Design and Evaluation of a Feedback Control EDF Scheduling Algorithm*

Chenyang Lu John A. Stankovic Gang Tao† Sang H. Son

Department of Computer Science, †Department of Electrical Engineering
University of Virginia, Charlottesville, VA22903
e-mail: {cl7v, stankovic, son}@cs.virginia.edu, †gt9s@virginia.edu

Abstract

Despite the significant body of results in real-time scheduling, many real world problems are not easily supported. While algorithms such as Earliest Deadline First, Rate Monotonic, and the Spring scheduling algorithm can support sophisticated task set characteristics (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 static or dynamic systems in which the workloads can be accurately modeled, they can perform poorly in unpredictable dynamic systems. In this paper, we present a feedback control real-time scheduling algorithm and its evaluation. Performance results demonstrate the effectiveness of the algorithm when execution times vary from the worst case and when there are major shifts of total load in the system. A key part of this feedback solution is its explicit use of deadline based metrics.

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 [Liu73][Leho89] 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) [Liu73] is an optimal dynamic scheduling algorithm in resource sufficient environments. EDF is a second major paradigm for real-time scheduling [Stan98]. While real-time system designers try to design the system with sufficient resources, because of cost and highly 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 [Rama84][Zhao87] can dynamically guarantee incoming tasks via on-line admission control and planning and thus is applicable in resource insufficient environments. Many other algorithms (e.g., RED algorithm [Butt95]) have also been developed to operate in this way. This planning-based set of algorithms represents 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 (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 static or dynamic systems in which the workloads (i.e., task sets) can be accurately modeled, they can perform poorly in *unpredictable* dynamic systems, i.e., systems whose workloads cannot be accurately modeled. For example, the Spring scheduling algorithm assumes complete knowledge of the task set

* Supported in part by NSF grant CCR-9901706 and contract IJRP-9803-6 from the Ministry of Information and Communication of Korea.

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.

Unfortunately, many real-world complex problems such as agile manufacturing, robotics, adaptive fault tolerance, and C4I and other defense applications are not predictable. Because of this, it is impossible to meet every task deadline. The objective of the system is to meet as many deadlines as possible[1]. For example, autonomous robotic systems always suffer from sudden variations in computational load and overload situations due to highly variable execution times in robot control algorithms (e.g., sensor interpretation, motion planning, inverse kinematics and inverse dynamics algorithms) [Becc99]. For another example, in information and decision support systems, accurate knowledge about transaction resource and data requirements is usually not known *a priori*. The execution time and resource requirements of a transaction may be dependent on user input or dependent on sensor values. For these applications, a design based on the estimation of worst case execution times will result in extremely expensive and underutilized system. It is more cost effective to design for less than worst case, but sometimes miss deadlines.

Another important issue is that these scheduling paradigms all assume that timing requirements are known and fixed. The assumption is that control engineers design the system front-end control loops and generate resulting timing requirements for tasks. The scheduling algorithms then work with this fixed set of timing requirements. Real control systems, in general, are much more flexible and robust, e.g., instead of choosing a single deadline for a task which is passed on to the scheduling system, a deadline range might be acceptable to the physical system. If this range was passed to the scheduling system, the on-line scheduling might be more robust.

We believe that due to all these problems, solutions based on a new paradigm of scheduling, which we call feedback control real-time scheduling, is necessary for some important systems. A case for this was made in [Stan99]. In this current paper we fully develop a feedback control scheduling algorithm, discuss stability, and present the evaluation of the algorithm.

In the past, many forms of feedback control have appeared in real-time and non-real-time scheduling systems, but it has not been elevated to a central principle; rather most of the time it is used more as an afterthought and is usually *ad hoc*. Most of these techniques also did not address the key performance metric of real-time systems - the deadline miss ratios. More discussion of this appears later in the related work section.

[1] If such systems have some critical tasks, they are treated separately by static allocation of resources.

2. Approach

The mapping of control theory methodology and analysis to scheduling provides a systematic and scientific method for designing scheduling algorithms. Many aspects of this mapping are straightforward, but others require significant insight and future research. We now describe how such a mapping can be done and what research is necessary. In section 3 we present an instance of this mapping by creating an actual algorithm and a runtime scheduling structure.

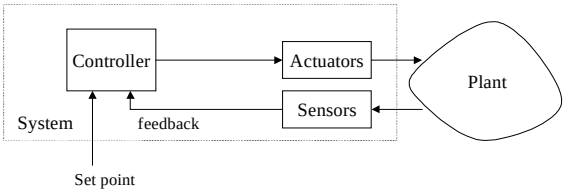

Figure 1 Architecture of Feedback Control Systems

2.1. Control theory and real-time scheduling

A typical feedback control system is composed of a controller, a plant to be controlled, actuators, and sensors (as illustrated in Figure 1). It defines a *controlled variable*, the quantity of the output that is measured and controlled. The *set point* represents the correct value of the controlled variable. The difference between the current value of the controlled variable and the set point is the *error*. The *manipulated variable* is the quantity that is varied by the controller so as to affect the value of the controlled variable. The system is composed of a feedback loop as follows. (1) The system periodically monitors and compares the controlled variable to the set point to determine the error. (2) The controller computes the required control with the control function of the system based on the error. (3) The actuators change the value of the manipulated variable to control the system. In the context of real-time scheduling problems, our approach is to regard a scheduling system as a feedback control system, and the scheduler as the controller. The scheduler utilizes feedback control techniques to achieve satisfactory system performance in spite of unpredictable system dynamics. We believe that the long term potential for a theory and practice of feedback control scheduling is significant; partly because we can also build upon the vast amount of knowledge and experience from control systems.

As a starting point, we will apply PID (Proportional-Integral-Derivative) control in schedulers. A basic form PID control formula is

$$Control\ (t) = C_P Error\ (t) + C_I \int Error\ (t)dt + C_D \frac{dError\ (t)}{dt} \qquad (1)$$

We choose PID control as the basic feedback control techniques in feedback control scheduling for the following reasons. (1) In term of control theory, the scheduling system is a *dynamic* system, i.e., a system whose output depends not only on the current input, but also on the previous system inputs. It is known that the current miss ratio of a scheduling system depends not only on the currently submitted task, but also on the previously submitted tasks that remain in the system for queuing, execution and blocking. It is known that PID control is a widely applicable control technique in dynamic systems. (2) Compared with other control techniques, an important feature of PID control is that it does not require a precise analytical model of the system being controlled. Instead, a PID controller designed based on an approximate model can achieve satisfactory performance. Due to the extremely complex behavior of current computer systems, it is impossible to precisely model the dynamics of real world scheduling systems. However, certain form of approximate modeling of the scheduling system can be built to help tune the PID control parameters more systematically (such an approximate model is presented in section 3.6 of this paper). (3) According to control theory, basic PID control can provide stable control in first and second order dynamic systems. In systems with higher order of dynamics, however, basic PID control can only provide approximate control, but an adaptive form of PID control can provide stable control for high order dynamic systems.

2.2. Feedback control real-time scheduling

To apply feedback control techniques in scheduling, we need to restructure schedulers based on the feedback control framework. We need to identify the controlled variable, the manipulated variable, the set point, the error, the control function, and the mechanisms of the actuators. We can then set up the feedback loops based on these selections. The choice of the controlled variable depends on the system goal. For example, the performance of real-time systems usually depends on how many tasks make (miss) their deadlines. We define the system deadline miss ratio as the percentage of tasks that miss their deadlines; this is a natural choice of the controlled variable. The manipulated variable must be able to affect the value of the controlled variable. In the real-time scheduling, it is a widely known fact that the deadline miss ratio highly depends on the system load, i.e., the requested CPU utilization of tasks in the system. Thus the requested CPU utilization can be used as the manipulated variable. Other possible choices of manipulated variables are the periods/deadlines of tasks in control applications when there exists the flexibility to adjust these task parameters [Seto96].

In summary, a feedback control scheduling system would start with a schedule based on the nominal assumptions of the incoming tasks (expected start time, expected execution time and deadline). The system would then monitor the actual performance of the schedule, compare it to the system requirements and detect differences. The system would call control functions to assess the impact of these differences and apply a correction to keep the system within an acceptable range of performance. Research is needed to answer the following open questions:

- What are the right choices of controlled variables, manipulated variables, set points, and effective control functions/mechanisms for feedback control scheduling?
- How to model a feedback control scheduling system?
- How to tune the control parameters to build a stable and high performance scheduler?
- How to integrate the flexible timing constraints derived from the front-end feedback control loops in control systems with an on-line feedback control scheduling algorithm?
- What is the impact of overhead in feedback control scheduling and how to minimize it?

3. Feedback Control EDF

We now present an algorithm called Feedback Control EDF (FC-EDF), which integrates PID control with an EDF scheduler. With FC-EDF, we demonstrate how to structure a real-time scheduler based on the feedback control framework. We will identify specific research issues related to feedback control scheduling using FC-EDF as a concrete example. The FC-EDF architecture (shown in Figure 2) can be generalized to be used as a framework of feedback control schedulers. For example, we can investigate feedback control RM by replacing the EDF scheduler in Figure 2 with a RM scheduler.

3.1. Overview of FC-EDF

To apply PID control to a scheduling system, we need to decide on the components of a scheduling system corresponding to those in a feedback control system (Figure 1). First, we need to choose the controlled variable and the set point of the system. The requirements of an ideal soft real-time scheduling algorithm should be to (1) provide (soft) performance guarantees to admitted tasks, i.e., maintain low miss ratio among admitted tasks; and (2) achieve high system throughput and utilization. To satisfy these requirements, FC-EDF chooses miss ratio among the admitted tasks, *MissRatio(t),* as the controlled variable and an (application dependent) small but non-zero value (e.g., $MissRatio_s$ = 1%) as the set point. Note that 0 is not chosen as the set point for the following reasons. A system with a set point of $MissRatio_s$ = 0 can ignore the second requirement of soft real-time scheduling, i.e., high utilization and throughput. When a system achieves a 0% deadline miss ratio but causes extremely low utilization (e.g., by unnecessarily rejecting too many tasks), a feedback control scheduler with $MissRatio_s$ = 0 will treat it as the correct state. In contrast, a feedback control scheduler with

a set point of $MissRatio_s \neq 0$ will always try to (lightly) overload the system to achieve high utilization. Note that in an unpredictable environment, it is impossible for a system to achieve 100% utilization and 0% miss ratio all the time and a tradeoff between miss ratio and utilization is unavoidable. There can be two approaches to deal with this tradeoff. The approach of admission control based on pessimistic estimation is the *pessimistic* approach, which always *avoids* deadline misses at the cost of low utilization and throughput. This approach has been widely used in hard real-time systems. On the other hand, the feedback control scheduling presented in this paper represents the *optimistic* approach, which maintains a low (but possibly non-zero) miss ratio and high utilization and throughput. When a high misses actually happens due to system load changes, the scheduler *corrects* the system state back to the satisfactory state, i.e., a state with low miss ratio and high utilization and throughput. This optimistic approach is especially preferable in soft real-time systems since it provides a soft performance guarantee in term of miss ratios while achieving high utilization and throughput at the same time (see performance evaluation in section 4).

Second, we choose the requested CPU utilization, i.e., the total CPU utilization requested by all the accepted tasks in the system, as the manipulated variable. The rational is that EDF can guarantee a miss ratio of 0% given the system is not overloaded, and in normal situations, the deadline miss ratios increases as the system load increases[2]. For simplicity of description, we will use requested utilization in place of requested CPU utilization in this paper. Third, we need to design the mechanisms (i.e., actuators) used by the scheduler to manipulate the requested utilization. An Admission Controller and a Service Level Controller are included in the FC-EDF scheduler as the mechanisms to manipulate the requested utilization. The Admission Controller can control the flow of workload into the system, and the Service Level Controller can adjust the workload inside the system.

The FC-EDF scheduler is composed of a PID controller, a Service Level controller, an Admission Controller and an EDF scheduler (Figure 2). The system performance *MissRatio(t)* is periodically fed back to the PID controller. Using the PID control formula (1), the PID controller computes the required control action $\Delta CPU(t)$, i.e., the total amount of CPU load that need to be added into (when $\Delta CPU(t)>0$) or reduced from (when $\Delta CPU(t)<0$) the system. Then the PID controller calls the Service-Level Controller and the Admission Controller to change the CPU load of the system by ΔCPU. This system control forms a feedback control loop in the scheduling system. The EDF scheduler schedules the accepted tasks according to the EDF policy.

[2] This is under the assumption that the domino effect is rare in real world applications.

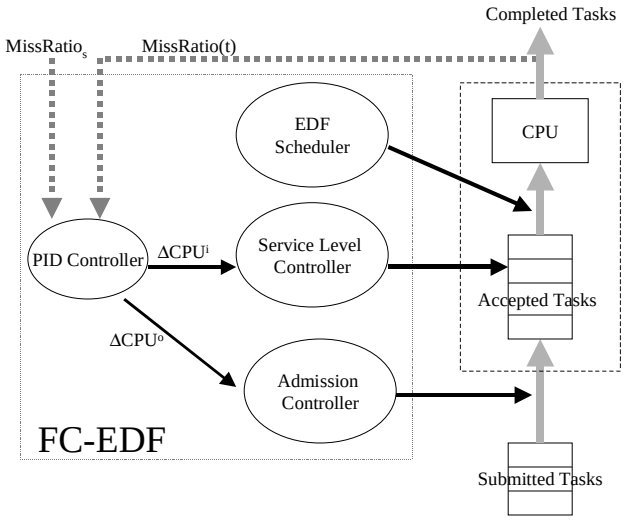

Figure 2 Architecture of FC-EDF

3.2. Task model

Our initial task model assumes that all tasks have soft deadlines and all the tasks are independent. For the convenience of description, this paper assumes a task model similar to the imprecise computation model [Liu91], but the scheduling algorithms presented do not depend on the imprecise computation model. Each task T_i submitted to the system is described with a tuple (I, ET, VAL, S, D). Each task T_i has one or more logical versions $I = (T_{i1}, T_{i2}, \ldots T_{ik})$. Note that when a task has multiple logical versions it does not necessarily mean that it has multiple implementations. An imprecise computation can have several different forms including milestone method, sieve function method or multiple version method [Liu91]. We call all these methods that can tradeoff computation value and time as multiple logical versions of a task for convenience of discussion. Each version has different execution time and different value. $ET = \{ET_{i1}, ET_{i2}, \ldots ET_{ik}\}$ (suppose $ET_{i1} \geq ET_{i2} \geq \ldots \geq ET_{ik}$) are the nominal execution times of different versions. Here, nominal execution time instead of worst case execution time is used in the system to achieve higher CPU utilization in the system. The execution time is described in the form of requested utilization. For example, $ET_{i1}=0.02$ means the 1st version of task T_i requires 2% of the CPU time. $VAL = \{VAL_{i1}, VAL_{i2}, \ldots VAL_{ik}\}$ represents the values of different implementation. In this research, different versions of a task are called service levels. We call a version with longer execution time and higher value a higher service level than another version with less execution time and lower value. Each task has a soft deadline D_i and a start time S_i.

Note that FC-EDF does not depend on the imprecise computation model. FC-EDF only needs a certain flexibility to adjust the CPU utilization. In our future work, we will extend the deadline D_i to a range of deadlines $(D_{i,min}, D_{i,max})$.

The deadline of a task T_i could be adjusted dynamically within the range. By adjusting the deadline of a task, the Service Level Controller can effectively change the requested CPU utilization in the system. This extension is based on the fact that digital control systems are usually robust, i.e., the task timing constraints are allowed to vary within a certain range without affecting critical control functions such as maintenance of system stability [Seto96]. Such extension is also applicable in multimedia systems, in which the QoS specifications of a multimedia application can be specified as intervals.

3.3. PID controller

The PID controller is the core of FC-EDF. It maps the miss ratio of accepted tasks (i.e., error) to the change in requested utilization (i.e., control signal) so as to drive the miss ratio back to the set point.

The PID controller periodically monitors the controlled variable *MissRatio(t)*, and computes the control $\Delta CPU(t)$ in terms of requested utilization with the following control formula, which is an approximation of formula (1).

$$\Delta CPU(t) = C_P error(t) + C_I \sum_{IW} error(t) + C_D \frac{error(t) - error(t - DW)}{DW} \quad (2)$$

where $error(t) = MissRatio_s - MissRatio(t)$. SP, C_P, C_I, C_D, IW and DW are tunable parameters of the PID controller. SP is the sampling period. The PID controller will be called every SP seconds. For convenience of presentation, we will take SP as the time unit in our following discussion. C_P, C_I and C_D are the coefficients of the PID controller. IW is the time window over which to sum the errors. Only errors in the last IW time units will be considered in the integral term. DW is the time window of derivative errors. Only the error change during the last DW time units will be considered in the derivative term (i.e., the derivative error is $(MissRatio(t\text{-}DW)\text{-}MissRatio(t))/DW$). The tuning of these parameters will be discussed in section 3.6. $\Delta CPU(t)$ is the output (i.e., control signal) of the PID controller. $\Delta CPU(t)>0$ means that the requested utilization should be increased, and $\Delta CPU(t)<0$ means that the requested utilization should be decreased.

The PID controller will call the Service Level Controller and the Admission controller to change the requested utilization of the system by ΔCPU. If the current requested utilization is $CPU(t)$, the Service Level controller and Admission Controller will change the requested utilization to $CPU(t)+\Delta CPU$. The PID controller always tries to change the requested utilization internally by calling Service Level Controller first so that the system could respond to the error faster. The admission controller is called only if the Service Level Controller cannot accommodate ΔCPU completely.

3.4. Service Level Controller

The Service Level Controller (SLC) changes the requested utilization in the system by adjusting the service levels of accepted tasks. For example, if it changes the service level of task T_i from T_{ik} to T_{ij}, it adjusts the requested utilization of the system by ET_{ij}-ET_{ik}, where ET_{ij} and ET_{ik} are the estimated CPU requirement of T_{ik} and T_{ij}, respectively. SLC returns the portion of ΔCPU not accommodated. This portion of control will be accommodated by the Admission Controller

3.5. Admission Controller

The Admission Controller (AC) controls the flow of workload into the system. When a new task T_i is submitted to the system, AC decides on whether it could be accepted into the system. Given current system-wide requested utilization $CPU(t)$ and the CPU requirement of the incoming task, the Admission Controller admits T_i with service level k if k is the highest level that satisfies $CPU(t)+ET_{ik}<1$; T_i is rejected if it cannot be admitted even with the lowest level.

The Admission Controller's parameter $CPU(t)$ may be adjusted when the PID controller cannot accommodate $\Delta CPU(t)$ completely with SLC. Suppose SLC changes the requested utilization by ΔCPU^i and $\Delta CPU^i<\Delta CPU$, AC will accommodate $\Delta CPU^o=\Delta CPU(t)-\Delta CPU^i$, the portion of ΔCPU not accommodated by SLC. It accommodates ΔCPU^o by adjusting the estimation of requested utilization $CPU(t)$ as following $est\_cpu\_util = 1 - \Delta CPU^o$. The Admission Controller will admit the workload of at most ΔCPU^o (when $\Delta CPU^o > 0$) or will not admit any new tasks until the system load has been reduced by amount of more than $|\Delta CPU^o|$ (when $\Delta CPU^o < 0$).

3.6. Modeling and Analysis of feedback control scheduling

3.6.1. Feedback control scheduling model

An important task of building a stable and high-performance PID-control-based scheduler is to tune the control parameters (i.e., C_P, C_I and C_D). One way is to tune the parameters by simulations. However, this approach would require large amount of experiments to gain enough confidence in the selected values of the parameters. A more scientific and systematic approach is to apply control theory analysis to select the PID control parameters. Such analysis requires an analytical model of the scheduling system. Before we present the model of a PID-control based scheduling system, we define the following notions in the discrete time domain:

1. SP is a constant sampling period, which is the time elapsed in interval [k, k+1], k being time instants.
2. *MissRatio(z)*: the miss ratio - the system output and the controlled variable.

3. $MissRatio_s$: the set point (i.e., the target) in term of miss ratio.
4. *CPU'(z)*: the estimated CPU utilization;
5. *CPU(z)*: the CPU utilization;
6. *ΔCPU'(z)*: the change in the estimated CPU utilization (*CPU'(z)*) . This is the system input;
7. *ug(k)*: the ratio of the actual total utilization to the estimation;
8. *mrg(k)*: the gain that maps the utilization (*CPU(z)*) to the miss ratio (*MissRatio(z)*).
9. *d(k)*: the disturbance (e.g., associated with the utilization bound).

Using the above notations, the estimated CPU utilization follows the following equation:

$$CPU'(z) = \Delta CPU'(z)/(z-1) \quad (3)$$

Since the precise execution time of each task is unknown and changes over time, the actual requested utilization is usually different from the estimated requested utilization. In term of control theory, *ug(k)* is a time-variant *gain* (called *utilization gain* in this paper) of the system. We can get the (actual) CPU utilization using *ug(k)*:

$$CPU(z) = ug(k)CPU'(z) \quad (4)$$

We can derive the deadline miss ratio based on the correlation between the deadline miss ratio and the system utilization. The relationship between the deadline miss ratio and the requested utilization can be modeled as

$$MissRatio(z) = mrg(k)CPU(z) - d(k) \quad (5)$$

where *mrg(k)* (*miss ratio gain*) is a time-variant *gain* that maps *CPU(z)* to *MissRatio(z)* in overload situations.

The transfer function of the PID controller is:

$$H(z) = C_P + C_I/(z-1) + C_D(z-1)/z \quad (6)$$

In summary, we present the block diagram of the FC-EDF scheduling system in Figure 3.

3.6.2. Stability Analysis

For a control system, there are two forms of stability: internal stability and BIBO stability. *Internal stability* is related to the system behavior due to initial conditions. If no input is applied, an internally stable system will settle to be close to an equilibrium set point within a definite amount of time. Although a real-time system could enter an internally unstable mode when the miss ratio rises monotonically over time (e.g., when deadlock occurs) even when there is no change in the workload, this paper will assume an independent task model and the system is assumed internally stable when it is made BIBO stable. The issue of internal stability will be investigated in our future research. *BIBO stability* means that the system output is always bounded for a bounded input. In the context of the scheduling system, this means that the miss ratio will be bounded for bounded changes in the workload. It is important to tune the PID controller such that the BIBO stability is satisfied to avoid uncontrollable performance degradation in a scheduling system. For convenience of description, we will use stability in place of BIBO stability in this paper. To demonstrate the importance of using a formal theory to describe feedback control real-time scheduling, we present the following stability analysis result (The mathematical proof is skipped due to the length limit on this paper).

Theorem 1: When *ug(k)mrg(k)* is close to 1, the FC-EDF scheduling system established by the block diagram in Figure 3 is stable if one of the following conditions is satisfied:

1. $C_I > 0$, $|C_D| < 1$, $2C_P - C_I + 4C_D < 4$, and $2 - 2C_D^2 > C_DC_P + C_P - C_I > 0$
2. $C_I = 0$, $|C_D| < 1$, and $0 < C_P + 2C_D < 2$

The assumption that *ug(k)mrg(k)* is close to 1 implies that (1) the estimated utilization is not too far from the actual utilization and (2) domino effect does not happen. Under this assumption (which we believe is true for most soft real-time systems), the stability condition gives the guidance on how to set the coefficients in the PID control such that the satisfactory scheduling performance can be maintained. This is one important reason for basing feedback control real-time scheduling on control theory.

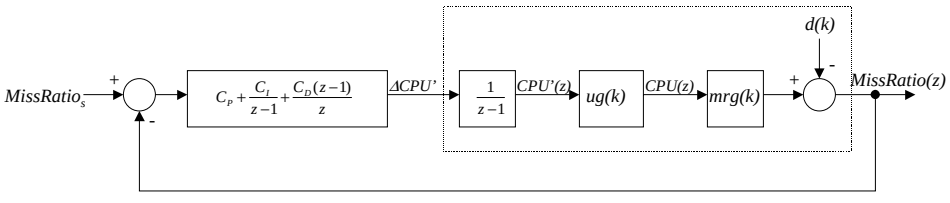

Figure 3. Block diagram of the FC-EDF scheduling system

4. Experiments and Results

4.1. Simulation Model

An uniprocessor simulator of a soft real-time system was used to study the performance of FC-EDF and baseline algorithms. The workload consists of independent periodic tasks. Each task instance has a deadline that equals its period. If a task instance is not completed by its deadline, it is immediately aborted. The simulator (Figure 4) has six components: *sources* that generate tasks; an *admission controller* that make admission/rejection decisions on submitted tasks; an *executor* that models the execution of the tasks; a *monitor* that periodically collects the performance statistics of the system; a *PID controller* that periodically computes control signals based on the performance errors; and a *service level controller* that adjusts the service levels of the tasks. A basic EDF scheduler is embedded in the executor. The admission controller, service level controller, and PID controller each can be turned on/off to emulate the different baseline scheduling algorithms.

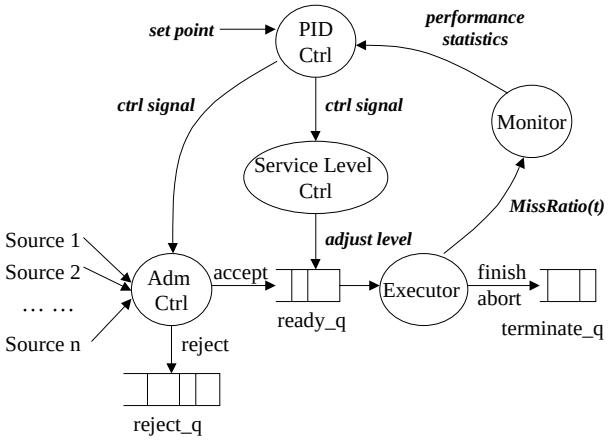

Figure 4. Feedback Control Scheduling Simulator

4.2. Workload Model

Each source is characterized with a period (P) (the deadline of each task instance equals its period), a set of worst case execution times $\{WCET_i\}$, a set of best case execution times $\{BCET_i\}$, a set of estimated execution times $\{EET_i\}$, a set of average execution times $\{AET_i\}$ ($WCET_i \geq AET_i \geq BCET_i$), and a set of values $\{VAL_i\}$, where $\#service\_level-1 \geq i \geq 0$. Each tuple ($P$, $WCET_i$, $BCET_i$, EET_i, AET_i, VAL_i) characterizes a service level and

$$EET_i = (WCET_i + BCET_i) * 0.5$$
$$AET_i = EET_i * etf$$

where *etf* (*execution time factor)* can be tuned to change the accuracy of the estimation. The larger the *etf* is, the more pessimistic the estimation is. For example, *etf* of 1.0 means that the estimated execution time is equal to the actual average execution time. The estimation is less than the average execution time when $etf < 1.0$; and greater than the average execution time when $etf > 1.0$. The actual execution time of each instance (which is unknown to the scheduler) is computed as a uniform random variable in interval $[AET_i, WCET_i]$ or $[BCET_i, AET_i]$ depending on a random *Bernoulli* trial with probability $(AET_i - BCET_i)/(WCET_i - BCET_i)$. In our experiments, the workload is composed of 40 periodic tasks and each task has two service levels. $WCET_0 = 2*WCET_1$, $WCET_i = 4*BCET_i$, $VAL_0 = 1.0$ and $VAL_1 = 0.5$. It follows that *etf* lies in the range of [0.4, 1.6]. A uniform distribution is used to generate $WCET_0$ and P of each task: $WCET_0 = uniform(5,10)$, $P = WCET_0 * uniform(10,15)$. P is adjusted such that 31 of the tasks are harmonious with least common multiplier of 2400 time units. The same workload is used for all the experiments. Each different experiment has varied *etf*. All the tasks arrive in the beginning of each experiment to overload the system. Each rejected task will attempt to get re-admitted all through the experiment.

4.3. Implementation of FC-EDF

The parameter settings of FC-EDF in all the experiments are listed in Table 1. The set point is the soft real-time performance target miss ratio that the scheduler will achieve. In practice, this value depends on the tolerance (to deadline misses) of the application. The *SP* is equal to the least common multiplier of the majority of the tasks such that similar number of task instances are submitted in each *SP*. *IW* controls the length of history that the controller need to consider. *DW* affects the agility of the controller to the sudden change in workload. C_P, C_I and C_D are coefficients of the PID controller. Note that the values of C_P, C_I and C_D are tuned so that they satisfy the stability condition established in Theorem 1 to guarantee the scheduling performance.

Set Point	0.01
SP	2400 (time unit)
IW	100 (SP)
DW	1 (SP)
C_P	0.5
C_I	0.05
C_D	0.1

Table 1 FC-EDF parameter setting

4.4. Baseline Algorithms

The following baseline algorithms are implemented and compared with FC-EDF in the experiments.

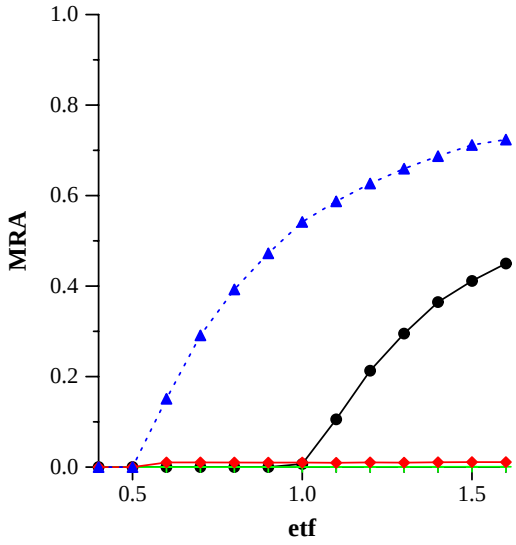

Figure 5 MRA vs. etf

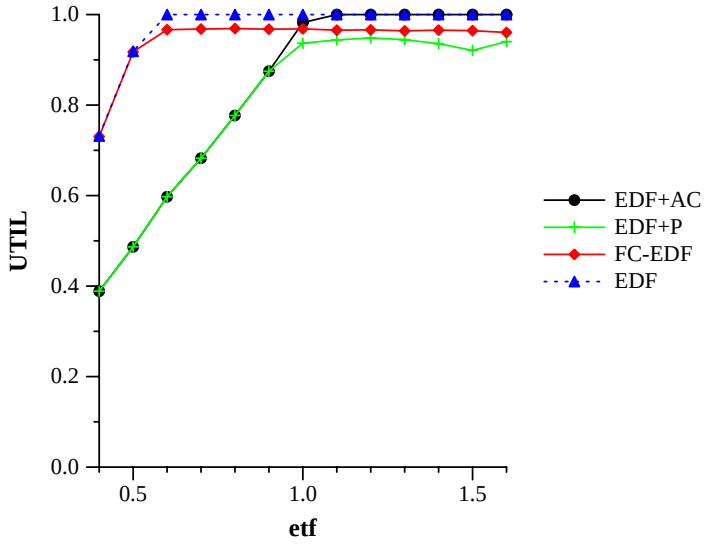

Figure 6 UTIL vs. etf

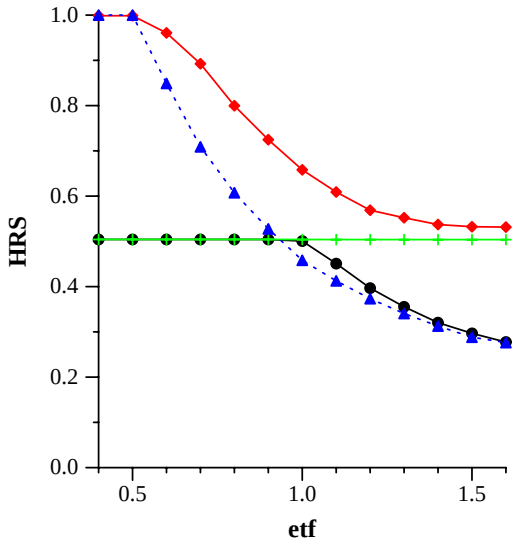

Figure 7 HRS vs. etf

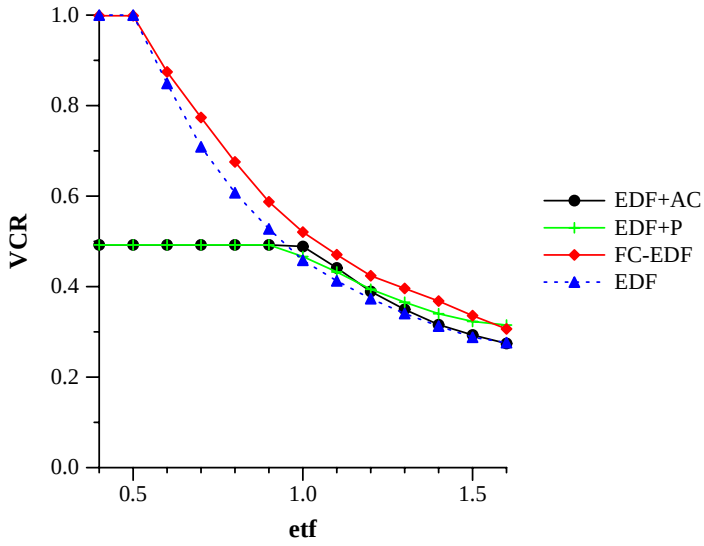

Figure 8 VCR vs. etf

- *EDF*: This is the basic EDF scheduling algorithm. It is implemented by turning off the admission controller, service level controller, and the PID controller.
- *EDF+AC* (EDF with static admission control): EDF+AC is implemented by turning off the service level controller and the PID controller. The admission decision is made once for each periodic task when it is submitted.
- *EDF+P* (EDF with proportional control): EDF+P is implemented and configured the same as FC-EDF except for $C_P = C_D = 0$. This algorithm is to test whether the simple proportional control can provide comparable performance as PID control.

4.5. Performance Metrics

- *Miss Ratio among admitted tasks* (*MRA = #Misses / #(Admitted Task Instances)*): Although a soft real-time task can tolerate a small percent of deadline misses, it usually requires a *soft guarantee* in term of miss ratio in order to maintain its normal functionality. Therefore, the average miss ratio among admitted tasks should be the most important performance metric.
- *CPU Utilization* (*UTIL*): Soft real-time systems should avoid under-utilizing the system resources to reduce costs, thus the CPU utilization is another metric under consideration.
- *Hit Ratio among submitted tasks* (*HRS = #Hits / #(Submitted Task Instances)*): Hit ratio among submitted tasks is a measure of the system throughput in term of the percentage of task instances completed in time among all the arrivals.
- *Value Completion Ratio* (*VCR =(Σ(Values of hit task instances) / Σ(Values of Submitted Task Instances)*): Value

completion ratio considers the quality of the results. A task that misses its deadline contributes 0 value. A task instance completed with a lower service level contributes a lower value.

4.6. Experiment A: Steady execution time

Experiment A evaluates the performance of the scheduling algorithms (EDF, EDF+AC, EDF+P and FC-EDF) when the average execution time of each periodic task is statistically steady, but different from the estimation. *etf* is constant through each run, but the actual execution time of each task instance is a random value that differs over time. Each algorithm made 13 set of runs with each set for a different *etf* values in the range [0.4, 1.6]. These experiments are interesting since, in real-world applications, it is impossible to make the estimation of execution time the same as the average actual execution time at all times. This problem is avoided in hard real-time systems by using worst-case estimation (*etf* << 1.0). However, soft real-time systems usually use *nominal* estimations instead of worst-case estimations due to cost considerations. It is thus important for a soft real-time system to maintain satisfactory performance even when the estimation is different from the average actual execution times. This is one place the value of feedback control is demonstrated. Experiment A evaluates the algorithms under situations when the execution time is statistically steady. The performance under conditions when the execution time changes dramatically is studied in Experiment B. This is another place where feedback proves very valuable. Experiment A results are illustrated in Figures 5-8. Each point in Figure 5-8 is the average value of 30 runs. The 90% confidence interval for each point of *MRA*, *UTIL*, *HRS*, and *VCR* is within ±0.0006, ±0.0058, ±0.0066 and ±0.0046, respectively. Each run lasts for 2880000 time units (1200 SP).

From Figure 5, we see that both FC-EDF and EDF+P maintains a very low *MRA* (<1%) throughout the *etf* interval. The *MRA* of FC-EDF is closer to the set point (1%) and thus slightly higher than EDF+P since the Integral control reduces the steady state error of the control. EDF's *MRA* increases sharply when *etf* > 0.5 and the system is overloaded. This shows pure EDF cannot provide acceptable performance in resource-insufficient systems. EDF+AC maintains low *MRA* when *etf* < 1, however, its performance degrades sharply when the average execution time is larger than its estimation (*etf* > 1). This shows that static admission control based on nominal estimations cannot provide (soft) performance guarantees under all situations. Figure 6 shows that EDF always achieves the highest CPU utilization since it never rejects any tasks. EDF+AC under-utilizes the system when the average execution time is lower than the estimation (*etf* < 1). This shows that static admission control cannot avoid wasting the system resources under all situations. Similarly, EDF+P also under-utilizes the utilization when *etf* < 1 similar to EDF+AC. This is because EDF+P does not consider the accumulated error, and the control signal (*ΔCPU*) computed with proportional control alone is too small to be accommodated when the task utilization can only be adjusted in discrete steps[3]. In comparison, with the PID control, FC-EDF maintains high CPU utilization in all the situations. We can also see that the CPU utilization of FC-EDF is consistently higher than EDF+P even when (*etf* > 1).

Figures 7 and 8 compare the throughput of the algorithms in term of *HRS* and *VCR*. EDF+AC and EDF+P has a low throughput when (*etf* < 1) as a result of admission control based on pessimistic estimation. EDF has the highest throughput when the system is lightly loaded and degrades fast as the system load increases. We should also note EDF achieves its throughput by greedily accepting all the tasks while at the cost of poor performance for *each* task (no performance guarantees for admitted tasks). FC-EDF's throughput is slightly lower than EDF when the system is lightly loaded (This is actually a result in the initial phase and the PID control drives the throughput to the same level as EDF after a short learning period). When the system is overloaded, FC-EDF consistently achieves the highest throughput in all situations (except for when *etf* = 1.6 when it is slightly lower than EDF+P).

In summary, Experiment A demonstrates that under situations when the estimation of execution time differs from the (statistically steady) actual execution time, FC-EDF is able to (1) provide a soft performance guarantee for admitted tasks; (2) achieving high system utilization; and (3) high throughput. None of the other algorithms under comparison can meet these goals simultaneously.

4.7. Experiment B: Dynamic execution time

In experiment B, the *etf* is dynamically changed every 720000 time units (300 SP) in order to study the response of the scheduling algorithms when the average system load changes dramatically. The *etf* settings in different intervals are listed in table 2.

Interval (SP)	0-300	300-600	600-900	900-1200
etf	0.8	1.3	0.8	1.2

Table 2 *etf* setting in Experiment B

[3] While a larger C_P can amplify the control signal, it will also cause severe oscillation and even instability in the system response.

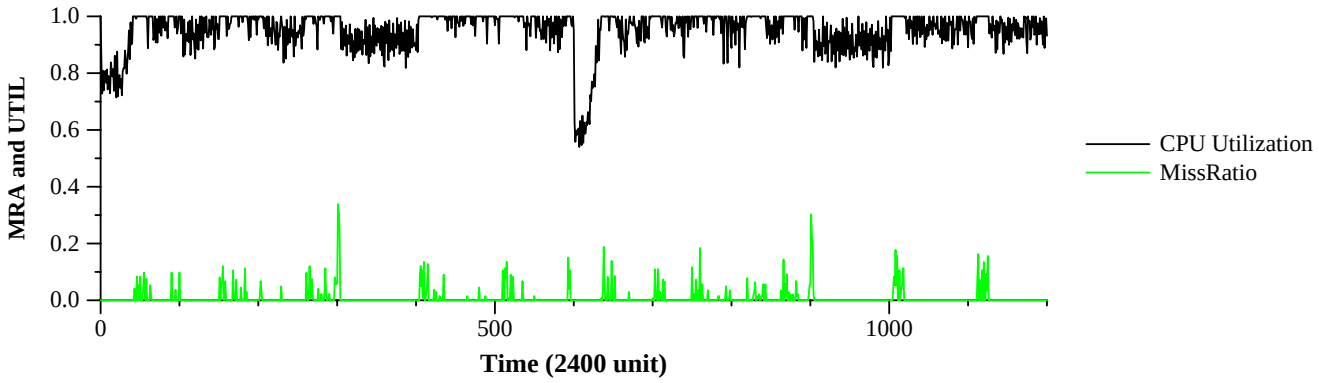

Figure 9 Sampling MissRatio and CPU Utilization of FC-EDF

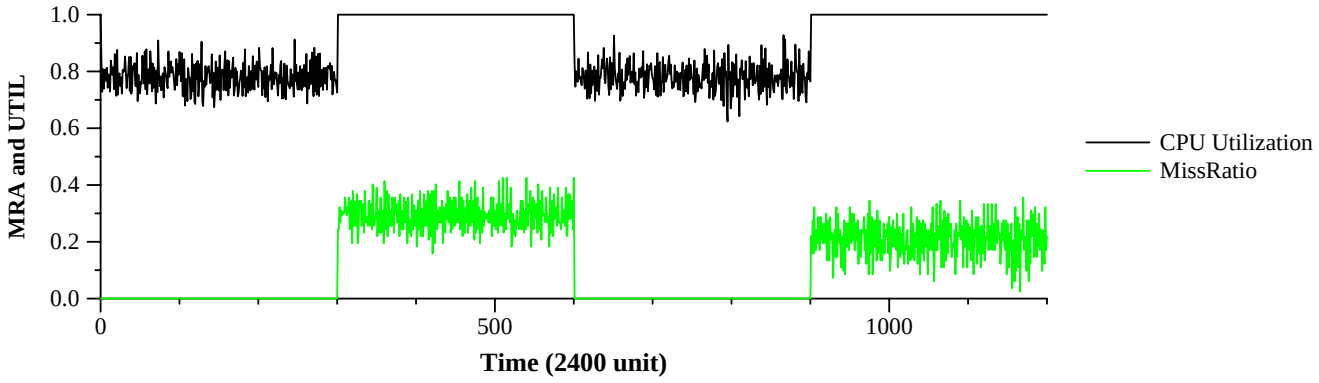

Figure 10 Sampling MissRatio and CPU Utilization of EDF+AC

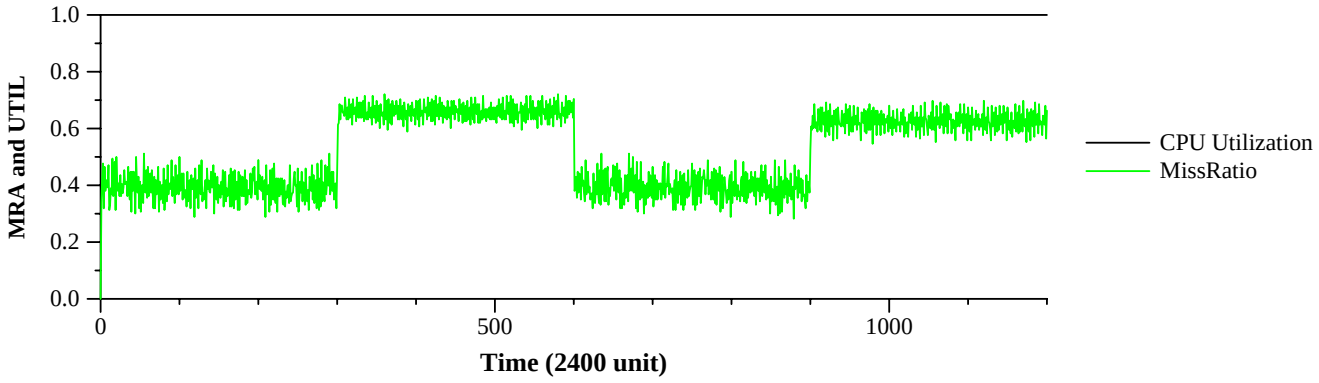

Figure 11 Sampling MissRatio and CPU Utilization of EDF

Figures 9-11 illustrates the sampled *MRA* and *UTIL* of FC-EDF, EDF+AC and EDF (EDF+P is dropped in this experiment since its performance is consistently worse than FC-EDF) in a typical run. From Figure 9, we can see that the system starts with approximately 80% of utilization (the top curve) and 0 miss ratio (the bottom curve) as a result of the pessimistic admission control. However, after a short learning period (43 SP), the PID control in FC-EDF causes the system to increase its utilization (by increasing service levels and admitting more tasks) to close to 100%. The system maintains a high utilization and low miss ratio until the end of 300 SP when the *etf* suddenly increases to 1.3, which implies that the system load suddenly increases by over 60%. This causes the big spike on miss ratio at time 300 SP in Figure 9. FC-EDF responds immediately by reducing the load (i.e., reducing service levels) to below 100% within 3 SP. In addition to the proportional control, the fast response is also due to the derivative control in FC-EDF which responses to the sudden increase in the miss ratio. The system comes back to low miss ratio and high utilization and stays this way until 600 SP when the *etf* is reduced to 0.8 (which corresponds to an over 60% loss in

system load). Again, the FC-EDF responds by increasing the system load. Within 35 SP, the utilization is driven back close to 100% and the system maintains low miss ratio and high utilization afterwards. At the end of 900 SP, the *etf* is increased to 1.2 and, similar to the time at 300 SP, the system is back to a satisfactory performance state. We can also see that all through the run, the *MRA* is 0 in most sampling periods but non-zero (around 10% except for the short transition periods between different *etf* intervals) in a small portion of sampling periods. This is the penalty for achieving high utilization (around 95% throughout the run except for the short transition periods) and throughput of the system. Note that due to the random nature of the system load, it is impossible to maintain both 0 miss ratio and 100% utilization all the time.

In contrast, from Figure 10, we see that EDF+AC achieves 0 miss ratio at the cost of low utilization (around 80%) in 0-300 SP and 600-900 SP (when *etf* < 1.0). However, the miss ratio is around 20-25% in 300-600 SP and 900-1200 SP (when *etf* > 1.0). From Figure 11, we can see the *MRA* is around 40% or 65% at different intervals for EDF, which performs even worse that EDF+AC.

	MRA	UTIL	HRS	VCR
FC-EDF	0.011	0.954	0.796	0.537
EDF+AC	0.127	0.889	0.440	0.431
EDF	0.518	1.000	0.482	0.482

Table 3 Overall performance in Experiment B

In summary, the overall performance of the algorithms is listed in Table 3 (Each data is the average value of 30 runs. The 90% confidence interval of each value of MRA, UTIL, HRS, and VCR is within ±0.0006, ±0.0023, ±0.0068, and ±0.0026, respectively). FC-EDF effectively adapts to the radical changes in the execution time and system load, and maintains satisfactory performance throughout the run time. In contrast, both EDF and EDF+AC are unable to handle the workload and render poor performance.

4.8. Discussions on overhead

Feedback control scheduling algorithms can introduce extra overhead compared with open-loop scheduling algorithms. The overhead is ignored in the simulation experiments for the following reasons. First, the feedback controller can be executed at a much lower rate than application tasks. In both experiments, FC-EDF maintains satisfactory performance while more than 100 task instances are executed between two subsequent invocations of the feedback controller. Second, the control algorithm is not complex. The bookkeeping of the miss ratio, the PID control function, and admission control adjustment all cost constant time. The service-level controller is a linear-time (in terms of the number of admitted tasks) algorithm. The overhead issue will be further investigated in our future research as we vary the period of the feedback controller.

5. Related Work

The idea of using feedback information to adjust the schedule has been used in general-purpose operating systems in the form of multi-level feedback queue scheduling [Blev76]. The system monitors each task to see if it consumes a time slice or does I/O and adjusts its priority accordingly. This type of control is a crude correction term that seems to work via *ad hoc* methods. No systematic study has been done. [Stee99] presented a feedback-based scheduling scheme that adjusts CPU allocation based on application-dependent *progress monitors*. This work is done in the context of general operating systems and the performance of real-time tasks is not addressed.

In the area of real-time databases, Haritsa et. al. [Hari91] proposed *Adaptive Earliest Deadline* (AED), a priority assignment policy based on EDF. In order to stabilize the performance of EDF under overload conditions, AED features a feedback control loop that monitors transactions' deadline miss ratio and adjusts transaction priority assignment accordingly. *Adaptive virtual deadline first* [Pang95] aims to achieve the fairness of an EDF based scheduler to transactions with different sizes. The linear correlation between deadline miss ratio and transaction size in the system is measured and fed back to the scheduler, which adjusts scheduling parameters dynamically. However, the feedback control in these algorithms is *ad hoc* and the issues of stability of the schedulers is not addressed.

[Seto96] and [Ryu98] both propose to integrate the design of a real-time controller with the scheduling of real-time control systems. [Seto96] introduced a system *performance index* to describe the control cost and tracking errors of a control system. Their approach was to assign frequencies to control tasks that optimize the system performance index under resource constraints in the system. [Ryu98] is based on a generic analytical feedback control system model, which expresses the system performance as functions of the end to end timing constraints at the system level. A heuristic algorithm is used to find the end-to-end timing constraints that can achieve the required system performance. They then use the *period calibration method* [Gerb95] to derive the timing constraints for each task in the system. Both [Seto96] and [Ryu98] aim at providing design tools that enable control engineers to take into consideration scheduling in the early design stage of control systems. The scheduling algorithms in both of these cases are still open loop scheduling algorithms.

[Becc99] and [Shin99] both utilized the flexible timing constraints as a mechanism for graceful performance degradation in control systems. Both of the works assumed the execution times are known and focused on how to reassign the periods for tasks to satisfy the utilization constraints. Instead, our work focuses on using feedback

control loops to maintain satisfactory deadline miss ratio when the task execution times change dynamically.

Jehuda and Israeli [Jehu98] proposed an *automated software meta-controller* to dynamically reconfigure real-time systems. Compared with feedback control real-time scheduling, the software meta-control is a higher level control that occurs only when the system mode changes.

In the area of multimedia communication, several papers [Meer97][Li98][Abde98] presented feedback control architectures and algorithms for QoS control. These works are targeted at supporting end-to-end QoS in distributed multimedia communication and they are not scheduling algorithms. Meeting task deadlines is not a focus of these works.

The idea and framework of feedback control scheduling has been presented in our earlier paper [Stan99]. As an evaluation and extension to our earlier work, this paper presents simulation experiments that verifies the advantage of feedback control scheduling in unpredictable dynamic environments. In this paper, we also present a discrete control model and stability analysis of the feedback control scheduling system.

6. Conclusion

In our work, we are systematically exploring the use of feedback control concepts in soft real-time scheduling systems. The goal is the development of a theory and practice of feedback control scheduling. This would be a new paradigm for real-time scheduling. We have also demonstrated feasibility of the basic approach by developing a specific algorithm called FC-EDF and scheduling architecture as presented here. Another contribution of our work is that the key performance metric of the real-time system – the deadline miss ratio – is *directly* controlled by the scheduler. The feedback control scheduler can achieve a soft performance guarantees in term of deadline miss ratios. An analytical model of the scheduling system is introduced and analyzed to facilitate tuning the scheduling algorithm systematically. The simulation experiments demonstrate that FC-EDF can maintain satisfactory deadline miss ratio and high system utilization when the workload changes dramatically.

Acknowledgment

The authors wish to thank Tarek Abdelzaher for finding an error in the earlier manuscript of this paper.

References

[Abde98] T. F. Abdelzaher and Kang G. Shin, "End-host Architecture for QoS-Adaptive Communication" *IEEE RTAS*, June 1998.

[Becc99] G. Beccari, et. al., "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems", *EuroMicro Conference on Real-Time Systems*, June 1999.

[Blev76] P. R. Blevins and C. V. Ramamoorthy, "Aspects of a dynamically adaptive operating systems", *IEEE Transactions on Computers*, Vol. 25, No. 7, pp. 713-725, July 1976.

[Butt95] G. Buttazzo and J. A. Stankovic, "Adding Robustness in Dynamic Preemptive Scheduling", *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems* (D. S. Fussell and M. Malek Ed.), Kluwer Academic Publishers, 1995.

[Gerb95] R. Gerber, S. Hong and M. Saksena, "Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes", *IEEE Transactions on Software Engineering*, Vol. 21, No. 7, July 1995.

[Hari91] J. R. Haritsa, M. Livny and M. J. Carey, "Earliest Deadline Scheduling for Real-Time Database Systems", *IEEE RTSS*, 1991.

[Jehu98] J. Jehuda and A. Israeli, "Automated Meta-Control for Adaptable Real-Time Software", *Real-Time Systems J.*, 14, 1998.

[Leho89] J. P. Lehoczky, L. Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm – Exact Characterization and Average Case Behavior", *IEEE RTSS*, 1989.

[Li98] B. Li, K. Nahrstedt, "A Control Theoretical Model for Quality of Service Adaptations", in *IEEE International Workshop on Quality of Service*, May 1998.

[Liu73] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *JACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[Liu91] J. W. S. Liu, et. al., "Algorithms for Scheduling Imprecise Computations", *IEEE Computer*, Vol. 24, No. 5, May 1991.

[Meer97] J. B. de Meer, "On the Specification of EtE QoS Control", (A. Campbell and K. Nahrstedt Eds.) *Building QoS into Distributed Systems*, Chapman & Hall, 1997.

[Pang95] H. Pang and M. J. Carey, "Multiclass Query Scheduling in Real-Time Database System", *IEEE Trans. on Knowledge and Data Engineering*, vol. 7, no. 4, August 1995.

[Rama84] K. Ramamritham and J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time systems", *IEEE Software*, Vol. 1, No. 3, July 1984.

[Ryu98] M. Ryu and S. Hong, "Toward Automatic Synthesis of Schedulable Real-Time Controllers", *Integrated Computer-Aided Engineering*, 5(3) 261-277, 1998.

[Seto96] D. Seto, et. al., "On Task Schedulability in Real-Time Control Systems", *IEEE RTSS*, December 1996.

[Shin99] 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.

[Stan98] 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.

[Stan99] 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*, June 1999.

[Stee99] D. C. Steere, et. al., "A Feedback-driven Proportion Allocator for Real-Rate Scheduling", *Operating Systems Design and Implementation*, Feb 1999.

[Zhao87] W. Zhao, K. Ramamritham and J. A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints", *IEEE Transactions on Computers* 36(8), 1987.