# The Case for Feedback Control Real-Time Scheduling

John A. Stankovic, Chenyang Lu and Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, VA22903
USA

e-mail: {stankovic, cl7v, son}@cs.virginia.edu
Fax: 1-804-982-2214

November 27, 1998

## 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, i.e., systems whose workloads cannot be accurately modeled. Unfortunately, many real-world complex systems are dynamic and unpredictable. In addition, most scheduling paradigms assume that timing requirements are known and fixed, while in real systems timing requirements are more flexible. For example, instead of a single fixed deadline for a task, a deadline range might be acceptable to the physical system. If this range was passed to the scheduling system, more robust scheduling could be possible.

In this paper, we present a new scheduling paradigm, which we call feedback control real-time scheduling. Feedback control real-time scheduling will define error terms for schedules, monitor the error, and continuously adjust the schedules to maintain stable performance. Feedback control real-time scheduling will be able to deal with dynamic systems that are both resource insufficient and with unpredictable workload. It can also be integrated with flexible real-time requirements. In this paradigm, the scheduling algorithms regard the computer system as a control system with the scheduler as the controller, and integrate practical feedback control techniques into scheduling algorithms. This paper presents a practical feedback control real-time scheduling algorithm, FC-EDF, which is a starting point in the long-term endeavor of creating a theory and practice of feedback control scheduling. Research results will apply to many real world systems including real-time database applications, defense applications, agile manufacturing, and distributed multimedia.

## 1 Motivation and Introduction

Real-time scheduling algorithms fall into two categories [Stan95]: *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][Spru89][Sha90] 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 can work in *resource sufficient* environments and those that can work in *resource insufficient* environments. Resource sufficient environments are those 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[1]. Earliest Deadline First (EDF) [Liu73] is an optimal dynamic scheduling in resource sufficient environments. EDF is a second major paradigm for real-time scheduling [Stan98b]. 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 (resource insufficient) situations [Jens85][Lock86][Abbo88][Huan89]. The Spring scheduling algorithm [Rama84][Rama90] can dynamically guarantee incoming tasks via on-line admission control and planning and thus is applicable in resource insufficient environments. Many other algorithms 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 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 either results in a highly underutilized system based on extremely pessimistic estimation of workload, or a system that performs poorly and possibly causes catastrophic results in unexpected overload situations.

Unfortunately, many real-world complex problems such as real-time database systems, agile manufacturing, robotics, adaptive fault tolerance, and C4I and other defense applications are not predictable. Because of this, it is impossible to meet every deadline of tasks. The objective of the system is to meet as many deadlines as possible[2]. For example, in complex real-time database systems, accurate knowledge about transaction resource and data requirements is usually not known *a priori* [Stan88][Best96][Kim96a]. The execution time and resource requirements of a transaction may be dependent on user input (e.g., in an information and decision support system) or dependent on sensor values (e.g., in a manufacturing system). A design based on the estimation of worst case execution time of transactions will result in extremely expensive and underutilized system. It is more cost effective to design for less than worst case, but sometimes miss deadlines. For another example, many real-time scheduling algorithms lay out tasks on a timeline into the future. The start times of the future tasks are assumed to be set. In manufacturing this future start time might be computed based on the availability

[1] Systems using static scheduling algorithms are always designed to be resource sufficient.

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

of a part on a conveyor belt by that time. However, due to accumulated errors (like the slowing of the belt due to the weight of objects on it), or a vast array of other real-world conditions that are not completely predictable, the start time may be wrong (the part may have passed or is not there yet). A missed deadline might just mean a particular product instance is now incorrect. A better solution would be to continuously monitor the location of the part and adjust the schedule as needed so that the part and the processing are synchronized. In fact, this type of solution is common in practice, but it is not part of the three scheduling paradigms listed above.

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 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. Recently, very interesting work has been done to tie the front-end control loop timing requirements analysis more closely with the on-line scheduling algorithm [Ryu98][Seto96]. However, in their work the scheduling algorithms themselves *are still open-loop* scheduling algorithms. Our new feedback control algorithms should be designed to work with these deadline ranges.

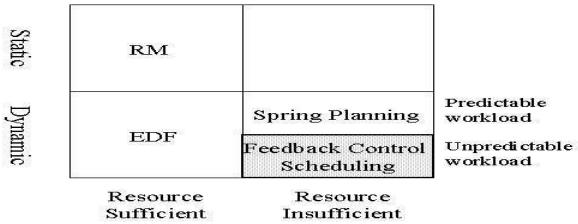


Figure 1 Real-time scheduling paradigms

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. Feedback control real-time scheduling will define error terms for schedules, monitor the error, and continuously adjust the schedules to maintain satisfactory performance. Feedback control real-time scheduling will be able to deal with dynamic systems that are both resource insufficient and with unpredictable workload (see Figure 1). Note that actually most dynamic real world applications fall into this category. The workload is usually not fully predictable in dynamic systems. It is usually extremely expensive to build a resource sufficient system to prevent *all* transient overloads caused by different reasons such as changes in the environment, simultaneous arrivals of asynchronous events, and faults in peripheral devices. In addition, due to the unpredictability in current computer architectures, operating systems and programming languages, the execution time and resource requirements of each task submitted into the system cannot be accurately estimated *a priori* [Lee96][Lee97][Li96][Heal98] [Huan96]. On-line algorithms such as RM and Spring algorithm deal with this problem by using worst case execution time of tasks, which leads to underutilized systems. Thus feedback control real-time scheduling will have significant impact on real-time systems research. We will be able to base this new paradigm on the theory and practice of control theory, adaptive control theory [Nare89][Astr95][Ioan95] [Krst95][Tao96], and stochastic control [Davi77][Mosc95]. We plan to integrate the front-end control loop timing requirements generation with the on-line feedback control scheduling. The result would be

that many applications can meet significantly more deadlines thereby improving the productivity of a manufacturing plant or an on-line real-time database system. General principles should also be applicable to feedback scheduling in non-real-time systems. This could improve the throughput and performance of general timesharing systems.

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 key central principle; rather most of the time it is used more as an afterthought and is usually ad hoc. More discussion of this appears later in the related work section.

In summary, feedback control real-time scheduling is required when the system workload is difficult to be accurately modeled. We should utilize scientific and systematic solutions to build stable feedback control schedulers in dynamic systems. In the long term, we plan to develop the theory and practice of feedback control real-time scheduling. In this paper, we begin with describing the mapping between feedback control systems with scheduling systems. We then present a class of real-time scheduling algorithm based on PID control as a case of feedback control scheduling algorithms. We present a general architecture for feedback control real-time scheduling and a new real-time scheduling algorithm called Feedback Control EDF (FC-EDF). With the PID control loop, FC-EDF can achieve robust performance in overload situations. By using nominal estimation of execution time of workload, FC-EDF can also achieve higher resource utilization than open-loop algorithms that base on worst case estimation of workload. To demonstrate how control theory could help us build stable feedback control schedulers, we describe and analyze a liquid tank model of scheduling systems, which facilitate tuning the PID control parameters systematically.

## *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 the next section we present an instance of this mapping by creating an actual algorithm and a runtime scheduling structure.

### 2.1 Control Theory and Real-Time Scheduling

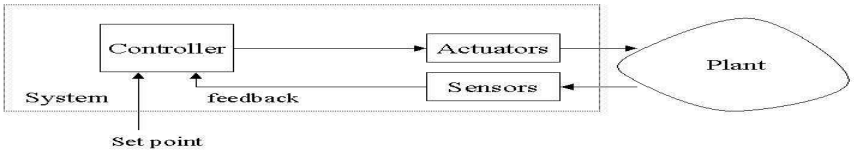


Figure 2 Architecture of Feedback Control Systems

A typical feedback control system is composed of a controller, a plant to be controlled, actuators and sensors (as illustrated in Figure 2). 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*, i.e., *error = set point – current value of controlled variable*. 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

$$Control(t) = C_P * Error(t) + C_I * \int_t Error(t) + C_D * dError(t)/dt \quad (1)$$

PID control is generally applicable to most control systems. 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. One hypothesis of our work is that we can control the arrival stream of tasks well enough that the PID control will be sufficient. However, this will likely require an adaptive form of PID control [Astr95] that can provide stable control for high order dynamic systems. Compared with other control theory techniques, an important feature of PID control is that it does not require a precise analytical model of the system being controlled. Due to the extremely complex behavior of current computer systems, it is very difficult, if not impossible, to accurately model the system dynamics of real world scheduling systems. However, certain form of approximate modeling of the scheduling system will help to tune the PID control parameters more systematically and accurately. Therefore, PID control is an appropriate and practical control technique to be used in the context of scheduling problems.

## 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 and the set point 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. For another example, if the tasks in the system are assigned different *values*, the controlled variables could be the total value of the tasks that meet their deadlines. 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 start time and period of tasks in control applications when there exists flexibility to adjust these task parameters [Ryu98][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 calls control functions to assess the impact of these differences and applies a correction to keep the system within an acceptable range of performance. Research is needed to fully develop all these ideas within the context of real world problems. More specifically, research is needed to answer the following open questions:

- What are the right choices of controlled variables, manipulated variables, and set points in real-world applications?

- What are the effective control mechanisms for feedback control scheduling?
- What are the appropriate control functions in scheduling systems? Is PID control applicable in most dynamic systems? Or are more advanced control techniques such as adaptive control needed in complex dynamic systems?
- How can we deal with the stability problem of feedback control in the context of real-time scheduling?
- How could the control parameters be tuned to achieve a stable scheduler? One of the reasons of the popularity of PID control in industry is that people have set up a set of methods for systematically tuning the control parameters. We can utilize the methodology developed by the control theory field and set up a similar tuning methodology for feedback control scheduling.
- How much performance improvement results from feedback control scheduling compared with open loop scheduling in different dynamic systems?
- How significant is the impact of overhead in feedback control scheduling and how to minimize it?
- How could feedback control scheduling be used in distributed systems?
- How can we integrate a runtime analysis of the timing constraints derived from the front-end feedback control loops in control systems with an on-line feedback control scheduling algorithm?

## *3. Feedback Control EDF*

We now present an algorithm called Feedback Control EDF (FC-EDF). FC-EDF 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 3) 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 3 with a RM scheduler. The PID controller can be enhanced with additional adaptation functions to tune the parameters of PID control on-line (i.e., adaptive control).

### 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 2). First, we need to choose the objective (i.e., set point and controlled variable) of the system. A good scheduling algorithm should always minimize the deadline miss ratio of all the submitted tasks. A classical EDF scheduler can achieve a deadline miss ratio of 0% when the system is not overloaded, however, its miss ratio increases dramatically in overload situations. FC-EDF stabilizes the performance of scheduler with admission control and service level control mechanisms. Specifically, the goal of FC-EDF is to keep the deadline miss ratio of the *accepted* tasks at or near 0, and tries to accept as many tasks as possible, thus in effect minimize the deadline miss ratio of all the *submitted* tasks. In terms of control systems, the scheduling system has MissRatio(t), i.e., the deadline miss ratio at time t, as the controlled variable, and $\text{MissRatio}_s = 0$ as the set point, thus error = $\text{MissRatio}_s$-MissRatio(t) = -MissRatio(t). Note that $\text{MissRatio}_s$ and MissRatio(t) refer to the deadline miss ratio of *accepted* tasks. 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. By controlling the requested CPU utilization below but near 100%, we can effectively minimize the miss ratio of all the submitted tasks. Third, we need to design the mechanisms (i.e., actuators) used by the scheduler to manipulate the requested CPU utilization. An Admission Controller (AC) and a Service Level Controller (SLC) are included in the FC-EDF scheduler as the mechanisms to manipulate the requested CPU utilization. The Admission Controller can control the

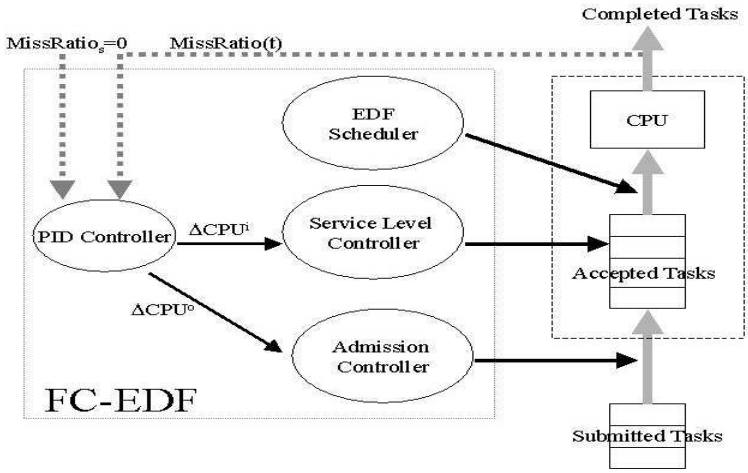


Figure 3 Architecture of FC-EDF

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 3). 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 ΔCPU, i.e., the total amount of CPU load that need to be added into or reduced from 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 flow forms a feedback control loop in the scheduling system. The EDF scheduler schedules the accepted tasks according to the earliest deadline first policy.

## 3.2 Task Model

Our initial task model assumes that all tasks have soft deadlines. The task model is similar to the imprecise computation model [Lin88][Chun90][Liu91][Shih91]. Each task $T_i$ submitted to the system is described with a tuple (I, ET, A, S, D). Each task Ti has one or more *logical* versions I = ($T_{i1}$, $T_{i2}$, … $T_{ik}$). 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]. *Milestone* methods use monotone algorithms that gradually refine the result and each intermediate result can be returned as an imprecise result. *Sieve function* methods can skip certain computation steps to tradeoff computation quality for time. *Multiple version* methods have different implementations with different cost and precision for a task. We call all these methods that can tradeoff computation precision and time as multiple logical versions of a task for convenience of discussion. Each version has different execution time and different accuracy. ET = {$ET_{i1}$, $ET_{i2}$, … $ET_{ik}$} (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. With the feedback control loop, we expect to achieve satisfactory performance even when tasks' execution times vary from their nominal execution times. Actually, the power of feedback control lies in the fact that it is possible to use relatively inaccurate control mechanisms to obtain the accurate control of a given system. The execution time is described in terms of the percentage of CPU time requirement. For example, $ET_{i1}$=0.02 means the 1st version of task Ti requires 2% of the CPU time. A = {$A_{i1}$, $A_{i2}$, … $A_{ik}$} represents the accuracy of different implementation. In this research, different versions of a task are called *service levels*. We call a version with longer execution time and better accuracy a higher service level than another version with less execution time and less accuracy. Each task has a soft deadline $D_i$ and a start time $S_i$. 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 Ti could be adjusted dynamically within the range. 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 and without affecting critical control functions such as maintenance of system stability [Seto96]. The lower bound of the deadline, $D_{i,min}$, is determined by the control functions of the control systems, e.g., the minimum sampling rate of a digital control system should be 5-10 times of the frequencies of the system characteristics. The upper bound of the deadline, $D_{i,max}$, comes from the constraints from the limited computing resources. This could be derived from the offline analysis of the system [Seto96][Gerb85b][Ryu98]. Such extension is also applicable in soft real-time systems such as distributed multimedia systems, in which the QoS specifications of a multimedia application are usually specified as intervals [Wolf97]. Note that by adjusting the deadline of a task, we also change its requested CPU utilization given that `requested_CPU_utilization = execution_time/(deadline – current_time)`. This technique can be combined with the Service Level Controller (described in section 3.4) as a mechanism to manipulate the requested CPU utilization CPU(t) of the system.

## 3.3 PID controller

The PID controller is the core of FC-EDF. It maps the miss ratio of accepted tasks (i.e., errors) to the necessary changes to requested CPU utilization (i.e., control) so as to drive the miss ratio back to 0 (i.e., set point).

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

$$\Delta CPU(t) = C_P*(-MissRatio(t))+C_I*\Sigma_{IW}(-MissRatio(t))+ C_D*(-MissRatio(t)+MissRatio(t-DW))/DW \quad (2)$$

There is a hidden tunable parameter of the PID controller - the sampling period $P_s$. The PID controller will be called every $P_s$ milliseconds. For convenience of presentation, we will take $P_s$ as the time unit in our discussion. $C_P$, $C_I$, $C_D$, IW and DW are also tunable system parameters. 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)+ MissRatio(t-DW))/DW ). ΔCPU(t)>0 means that the requested CPU utilization should be increased, and ΔCPU(t)<0 means that the requested CPU utilization should be decreased.

Note that PID control based on formula (2) can only control the miss ratio of the accepted tasks. It is also necessary to accept as many tasks as possible to minimize the miss ratio of all the submitted tasks. The following adjustment to ΔCPU(t) is included in the PID controller for this purpose.

```
if(ΔCPU(t) ≥ 0)
   ΔCPU(t) = ΔCPU(t) + ΔCPU_A
```

$\Delta CPU_A$ is a tunable parameter. The fact that $\Delta CPU(t) \geq 0$ indicates that the system is performing well, i.e., the miss ratio of accepted tasks has stabilized at 0. For example, when the system has been in a steady state and the miss ratio of accepted tasks has been 0 for a period longer than max(IW, DW), $\Delta CPU(t) = C_P*0 + C_I*0 + C_D*(0 - 0)/DW = 0$. The reasoning of the adjustment is that, if the system performs well on accepted tasks, the system attempts to increase the system load by additional $\Delta CPU_A$ to get more work done, i.e., increase the service level of accepted tasks and/or accept more tasks.

The PID controller will call the Service Level Controller and the Admission controller to change the requested CPU utilization of the system by ΔCPU. That is, if the current requested CPU utilization is CPU(t), the Service Level controller and Admission Controller will change the requested

CPU utilization to CPU(t)+ ΔCPU. Note that the PID controller always tries to change the requested CPU utilization internally by calling Service Level Controller first so that the system could response to the control faster (assuming that the earlier accepted tasks tend to have earlier deadlines than tasks accepted later by the Admission Controller). Admission controller is called only if Service Level Controller cannot accommodate ΔCPU completely.

The following is the pseudo code of PID controller.

```
void PID_ctrl()
{
      Get MissRatio(t) during last sampling period P_S;

      ΔCPU(t) = C_P*(-MissRatio(t))+C_I*Σ_IW(-MissRatio(t))+
              C_D*(-MissRatio(t)+MissRatio(t-DW))/DW

      if(ΔCPU(t) ≥ 0)
        ΔCPU(t) = ΔCPU(t) + ΔCPU_A

      ΔCPU^i = ServiceLevelCtrl(ΔCPU(t));
      AdmissionCtrl_adjust(ΔCPU(t), ΔCPU^i);
}
```

## 3.4 Service Level Controller

The Service Level Controller changes requested CPU 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 CPU 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. The PID controller first calls the Service Level Controller to accommodate the computed ΔCPU. If the Service Level Controller cannot accommodate ΔCPU completely, i.e., all the accepted tasks have been degraded to the lowest service level (when ΔCPU<0), or all the accepted tasks have been enhanced to the highest service level (when ΔCPU>0), the system turns to the Admission Controller to accommodate the uncompleted part of ΔCPU.

The following is the pseudo code of Service Level controller.

```
// Service Level controller
double ServiceLevelCtrl(double ΔCPU)
{
      /* ΔCPU^L and ΔCPU^H are thresholds to define the sensitivity of the
      control. To reduce the overhead and thrashing of control, no control
      actions are made if ΔCPU is too small. */
      if (ΔCPU^L < ΔCPU < ΔCPU^H)
        return ΔCPU;

      if (ctrl<ΔCPU^L)
      {
        while(ΔCPU < ΔCPU^L && Exist Degradable Task) {
              T_i = Select_Degraded_Task();
              ChangeSeviceLevel(task,current_level,new_level);
              ΔCPU = ΔCPU - ET_i,new_level + ET_i,current_level;
        }
       } else {
        while(ΔCPU > ΔCPU^H && Exist Enhancable Task) {
              T_i = Select_Enhanced_Task();
              ChangeSeviceLevel(task,current_level,new_level);
```

```
                ΔCPU = ΔCPU - ET_{i,new_level} + ET_{i,current_level};
          }
        }
        return ΔCPU; //return part of ΔCPU not accommodated by SLC}
}
```

## 3.5 Admission Controller

The Admission Controller controls the flow of workload into the system based on the liquid tank model (to be explained in section 3.6). When a new task $T_i$ is submitted to the system, the Admission Controller decides on whether it could be accepted into the system. Given current requested CPU utilization CPU(t) and the CPU requirement of the incoming task $ET_{i1}$[3], the Admission Controller can admit $T_i$ if $CPU(t)+ET_{i1}<1$; else reject $T_i$.

The Admission Controller's parameter CPU(t) may be adjusted when PID controller cannot accommodate ΔCPU completely with the service level control alone. Suppose the Service Level Controller changes the requested CPU utilization by $\Delta CPU^i$ and $\Delta CPU^i < \Delta CPU$, the Admission Controller will accommodate $\Delta CPU^o = \Delta CPU - \Delta CPU^i$, the part of ΔCPU that is not accommodated by SLC ($\Delta CPU^o$ is computed and returned in `ServiceLevel_Ctrl()`). It accommodates $\Delta CPU^o$ by adjusting the estimation of requested CPU utilization CPU(t).

The rationale of this action is as follows. Suppose that without any control action, the Admission Controller will, at most, let a workload of amount 1-CPU(t) into the system. With the previous control action, the Admission Controller will admit the workload at most $1\text{-}CPU(t)+ \Delta CPU^o$, which means that Admission Control will change the requested CPU utilization of the system by $\Delta CPU^o$ in the future[4]. For example, suppose the current requested CPU utilization CPU(t) = 80%, and $\Delta CPU^o$ = -10%, which means that the Admission Controller should reduce 10% of the requested CPU utilization. Without any adjustment, the Admission Controller would admit the workload requiring 1-CPU(t) = 20% of CPU into the system (assuming the submitted workload is always available). With the adjustment, however, Admission Controller will only admit a workload requiring $1\text{-}CPU(t)+ \Delta CPU^o$ = 10% of the CPU. The adjustment thus, in effect, reduces the requested CPU utilization by 10% in the future.

## 3.6 Modeling and Tuning of Feedback Control Scheduling

An important task of building a stable PID-control-based scheduler is to tune the control parameters (i.e., $C_P$, $C_I$ and $C_D$) so that the PID controller is able to achieve the set point as fast as possible. One way is to tune the parameters manually by simulations. However, this approach will 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 systems. In this work, we use a liquid tank model (Figure 4) as an (approximate) abstraction of the scheduling system. In this model, the accepted tasks are viewed as a liquid flowing into the CPU, and the completed tasks are liquid flow out of CPU. The CPU is viewed as a liquid tank. The height of the liquid level CPU(t) corresponds to requested CPU utilization of all current accepted tasks. The requested CPU utilization is 100% if the liquid tank's height is 1. Each accepted task contributes to an amount of liquid in the tank corresponding to its CPU requirements. Suppose an accepted task requires x% of the CPU time, it adds 0.0x to the liquid level. The liquid level decreases 0.0x when a task requiring x% CPU is completed. The Admission Controller controls the flow into the system. At any time, the Admission Controller can

[3] In the current design, the admission control is based on the CPU requirement of the highest service level of a task. A renegotiation process could be added to the admission control in the future so that tasks could be accepted with lower service level.

[4] Note that CPU(t) is the *estimation* of the CPU utilization. The adjustment of CPU(t) is actually the adjustment of the estimation of the CPU utilization.

admit tasks with the total CPU requirements of no more than 1-CPU(t). Ideally, the tank should never exceed its capacity. The Service Level Controller can "magically" expand or shrink the liquid inside the tank. This corresponds with the action of adjusting service levels of the accepted tasks. The manipulation of requested CPU utilization by admission control and service level control corresponds to the control of the liquid level in the tank. The EDF scheduler can guarantee the accepted tasks make their deadlines as long as water level is below 1. In terms of control theory, the liquid tank model is a first-order control system. It can be used to systematically tune the parameters of PID control to achieve stable control over the scheduling system as described below. However, the liquid tank model is only an

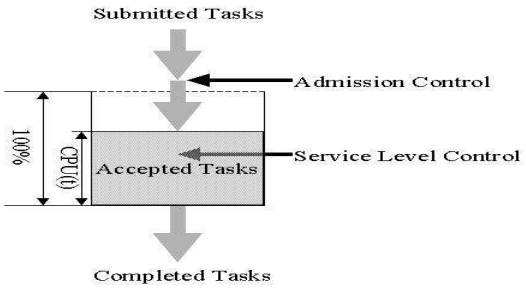

Figure 4 Liquid Tank Model of Scheduling System

approximate model of the real world scheduling system. CPU(t) is only the *estimation* of the requested CPU utilization because the system does not know the precise execution time of the accepted tasks[5]. The difference between the estimated execution time and the actual execution time is the *disturbance* and *uncertainty* in the scheduling system. Modeling such disturbance and uncertainty can enhance the accuracy of the PID control. More accurate modeling can facilitate tuning the PID control parameters systematically and precisely. In the following part of this section, we present the mathematical analysis and extension of the liquid tank model and demonstrate how PID control parameters can be tuned systematically.

For a liquid tank model which is used to represent the real-time scheduling system, we have the following mathematical model

$$y(s) = G(s)\, u(s) \qquad (3)$$

where the output $y(t)$ (or $y(s)$ in the Laplace domain) is the liquid level, $u(t)$, the control signal, is the difference between the inflow (i.e., requested CPU utilization of accepted tasks) and outflow (i.e., CPU utilization of completed tasks) of the liquid tank, and $G(s)$ is the system transfer function. In the context of scheduling systems, $y(t)$ should represent the total requested CPU utilization of the accepted tasks. However, in the current design of FC-EDF, we use deadline miss ratio of accepted tasks (i.e., MissRatio(t)) as the meaning of $y(t)$ for the following reasons. First, the ultimate goal of scheduling systems is to minimize the deadline miss ratio. Second, it is impossible to monitor the *requested* CPU utilization of the accepted tasks precisely because the actual execution time of tasks is unknown before they are completed. Third, suppose $CPU(t)$ is the total requested CPU utilization of the current set of accepted tasks, then $MissRatio(t) = b(t)CPU(t)$ where $b(t)$ is a transfer function between $MissRatio(t)$ and $CPU(t)$. According to scheduling theory, when EDF is used to schedule tasks, we know that $b(t) =$

[5] The estimated CPU(t) can be adjusted on-line as part of the PID control. See section 3.5 Admission Controller.

0 when *CPU(t)*<1.0, and when *CPU(t)*>1.0, *b(t)* depends on the characteristics of the set of current accepted tasks, e.g., arrival times of tasks, resource contention between tasks, inter task communication, precedence constraints, cache misses and page faults. This is a form of uncertainties in scheduling systems.

Let *r(t)* be the set point function to be tracked by the system and the tracking error be *e(t) = r(t) - y(t)*. As an illustration to show how a PID controller can achieve desired control system performance, we present the following analysis.

The PID controller structure is

$$D(s) = C_P + \frac{C_I}{s} + C_D s \tag{4}$$

where $C_P$, $C_I$ and $C_D$ are the P, I, D coefficients respectively. Before applying this controller, we need to model the liquid tank system. The ideal system transfer function for a liquid tank model is *G(s) = k/s* with a coefficient k > 0. Its input *u(t)* can be generated from a PID controller. However, in a real-life situation, the transfer function may be of a higher order, due to additional dynamics in the system; this gives rise to a more complicated transfer function, *G(s) = k/(s(s + a))* for some constant *a*. Moreover, there may be disturbances in the control signal *u(t)*. In real-time scheduling systems, there are two forms of system uncertainties and disturbances. The first form of uncertainty is the relationship between the deadline miss ratio and the total requested CPU utilization as described previously in this section. The other form of uncertainties or disturbances is the difference between the *estimated* execution time of tasks and the *actual* execution time of tasks because of unpredictability in computer systems and programming languages, interactions between tasks, and different user inputs. For example, when the Admission Controller accepts a task with an estimated CPU utilization of 1%, the tasks might actually consume 2% of CPU. Taking into account such modeling errors, we have the PID feedback control system structure shown in Figure 5.

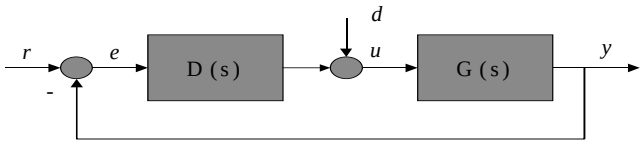


Figure 5: The PID feedback control system

To demonstrate the importance of using a formal theory to describe feedback control real-time scheduling, we present the following two cases. Assume that the desired system performance is to make the system output *y(t)* track a given set point *r(t) = $r_0$* (that is, *r(s) = $r_0$/s*). In other words we want to meet some target deadline miss ratio (i.e., $r_0$=*MissRatio*$_s$ in FC-EDF). We can achieve this goal with PID control in the following ways.

**Case (i):** Assume there is no modeling error, that is, when *G(s) = k/s*, and *d(t) = 0*. In scheduling terms this means that task execution times are known and that tasks always use that precise amount of time. In this case, the system output is

$$y(s) = \frac{k(C_D s^2 + C_P s + C_I)}{s^2 + k(C_D s^2 + C_P s + C_I)} \frac{r_0}{s} \tag{5}$$

Feedback control theory tells us that as long as we choose $C_D > 0$, $C_P > 0$, $C_I > 0$ and $k > 0$, that the system will achieve the target deadline miss ratio. Using empirical methods (or a model), it is possible to fine-tune the coefficients so that the movement of the system to the target missed deadline percentage is as fast as possible. In other words the theory tells us how to set the coefficients in the PID control, and this is one important reason for basing feedback control real-time scheduling on control theory.

**Case (ii):** The above case is unrealistic. Assume that there are both constant but unknown disturbance $d(t) = d_0$ and unmodeled dynamics in $G(s)$, that is, $G(s) = k/s(s + a)$. The unmodeled dynamics might be that tasks run longer or shorter than expected. This gives rise to the $(s+a)$ term.

In this case, the system output is

$$y(s) = \frac{k(C_D s^2 + C_P s + C_I)}{s^2(s+a) + k(C_D s^2 + C_P s + C_I)} \frac{r_0}{s} + \frac{k}{s^2(s+a) + k(C_D s^2 + C_P s + C_I)} d_0 \quad (6)$$

From control theory we know we need to choose $C_D > 0$, $C_P > 0$, $C_I > 0$ such that

$$(a + k C_D)C_P > C_I. \quad (7)$$

In other words, we are given guidance on how to choose the coefficients. From this formula one can see that the additional condition (7) for a desired PID controller design depends on the system parameters $k$ and $a$. $k$ is effectively the CPU speed (and is related to the size of the tank in the fluid model, or the flow rate of the stream assuming the tank has a height of 1.0), and $a$ is the dynamics of the execution times (and is related to leaking water in the tank model). When these system parameters are unknown, an adaptive PID controller is needed.

Adaptive control is a control methodology which provides an adaptation mechanism for adjusting the controller for a system with parametric, structural and environmental uncertainties, to achieve desired system performance [Nare89][Astr95][Ioan95][Krst95][Tao96]. Payload variation or component aging causes parametric uncertainties, component failures lead to structural uncertainties, and external noises are typical environmental uncertainties. Adaptive control has experienced many successes in both theory and practical applications and is developing rapidly with emergence of new challenging problems. When applied to this real-time scheduling control problem, an adaptive algorithm will update the PID coefficients $C_P$, $C_I$ and $C_D$ on line, using the knowledge of the tracking error $e(t) = r(t) - y(t)$, so that the design condition (7) will be met adaptively, without the knowledge of the system parameters $k$ and $a$. There have been many design techniques used in adaptive control such as direct design and indirect design, which can be applied to the control of real-time scheduling systems. When the disturbance $d(t)$ is not constant but has some statistical properties, the stochastic control method [Davi77][Mosc95] can be used to handle such disturbances to achieve some desired system performance.

Based on previous analysis, there are three different approaches of tuning PID-control-based real-time schedulers such as FC-EDF. First, we can tune the PID control parameters with large amount of simulation experiments. Second, we can choose the parameters based on modeling analysis (e.g., equations (5)(6)(7)). This approach requires that the model used in the analysis is close enough to the real scheduling system. A more practical method would be to use the modeling analysis to get a starting point of the control parameters, and then fine-tune the parameters with simulation experiments[6]. Note that with a good starting point of these parameters, the amount of experiments should be less than the first approach. Third, when the system parameters (e.g., $a$ and $k$ in equations (6)(7)) are unknown or change dynamically, an adaptive algorithm can be applied to tune the PID control parameters on-line.

[6] Actually this is the most common tuning approach that is used in industry control systems.

The previous two approaches both assume the PID control to be fixed at run-time. When the system has high order (>2) dynamics, e.g., *a* and *k* changes over time, such adaptive form of PID control can provide more stable control than basic PID control with fixed parameters. Strictly speaking, scheduling systems are high order control systems, thus the on-line tuning in the third approach is the most accurate (and also most complex). More research is necessary to explore whether adaptive PID control based schedulers are necessary in different dynamic systems.

## *4 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 [Klei70] [Blev76][Poti76]. 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 very crude correction term that seems to work via ad hoc methods. No systematic study has been done.

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. *Hierarchical Earliest Deadline* [Hari91] extends AED to maximize the *value* realized by the in-time transactions. *Adaptive virtual deadline first* [Pang92][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 adjust scheduling parameters dynamically. However, the feedback controls in these algorithms are ad hoc and the issues of stability of the schedulers are not addressed.

[Saks94] and [Gerb95a] presented a *parametric dispatching* algorithm that adjusts the dispatching time of tasks at run-time depending on the execution condition of previous tasks in the system. Parametric dispatching is not a scheduling algorithm as it assumes a predefined total order of the tasks. However, adjusting dispatching time of tasks is a possible control mechanism that could be used in feedback control scheduling.

[Seto96] and [Ryu98] both propose to integrate the design of a real-time controller with the scheduling of real-time control systems. Seto et. al. [Seto96] investigated the interaction between control performance and task scheduling. They 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. Ryu and Hong [Ryu98] addressed the similar problem in feedback control systems. Their work 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 *period calibration method* [Gerb95b][Kim96b] to derive the timing constraints (i.e. periods and deadlines) for each task in the system. Both of [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. Their algorithms are both *off-line* algorithms based on different analytical models of control systems. The scheduling algorithms in both of these cases are still open loop scheduling algorithms such as RM and EDF, which tend to perform poorly when unexpected disturbance occurs. On the contrary, feedback control scheduling in our work is *on-line* scheduling algorithms that dynamically adjust schedules based on feedback at run-time. We will also investigate how to integrate feedback control scheduling with this front-end solution by bringing that solution on-line.

Jehuda and Israeli [Jehu95][Jehu96][Jehu98] proposed an *automated software meta-controller* to adapt real-time systems when the system state changes. A system state includes the *application state* (i.e., CPU requirements and values of jobs) and the *platform state* (i.e., the computation capacity of each node in the computing system). The software meta-controller is an online agent that dynamically adapts an application's software configuration, e.g. altering operational modes and migrating tasks, to

accommodate varying runtime circumstances. Compared with feedback control real-time scheduling, the software meta-control is a higher level control that occurs only when the system state changes, e.g., when the system is ported to a different computing platform, or when the CPU requirements or values of jobs change in different environments.

In the networking area, feedback control has been used for traffic flow control (e.g., ABR service in ATM networks [ATM96][ATM97]). In 1997, Meer [Meer97] sketched a feedback control architecture for QoS control in multimedia networks. A discrete control component based on feedback from a water level monitor [Alur95] is integrated with a continuous stream model to form a close-loop control system. These works are targeted at supporting end-to-end QoS in multimedia networks and they are not scheduling algorithms.

Admission control to avoid overloading the system is widely used in real-time scheduling such as in the Spring planning algorithm [Rama97][Rama90] and the Robust Earliest Deadline (RED) algorithm [Butt95][Butt97]. These algorithms decide on whether a submitted task can be accepted based on the worst case resource requirements of the submitted task and the accepted tasks. No feedback information is considered in the admission control process.

A crucial requirement of feedback control is stability. However, ad hoc controls may become unstable and brings unsatisfactory performance in complex dynamic systems. For example, some EDF schedulers [Hari91] monitor the system performance and if it goes down they drop tasks to avoid the potential catastrophic failure of EDF in overload. This is a form of feedback control, but the system errors and the correspondent system reaction are not correlated in a systematic way. The stability issue is not explicitly addressed in these algorithms.

Many high level manufacturing scheduling tools create schedules and adjust them based on changes such as when inventory fails to arrive on-time [Zweb94]. This is sometimes called intelligent scheduling or reactive scheduling. While much can be learned from these algorithms, they are usually not on-line, don't define stability and error terms, and deal with deadlines at a high level of granularity (days, or even months).

Bestavros and Nagy [Best96a][Best96b] decompose transactions into two components: a primary task and a compensating task. The admission controller guarantees that the compensating tasks of all accepted transactions will complete before deadlines based on the CPU bandwidth allocated for compensating tasks in the system. The admission controller gauges the CPU bandwidth allocated to compensating tasks based on measured variables such as arrival rates of transactions. This feedback control scheme is based on a table, which is computed offline (using simulations) to determine the appropriate CPU bandwidth allocated to compensating tasks under current workload and system conditions. Such table-driven feedback control can provide coarse-grained control when system mode changes. However, it will be too expensive to set up a big control table via simulations because the amount of simulation testing is proportional to the number of entries in the table.

## *5. Conclusion and Future Work*

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. Results would have wide applicability since most computer systems of various types are now dealing with soft real-time constraints. We have also demonstrated feasibility of the basic approach by developing a specific algorithm and scheduling architecture as presented here. An analytical model of the scheduling system is introduced and analyzed to facilitate tuning the scheduling algorithm systematically. The future work includes the evaluation of the feedback control algorithms in different dynamic systems with realistic workloads, and the accuracy of the analytical model for scheduling systems. We will investigate the applicability of adaptive control in feedback control real-time scheduling. We will also investigate how to integrate feedback control scheduling with the flexible timing constraints in control systems.

### *Acknowledgement*

We would like to thank Professor Gang Tao for his discussions on the theory of feedback control and his help on the control theory part of this paper.

## References

**[Abbo88]** R. Abbott and H. Garcia-Molina, Scheduling Real-Time Transactions: A Performance Evaluation, 14th Intl. Conference on Very Large Database Systems, August 1988.

**[Alur95]** R. Alur et. al., The Algorithmic Analysis of Hybrid Systems, Theoretical Computer Science 138, pp. 3-34, 1995.

**[Astr95]** Astrom, K. J. and B. Wittenmark, Adaptive Control, 2nd ed., Addison-Wesley, Reading, MA, 1995.

**[ATM96**] ATM Forum, Traffic Management Group, Traffic Management 4.0. <af-tm-0056.000>, April 1996.

**[ATM97**] ATM Forum, Traffic Management Group, ABR Addendum, <af-tm-0077.000>, January 1997.

**[Benn88]** S. Bennett, Real-time Computer Control, Prentice Hall International (UK) Ltd, 1988.

**[Best96a]** A. Bestavros and S. Nagy, Value-cognizant Admission Control Strategies for Real-Time Database Management Systems, Proceedings of RTDB'96: The 1996 Workshop on Real-Time Databases, Newport Beach, California. March 1996.

**[Best96b]** A. Bestavros and S. Nagy, An Admission Control Paradigm for Value-cognizant Real-time Databases, Proc. of AAAI'96, November 1996.

**[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.

**[Butt97]** G. Buttazzo, Hard Real-Time Computing Systems – Predictable Scheduling Algorithms and Applications, pp. 245-248, Kluwer Academic Publishers, 1997.

**[Chun90]** J. Y. Chung, J. W. S. Liu and K. J. Lin, Scheduling Periodic Jobs That Allow Imprecise Results, IEEE Transaction on Computers, Vol. 19, No. 9, pp. 1.156-1.173, Sept. 1990.

**[Davi77]** M. H. A. Davis, Linear Estimation and Stochastic Control, John Wiley and Sons, 1977.

**[Gene94]** Gene F. Franklin, J. David Powell and Abbas Emani-Naeini, Feedback Control of Dynamic Systems, 3rd ed., Addison Wesley, 1994.

**[Gerb95a]** R. Gerber, W. Pugh and M. Saksena, Parametric Dispatching of Hard Real-Time Tasks, IEEE Transactions on Computers, Vol. 44, No. 3, March 1995.

**[Gerb95b]** 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 12th Real-Time Systems Symposium, 1991.

**[Heal98]** C. Healy et. al., Bounding Loop Iterations for Timing Analysis, IEEE Real-Time Technology and Applications Symposium, June 1998.

**[Huan89]** J. Huang et. al., Experimental Evaluation of Real-Time Transaction Processing, IEEE Real-Time Systems Symposium, December 1989.

**[Huan96]** T. –Y. Huang, J. W. –S. Liu and D. Hull, A Method for Bounding the Effect of DMA I/O Interference on Program, IEEE Real-Time Systems Symposium, December 1996.

**[Ioan95]** Ioannou, P. A. and J. Sun, Robust Adaptive Control, Prentice-Hall, Englewood Cliffs, NJ, 1995.

**[Jehu95]** J. Jehuda and D. Berry, A Top Layer Design Approach to Adaptive Real-Time Software, 20th IFAC/IFIP Workshop on Real-Time Programming, 1995.

**[Jehu96]** J. Jehuda, A top layer design approach to complex real-time systems, Submitted. PhD thesis, Department of Electrical Engineering, Israel Institute of Technology, 1996.

**[Jehu98]** J. Jehuda and A. Israeli, Automated Meta-Control for Adaptable Real-Time Software, Real-Time Systems, 14, 107-134, 1998.

**[Jens85]** E. Jensen, C. Locke, and H. Tokuda, A Time-Driven Scheduling Model for Real-Time Operating Systems, IEEE Real-Time Systems Symposium, December 1985.

**[Kim96a]** Y. Kim and S. H. Son, Supporting Predictability in Real-Time Database Systems, In IEEE RTAS'96, pages 38-48, Brookline, MA. June 1996.

**[Kim96b]** N. Kim et. al., Visual Assessment of a Real-Time System Design: A Case Study of a CNN Controller, IEEE Real-Time Systems Symposium, pp. 300-310, 1996.

**[Klei70]** L. Kleinrock, A Continuum of Time-Sharing Scheduling Algorithms, Proc. Of AFIPS, SJCC, pp. 453-458, 1970.

**[Krst95]** Krstic, M., I. Kanellakopoulos, P. V. Kokotovic, Nonlinear and Adaptive Control Design, John Wiley and Sons, New York, 1995.

**[Lee96]** C. Lee et. al., Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling, IEEE Real-Time Systems Symposium, December 1996.

**[Lee97]** C. Lee et. al. Enhanced Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling, IEEE Real-Time Systems Symposium, December 1997.

**[Lin88]** K. J. Lin and S. Natarajan, Expressing and Maintaining Timing Constraints in Flex, Proc. 9th IEEE Real-Time Systems Symposium, pp. 96-105, 1988.

**[Leho89]** Lehoczky J. P., L. Sha and Y. Ding, The Rate Monotonic Scheduling Algorithm – Exact Characterization and Average Case Behavior, IEEE Real-Time Systems Symposium, 1989.

**[Li96]** Y. –T. S. Li, S. Malik and A. Wolfe, Cache Modeling for Real-Time Software: Beyond Direct Mapped Instruction Caches, IEEE Real-Time Systems Symposium, December 1996.

**[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, pp. 58-68, May 1991.

**[Lock86]** C. D. Locke. Best-effort Decision Making for Real-Time Scheduling. PhD thesis, Carnegie Mellon University, Computer Science Department, 1986.

**[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.

**[Mosc95]** Mosca, E., Optimal, Predictive, and Adaptive Control, Prentice-Hall, Englewood Cliffs, NJ, 1995.

**[Nare89]** Narendra, K. S. and A. M. Annaswamy, Stable Adaptive Systems, Prentice-Hall, Englewood Cliffs, NJ, 1989.

**[Ogat97]** K. Ogata, Modern Control Engineering (3rd Ed.), Prentice Hall, 1997.

**[Pang92]** H. Pang, M. Livny and M. J. Carey, Transaction Scheduling in Multiclass Real-Time Database Systems, IEEE 13th Real-Time Systems Symposium, December 1992.

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

**[Poti76]** D. Potier, E. Gelenbe and J. Lenfant, Adaptive Allocation of Central Processing Unit Quanta, Journal of ACM, Vol. 23, No. 1, pp. 97-102, January 1976.

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

**[Rama90]** K. Ramamritham, J. A. Stankovic and P. Shiah, Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems, IEEE transactions on Parallel and Distributed Systems, Vol. 1, No. 2, pp. 184-194, April 1990.

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

**[Saks94]** M. Saksena, Parametric Scheduling for Hard Real-Time Systems, Ph.D. Dissertation, Department of Computer Science, University of Maryland, July 1994.

**[Seto96]** D. Seto, et al, On Task Schedulability in Real-Time Control Systems, Proceedings of Real-Time Systems Symposium, December 1996.

**[Sha90]** L. Sha, R. Rajkumar and J. P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, Vol. 39, No. 9, September 1990.

**[Shih91]** W. K. Shih, J. W. S. Liu and J. Y. Chung, Algorithms for scheduling imprecise computations with timing constraints, SIAM J. Computing, July 1991.

**[Spru89]** B. Sprunt, L. Sha and J. Lehoczky, Aperiodic Task Scheduling for Hard Real-Time Systems, Journal of Real-Time Systems, Vol. 1, No.1, pp. 27-60, 1989

**[Stan88**] J. A. Stankovic and W. Zhao, On Real-Time Transactions, ACM SIGMOD Record, March 1988.

**[Stan95]** J. A. Stankovic et. al., Implications of Classical Scheduling Results for Real-Time Systems, IEEE Computer, Vol. 28, No. 6, pp. 16-25, June 1995.

**[Stan97]** J. A. Stankovic, S. H. Son and J. Liebeherr, BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications, chapter in Real-Time Databases and Information Systems, A. Bestavros, K. J. Lin, and S .H. Son (eds.), Kluwer Academic Publishers, 1997, pp. 409-422.

**[Stan98a]** J. A. Stankovic and S. H. Son, Architecture and Object Model for Distributed Object-Oriented Real-Time Databases, IEEE Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98), Kyoto, Japan, April 1998.

**[Stan98b]** J. A. Stankovic, et. al., Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms, Kluwer Academic Publishers, 1998.

**[Tao96]** G. Tao and P. V. Kokotovic, Adaptive Control of Systems with Actuator and Sensor Nonlinearities, John Wiley and Sons, 1996.

**[Wolf97]** L. C. Wolf, C. Griwodz and R. Steinmetz, Multimedia Communication, Proceedings of the IEEE, Vol. 85, No. 12, December 1997.

**[Zweb94]** M. Zweben and M. S. Fox Ed., Intelligent Scheduling, Margan Kaufmann Publishers, 1994.