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 Engineering*
University of Virginia, Charlottesville, VA 22903
e-mail: {chenyang, stankovic, gt9s, son}@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 design methodology to systematically design FCS algorithms to satisfy their transient and steady state performance specifications. In particular, we establish a dynamic model and performance analysis of several feedback control scheduling algorithms, which is a major challenge and key step for the control design of adaptive real-time systems. We also generalize a FCS architecture that allows plug-ins of different real-time scheduling policies and QoS optimization algorithms. 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 considerably from the estimation.

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 [19][23] are static scheduling algorithms and represent one major paradigm for real-time scheduling. In dynamic scheduling, however, the scheduling algorithm does not have the 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) [23][36] 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 [41] can dynamically guarantee incoming tasks via on-line admission control and planning and thus is applicable in resource insufficient environments. Many other algorithms [36] 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 and EIA-9900895, and contract IJRP-9803-6 from the Ministry of Information and Communication of Korea. This paper is an extension to previous papers published in the Proceedings of IEEE Real-Time Systems Symposium [25][26].

(such as deadlines, precedence constraints, shared resources, jitter, etc.), 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, the Spring scheduling algorithm assumes complete knowledge of the task set except for their future release times. 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 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 [37]. 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 many 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 and estimation of worst-case resource requirements can result in an extremely expensive and underutilized system.

As a cost-effective approach to achieve performance guarantees in unpredictable environments, several adaptive scheduling algorithms have been recently developed (e.g., [4][6][7][13][20][22][27]). 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 he systematically design a scheduling algorithm to satisfy its performance specifications? The design methodology for automatic adaptive systems has been developed in feedback control theory [15][16]. However, feedback control theory has been mostly applied in mechanical and electrical systems. On the other hand, the modeling, analysis and implementation of adaptive real-time systems lead to significant research challenges to both real-time and control theory community. Recently, several works applied control theory to computing systems. For example, several papers [3][9][11][12][14][28][30][33][38][40] presented flexible or adaptive real-time (CPU) 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. Several other papers [5][10][18][20][30][31][39] presented adaptive CPU scheduling algorithms or QoS management architectures for computing systems such multimedia and communication systems. Transient and steady state performance of adaptive real-time systems has received special attention in recent years. For example, Brandt et. al. [10] evaluated a dynamic QoS manager by measuring the transient performance of applications in response to QoS adaptations. Rosu et. al. [31] proposed a set of performance metrics to capture the transient responsiveness of adaptations and its impact on applications. The paper proposed metrics that is 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 system from performance specifications of desired dynamic response. In this paper we present *feedback control real-time scheduling* (*FCS*), a unified framework that maps QoS control in adaptive real-time systems to feedback control theory. Our control theoretical framework includes the following elements:

- A feedback control scheduling architecture that maps the feedback control structure to adaptive resource scheduling in real-time systems [26],

- A set of performance specifications and metrics to characterize both transient and steady state performance of adaptive real-time systems [25], and
- A control theory based design methodology for resource scheduling algorithms to satisfy their performance specifications.

In contrast with *ad hoc* approaches that often rely on laborious design/tuning/testing iterations, our framework enables system designers to systematically design adaptive real-time systems with established analytical methods to achieved desired performance guarantees in unpredictable environments. Our previous work on feedback control real-time scheduling has appeared in [25][26]. The contributions of the previous work covered the first two elements of the FCS framework listed above. In this paper, we present recent advances in this framework including the following new major contributions:

- An analytical model and analysis of feedback control scheduling algorithms, which is a major challenge and key step for the control design of adaptive real-time systems,
- A generalized scheduling architecture that allows plug-ins of different real-time scheduling policies and QoS optimization algorithms,
- Identification of the applicability of FCS algorithms to different types of real-time applications, and
- Performance evaluation results that shows that the analytically tuned FCS algorithms can achieve desired steady and transient state performance for unpredictable periodic and aperiodic tasks.

In the rest of this paper, the feedback control real-time scheduling architecture is described in Section 2. We describe the performance specifications and metrics in Section 3. The control theory based design methodology is presented in Section 4. We establish an analytical model for a real-time system in Section 5. Based on the model, we present the design and control analysis of a set of FCS algorithms in Section 6. We present the performance evaluation results of these scheduling algorithms in Section 7. We then qualitatively compare FCS algorithms with several existing scheduling paradigms in Section 8. Finally, we conclude this paper in Section 9.

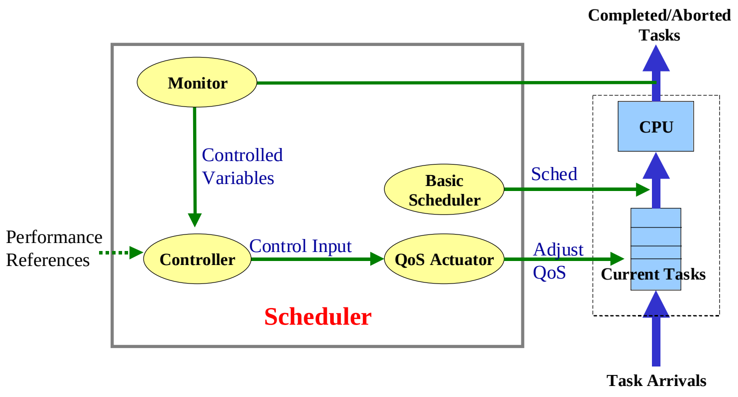


Figure 1: Feedback Control Real-Time Scheduling Architecture

2. Feedback Control Real-Time Scheduling Architecture

Our feedback control real-time scheduling (FCS) architecture is composed of four parts: a task model, a set of control related variables, a feedback control loop that maps a feedback control system structure to real-time scheduling, and a Basic Scheduler (Figure 1).

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 [10], embedded digital control systems [12], and systems that support imprecise computation [23] or flexible security [35], 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 [10][12][33]. 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 in order to achieve desired system performance. 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 $((k\text{-}1)W, kW)$. 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 $((k\text{-}1)W, kW)$. CPU utilization is regarded as a controlled variable for real-time systems due to cost and throughput considerations. CPU utilization is important also because the 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 rational for choosing the total estimated utilization as a manipulated variable is that most real-time scheduling policies (such as EDF and Rate/Deadline Monotonic) can guarantee no deadline misses when the system 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)*. This 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 exceeds 100% while *B(k)* does not have this boundary.

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, i.e., $B(k+1) = B(k) + D_B(k)$, 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)*, the QoS Actuator calls a QoS optimization algorithm (see Section 7.1) to maximize the system value. In the simplest form, each task only 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 [25].

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 impacts 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 (see Section 6.4).

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. Our FCS architecture establishes a mapping from real-time scheduling to a typical structure of feedback control systems. This step enables us to treat a real-time system as a feedback control system and utilize feedback control theory to design the system rather than developing *ad hoc* algorithms.

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. Dynamic (transient and steady state) behavior of adaptive real-time systems upon load or resource changes has received special attention in recent years. Transient behavior of an adaptive system represents the responsiveness and efficiency of QoS adaptation in reacting to changes in run-time conditions, and steady-state behavior describes a system's long-term performance after its transient response settles. 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 [25][31] 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 [15].

3.1. Performance Profile

The performance profile characterizes important transient and steady state properties of a system in terms of its controlled variables. Note that when the sampling period *W* is small, *M(k)* and *U(k)* approximates the instantaneous system performance at the sampling instant *k*. 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 [25]. From the control theory point of view, a real-time system transits from the *steady state* to the *transient state* when the Controller changes a controlled variable significantly from its initial value. After a time interval in the transient state, the system may settle down to a new steady state. For real-time systems, the steady state can be defined as a state when *M(k)* is within ε% (e.g., we assume ε% = 2% in this paper) of the reference. The performance profile describes the system performance in both transient state and steady state as follows.

[1] The performance profile has been called the *miss-ratio profile* in [25]. The performance profile can be generalized to other metrics such as response time, throughput, and value-cognizant metrics.

- *Stability*: A system is (BIBO) stable if its controlled variables, miss ratio $M(k)$ or utilization $U(k)$, are always bounded for bounded references. Although both miss ratio $M(k)$ and utilization $U(k)$ are naturally bounded in the range [0, 1], stability is a necessary condition to prevent the controlled variables from significant divergence from the reference values.

- *Transient-state response* represents the responsiveness and efficiency of QoS adaptation in reacting to changes in run-time conditions.
  - *Overshoot* M_o and U_o: The highest values of controlled variables in transient state. Overshoot represents the worst-case transient performance of a system in response to the load profile. Overshoot is often defined as the maximum amount that the system overshoots its reference divided by its reference, i.e., $M_o = (M_{max} - M_S) / M_S$, $U_o = (U_{max} - U_S) / U_S$ Overshoot is an important metric because a high transient miss-ratio or utilization can cause system failure in many systems such as robots and media streaming [10].
  - S*ettling time* T_s*:* The time it takes the system miss-ratio to enter a steady state after the Controller is turned on. The settling time represents how fast the system can settle down to steady state with desired performance. This metric has also been called reaction time or recovery time [31].

- *Steady-state error* E_{SU} and E_{SU}: The difference between the average values of a controlled variable in steady state and its reference. The steady state error characterizes how precise the system can enforce the reference (desired performance) 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 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.

The performance profile establishes a mapping from metrics of adaptive real-time systems to dynamic response of control systems. This mapping enables system designers to apply established control theory techniques to achieve stability, and meet transient and steady state specifications.

3.2. Load Profile

Although system load is not known *a priori* for real-time systems in unpredictable environments, its performance can be specified under a set of representative load profiles borrowed from control theory; namely, the *step load* and the *ramp load*. Similar types of signals have been widely used to generate canonical system responses in control theory. In the context of real-time systems, the step load represents the worst-case load variation, and the ramp load represents a nominal form of load variation. The *load profile* $L(k)$ of a system at sampling instant k is defined as the total actual utilization *at the highest QoS level* of all the tasks in the system, i.e., $L(k) = \sum_i A_i[N\text{-}1](k)$. 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 load L_m ($>L_n$) and stays constant after the jump. In real systems, load variations typically occur gradually over a finite amount of time. Gradual load changes are easier to control and adapt to than sudden load changes.

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

In practice the load profiles are application-specific based on the load characteristics and system requirement. It is usually necessary to specify and test the system performance under a series of load profiles with different types/parameters. The load profile is an *abstraction* 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.

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, we now establish a design methodology based on feedback control theory. Using this design methodology, a system designer can systematically design an adaptive scheduler to satisfy the system's performance specifications with established analytical methods. This methodology is in contrast with existing *ad hoc* approaches that depend on laborious design/tuning/testing iterations. Our design methodology works as follows.

1) The system designer specifies the desired dynamic behavior with transient and steady state performance metrics. This step maps from the existing metrics of adaptive real-time systems to the dynamic responses of control systems in control theory.

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 performance (controlled variables) of a system with differential/difference equations or state matrices. Note that for the purpose of control design, the scheduling policy of the Basic Scheduler is part of the system model. Modeling is important because it provides a basis for the analytical design of the Controller. Two different approaches can be used to establish the dynamic model of a system. In the analytical approach, a system designer describes a system directly with mathematical equations based on the knowledge of the system dynamics. When such knowledge is not available, the system identification approach [8] can be used to estimate the system model based on profiling experiments. In this paper, we adopt the analytical approach to model a real-time system (Section 5). An example of system identification to model web servers is presented in [2].

3) Based on the performance specs and system model from step 1) and 2), the system designer applies existing mathematical techniques (i.e., the Root Locus method, frequency design, or state based design) of feedback control theory [15] to design the feedback scheduling algorithm with *analytic* guarantees on the desired transient and steady-state behavior at run-time. This step is similar to the process that a control engineer uses to design a Controller for a feedback-control system to achieve desired dynamic responses.

To demonstrate the strength of our control theory based methodology, we apply this methodology (Section 6) to design scheduling algorithms to analytically guarantee the satisfaction of a set of performance specifications in face of workload variations.

5. Modeling the Controlled Real-Time System

Before applying analytical methods to design the Controller, we need to establish a mathematical model to approximate the *controlled system* in the FCS architecture (Figure 1). The controlled system includes the QoS Actuator, the scheduled real-time system, the Basic Scheduler, and the Monitor. The input to the controlled system is the control input, i.e., the change in the total estimated utilization $D_B(k)$. The output of the controlled system includes the controlled variables, i.e., 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, it is usually adequate to approximate such a system with a linear model for the purpose of control design because of the robustness of feedback control with regard to nominal system variations. The block diagram of the controlled system model is illustrated in Figure 2. We now derive the model from the control input, $D_B(z)$, through each block in Figure 2. The goal is to derive the transfer function from the control input to the output, the controlled variables $U(z)$ and $M(z)$. While the block diagram in Figure 2 is expressed in the z-domain that is amenable to control design, we describe the equivalent notations and formula in time domain in the following for clarity of presentation (e.g., $D_B(k)$ in the time domain is equivalent to $D_B(z)$ in the z-domain).

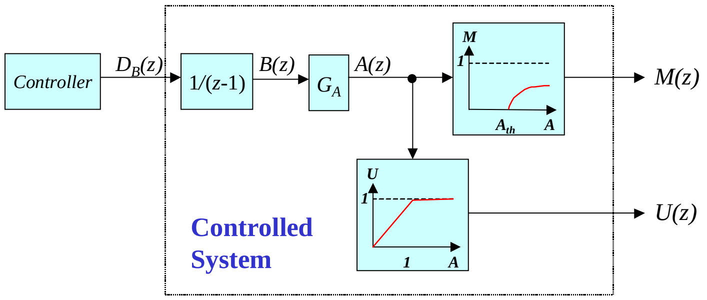


Figure 2: The Model of the Controlled System

Staring from the control input $D_B(k)$, the total estimated utilization $B(k)$ is the integration of the control input $D_B(k)$:

$$B(k+1) = B(k) + D_B(k) \qquad (1)$$

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 $B(k)$:

$$A(k) = G_a(k)B(k) \qquad (2)$$

where $G_a(k)$, called the *utilization ratio*, is a time-variant ratio that represents the extent of workload variation in term 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 should 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:

$$A(k) = G_A B(k) \qquad (3)$$

For systems without arriving-time admission control, the load profile $L(k)$ should be modeled as an *external disturbance* to the total requested utilization, i.e., $A(k) = G_A B(k) + L(k)$. This is because new tasks' requested utilization is not accounted for in the control input if the QoS Actuator is not invoked upon their arrivals. However, if the QoS Actuator is invoked upon the arrival of each task, the requested utilization of admitted tasks is part of the control input. The arriving-time admission control thus cancels the disturbance caused by load profiles. Since we focus on systems with arriving-time admission control in this paper, the load profile is not part of our model.

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 $D_B(k) \neq 0$. Saturation complicates the control design because the controlled variables become unresponsive to the control in their saturation zones. When CPU is underutilized, the utilization *U(k)* is outside its saturation zone and stays close to the total requested utilization *A(k)*:

$$U(k) = A(k) \qquad (A(k) \le 1) \qquad (4)$$

However, Since utilization can never exceed 100%, *U(k)* saturates when CPU is *overloaded*:

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

In contrast, the miss ratio *M(k)* saturates at 0 when the CPU is *underutilized*, i.e., the total requested utilization is below a threshold $A_{th}(k)$, called the (schedulable) *utilization threshold*:

$$M(k) = 0 \qquad (A(k) \le A_{th}(k)) \qquad (6)$$

In existing real-time scheduling theory, *schedulable utilization bounds* have been derived for various real-time scheduling policies under different workload assumptions (e.g., [1][19][23][36]). 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 $A_b \le A_{th}(k)$). Since it is always true that $A_{th}(k) \le 1$, the saturation zones of CPU utilization ($A(k) \ge 1$) and that of Miss Ratio ($A(k) \le A_{th}(k)$) are guaranteed to be mutually exclusive. This property means that at any instant of time, at least one of the controlled variables does not saturate. Note that different scheduling policies in the Basic Scheduler usually lead to 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 reference (see Section 6.4.4).

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 *operation point* ($A(k) = A_{th}(k)$).

$$G_M = \frac{dM(k)}{dA(k)} \qquad (A(k) = A_{th}(k)) \qquad (7)$$

In practice, G_M, the *miss ratio factor*, can be estimated experimentally by plotting a *M(k)* curve as a function of *A(k)* based on experimental data and measuring its slope at the vicinity of the point where *M(k)* starts to become nonzero (see Section 7.3). At the vicinity of $A(k) = A_{th}(k)$, we have the following linearized formula:

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

Note that different scheduling policies in the Basic Scheduler usually have a different miss ratio factor, and hence the choice of the scheduling policy has a direct impact on the Controller parameters (see Section 6.4.4).

From formula (1)-(8), we can derive a transfer function for each controlled variable when it is outside its saturation zone:

Utilization control: Under the condition that $A(k) < 1$, there exists a transfer function $H_U(z)$ from the control input $D_B(z)$ to CPU utilization $U(z)$, i.e., $U(z) = P_U(z)D_B(z)$ and

$$P_U(z) = G_A / (z\text{-}1) \qquad (A(k) < 1) \qquad (9)$$

Miss ratio control: Under the condition that $A(k) > A_{th}(k)$, there exists a transfer function $H_M(z)$ from the control input $D_B(z)$ to Miss Ratio $M(z)$, i.e., $M(z) = P_M(z)D_B(z)$ and

$$P_M(z) = G_A G_M / (z-1) \qquad (A(k) > A_{th}(k)) \quad (10)$$

In summary, the controlled system can be modeled as a first order transfer function (Equations (9)(10)) with a saturation zone (Equations (5)(6)) for each control variable, utilization $U(k)$ and miss ratio $M(k)$. Note that the saturation properties cause the controlled system to be non-linear and lead to special challenges for the Controller design.

6. Design of Feedback Control Real-Time Scheduling Algorithms

In this section, we apply control design methods and analysis to the Controller, the key component of feedback control scheduling algorithms. First, we define the performance specifications for a real-time system. We then present the control algorithm and the model of the feedback control loop for each controlled variable. Based on the analytical system models, we apply a control design method called the Root Locus method to tune the Controller and develop mathematical analysis on the performance properties of the resultant Controller. We then design several FCS algorithms to handle the saturation zones in different types of real-time systems.

6.1. Performance Specifications of a Real-Time System

As an example of performance specifications of real-time systems, we assume a specific real-time system has the performance specifications as listed in Table 1. The sampling period W = 0.5 sec. The transient and steady state performance requirements include the following. (1) The miss ratio of the system should remain stable in face of a step load of 200%. (2) The system should settle down to steady state within 15 sec after the beginning of the step load, and the highest miss-ratio during the transient state should be lower than 15%. (4) The system should have an average miss-ratio of less than 1% in steady state, and this steady state miss ratio should be achieved regardless of variations in task execution time.

Load Profile	***SL(0, 200%)***
M_{mox}	< 15%
T_s	< 30 sec
M_s	< 1%
S_{AE}	= 0%
W	0.5 sec

Table 1: Performance Specifications of a Real-Time System

6.2. Design of the Controller

At each sampling point, the Controller computes a control input $D_B(k)$, the change in the total estimated requested utilization, based on the miss ratio error $E_M(k) = M_S(k) - M(k)$ and/or the CPU utilization error $E_U(k) = U_S(k) - U(k)$. In this section, we focus on a Controller for a single controlled variable (see Figure 3). The goal of a Controller is to satisfy the performance specifications in Table 1. Since a same control function (with different parameters) can be used for both controlled variables, we use the same symbol $E(k)$ to represent the miss ratio error $E_M(k)$ and the utilization error $E_M(k)$ in the rest of this section.

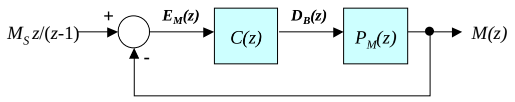
The Controller uses a PI (Proportional-Integral) control function [15] to compute the control input. A digital form of the PI control function is in Equations (11) and (12). Equations (11) and (12) are equivalent but Equation (12) is more efficient than Equation (11) at run-time. The transfer function of PI control $C(z)$ in the z-domain is in Equation (13).

$$D_B(k) = K_P(E(k) + K_I \sum_{j=0}^{k} E(j)) \quad (11)$$

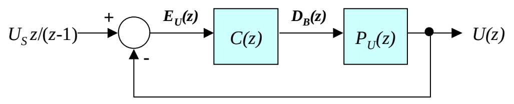
$$D_B(k) = D_B(k-1) + K_P((K_I+1)E(k) - E(k-1)) \quad (12)$$

$$C(z) = \frac{g(z-r)}{z-1} \quad (g = K_P(K_I+1),\ r = \frac{1}{K_I+1}) \quad (13)$$

PI control is a widely used control function that can achieve robust performance in first and second order systems. The performance of the real-time system depends on the values of the Controller parameters. An *ad hoc* approach to design the Controller is to repeat numerous experiments on different values of the parameters. In our work, we apply established control theory methods to tune the parameters analytically to guarantee the performance specifications. We first tune the Controller for each of the controlled variables in Section 6.3 based on the linear models of the controlled system (Equations (14)). Due to the saturation properties, the performance of the closed loop system may deviate from the linear case. We address this issue in Section 6.4.



(a) Miss Ratio Feedback Control Loop



(b) CPU Utilization Feedback Control Loop

Figure 3: Models of Feedback Control Loops

6.3. Control Tuning and Analysis

For the purpose of control design, the input of a closed loop system with a single (utilization or miss ratio) Controller is the performance reference M_S or U_S, and the output is the controlled variable miss ratio *M(k)* or utilization *U(k)*. Given the model of the controlled system *P(z)* (Equations (9)(10)) and the Controller *C(z)* (Equation (13)), the transfer function of each feedback control loop can be established:

$$H_M(z) = \frac{C(z)P_M(z)}{1 + C(z)P_M(z)} \qquad H_U(z) = \frac{C(z)P_U(z)}{1 + C(z)P_U(z)} \quad (14)$$

$$M(z) = \frac{M_S z}{z-1} H_M(z) \qquad U(z) = \frac{U_S z}{z-1} H_U(z) \quad (15)$$

Based on the above analytical models, we can apply control theory to tune the Controller parameters and analyze the properties of the system performance. We now present the tuning and analysis of the utilization Controller and the miss ratio Controller.

According to control theory, the performance of a system depends on the poles of its closed loop transfer function. Thus the control design problem can be viewed as a pole placement problem. The Root Locus method is a graphical technique that plots the traces of poles of a closed-loop system on the *z*-plane as its Controller parameters changes. We used the Root Locus tool of MATLAB [17] to tune the Controller parameters so that the performance specifications can be satisfied. The detailed tuning procedure can be found in control textbooks such as [15]. Consider the utilization control loop and assume the system workload has a worst-case utilization ratio $G_A = 2$, we can place the closed-loop poles to $p_0 = 0.89$ and $p_1 = -0.19$ with the following Controller parameters:

$$K_P = 0.5, K_I = 0.1 \qquad \text{(Utilization Controller)} \qquad (16)$$

Because the transfer function of the miss ratio control loop has an extra process gain G_M (Equation (9)(10)), the miss ratio Controller should be set to:

$$K_P = 0.5/G_M, K_I = 0.1 \quad \text{(Miss Ratio Controller)} \qquad (17)$$

The miss ratio control loop with the parameters in Equation (17) has the same closed loop poles and same performance profiles as the utilization control loop with the parameters in Equation (16). We now use control theory to analyze the performance of the utilization and miss ratio control loops.

6.3.1. Stability

With the Controller parameters in Equations (16)(17), both the utilization control and miss ratio control can guarantee stability because all their closed-loop poles are in the unit circle of the z-plane, i.e., $|p_j| < 1$ $(0 \le j \le 1)$.

6.3.2. Settling Time

The settling time decreases as the radius of the closed-loop poles $|p_j|$ $(0 \le j \le 1)$ decreases. Our closed loop poles lead to a theoretical settling time $T_s = 7.5$ sec. This result means that with the utilization Controller, the utilization *U(k)* can settle down to within ±2% of the desired utilization U_S within 7.5 sec after the Controller is turned on. Similarly, with the miss ratio Controller, the miss ratio *M(k)* can also settle down to within ±2% of the desired miss ratio M_S within 7.5 sec after the Controller is turned on.

6.3.3. Overshoot

The overshoot decreases as the damping ratio of the closed-loop transfer function (Equation (14)) decreases. Our closed loop poles result in a theoretical overshoot of 29% in term of utilization *U(k)* for utilization control and miss ratio *M(k)* for miss ratio control, respectively. For example, if the reference utilization is $U_S = 80\%$, the utilization control loop should achieve a theoretical maximum CPU utilization of $U_{max} = 0.8*(1+0.29) = 103\%$. Practically this means that the CPU utilization will saturate at 100% and cause transient deadline misses at the overshoot instant. In the case of the miss ratio control loop, suppose the reference miss ratio $M_S = 1\%$, the system should get a maximum miss ratio of $M_{max} = 0.01*(1+0.29) = 1.29\%$ at the overshoot instant.

6.3.4. Steady State Error

Both FC-U and FC-M can achieve zero steady state error, i.e., $E_s = 0$. Applying the Final Value Theorem [14] to the closed loop transfer functions (Equation 14), we have the following result regarding the steady state error.

Theorem 1. Assuming stability, let R_S denote the reference value (U_S or M_S), the final values of the error *E(k)* of the system scheduled by FC-U ($E(k) = U_S - U(k)$) or FC-M ($E(k) = M_S - M(k)$) modeled by Equation (14) is

$$E(\infty) = \lim_{z \to 1} (z-1) \frac{R_S z}{z-1} \frac{1}{1 + C(z) P_U(z)} = 0 \qquad (18)$$

This result means that the miss ratio control achieves its desired miss ratio M_S in steady state, and the utilization control achieves the desired utilization U_S in steady state. If U_s is less than the utilization threshold A_{th}, this result also guarantees that the utilization control can achieve zero miss ratio in steady state.

6.3.5. Sensitivity

Because the final values of the error *E(k)* is *independent* of the utilization ratio G_a and the miss ratio G_M (Equation (18)), we have the following theorem regarding the sensitivity of both feedback control loops.

> **Theorem 2.** Under the condition that the system remains stable, systems scheduled by the utilization control or miss ratio control always achieves the desired CPU utilization U_S or the desired miss ratio M_S *regardless* of the variations of task execution times, inter-arrival-times, or miss ratio factors.

Theorem 2 verifies the robustness of utilization control and miss ratio control in guaranteeing desired steady state performance in unpredictable environments.

6.4. FCS Algorithms

We present the design of FCS algorithms based on the utilization and/or miss ratio control to achieve the performance specifications (Section 6.1) in different types of real-time systems. We also discuss the impacts of the basic scheduling policy and workloads on the FCS algorithms design.

6.4.1. FC-U: Feedback Utilization Control

The FC-U scheduling algorithm uses a utilization control loop (see Figure 3(b)) to control the utilization *U(k)*. FC-U can guarantee that the system has zero miss ratio in steady state if its reference $U_S \leq A_{th}$ where A_{th} is the schedulable utilization threshold of the system.

Because CPU utilization *U(k)* saturates at 100%, FC-U cannot detect how severely the system is overloaded when *U(k)* remained at 100%. The consequence of this problem is that FC-U can have a longer settling time than the analysis results based on the linear model (Figure 3) in severely overload conditions. 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 (14)). For example, suppose the total requested utilization *A(k)* = 200% and the utilization reference is 99%, the error measured by the Controller would be $E_U = U_S - U(k) = 0.99 - 1 = -0.01$; however, the error would have been $E_U = U_S - U(k) = 0.99 - 2 = -1.01$ 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% (e.g., $U_S \leq$ 90%) 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 zero miss ratio in steady state if its reference $U_S \leq A_b \leq A_{th}$. For example, FC-U can perform well in a system with EDF scheduling and periodic and independent tasks because its utilization bound is 100%. However, FC-U is not applicable for systems whose utilization bounds are unknown or significantly 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

The FC-M scheduling algorithm uses a miss ratio control loop (see Figure 3(a)) to *directly* controls the system miss ratio *M(k)* (FC-M has been called FC-EDF if EDF is plugged into the Basic Scheduler [26]). Compared with FC-U, the advantage of FC-M is that it does not depend on any knowledge about the utilization bound and thus is applicable in many real-world systems. 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. Therefore FC-M can have a longer settling time than the analysis results based on the linear model (Figure 3) in severely underutilized conditions. The smaller the reference is, the longer the settling time will be. 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 (Equation (14)). For example, suppose the total requested utilization $A(k)$ = 10% and the miss ratio reference is 1%, the error measured by the Controller would be $E_M = M_S - M(k) = 0.01 - 0 = 0.01$; 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 stay 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 (e.g., $M_S \geq 1\%$) 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 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* [26]. Since FC-M cannot guarantee zero deadline miss ratio in steady state, it is applicable only to soft real-time systems that can tolerate sporadic deadline misses in steady state.

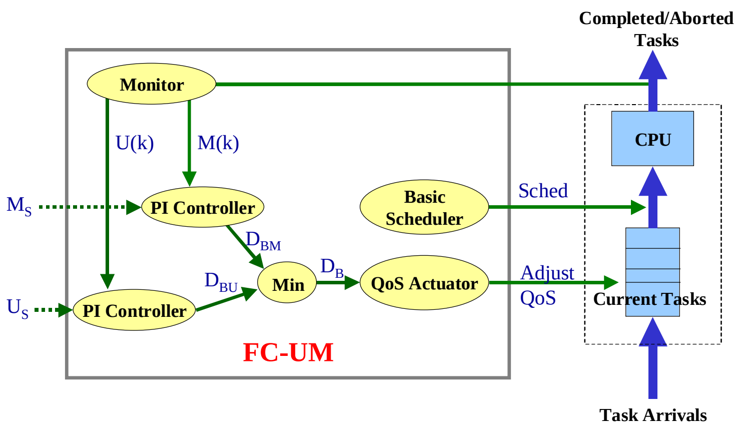


Figure 4: The FC-UM Algorithm

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

The FC-UM algorithm (also called FC-EDF^2 if EDF is plugged into the Basic Scheduler [25]) integrates miss-ratio control and utilization control (Figure 4) together to combine the advantages of FC-U and FC-M. 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 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. FC-UM utilizes the *min* operator and an *integrator anti-windup* [16] technique to achieve smooth transition between the two Controllers. *Integrator anti-windup* means that each PI Controller turns off the error integration at a sampling instant if its control input is larger than the other control input.

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 due to 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 *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 $U_S \le A_{th}(k)$, the utilization control dominates in steady state and guarantees the total requested utilization to stay close to its utilization threshold $A_{th}(k)$ and miss ratio $M(k) = M_S$ in steady state.

Therefore, 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. 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 demonstrate that FC-UM achieves satisfactory performance. The rigorous analysis of the integrated Controller is left for our future work.

6.4.4. Impacts of Scheduling Policies and Applications on the Design FCS algorithms

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 all tasks are periodic and independent, Liu and Layland proved that the schedulable utilization bound of EDF and RM is 100% and 69%, respectively [23]. Recently, Abdelzaher proved that the schedulable utilization bound for Deadline Monotonic scheduling is 62.5% for independent aperiodic tasks [1]. Other papers established schedulable utilization bounds for other types of workloads (e.g., [19][36]). Since FC-U can guarantee miss ratio $M(k) = 0$ in steady state if its utilization reference $U_S \le A_b$, 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 processing can be blocked frequently due to concurrency control, disk I/O, and network congestion. The task arrivals patterns may 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 become even more difficult due to unpredictable kernel activities such as interrupt handling. Another issue is the utilization bound can be significantly pessimistic for the current specific workload 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% [20]. 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 policy 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 (17)), the scheduling policy and workload can directly affect the parameter of the miss ratio Controller. For example, our previous experiments showed that while EDF with a periodic task set lead to a miss ratio factor $G_M = 1$ (details of experiments is skipped due to space limitations), Deadline Monotonic (DM) with a mixed periodic and aperiodic task set has a much smaller miss ratio factor $G_M = 0.447$ (see Section 7.3). This result means that if the case of DM with the mixed task set, K_P of the miss ratio Controller should be 2.24 times of the K_P in the case of EDF with the aperiodic task set in order to achieve similar performance profiles.

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 proved that the resultant FCS algorithms could satisfy the transient and steady state performance specifications. This design methodology is in contrast with existing *ad hoc* design methods that depend on laborious design and testing iterations. We also investigated the impacts of scheduling policies and workloads on the design of FCS algorithms.

7. Experiments

In this section, we describe simulation experiments to evaluate the performance of our FCS algorithms and the correctness of our control design. Our previous evaluation of FCS algorithms focused on the EDF scheduling policy and periodic task sets [25][26]. To demonstrate the robustness of our FCS algorithms, we present experimental results using the Deadline Monotonic policy as the underlying Basic Scheduler and mixed periodic/aperiodic task sets.

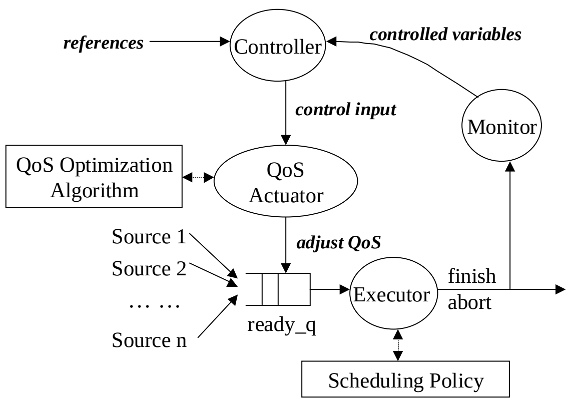


Figure 5: The FCS Simulator

7.1. Simulation Model

The FCS architecture is implemented on a uniprocessor simulator called FECSIM [26] of a soft real-time system. The simulator (Figure 5) 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 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. The QoS Actuator can also be turned off for system profiling experiments (see Section 7.3). The configurations of the simulator are as follows.

7.1.1. Scheduling policy

A Deadline Monotonic [1] policy is used in all the experiments in this paper. In the DM policy, each periodic or aperiodic task is assigned a fixed priority based on its (fixed) relative deadlines. Deadline

Monotonic has been proved to be the optimal static scheduling policy in term of maximizing the utilization bound [1].

7.1.2. QoS optimization

A Highest-Value-Density-First (HVDF) QoS assignment algorithm [34] 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 (e.g., 80%) is used by open loop scheduling algorithms, while our FCS algorithms dynamically change the threshold $U_C = B(k+1) = B(k) + DB(k)$ at each sampling instant. When tasks' utilization is small (e.g., < 1%) and there is no deadline misses, the HVDF can approximate the optimal value under the utilization constraint. However, if the actual requested utilization is unknown (as the case in unpredictable environments), the QoS optimization algorithm cannot always guarantee no deadline misses and maximize the system value when used with an open loop scheduling algorithm.

We should note that FCS has a general architecture that can incorporate different real-time scheduling policies and QoS optimization algorithms (although the scheduling policy does affect the design of FCS algorithms and the Controller parameters as discussed in Section 6.4.4)). Our work focuses on the steady and transient state performance of the feedback control loop rather than evaluating the best basic scheduling policy or QoS optimization algorithm.

7.2. Workload

The workload is composed of 50% aperiodic tasks and 50% periodic tasks where each task follows the task model described in Section 2.1. This task model 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. 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 in this paper.

$EE_i[j]$:	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 followed a normal distribution $N(AE_i, AE_i^{1/2})$, where the average execution time $AE_i[j] = G_a'ET_i[j]$. G_a', called the *execution time factor,* is a tunable workload parameter that approximated the utilization ratio G_a in Equation (2). The larger G_a' is, the more pessimistic is the estimation of execution time. For example, in our experiments, we set $G_a' = 2.0$, which means that the estimated execution time is twice the average execution time.
$D_i[j]$:	All QoS levels of a task T_i have a same relative deadline $D_i = 10(F_i + 1)ET_i[2]$, 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., $V_i[j] = w_i\ ET_i[j]$. The weight w_i follows a uniform distribution in the range [1, 5].

For periodic tasks:

$P_i[j]$:	All QoS levels of a task T_i have a same *period* that equals its deadline $P_i = D_i$.

For 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 $AI_i = D_i$.

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

7.3. Saturation of Control Variables

In this section, we profile the controlled system with a set of experiments to verify the saturation properties of the controlled variables and measure the miss ratio factor G_M in Equation (8). Different step loads are used to stress the system (with both the Controller and the QoS Actuator turned off) for 60 sec. We plot the average CPU utilization and miss ratio in Figure 6. Each point in Figure 6 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. We can see that the CPU utilization saturates at 100% after the total requested utilization $A(k)$ exceeded 100%. Miss ratio saturates at zero when the total requested utilization $A(k)$ is below 90%. Deadline misses starts to occur when the total requested utilization reaches 90%. This shows that the utilization threshold A_{th} located between 80% and 90% for our workload. When the miss ratio is outside of its saturation zone, it increases as the total requested utilization increases. Because the miss ratio reference is usually small, 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 6, the maximum slope is 0.447 occurred when the total requested utilization $A(k)$ increases from 100% to 110%, i.e., $G_M \approx 0.447$. This result provides a basis for tuning the miss ratio Controller (Equation (17)). Note that although the actual miss ratio factor may deviate from the profiling results at run-time, the steady state performance of the FCS algorithms are robust with regard to the miss ratio factor according to Theorem 2. Therefore, the measured miss ratio factor is usually adequate for the purpose of Controller tuning.

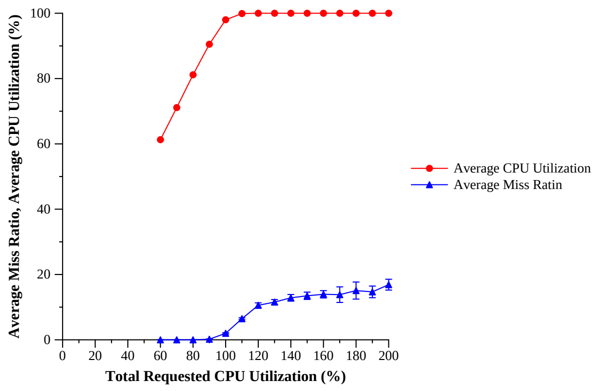


Figure 6: Total Requested Utilization vs. Control Variables

7.4. Evaluation Results of FCS Algorithms

In this Section, we present evaluation results of three FCS algorithms, FC-U, FC-M, and FC-UM. An open loop QoS optimization algorithm with a fixed threshold U_{th} = 80% is used as a baseline. All the FCS algorithms start with an initial threshold $B[0] = 0$. The same scheduling policy (Deadline Monotonic) and QoS optimization algorithm (the HVDF algorithm) is used for all FCS algorithms and the baseline. The configurations of the FCS algorithms are summarized in Table 2. The Controller parameters are based on the analytical tuning results in Section 6.3.

The sampled miss ratio $M(k)$ and CPU utilization $U(k)$ of a typical run for each of the scheduling algorithms are illustrated in Figure 7. A step load SL(0, 200%) was used to stress the system in all the runs. The execution time factor $G_a' = 2$, i.e., the average execution time of each task was twice of the estimation. Each run lasted for 60 sec. We now describe the results for each of the scheduling algorithms.

	FC-UM	
	FC-U	FC-M
K_P	0.5	1.1
K_I	0.1	0.1
Reference	$U_S = 80\%$	$M_S = 1\%$

Table 2: Configurations of FCS Algorithms

7.4.1. FC-U

The performance of FC-U is illustrated in Figure 7(a). We can see that after the Controller is turned on at time 0, FC-U increases task QoS levels and causes the CPU utilization to overshoot to $U_O = 93\%$ at time 0.5 sec. The overshoot also causes a transient miss ratio of 0.5%. The overshoot is lower than the theoretical prediction of 100% possibly due to the system noise caused by random execution times and inter-arrival-times of aperiodic tasks. The settling time is difficult to measure due to the system noise. However, we can see that the CPU utilization *U(k)* settles to the vicinity of the reference $U_S = 80\%$ within 5 sec. The CPU utilization *U(k)* remains stable all through the run and close to 80% after 5 sec, which shows the system error is close to zero in steady state. Because the utilization threshold $A_{th} < 80\%$, the miss ratio *M(k)* = 0 in all the sampling instant except for the overshoot at time 0.5 sec.

7.4.2. FC-M

The performance of FC-M is illustrated in Figure 7(b). After the Controller is turned on at time 0, FC-M dynamically increases task QoS levels and causes the CPU utilization to increase. Because the miss ratio *M(k)* remains in the saturation zone in the starting phase, the miss ratio stays at zero and the CPU utilization increases slower than the case of FC-U. At time 11 sec, the miss ratio overshot to $M_O = 6.09\%$ while the CPU utilization reached 99.36%. The overshoot is higher than the theoretical prediction of 1.23% (but still well within the performance specification of 15%) due to saturation and system noise. The system remains stable all through the run. The miss ratio *M(k)* remains close to 1% and below 5% at most of the sampling instants after 15 sec, which shows that the miss ratio error is close to zero in steady state. We should note that CPU utilization in the run of FC-M stays close to 95% after the system settles, which is significantly higher than the CPU utilization (close to 80%) in the case of FC-U. This is because by directly controlling the miss ratio, FC-M can change the CPU utilization to the vicinity of the utilization threshold, which is higher than the utilization reference of FC-U that is set to 80% *a priori*. Therefore, compared with FC-U, FC-M achieves higher CPU utilization and robustness with regard to utilization threshold variations at the cost of a small (but bounded) miss ratio in steady state.

7.4.3. FC-UM

The performance of FC-UM is illustrated in Figure 7(c). After the Controller is turned on at time 0, FC-UM dynamically increases task QoS levels and causes the CPU utilization to increase. Similar to the case of FC-M, the miss ratio stays at zero and the CPU utilization increases slower than the case of FC-U in the starting phase. Note that in the starting phase, the (saturated) miss ratio control has a smaller error ($E_M(k) = 0.01 – 0 = 0.01$) than the error of the utilization control (e.g., $E_U(0) = 0.9 – 0 = 0.9$), and hence the miss ratio dominates the control loop due to the *min* operation on control inputs from both Controllers. At time 11 sec, the miss ratio overshoots to $M_O = 1.19\%$ while the CPU utilization overshoots to 95.76%. Because the utilization threshold is lower than the utilization reference $U_S = 80\%$, the CPU utilization *U(k)* settles to 80% while the miss ratio settles to 0 by 15 sec. The system remains stable all through the run. The miss ratio *M(k)* remains close to 1% and below 5% at most of the sampling instants after 15 sec, which showed the system error is close to zero in steady state. This is because the utilization control has smaller errors and dominates the control loop when the system is approaching the steady state. For example, consider time 12 sec, the utilization error is $E_U(k) = 0.8 – 0.913 = -0.113$ while the miss ratio error is $E_M(k) = 0.01 – 0 = 0.01 > E_U(k)$. Note that if the utilization threshold were higher than the utilization reference, the miss ratio control would dominate the control loop and therefore miss ratio would stay close to the miss ratio set point M_S = 1%. In summary, FC-UM combines the advantages of both FC-U and FC-M by achieving zero miss ratio in the nominal case when the utilization reference is lower than the utilization threshold, and robustness with regard to utilization threshold variations.

7.4.4. Open Loop QoS Optimization Algorithm

In comparison with the FCS algorithms, the system scheduled by the open loop QoS optimization algorithm suffers from high miss ratios (see Figure 7(d)). This is because the task execution time is higher than the estimation (by a factor of 2) 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 considerably lower than the estimation [26]. This result demonstrates that open loop QoS optimization algorithms are incapable of maintaining satisfactory performance in face of workload variations.

In summary, our evaluation results demonstrate that our FCS algorithms achieve robust performance guarantees even when the workload significantly varies from the estimation, while an open loop QoS optimization algorithm fails to provide performance guarantees. The results also demonstrate that the FCS algorithms can satisfy transient and steady state performance specifications in Table 1 and verify the correctness of our analytical control design. In addition, the experiments also demonstrate the advantages of different FCS algorithms. In particular, FC-UM combines the advantages of both FC-U and FC-M by achieving zero miss ratio in the nominal case when the utilization reference is lower than the utilization threshold, and robustness with regard to utilization threshold variations.

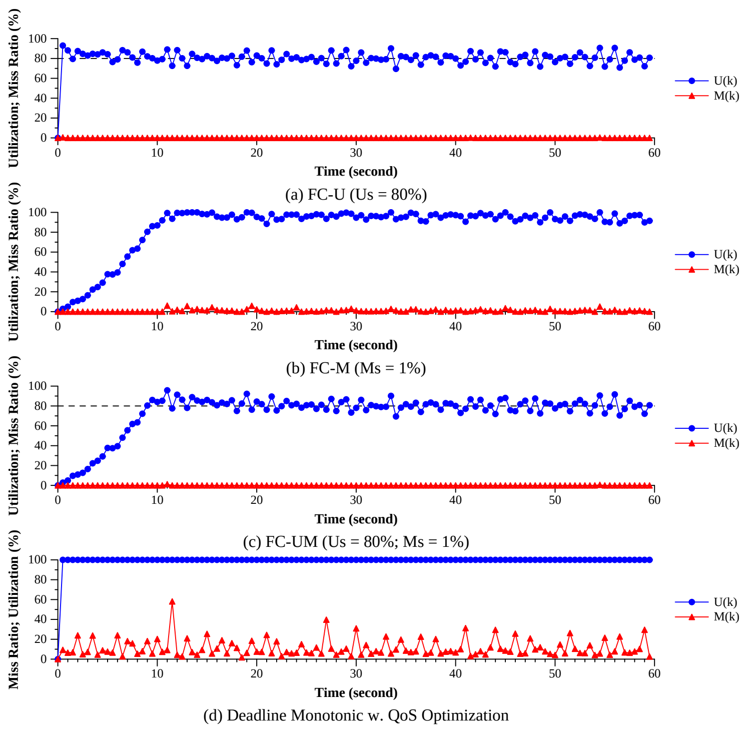


Figure 7: Typical Runs of Scheduling Algorithms

8. Comparison of Real-Time Scheduling Algorithms in Overload Conditions

We now qualitatively compare several existing real-time scheduling paradigms/algorithms (see Table 3). Our comparison is based two criteria, the required knowledge of the workload by a scheduler and its performance in overload conditions. Simple algorithms such as Rate (Deadline) Monotonic based on offline schedulability analysis depend complete knowledge about the workload and the system including tasks' resource requirements, future arrivals and the system's schedulable utilization bound. These algorithms cannot work in overload conditions because of their lack of overload handling mechanisms. The (open loop) on-line admission control or QoS optimization based algorithms add flexibility to real-time systems by not requiring knowledge about task future arrivals, although the tasks' resource requirements and utilization bound still need to be known *a priori*. FCS algorithms accomplished the next level of flexibility by providing robust performance guarantees without requiring *a priori* knowledge about tasks' resource requirements and even the utilization bound as in the case of FC-M and FC-UM. Therefore, feedback control real-time scheduling provides the most appropriate solution for soft real-time systems in unpredictable environments. Such systems include online trading and e-business servers, and data-driven systems such as smart spaces, agile manufacturing, and many defense applications such as C4I.

	Knowledge of the Workload/System			***Performance in Overload***		
	Task resource requirement	***Future arrival time***	***Utilization Bound***	***Miss Ratio***		***CPU utilization***
				Steady state	***Transient state***	
RM, EDF	Yes	Yes	Yes	N/A		N/A
Open Loop Admission Control/QoS Optimization	Yes	No	Yes	0		High if estimation of resource requirement is not pessimistic; Low otherwise
FC-U	No	No	Yes	0	Bounded by overshoot	High
FC-M	No	No	No	Small	Bounded by overshoot	High
FC-UM	No	No	No	0 nominally; Guaranteed to be small	Bounded by overshoot	High

Table 3: Comparison of Real-Time Scheduling Paradigms in Overload Conditions

9. Conclusion

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 their transient and steady state performance specifications. In particular, we establish an analytical model and complete analysis of feedback control scheduling algorithms, which is a major challenge and key step for the control design of adaptive real-time systems. We also generalize the FCS scheduling architecture that allows plug-ins of different real-time scheduling policies and QoS optimization algorithms. 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 guarantee for periodic and aperiodic tasks even when the task execution time varied considerably from the estimation. In our future work, we are developing theoretical analysis of the nonlinearities of real-time systems. We will also extend our solutions to networked embedded systems.

10. Reference

[1] T. F. Abdelzaher, "A Schedulable Utilization Bound for Aperiodic Tasks," *University of Virginia, Technical Report CS-2000-21*, August 2000.

[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] K. J. Astrom and B. Wittenmark, *Adaptive control (2nd Ed.)*, Addison-Wesley, 1995.

[9] G. Beccari, et. al., “Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems,” *EuroMicro Conference on Real-Time Systems*, June 1999.

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

[11] 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.

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

[13] S. Cen, "A Software Feedback Toolkit and its Application In Adaptive Multimedia Systems," *Ph.D. Thesis*, Oregon Graduate Institute, October 1997.

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

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

[16] G. F. Franklin, J. D. Powell and A. Emami-Naeini, *Feedback Control of Dynamic Systems (3rd Ed.)*, Addison-Wesley, 1994.

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

[18] D. Hull, A. Shankar, K. Nahrstedt, and J. W. S. Liu, “ An end-to-end QoS model and management architecture,” *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, Dec 1997.

[19] 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.

[20] 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.

[21] 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.

[22] B. Li, D. Xu, K. Nahrstedt, J. W. S. Liu, “End-to-End QoS Support for Adaptive Applications Over the Internet”, *SPIE International Symposium on Voice, Video and Data Communications*, Nov 1998.

[23] 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.

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

[25] 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.

[26] 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.

[27] P. Mejia-Alvarez, R. Melhem, and D. Mosse, “An Incremental Approach to Scheduling during Overloads in Real-Time Systems,” *IEEE Real-Time Systems Symposium*, Orlando, FL, Dec 1999.

[28] L. Palopoli, L. Abeni, F. Conticelli, M. D. Natale, and G. Buttazzo, “Real-Time control system analysis: An integrated approach,” *IEEE Real-Time Systems Symposium*, Orlando, FL, Dec 2000.

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

[30] D. Rosu, K. Schwan, and S. Yalamanchili, “FARA–a framework for adaptive resource allocation in complex real-time systems,” *IEEE Real-Time Technology and Applications Symposium*, June 1998.

[31] 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.

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

[33] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, “On Task Schedulability in Real-Time Control Systems,” *IEEE Real-Time Systems Symposium*, December 1996.

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

[35] 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.

[36] 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.

[37] 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.

[38] K. G. Shin and C. L. Meissner, “Adaptation and Graceful Degradation of Control System Performance by Task Reallocation and Period Adjustment,” *EuroMicro Conference on Real-Time Systems*, June 1999.

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

[40] L. R. Welch, B. Shirazi and B. Ravindran, “Adaptive Resource Management for Scalable, Dependable Real-time Systems: Middleware Services and Applications to Shipboard Computing Systems,” *IEEE Real-time Technology and Applications Symposium*, June 1998.

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