to control theory, for the system transfer function Equation 13a, the overshoot remains zero in response to arrival overload if the closed loop pole $p_0 \geq 0$. From Equation 16, the utilization control and the miss ratio control achieves zero overshoot if and only if

$$0 < K_P \leq 1/G$$

The settling time increases as the Controller parameter increases in the above range.

We place the pole $p_0 = 0.63$ by settling $K_P = 0.37/G$, or:

$$\begin{aligned} &\text{Miss Ratio Control:} \quad K_P = 0.37/(G_A G_M) \quad &\text{(a)}\\ &\text{Utilization Control:} \quad K_P = 0.37/G_A \quad &\text{(b)} \end{aligned} \qquad (18)$$

The above values for the Controller parameter K_P has the following properties based on control analysis.

1) The parameters in Equations 18ab satisfy the stability condition in Equation 15.

2) Since the control parameter value in Equations 18a and 18b satisfy the zero overshoot condition, the overshoot in response to the reference input is:

Miss Ratio Control: $M_O = 0$ $M_{max} = M_S$ (a)
Utilization Control: $U_O = 0$ $U_{max} = U_S$ (b) (19)

3) The Controller cannot affect the overshoot in response to the disturbance input, which directly changes the output before any control action could take place:

Miss Ratio Control: $M_O = G_M(L_m - L_n)/M_S$ $M_{max} = M_S + G_M(L_m - L_n)$ (a)
Utilization Control: $M_O = (L_m - L_n)/U_S$ $U_{max} = U_S + L_m - L_n$ (b) (20)

4) Regarding the system to be in steady state if its output $y(k)$ is within ε% = 2% of its final value, the above pole placement corresponds to a same settling time in response to the reference input and the disturbance input.

Miss Ratio/Utilization Control: $T_s = 4.5$ sec (21)

The above settling time is not applicable to the miss ratio control in response to arrival overload because the miss ratio $M(k)$ saturates at 0. Assume an arrival overload occurs to an idle system at time 0, the miss ratio control observes $M(0) = 0$, which results in a control signal of $K_P(M_S - M(0)) = K_P M_S$. Since M_S is typically small, the control signal is also small. Due to the saturation problem, the miss ratio will stay at 0 and cause the control signal to remain small. This property can cause the utilization and miss ratio to increase slower than in case of the linear model and result in a longer settling time than Equation 21. One solution is to assign a high initial value to the estimated requested utilization $B(k)$ when the system is idle, which will help to push the system out of the saturation zone faster than a zero initial $B(k)$.

Based on the above analysis, we have the following conclusions on the transient performance of the closed-loop system.

- **Transient Performance in Response to Arrival Overload:** From Equation 21, in response to an arrival overload the output settles to within 2% the performance reference in 4.5 sec. Furthermore, Equation 20a ensures that with miss ratio control, the miss ratio never exceeds the miss ratio reference in response to an arrival overshoot. Similarly, Equation 20b ensures that with utilization control, the CPU utilization never exceeds the utilization reference in response to an arrival overload.

- **Internal Overload**: From Equation 21, the system output can recover to within 2% the performance reference in 4.5 sec after the beginning of an internal step-overload. However, Equations 20a and 20b show that the system suffers from a non-zero overshoot during transient state in response to an internal step-overload. With miss ratio control, the system miss ratio $M(k)$ can overshoot the reference M_S by $G_M(L_m\text{-}L_n)$. With utilization control, the CPU utilization can overshoot the reference U_S by $G_M(L_m\text{-}L_n)$.

6.3.1. Impact of System/Workload Variations on Performance Profiles

Because a real-time system is usually a time-varying system (as discussed in Section 5), an important issue is how the variations in system/workload parameters (e.g., task execution times and miss ratio factor) affect the above analysis based on fixed values of the parameters. Specifically, since $G_a(k)$ and $G_m(k)$ may be different from the worst-case utilization ratio G_A and the worst-case miss-ratio factor G_M. We need to analyze how the changes in miss ratio factor $G_m(k)$ and utilization ratio $G_a(k)$ affect the performance profile of the closed-loop system in the following.

- **Stability**

Based the stability condition in Equation 15 and the Controller parameter in Equations 18a and 18b, we can derive the range of $G_m(k)$ and $G_a(k)$ such that the system stability is guaranteed.

Miss Ratio Control: $0 < G_a(k)G_m(k) < 5.4G_AG_M$
Utilization Control: $0 < G_a(k) < 5.4G_A$ (22)

Note that since we usually compute the Controller parameter K_P based on the worst case estimation such that, $G_A > G_a(k) > 0$ and $G_A > G_a(k) > 0$, our closed-loop system guarantees stability. Furthermore, even if the actual system parameter can exceed the design-time estimations (due to estimation error or dramatic system change), stability is still guaranteed by the closed loop system as long as $G_a(k)$ and $G_m(k)$ stay within the above stability range.

- **Steady State Performance**

We have proven that both miss ratio control and utilization control can achieve their performance references in steady state as long as the systems remain stable. Therefore, both the miss ratio control and utilization control provide robust and accurate performance guarantees in steady state regardless of the actual values of miss ratio factor and utilization ratio if they stay in the stability range (Equation 22).

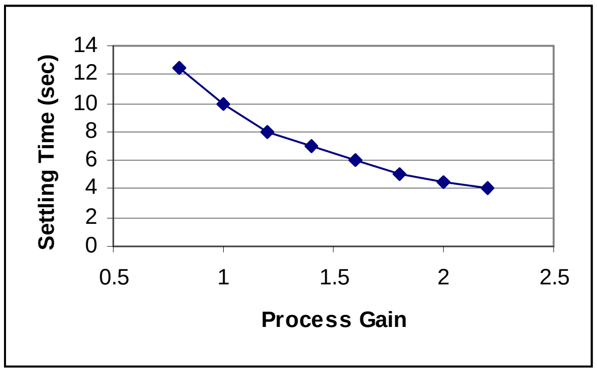


Figure 3: Settling Time vs. Process Gain

- **Transient Performance**

Unlike stability and steady state performance, the closed loop system's settling time is sensitive to the variations in miss ratio factor G_m and utilization ratio G_a. Assume we use an estimation of $G_A = 2.0$ to compute the utilization control parameter $K_P = 0.37/ G_A = 0.185$ (as in our experiments in Section 7), then we plot the theoretical settling time corresponding to different process gains $G = G_a$, shown in Figure 3. The process gain decreases from 12.5 sec to 4 sec as the process gain G increases from 0.8 to 2.2. This result shows that with a same Controller parameter K_P, the system reacts faster to overload when its utilization ratio and miss ratio factor are larger. Therefore, a P Controller with a fixed parameter K_P cannot guarantee a fixed settling time. Instead, if the range of the process gain G is known, a range of settling times can be guaranteed. For example, if we know that the process gain stays in the range $0.8 \leq G \leq 2.0$, the settling time can be guaranteed to be in the range of $4.5 \leq T_S \leq 12.5$ (sec) as shown in Figure 3.

Similarly, the overshoot is also sensitive to the variations in the process gain. For our closed-loop transfer function in response to arrival overload (Equation 13a), the overshoot remains zero if the closed loop pole $p \geq 0$. Therefore, the system can achieve zero overshoot in response to an arrival overload if miss ratio factor and

Miss Ratio Control: $0 < G_a(k)G_m(k) < 2.7G_AG_M$
Utilization Control: $0 < G_a(k) < 2.7G_A$ (23)

In summary, given the system parameters, the worst-case utilization ratio G_A, and the miss ratio factor G_M, we can directly derive the control parameter K_P based on Equations 18a and 18b to guarantee a set of performance profiles including stability, zero steady state error, and a satisfactory range of transient performance. Note that the analytical tuning method of the control parameter is significantly easier and less time consuming than *ad hoc approaches* based on repeated simulation experiments. This is one important advantage of using our control-theory based FCS framework instead of *ad hoc* solutions.

6.4. FCS Algorithms

In this section, we present FCS algorithms based on utilization and/or miss ratio control to achieve satisfactory performance profiles in different types of real-time systems. We also discuss the impact of the basic scheduling policy and workloads on the design of FCS algorithms.

6.4.1. FC-U: Feedback Utilization Control

FC-U embodies a utilization control loop to control the utilization $U(k)$. The utilization control loop periodically samples the utilization, computes a change in the total estimated utilization, and calls an value optimization algorithm to assign QoS levels to tasks under the constraint of the new total estimated utilization. The pseudo code of FC-U is as following.

```
// Invoked periodically at every sampling instant
// $U_S$: utilization reference
// $K_{PU}$: utilization control parameter computed using Equation 18b
FC-U($U_S$, $K_{PU}$) {
   //Monitor
   $U$ = utilization in the last sampling period (($k$-1)$W$, $kW$);
   //Controller
   $E_U = U_S - U$;
   $D_B = K_{PU}*E_U$;
   //Actuator
   $B = B + D_B$;
   //decide task QoS levels to optimize total value under the constraint that
   //the total estimated utilization of all tasks ≤ $B$;
   AssignQoS($B$);
}
```

FC-U guarantees that the miss ratio $M(k) = 0$ in steady state if its reference $U_S \leq A_{th}$ where A_{th} is the schedulable utilization threshold of the system.

Because utilization $U(k)$ saturates at 100%, FC-U cannot detect how severely the system is overloaded when $U(k)$ remains at 100%. The consequence of this problem is that in severely overload conditions FC-U can have a longer settling time than the analysis results based on the linear model. The closer the reference is to 100%, the longer the settling time will be. This is because the utilization control measures an error with a smaller magnitude and thus generates a smaller control input than the ideal case described by the linear model (Equation 11). For example, suppose the total requested utilization $A(k)$ = 200% and the utilization reference is 99%, the error measured by the Controller would be $E_U = 0.99–1 = -0.01$; however, the error would have been $E_U = 0.99 – 2 = -1.01$ according to the linear model. In the extreme case, U_S = 100% can cause the system to stay in overload (a settling time of infinity) because the error E_U=0 even when the system is severely overloaded. Therefore, the reference U_S should have enough distance from 100% to alleviate the impact of saturation on the control performance.

FC-U is especially appropriate for systems with a utilization bound that is *a priori* known and not pessimistic. In such systems, FC-U can guarantee a zero miss ratio in steady state if $U_S \leq A_b \leq A_{th}$. For example, FC-U performs well in a system with EDF scheduling and a periodic and independent task set because its utilization bound is 100%. FC-U is not applicable for systems whose utilization bounds are unknown or pessimistic. In such systems, a reference that is too optimistic (higher than the utilization threshold) can cause high miss ratio even in steady state. On the other hand, a reference that is too pessimistic can unnecessarily underutilize the system.

6.4.2. FC-M: Feedback Miss Ratio Control

Unlike FC-U that controls miss ratio indirectly through utilization control, FC-M utilizes a miss ratio control loop to *directly* control miss ratio[2]. The pseudo code of FC-M is as following.

```
// Invoked periodically at every sampling instant
// $M_S$: miss ratio reference
// $K_{PM}$: miss ratio control parameter computed using Equation 18a
$FC\text{-}M(M_S, K_{PM})$ {
   //Monitor
   $M$ = miss ratio in the last sampling period (($k$-1)$W$, $kW$);
   //Controller
   $E_M = M_S - M$;
   $D_B = K_{PM} * E_M$;
   //Actuator
   $B = B + D_B$;
   AssignQoS($B$);
}
```

Compared with FC-U, the advantage of FC-M is that it does not depend on any knowledge about the utilization bound. In the process of directly controlling the miss ratio, the miss ratio control loop always changes the total requested utilization $A(k)$ to the vicinity of the (unknown) utilization threshold $A_{th}(k)$. An additional advantage of FC-M is that it can achieve higher CPU utilization than FC-U because the utilization threshold is often higher than the utilization bound.

Similar to FC-U, FC-M has restrictions on the miss ratio reference M_S due to saturation. Because miss ratio $M(k)$ saturates at 0, FC-M cannot detect how severely the system is underutilized when $M(k) = 0$. Consequently, FC-M can have a longer settling time than the analysis results based on the linear model (Equation 11) in severely underutilized conditions, and the settling time increases as the miss ratio reference decreases. This is because the miss ratio control measures an error of a smaller magnitude and generates a smaller control input than the case of the linear model. For example, suppose the total requested utilization $A(k) = 10\%$ and the miss ratio reference is $M_S = 1\%$, the error measured by the Controller would be $E_M = 0.01 – 0 = 0.01$; however, the error would have been much larger according to the linear model because it would have a "negative" miss-ratio. In the extreme case, $M_S = 0$ can cause the CPU to remain underutilized because the error $E_M = 0$ even when the system is severely underutilized. Therefore, the miss ratio reference should have some distance from the saturation boundary 0 to alleviate the impact of saturation on the control performance. Unfortunately, a positive miss ratio reference also means that the system cannot achieve zero miss ratio in steady state.

In summary, the FC-M scheduling algorithm (with a small positive miss ratio reference) can achieve low deadline miss ratio (close to M_S) and high CPU utilization even if the system's utilization bound is *unknown* or *time varying*. Since FC-M cannot guarantee zero deadline miss ratio in steady state, it is only applicable to soft real-time systems that can tolerate sporadic deadline misses in steady state.

6.4.3. FC-UM: Integrated Utilization/Miss Ratio Control

FC-UM integrates miss-ratio control and utilization control to combine the advantages of both FC-U and FC-M [3]. In this integrated control scheme, both miss-ratio $M(k)$ and utilization $U(k)$ are monitored. At each sampling instant, $M(k)$ and $U(k)$ are fed back to two separate Controllers, the miss ratio Controller and the utilization Controller, respectively. Each Controller then computes its control signal

[2] FC-M was also called FC-EDF in [23] when working with EDF as the Basic Scheduler.

[3] FC-UM was also called FC-EDF^2 in [22] when working with EDF as the Basic Scheduler.

independently. The control input of the utilization control $D_{BU}(k)$ is compared with the miss-ratio control input $D_{BM}(k)$, and the smaller one $D_B(k) = \min(D_{BU}(k), D_{BM}(k))$ is sent to the QoS Actuator.

```
// Invoked periodically at every sampling instant
// $M_S$: miss ratio reference
// $U_S$: utilization reference
// $K_{PM}$: miss ratio control parameter computed using Equation 18a
// $K_{PU}$: utilization control parameter computed using Equation 18b
FC-UM($M_S$, $U_S$, $K_{PM}$, $K_{PU}$) {
   //Monitors
   $M$ = miss ratio in the last sampling period (($k$-1)$W$, $kW$);
   $U$ = utilization in the last sampling period (($k$-1)$W$, $kW$);
   //Controllers
   $E_M = M_S - M$;
   $D_{BM} = K_{PM}*E_M$;
   $E_U = U_S - U$;
   $D_{BU} = K_{PU}*E_U$;
   $D_B$ = min($D_{BM}$,$D_{BU}$)
   //Actuator
   $B = B + D_B$;
   AssignQoS($B$);
}
```

Note that the advantage of FC-U is that it can achieve excellent performance ($M(k) = 0$) in steady state if the utilization reference is correct, while the advantage of FC-M is that it can always achieve low (but, non-zero) miss ratio and, therefore, is more robust in face of utilization threshold variations. The integrated control structure can achieve the advantages of both controls because of the following reasons. If used alone, the utilization control would change the total requested utilization $A(k)$ to its reference U_S in steady state, and the miss ratio control loop would change $A(k)$ to the vicinity of the utilization threshold $A_{th}(k)$ in steady state. Due to the min operation on the two control inputs, the integrated control loop would change the total requested utilization to the *lower* value caused by the two control loops, $\min(A_{th}(k), U_S)$. The implication of this feature is that the integrated control loop always achieves the performance of the relatively more *conservative* control loop in steady state. Specifically, in a system scheduled by FC-UM, if $U_S \le A_{th}(k)$, the utilization control dominates in steady state and guarantees that the total requested utilization $A(k)$ stays close to its utilization reference U_S and thus miss ratio $M(k) = 0$ in steady state. On the other hand, if $U_S > A_{th}(k)$, the utilization control dominates in steady state and guarantees that the total requested utilization stays close to its utilization threshold $A_{th}(k)$ and miss ratio $M(k) = M_S$ in steady state.

In a system with the FC-UM scheduler, the system administrator can simply set the utilization reference U_S to a value that causes no deadline misses in the *nominal* case (e.g., based on system profiling or experiences), and set the miss ratio reference M_S according to the application's requirement on miss ratio. FC-UM can guarantee zero deadline misses in the nominal case while guaranteeing that the miss ratio stay close to M_S even if the utilization threshold of the system becomes lower than the utilization threshold. Our experimental results (Section 7) demonstrate that FC-UM achieves satisfactory performance.

6.4.4. Impacts of Scheduling Policies and Applications on FCS algorithm Design

An important factor that affects the design of FCS algorithms is whether an *a priori known* and *non-pessimistic* utilization bound exists for the scheduling policy and workload of a system. Existing real-time scheduling theory has derived the schedulable utilization bound for various scheduling policies based on different workload assumptions. For example, assuming a periodic and independent task set, it has been established that the schedulable utilization bound of EDF and RM is 100% and 69%, respectively [19]. Recently, the schedulable utilization bound for Deadline Monotonic scheduling is also derived for general aperiodic and periodic tasks in the ideal case [1]. Other papers established schedulable utilization bounds for other types of workloads (e.g., [15][31]). Since FC-U can guarantee a miss ratio $M(k) = 0$ in steady state if its utilization reference $U_S \leq A_b$, the utilization reference should be determined based on the scheduling policy and workload of a system. For example, for an independent and periodic task set scheduled by EDF, a $U_S = 90\%$ is sufficient to guarantee that miss ratio stays at 0 in steady state. Because FC-U can achieve zero steady state miss–ratio, it is the most appropriate FCS algorithm for systems with a known and non-pessimistic utilization bound. FC-UM can also achieve zero steady state miss-ratio in this type of system, but it is more complicated than FC-U.

Unfortunately, the utilization bounds of many unpredictable real-time systems are still *unknown*. For example, in a typical on-line trading server, database transactions and Web request processing can be blocked frequently due to concurrency control, disk I/O, and TCP congestion control. The task arrival patterns may also vary considerably because its workload is composed of periodic price updating tasks and unpredictable and aperiodic stock trading request processing. Deciding a utilization bound on top of commercial OS's can be even more difficult due to unpredictable kernel activities such as interrupt handling. Another issue is a theoretical utilization bound can be severely pessimistic for the specific workload currently in a system. For example, although the utilization bound of Rate Monotonic is 69% for periodic independent tasks, uniformly distributed task sets often do not suffer deadline misses even when the CPU utilization reaches 88% [17]. Enforcing the utilization at the utilization bound may not be cost-effective in soft real-time systems. FC-M and FC-UM are more appropriate than FC-U for systems without a known and non-pessimistic utilization bounds.

We should note that different scheduling policies and workloads usually introduce different miss ratio factors G_M. Because the gain K_P of the miss ratio Controller should be inversely proportional to the miss ratio factor (Equation 18a), the scheduling policy and workload can directly affect the correct parameter settling of the miss ratio Controller. For example, while our previous experiments showed that while the EDF algorithm with a periodic task set led to a miss ratio factor $G_M = 1.254$, the Extended Deadline Monotonic (DM) algorithm with a mixed periodic and aperiodic task set has a much smaller miss ratio factor $G_M = 0.447$ (see Section 7.5). This result means that for DM with the mixed task set, the K_P of the miss ratio Controller should be 2.81 times the K_P of EDF with the aperiodic task set in order to achieve similar performance.

In summary, we have designed three FCS algorithms (FC-U, FC-M, and FC-UM) using control theory based on an analytical model for a real-time system. Our control theory analysis proves that the resultant FCS algorithms can achieve the following performance guarantees under the stability condition of Equation 22:

1) Guaranteeing stability,
2) Guaranteeing that the system miss ratio and utilization remains close to the corresponding performance reference in steady state, and
3) Achieving satisfactory settling time (Figure 3) and zero overshoot under condition of Equation 23 in transient state.

7. Experiments

In this section, we describe the simulation experiments that evaluate the performance of our FCS algorithms and the correctness of our control design. We first describe a real-time CPU scheduling simulator used for our experiments. We then describe the configurations of the experiments and workloads. A set of profiling experiments on the controlled system is presented next. We then present two sets of evaluation experiments for our FCS algorithms.

7.1. FECSIM Simulator

The FCS architecture is implemented on a generic uniprocessor real-time system simulator called FECSIM [23]. FECSIM (Figure 4) has five components: a set of *Sources* that each generates a periodic or aperiodic task; an *Executor* that emulate the Basic Scheduler and the execution of the tasks; a *Monitor* that periodically measures controlled variables; a *Controller* that periodically computes the control input based on the performance errors; and a *QoS Actuator* that adjusts the QoS levels of the tasks to optimize the system value (based on estimated utilizations) under the utilization constraints. Different basic real-time scheduling policies can be plugged into the Executor. The Controller can be turned on/off to emulate the closed loop or open loop scheduling.

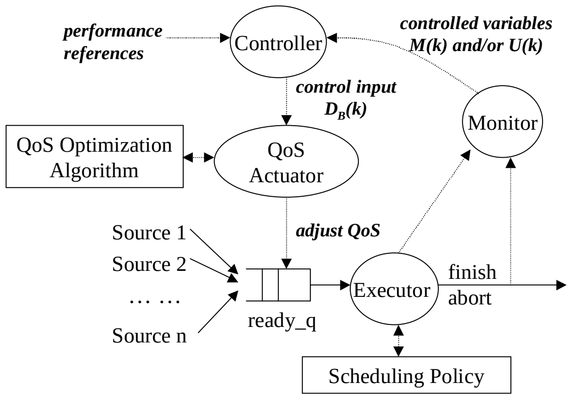


Figure 4: The FECSIM Simulator

7.2. Scheduling Policy of the Basic Scheduler

To demonstrate the generality of our FCS architecture and the robustness of our FCS algorithms, we present experimental results with two combinations of task sets and scheduling policies for the basic scheduler. We denote these two combinations as DM/PA and EDF/P (see Table 1. We describe the scheduling policies in this section, and the workloads in Section 7.3.

Configuration	Basic Scheduling Policy	Task Set
EDF/P	EDF	Periodic (P)
DM/PA	Extended Deadline Monotonic	Periodic/Aperiodic (PA)

Table 1 : Testing Configurations

Two different scheduling policies, Extended Deadline Monotonic (DM) and EDF are used in the Basic Scheduler.

- *DM*: Each task is assigned a fixed priority that equals its *relative* deadline. A shorter relative deadline leads to a higher priority. DM has been proved to be the optimal static scheduling policy in term of maximizing schedulable utilization bound under certain conditions [1]
- *EDF:* Each task is dynamically assigned a priority that equals its *absolute* deadline. An earlier absolute deadline leads to a higher priority. EDF is a dynamic real-time scheduling policy [19].

7.3. Workload

Two different task sets are used in our evaluation experiments.

- *Periodic/Aperiodic* (*PA*): the workload is composed of 50% aperiodic tasks and 50% periodic tasks. This type of task set can be found in a typical on-line trading server whose workload is composed of periodic stock updating tasks and aperiodic user requests such as trading and information queries.

- *Periodic* (*P*): all the tasks are periodic tasks. This type of task set emulates real-time applications such as multi-media streaming and process control where most of the system operations are periodic.

Each task follows the task model described in Section 2.1. Each task is assumed to have three QoS levels (0, 1, 2) including the lowest level 0 that represents service rejection. For the rejection level, both the task execution time and value are set to 0. The distributions of the task parameters are as follows. For the purpose of presentation, we assume each time unit is 0.1 ms.

$EE_i[j]$: The estimated execution time $ET_i[2]$ of task T_i at the QoS level 2 follows a uniform distribution in the range [0.2, 0.8] ms, and $ET_i[1] = 0.2ET_i[2]$.

$AE_i[j]$: The actual execution time $AE_i[j]$ of task T_i at QoS level j follows a normal distribution $N(AAE_i, AAE_i^{1/2})$, where the average execution time $AAE_i[j] = G_a'ET_i[j]$. G_a', called the *execution time factor,* is a tunable workload parameter that approximates the utilization ratio G_a. The larger G_a' is, the more pessimistic is the estimation of execution time. The maximum value of G_a' is 2.0 in all of our experiments, which means that the estimated execution time is twice the average execution time, i.e., worst-case utilization ratio: $G_A = 2.0$. This value is used to compute the Controller parameters based on Equations 18a and 18b.

$D_i[j]$: All QoS levels of a task T_i have a same and fixed relative deadline $D_i = (10F_i + 10)ET_i[2]$, where F_i, follows a uniform distribution in the range of [10, 15]. A task instance is immediately aborted once it misses its deadline.

$V_i[j]$: The value $V_i[j]$ of task T_i at QoS level j is computed as a weight w_i times its estimated execution time, i.e., $V_i[j] = w_iET_i[j]$. The weight w_i follows a uniform distribution in the range [1, 5].

Periodic tasks:

$P_i[j]$: All QoS levels of a task T_i have a same *period* that equals its deadline $P_i = D_i$. The average utilization of each periodic task i at QoS level j is $AA_i[j] = AAE_i[j]/P_i$.

Aperiodic tasks:

$AI_i[j]$: The inter-arrival-time of an aperiodic task T_i follows an exponential distribution with an average inter-arrival-time of $AI_i = D_i$. The average utilization of each periodic task i at QoS level j is $AA_i[j] = AAE_i[j]/AI_i$.

$EI_i[j]$: The estimated inter-arrival-time of an aperiodic task T_i equals the average inter-arrival-time, i.e., $EI_i = AI_i = D_i$.

7.4. QoS Actuator

A Highest-Value-Density-First (HVDF) QoS assignment algorithm [29] is used in the QoS Actuator. The *value density* of QoS level j of a task T_i is defined as $VD_i[j] = V_i[j]/B_i[j]$. The HVDF algorithm assigns QoS levels to all the current tasks in the order of the decreasing value density until the total estimated requested utilization reaches a *utilization constraint* U_C. A fixed threshold of 80% is used by open loop scheduling algorithms. In comparison, our FCS algorithms dynamically change the threshold at each sampling instant.

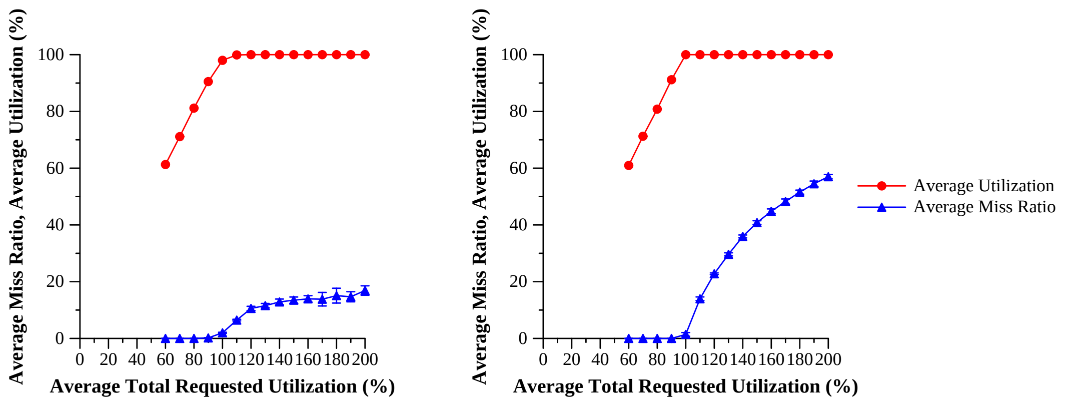
When each task's utilization is small and there are no deadline misses, HVDF approximates the optimal value under the utilization constraint. However, if the task execution times and/or task inter-arrival rates are unknown, an open-loop QoS optimization algorithm may not achieve maximize values due to deadline misses.

Note that our FCS architecture can incorporate different real-time scheduling policies and QoS optimization algorithms. Our work focuses on the steady and transient state performance of the feedback control loop rather than evaluating the basic scheduling policies or QoS optimization algorithms.

7.5. Profiling the Controlled Real-Time Systems

In the first set of experiments, we profile the controlled system to verify the saturation properties of the controlled variables, miss ratio $M(k)$ and CPU utilization $U(k)$, and measure the miss ratio factor G_M, which is a key system parameter used for computing the Controller parameter K_P in miss ratio control (Equation 18a).

Since we are interested in the properties of the controlled system, we turn off the Controller and the QoS Actuator of FECSIM in the profiling experiments. A set of step loads $SL(0, L_m)$ with different overload level L_m are used to stress FECSIM with for 60 sec. Each step load is composed of a set of tasks with an average total requested utilization of L_m. The experiments are repeated for both EDF/P and DM/PA configurations. We plot the measured average CPU utilization and average miss ratio corresponding to each step load level L_m in Figure 5a (DM/PA) and Figure 5b (EDF/P). Each point in the figures represents the average value of 5 runs. The 90% confidence intervals of the average miss ratio are also shown, while the confidence intervals of the average utilization are skipped because it is always within ±1% from corresponding average values.



(a) DM/PA (b) EDF/P

Figure 5: Controlled Variables vs. Total Requested Utilization

7.5.1. Profiling Results on DM/PA

First, we study the profiling results on DM/PA (Figure 5a). CPU utilization $U(k)$ saturates at 100% after the step load level L_m exceeds 100%. Miss ratio $M(k)$ saturates at 0 when the average total requested utilization A' is below 90%, and deadline misses start to occur when A' reaches 90%.

When $A' > 90\%$, the system's average miss ratio increases as the total requested utilization increases. We measure the maximum slop of the miss ratio curve near the boundary of the saturation zone to approximate the miss ratio factor G_M. In Figure 5(a), the maximum slope is 0.447 when the average total requested utilization increases from 100% to 110%. Therefore, the worst-case miss ratio factor G_M = 0.447 for the DM/PA settling.

7.5.2. Profiling Results on EDF/P

Second, we study the profiling results on EDF/P (Figure 5b). CPU utilization $U(k)$ saturates at 100% after the step load level L_m exceeds 100%. Miss ratio $M(k)$ saturates at 0 when the average total utilization A' is below 100%, and deadline misses start to occur when A' reaches 100% (the deadline misses when A' = 100% is due to random execution times of the workload).

When A' is above 100%, the system's average miss ratio increases as the total requested utilization increases. In Figure 5b, the maximum slope is 0.447 when the average total requested utilization increases from 100% to 110%. Therefore, the worst-case miss ratio factor G_M = 1.254 for the EDF/P setting.

	FC-UM	
	FC-U	**FC-M**
$\boldsymbol{K_P}$ (DM/PA)	0.185	0.414
$\boldsymbol{K_P}$ (EDF/P)		0.148
W (Sampling Window)	0.5 sec	

Table 2: Controller Parameters of FCS Algorithms

7.6. Controller Parameters

Based on the worst-case utilization ratio G_A and the worst-case miss ratio factor G_M, we compute the Controller parameter using Equations 18a and 18b. The resultant Controller parameter K_P for each FCS algorithm is listed in Table 2.

7.7. Performance References

The miss ratio reference depends on the application's requirement and tolerance to deadline misses in steady state. For example, Amazon.com may accept a higher miss ratio reference than E-Trade.com because usually a merchandize purchase has less strict timing constraints than stocking trading transactions. We assume that a miss ratio reference M_S = 2% (in both FC-M and FC-UM) is appropriate in our simulated applications. The utilization reference U_S should be lower than the nominal utilization threshold of the basic scheduling policy and the task set. U_S should also be lower than 100%, the saturation boundary of the utilization control. Since the theoretical utilization bound of EDF and a periodic task set is 100% in the ideal case [19], we set U_S = 90% in both FC-M and FC-UM in the EDF/P case. Although it has been shown that DM and general (aperiodic and/or periodic) task sets have a theoretical utilization bound of 58%, this bound is too pessimistic for our mixed aperiodic/periodic task set. For example, in our profiling experiments (Figure 5a), the utilization threshold A_{th} appears to be in the range (90%, 100%). We choose U_S = 80% in FC-U and U_S = 90% in FC-UM. FC-UM has a more optimistic utilization reference than FC-U because the miss ratio control in FC-UM provides a worst-case bound for the closed-loop performance even if the utilization reference becomes higher than the actual utilization threshold. The chosen performance references are summarized in Table 3.

	FC-U	**FC-M**	**FC-UM**
$\boldsymbol{U_S}$	80% (DM/PA)	N/A	90%
	90% (EDF/P)		
$\boldsymbol{M_S}$	N/A	2%	2%

Table 3: Performance References of FCS Algorithms

7.8. Evaluation Experiment A: Arrival Overload

In this section, we present the performance evaluation results of three FCS algorithms, FC-U, FC-M, and FC-UM in response to an arrival overload SL(0, 150%). The execution time factor is G_a' = 2. Therefore, the average execution time of each task was twice the estimation. An open loop scheduling algorithm using a fixed utilization constraint B = 80% for QoS Optimization is also evaluated as a baseline. The same scheduling policies (DM and EDF) and QoS optimization algorithm (the HVDF algorithm) are used for all FCS algorithms and the baseline. A zero initial value for $B(0) = 0$ for the total estimated utilization $B(k)$ is used in this section. A larger initial value for $B(k)$ is used in Experiment B (Section 7.9) to reduce the settling time of FC-M and FC-UM. We now describe the results for each of the scheduling algorithms.

7.8.1. FC-U

First, we look at FC-U with DM/PA (Figure 6a). In response to the arrival overload, FC-U increases the CPU utilization $U(k)$ by increasing the total estimated utilization $B(k)$ of the tasks in the system. The increasing $B(k)$ is enforced by the QoS Actuator that increases task QoS levels with the QoS optimization algorithm HVDF. By 4.5 sec, the settling time predicted by our control analysis, $U(k)$ reaches 77.1%, which is within 3.6% of the reference $U_S = 80\%$. This result is close to our prediction that the $U(k)$ should reach within 2% of the reference by 4.5 sec. The small difference between the experimental results and the theoretical prediction is due to the randomness of our workload. $U(k)$ never reaches beyond 80% in the transient state (before 4.5 sec). This result is also consistent with our theoretical prediction of zero overshoot, $U_O = 0$.

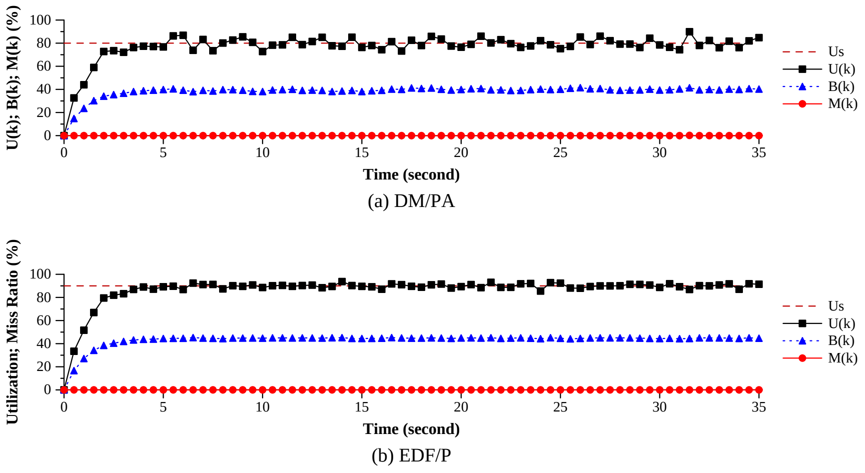
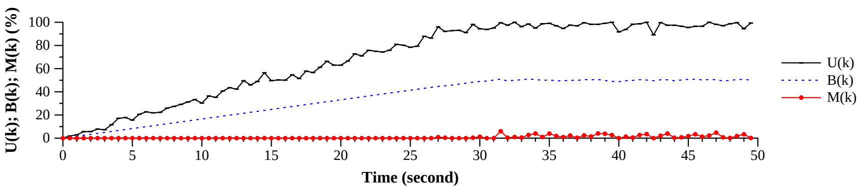


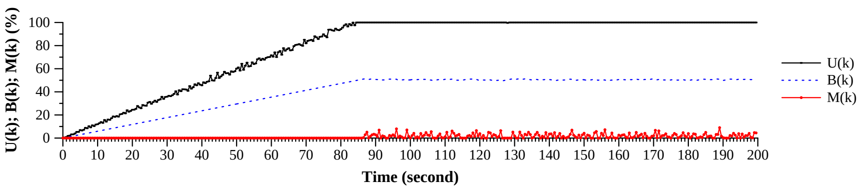
Figure 6: FC-U in response to an arrival overload

The CPU utilization *U(k)* remains stable all through the run. After 4.5 sec, the utilization stays close to 80% and the system error stays close to zero. Because $U(k)$ stays below the utilization threshold, the miss ratio $M(k) = 0$ in throughout the run.

The performance of FC-U with EDF/P (see Figure 6b) is similar to that of FC-U with DM/PA. At 4.5 sec, FC-U increases the CPU utilization $U(k)$ to 87.14%, within 3.2% of the reference $U_S = 90\%$. $U(k)$ never reaches beyond 90% in the transient state (before 4.5 sec). The CPU utilization $U(k)$ remains stable all through the run and close to 90% after 4.5 sec. Because $U(k)$ stays below the utilization threshold, the miss ratio $M(k) = 0$ throughout the run.



(a) DM/PA



(b) EDF/P

Figure 7: FC-M in response to an arrival overload

7.8.2. FC-M

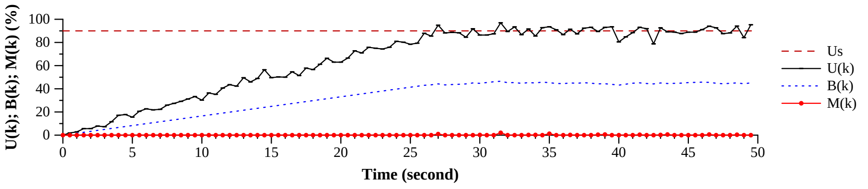
The performance evaluations of FC-M with DM/PA and EDF/P are illustrated in Figure 7. We first study FC-M with DM/PA. In response to the arrival overload, FC-U increases the total estimated utilization $B(k)$ by increasing the QoS levels of arriving or admitted tasks. Due to the saturation of the miss-ratio control, the settling time of FC-M in response to the arrival overload is longer than the prediction based on the linear model (Equation 11). $M(k)$ stays at 0 for the first 26.5 sec since the beginning of the arrival overload. The system settles at approximately 30 sec when $M(k)$ reaches 1.23% (within 0.77% to the reference M_S = 2%) and $U(k)$ reaches 94.44%. We can shorten the settling time of FC-M in response to arrival overload by assigning a larger initial value to the total estimated utilization $B[0]$ (as shown in Section 7.9). $M(k)$ never reaches beyond 2% and therefore achieves zero miss ratio overshoot in the transient state.

$M(k)$ remains stable throughout the run. In steady state (after 30 sec), $M(k)$ stays close to 2% and below 5% throughout the run except for $M(k)$ = 5.97% at 31.5 sec. This result shows that the steady state error is close to 0 as predicted by our analysis.

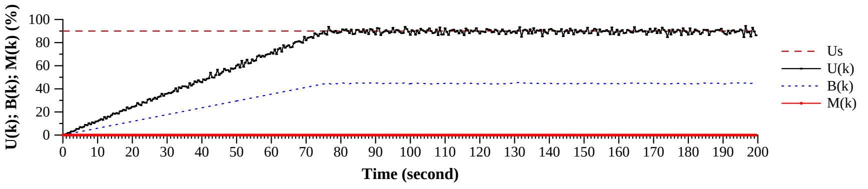
We also observe that with FC-M, the CPU utilization $U(k)$ in steady state is clearly higher than the CPU utilization (close to 80%) in the run of FC-U. This is because by directly controlling the miss ratio, FC-M can change the CPU utilization to the vicinity of the (*a priori* unknown) utilization threshold, which is higher than the utilization reference of FC-U that is set to 80% *a priori*.

The performance of FC-M with EDF/P (Figure 7b) is similar to the case of DM/PA. The settling time is approximately 87 sec when the miss ratio reaches 2.88%. FC-M with EDF/P achieves zero overshoot in transient state. The miss ratio stays close to 2% in steady state and remains stable throughout the run. Similar to the case of DM/PA, FC-M with EDF/P also has a higher CPU utilization (close to 100%) than FC-U with EDF/P (close to 90%) in steady state.

In summary, compared with FC-U, FC-M achieves higher CPU utilization and robustness with regard to utilization threshold variations at the cost of a low, but non-zero miss ratio in steady state.



(a) DM/PA



(b) EDF/P

Figure 8: FC-UM in response to an arrival overload

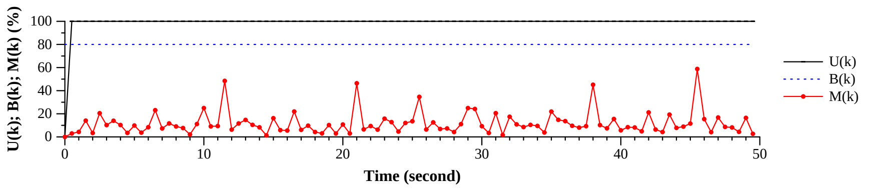
7.8.3. FC-UM

The performance evaluations of FC-UM with DM/PA and EDF/P are illustrated in Figure 8. First, we study the performance of FC-UM with DM/PA. After the overload arrives, FC-UM increases the utilization $U(k)$. Similar to FC-M, the miss ratio $M(k)$ stays at 0 and the CPU utilization $U(k)$ increases slower than FC-U. In the beginning of the run, the (saturated) miss ratio control computes a smaller control signal $D_{BM}(0) = K_P(M_S – M(0)) = 0.414*(0.02–0) = 0.008$ than of the utilization control's signal $D_{BM}(0) = K_P(U_S – U(0)) = 0.185*(0.9–0) = 0.167$. Due to the *min* operation on control inputs from the two Controllers, the miss ratio control dominates the control loop in the starting phase. The miss control signal remains 0.008 and stays smaller than the utilization control signal, which decreases as the utilization $U(k)$ increases. At time 27 sec, the utilization $U(54)$ reaches 94.9% and the miss ratio $M(54)$ = 0.93%. Now the utilization control signal $D_{BU}(54) = -0.009$ becomes smaller than the miss ratio control signal $D_{BM}(54) = 0.004$ and takes over the control loop. Because the utilization threshold is higher than the utilization reference U_S = 90%, the utilization control dominates the control loop, and $U(k)$ stays close to 90% while the miss ratio stays at 0 after 27 sec. Therefore, the settling time is approximately 27 sec. Since neither $U(k)$ nor $M(k)$ surpasses its corresponding reference in transient state (before 27 sec), FC-UM achieves 0 overshoot in both $U(k)$ and $M(k)$.

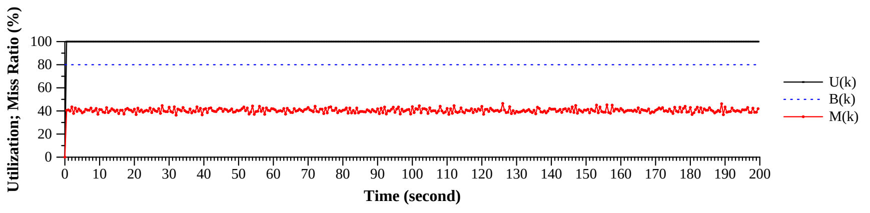
In the steady state, the utilization $U(k)$ stays close to 90% and, hence, FC-UM achieves zero steady state error in term of the utilization. The miss ratio $M(k)$ remains close to 0%, lower than the miss ratio reference M_S = 2% throughout the steady state except M(63) = 2.04%. This is because the utilization reference is lower than the utilization threshold and therefore dominates the control loop in the steady state. Note that if the utilization reference were higher than the utilization threshold, the miss ratio control would dominate the control loop and FC-UM would achieve zero steady error in term of miss ratio and a steady state utilization close to the utilization threshold. The system remains stable throughout the run.

The performance of FC-UM with EDF/P (Figure 8b) is similar to the case of FC-UM with DM/PA. The miss-ratio control dominates the control loop in the beginning of the experiment until 75 sec (the settling time) when the utilization control starts to take over the control loop. FC-UM with EDF/P achieves zero overshoot in both utilization $U(k)$ and $M(k)$. Because the utilization reference U_S is lower than the utilization threshold, FC-UM with EDF/P achieves zero steady state error in term of utilization and the miss ratio stays at 0 throughout the steady state.

In summary, FC-UM combines the advantages of both FC-U and FC-M and achieves zero steady state miss ratio in the nominal case when the utilization reference is lower than the utilization threshold. Furthermore, FC-UM can also achieve a low steady state miss ratio even if the system's utilization threshold changes to lower than the utilization reference.



(a) DM/PA



(b) EDF/P

Figure 9: Open loop QoS optimization in response to an arrival overload

7.8.4. Open Loop QoS Optimization

In comparison with the FCS algorithms, the system scheduled by the open loop QoS optimization algorithm suffers from high miss ratios with both DM/PA and EDF/P (see Figure 9). This is because the task execution time is on average twice that of the estimation and the QoS optimization algorithm overloaded the CPU due to the incorrect estimations on task execution time. On the other hand, the system would suffer from low CPU utilization if the task execution time were lower than the estimation (see Section 7.9). This result demonstrates that the open loop QoS optimization algorithm is incapable of maintaining satisfactory performance in face of unpredictable workload.

In summary, we have demonstrated that all of our three FCS algorithms, FC-U, FC-M, FC-UM can provide desired performance guarantees in terms of miss ratio and CPU utilization in steady state and achieve satisfactory performance profiles in response to an arrival overload *SL*(0, 150%) when the average task execution times is different from the estimation. In contrast, the open-loop QoS optimization fails to provide such performance guarantees in face of the same overload.

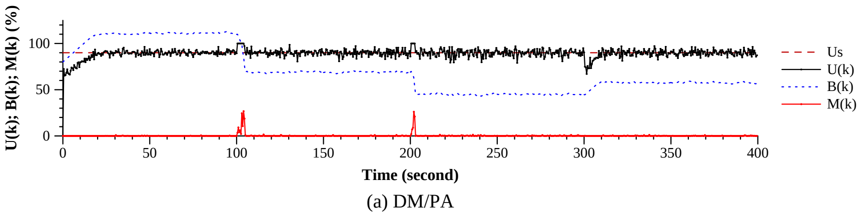
Interval (sec)	0-100	100-200	200-300	300-400
G_a’	0.8	1.26	2	1.5

Table 4: Execution time factor G_a’ in Experiment B

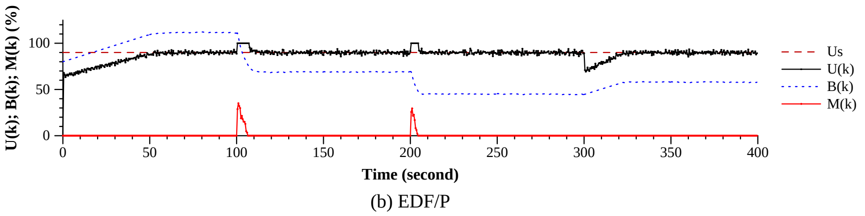
7.9. Evaluation Experiment B: Arrival/Internal Overload

In the second set of evaluation experiments, we stress our FCS algorithms and the baseline with a more unpredictable load profile than the one used in Experiment A. The new load profile causes an arrival overload of *SL*(0, 150%) in the beginning of each run. Furthermore, the average task execution times of all tasks vary every 100 sec to create internal overload in the system. The execution time factor G_a' throughout the run is shown in Table 4. The execution time factor G_a' instantaneously jumps from 0.8 to 1.26 at time 100 sec. This change causes a 57.5% increase in the average execution time of every task. Suppose the total requested utilization of the system is *A*(200) before the jump, the execution time change corresponds to an internal overload of *SL*(*A*(200), 1.575*A*(200)). A similar step load *SL*(*A*(400), 1.575*A*(400)) occurs again at time 200 sec when G_a' jumps from 1.26 to 2. The jump at time 300 sec, on the other hand, creates an internal underload *SL*(*A*(600), 0.75*A*(600)) (modeled as a negative step signal) when G_a' instantaneously decreases from 2 to 1.5.

In this set of experiments, a larger initial value $B(0)$ = 80% is assigned to the estimated requested utilization $B(k)$ to shorten the settling time of FC-M and FC-UM in response to arrival overloads. The open-loop baseline uses a fixed $B(k)$ = 80% and $B(k)$ = 90% for QoS optimization with DM/PA and EDF/P, respectively. Due to space limitations, only the results of FC-UM and the open-loop baseline are presented in detail. The detailed results of FC-U and FC-M are available in [24].



(a) DM/PA



(b) EDF/P

Figure 10: FC-UM in response to an arrival/internal overload

7.9.1. FC-UM

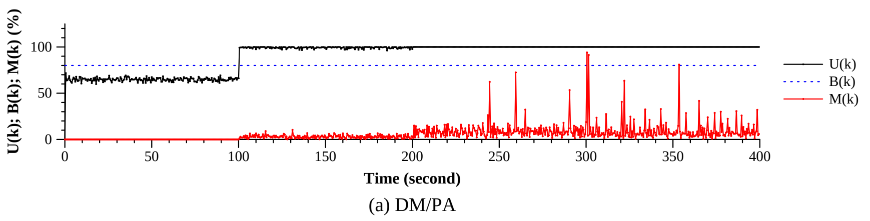
The run of FC-UM with DM/PA and EDF/P is illustrated in Figure 10. We first study the run of FC-UM with DM/PA. In response to the arrival overload at time 0, the miss ratio control dominates the control loop in the transient state until the utilization approaches the utilization reference U_S = 90% when the utilization takes over the control loop. Because the utilization reference is lower than the utilization threshold, the utilization control dominates the control loop and the system settles to steady state at 17.5 sec. The miss ratio $M(k)$ stays at 0% most of the time while in steady state, and the utilization $U(k)$ stays close to 90%.

The system stays in the steady state until 200 sec when the average execution time of every task increases from 0.8 to 1.26. The utilization $U(k)$ overshoots to 100% and the miss ratio overshoots to 24.53%. Although both the miss ratio control and the utilization control compute negative control signals in response to the internal overload, the miss ratio takes over the control loop because the utilization saturates at 100% resulting in a control signal with a smaller magnitude. The miss ratio control dominates the control loop until the utilization approaches 90% and the miss ratio becomes zero. The FC-UM then takes over and the system settles to a new steady state at 105 sec with an average miss ratio of 0.07% and an average utilization of 89.85% (steady state error E_{SU} = 0.15%).

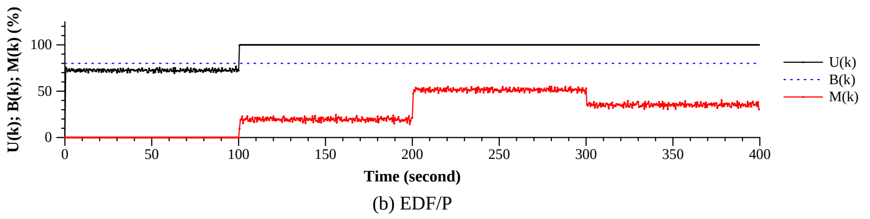
FC-UM responds similarly to the internal overload at 200 sec when the execution factor increases from 1.26 to 2. The system settles down to a satisfactory steady state within 2.5 sec. In the steady state (from 203 sec to 300 sec), the average miss ratio is 0.12% and the average utilization is 89.71% (steady state error E_{SU} = 0.29%).

At time 300 sec, the execution time factor decreases from 2 to 1.5 and the utilization $U(k)$ drops to 69.24%. Similar to the beginning of the run, FC-UM increases total estimated utilization $B(k)$ by improving task QoS levels. At time 308.5 sec, $U(k)$ increases to 92.02% while the system resettles in a steady state with an average miss ratio of 0.07% and an average utilization close to 89.90% (the steady state error E_{SU} = 0.10%).

The performance of FC-UM with EDF/P is similar to the above case with DM/PA. FC-UM with EDF/P successfully reacts to both the arrival overload and the internal overload and (re)settles to satisfactory steady states while the miss ratio stays close to 0% and the utilization stays close to 90% despite of the difference in execution times. This observation verifies that FC-UM has zero sensitivity with regard to execution time variations and provides robust performance guarantees in face of unpredictable workloads.



(a) DM/PA



(b) EDF/P

Figure 11: Open loop QoS optimization in response to an arrival/internal overload

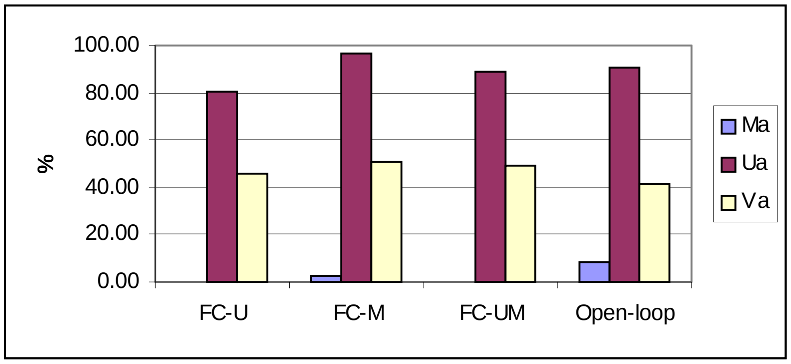
7.9.2. Open-Loop QoS Optimization

The performance of the open-loop baseline with DM/PA and EDF/P is illustrated in Figure 11. In contrast with our FCS algorithms, the open-loop baseline fails to provide performance guarantees in miss ratio or utilization in both EDF/P and DM/PA tests. When task execution times are lower than the estimations (from 0 to 100 sec), the baseline algorithm underutilizes the CPU (with utilization $U(k)$ close to 72%). On the other hand, when the execution exceeds the estimations (from 100.5 sec to 400 sec), the system suffers from persistent deadline misses. For example, the baseline with DM/PA has an average miss ratio of 9.23% from 200.5 sec to 300 sec and the miss ratio reaches 94.1%. The baseline with EDF/P has an average miss ratio of 51.39% in the same period.

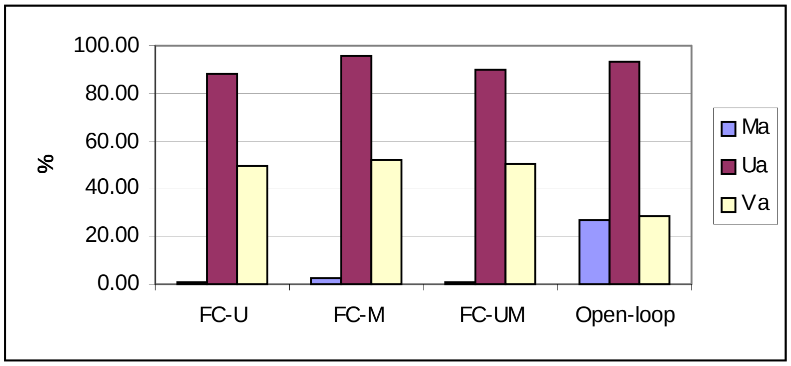
In summary, our evaluation results verify that our FCS algorithms can provide the following performance guarantees under the stability condition in Equation 22:

1) Stability in face of arrival overload and internal overload
2) System miss ratio and utilization stay close to the corresponding performance reference in steady state regardless of variations in task execution times
3) Satisfactory settling time and low overshoot in transient state

In addition to the performance profiles, the average performance of the FCS algorithms and the baseline are shown in Figure 12a (DM/PA) and Figure 12b (EDF/P). The considered performance metrics include the average miss ratio M_a, average CPU utilization U_a, and the Average Value Completion Ratio V_a defined as the total completed value divided by the total values of all the arriving tasks at the highest QoS level. V_a characterizes the utility and throughput of the system throughout the run. All of the above metrics is computed based on the performance throughout the run. Every data point in Figure 12ab is the mean of 5 repeated runs. The 90% confidence interval of each M_a, U_a, and V_a is within ±0.91%, ±0.23%, and ±1.25%, respectively, to its mean. We can see that all the FCS algorithms consistently outperform the open-loop baseline in terms of average miss ratio and the value completion ratio.



(a) DM/PA



(b) EDF/P

Figure 12: Average performance of FCS algorithms and the open-loop baseline
(*Ma*: Average Miss Ratio; *Ua*: Average Utilization; *Va*: Average Value Completion Ratio)

In summary, our evaluation results demonstrate that our three FCS algorithms provide robust and precise performance guarantees in term of utilization and miss ratio even when the workload significantly varies from the estimation. Furthermore, they also achieve satisfactory transient state performance profiles in response to arrival and internal overload. In contrast, an open loop QoS optimization algorithm fails to provide such guarantees when the workload deviates from the *a priori* estimation.

8. Conclusions

In summary, this paper presents a feedback control real-time scheduling (FCS) framework for adaptive real-time systems. An advantage of the FCS framework is its use of feedback control theory (rather than *ad hoc* solutions) as a scientific underpinning. We apply a control theory based design methodology to systematically design FCS algorithms to satisfy desired transient and steady state performance specifications of real-time systems. In particular, we establish an analytical model and analyses of FCS algorithms, which are major challenges and key steps for the design of adaptive real-time systems. Based on our model, we identify different types of real-time applications where each FCS algorithm can be applied. Performance evaluation results demonstrate that our analytically tuned FCS algorithms provide robust steady and transient state performance guarantees for periodic and aperiodic tasks even when the task execution time varied by as much as 100% from the estimation.

Acknowledgements

We want to especially thank Tarek Abdelzaher for his inspiring ideas and valuable discussions to this project. We also thank anonymous reviewers for helpful suggestions for improving this paper.

9. Reference

[1] T. F. Abdelzaher and C. Lu, “Schedulability Analysis and Utilization Bounds for Highly Scalable Real-Time Services,” *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, June 2001.

[2] T. F. Abdelzaher, “An Automated Profiling Subsystem for QoS-Aware Services,” *IEEE Real-Time Technology and Applications Symposium*, Washington D.C., June 2000.

[3] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, “QoS negotiation in real-time systems and its application to automatic flight control,” *IEEE Real-Time Technology and Applications Symposium*, June 1997.

[4] T. F. Abdelzaher and N. Bhatti, “Web Server QoS Management by Adaptive Content Delivery,” *International Workshop on Quality of Service*, 1999.

[5] T. F. Abdelzaher and C. Lu, “Modeling and Performance Control of Internet Servers”, *39th IEEE Conference on Decision and Control*, Sydney, Australia, December 2000.

[6] T. F. Abdelzaher and K. G. Shin, “End-host Architecture for QoS-Adaptive Communication,” *IEEE Real-Time Technology and Applications Symposium*, Denver, Colorado, June 1998.

[7] T. F. Abdelzaher and K. G. Shin, “QoS Provisioning with qContracts in Web and Multimedia Servers,” *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999, pp. 44-53.

[8] S. Brandt and G. Nutt, “A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage,” *IEEE Real-Time Systems Symposium*, December 1998.

[9] G. Buttazzo, G. Lipari, and L. Abeni, “Elastic Task Model for Adaptive Rate Control,” *IEEE Real-Time Systems Symposium*, Madrid, Spain, pp. 286-295, December 1998.

[10] M. Caccamo, G. Buttazzo, and L. Sha, “Capacity Sharing for Overrun Control,” *IEEE Real-Time Systems Symposium,* Orlando, FL, December 2000.

[11] J. Eker: "Flexible Embedded Control Systems-Design and Implementation." PhD-thesis, Lund Institute of Technology, Dec 1999.

[12] G. F. Franklin, J. D. Powell and M. L. Workman, *Digital Control of Dynamic Systems (3rd Ed.),* Addison-Wesley, 1998.

[13] C. V. Hollot, V. Misra, D. Towsley, and W. Gong, ”A Control Theoretic Analysis of RED,” *IEEE INFOCOM*, Anchorage, Alaska, April 2001.

[14] Mathworks Inc., http://www.mathworks.com/products/matlab.

[15] M. Klein, T. Ralya, B. Pollak, R. Obenza, M. G. Harbour, *A Practitioner's Handbook for Real-Time Analysis – Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, August 1993.

[16] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, “A Scalable Solution to the Multi-Resource QoS Problem,” *IEEE Real-Time Systems Symposium*, Phoenix, AZ, Dec 1999.

[17] J. P. Lehoczky, L. Sha and Y. Ding, “The Rate Monotonic Scheduling Algorithm – Exact Characterization and Average Case Behavior,” *IEEE Real-Time Systems Symposium*, 1989.

[18] B. Li, K. Nahrstedt, “A Control-based Middleware Framework for Quality of Service Adaptations,” *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms,* 17(9), pp. 1632-1650, Sep. 1999.

[19] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *Journal of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[20] J. W. S. Liu, et. al., “Algorithms for Scheduling Imprecise Computations”, *IEEE Computer*, Vol. 24, No. 5, May 1991.

[21] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, “A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers,” *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, June 2001.

[22] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son and M. Marley, “Performance Specifications and Metrics for Adaptive Real-Time Systems,” *IEEE Real-Time Systems Symposium*, Orlando, FL, Dec 2000.

[23] C. Lu, J. A. Stankovic, G. Tao and S. H. Son, “Design and Evaluation of a Feedback Control EDF Scheduling Algorithm,” *IEEE Real-Time Systems Symposium*, Phoenix, AZ, Dec 1999.

[24] C. Lu, *Feedback Control Real-Time Scheduling, University of Virginia CS PhD Dissertation 2001-6,* May 2001.

[25] S. Parekh, N. Gandhi, J. L. Hellerstein, D. Tilbury, T. S. Jayram, J. Bigus, “Using Control Theory to Achieve Service Level Objectives in Performance Management,” *IFIP/IEEE International Symposium on Integrated Network Management*, 2001

[26] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, “Practical solutions for QoS-based resource allocation problems,” *IEEE Real-Time Systems Symposium*, December 1998.

[27] D. Rosu, K. Schwan, S. Yalamanchili and R. Jha, "On Adaptive Resource Allocation for Complex Real-Time Applications," *IEEE Real-Time Systems Symposium*, Dec 1997.

[28] M. Ryu and S. Hong, “Toward Automatic Synthesis of Schedulable Real-Time Controllers”, *Integrated Computer-Aided Engineering*, 5(3) 261-277, 1998.

[29] S. S. Skiena and S. Skiena, *The Algorithm Design Manual*, Telos/Springer-Verlag, New York, November 1997.

[30] S. H. Son, R. Zimmerman, and J. Hansson, “An Adaptable Security Manager for Real-Time Transactions,” *Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, June 2000.

[31] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms*, Kluwer Academic Publishers, 1998.

[32] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao, “The Case for Feedback Control Real-Time Scheduling,” *EuroMicro Conference on Real-Time Systems*, York, UK, June 1999.

[33] D. C. Steere, et. al., "A Feedback-driven Proportion Allocator for Real-Rate Scheduling," *Symposium on Operating Systems Design and Implementation,* Feb 1999.

[34] W. Zhao, K. Ramamritham and J. A. Stankovic, “Preemptive Scheduling Under Time and Resource Constraints,” *IEEE Transactions on Computers* 36(8), 1987.