The Case for Feedback Control Real-Time Scheduling

John A. Stankovic, Chenyang Lu, Sang H. Son and Gang Tao*
Department of Computer Science
**Department of Electrical Engineering*
University of Virginia
Charlottesville, VA22903
*e-mail: {stankovic, cl7v, 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 new scheduling paradigm, which we call feedback control real-time scheduling. Feedback control real-time scheduling defines error terms for schedules, monitors the error, and continuously adjusts the schedules to maintain stable performance. This paper also 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.

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 can work in *resource sufficient* environments and those that can 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[1]. 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 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)

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

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 can result in a highly underutilized system based on extremely pessimistic estimation of workload.

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 task deadline. 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*. 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 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.

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

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. 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 and operating systems, the execution time and resource requirements of each task submitted into the system cannot be accurately estimated *a priori*. On-line algorithms such as RM and the 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, and stochastic control. 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 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 and scheduling systems. We then 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 build stable feedback control schedulers, we describe and analyze a liquid tank model of scheduling systems, which facilitates 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 section 3 we present an instance of this mapping by creating an actual algorithm and a runtime scheduling structure.

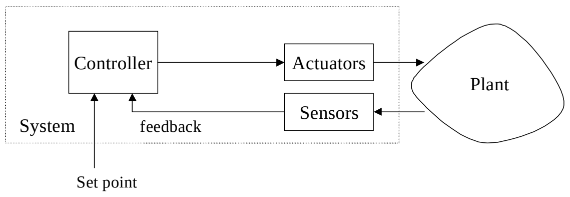

Figure 1 Architecture of Feedback Control Systems

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,* 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

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

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 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. 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 the 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 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 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? Can basic PID control achieve satisfactory performance in dynamic real-time applications? Is adaptive PID control necessary in these applications?
- 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 significant is the impact of overhead in feedback control scheduling and how to minimize it?
- 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), 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.

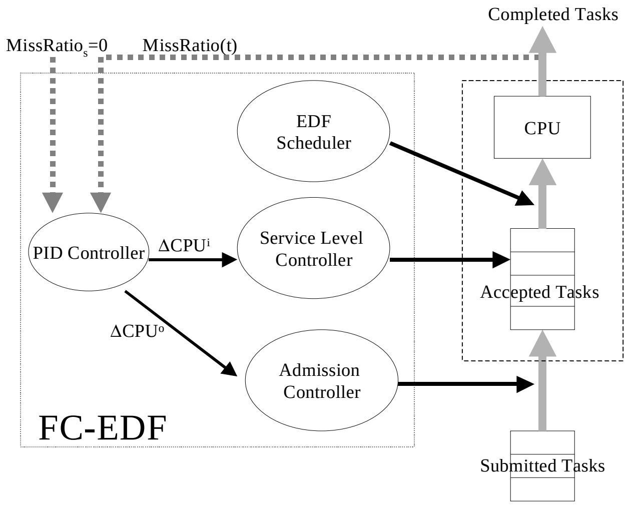

Figure 2 Architecture of FC-EDF

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 objective (i.e., the set point and the controlled variable) of the system. An ideal real-time scheduling algorithm should always minimize the deadline miss ratio of submitted tasks. In this paper, all tasks are assumed to be independent (the blocking factor will be explored in our future research). Under this assumption, an 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 improves the performance of scheduling systems with admission control and service level control. Specifically, the goals of FC-EDF is to (1) keep the deadline miss ratio of the *accepted* tasks at or near 0, and (2) accept as many tasks as possible. The first goal is achieved by feedback control. In terms of control systems, the scheduling system uses *MissRatio(t)*, i.e., the deadline miss ratio at time t, as the controlled variable, and $MissRatio_s = 0$ as the set point, thus $error = MissRatio_s - MissRatio(t) = -MissRatio(t)$. Here MissRatio_s and *MissRatio(t)* refer to the deadline miss ratio of *accepted* tasks. The second goal is necessary to minimize the deadline miss ratio of all the *submitted* tasks. A system that achieves a 0% deadline miss ratio but rejects too many tasks is not acceptable. This condition is avoided in FC-EDF by greedily increasing the CPU utilization when the system is lightly loaded (see formula (3)). 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. For simplicity of discription, 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 *ΔCPU(t)*, i.e., the total amount of CPU load that need to be added into (when *ΔCPU(t)*>0) or reduced from (when *ΔCPU(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.

3.2. Task model

Our initial task model assumes that all tasks have soft deadlines and all the tasks are independent. This paper assumes the imprecise computation model [Liu91]. Each task T_i submitted to the system is described with a tuple (I, ET, A, S, D). Each task T_i 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 $ET_{i1} \geq ET_{i2} \geq … \geq ET_{ik}$) 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. 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. 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.

Note that FC-EDF actually 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 change the requested utilization based on the following formula.

$$requestedCPUutiliztation = \frac{executionTime}{deadline - currentTime}$$

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 distributed multimedia systems, in which a QoS specifications of a multimedia application can be specified as an interval (QoS_{min}, QoS_{max}).

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 change in requested 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 utilization with the following control formula, which is an approximation of formula (1).

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

P_s, C_P, C_I, C_D, IW and DW are tunable parameters of the PID controller. P_s is the sampling period. The PID controller will be called every P_s seconds. For convenience of presentation, we will take P_s 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-DW)-MissRatio(t)*)/*DW*). The tuning of these parameters will be discussed in section 3.6. *ΔCPU(t)* is the output of the PID controller, i.e., the control signal. *ΔCPU(t)*>0 means that the requested utilization should be increased, and *ΔCPU(t)*<0 means that the requested utilization should be decreased.

The 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.

$$if(\Delta CPU(t) \geq 0)\ \Delta CPU(t) = \Delta CPU(t) + \Delta CPU_A \qquad (3)$$

ΔCPU_A is a tunable parameter called the *greedy-factor*. *ΔCPU(t)*≥0 indicates the system is lightly loaded and performing well. 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 reason for the adjustment is that, if the system is lightly loaded, the system greedily increases the system load by additional ΔCPU_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 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)+ΔCPU*. 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 (assuming that the earlier accepted tasks tend to have earlier deadlines than tasks accepted later by the Admission Controller). The admission controller is called only if the Service Level Controller cannot accommodate ΔCPU completely. The following is the pseudo code of the PID controller.

```
/* called every sampling period $P_S$ */
void PID()
{
  Get MissRatio(t) during last sampling period $P_S$;
  /*PID control function*/
  $\Delta CPU(t) = -C_P*MissRatio(t)-C_I*\Sigma_{IW}MissRatio(t)+$
        $C_D*(MissRatio(t-DW) -MissRatio(t))/DW$
  /* greedily increase system load when lightly loaded */
  if($\Delta CPU(t) \geq 0$)
    $\Delta CPU(t) = \Delta CPU(t) + \Delta CPU_A$
  /* call the Service Level Controller, which returns the portion
  of ΔCPU(t) that is not completed in it */
  $\Delta CPU^o = SLC(\Delta CPU(t))$;
  /* call the admission controller to accommodate the portion of
  ΔCPU(t) that is not completed by SLC, if there is any */
  if($\Delta CPU^o$ != 0)
    ACadjust($\Delta CPU^o$);
}
```

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 (see section 3.5). The following is the pseudo code of Service Level controller.

```
double SLC(ΔCPU)
{
  $\Delta CPU_1$=ΔCPU;
  if ($\Delta CPU_1$ <0)
    while($\Delta CPU_1$<0 && Exist Degradable Task) {
      $T_i$ = Select_Degraded_Task();
      ChangeSeviceLevel(task,current_level,new_level);
      $\Delta CPU_1 = \Delta CPU_1 - ET_{i,new\_level} + ET_{i,current\_level}$;
    }
  else
    while($\Delta CPU_1$ > 0 && Exist Enhancable Task) {
      $T_i$ = Select_Enhanced_Task();
      ChangeSeviceLevel(task,current_level,new_level);
      $\Delta CPU_1 = \Delta CPU_1 - ET_{i,new\_level} + ET_{i,current\_level}$;
    }
  /* return $\Delta CPU_1$, i.e., the portion of ΔCPU not accommodated.
  $\Delta CPU_1$ will be accommodated by the Admission Controller */
  return $\Delta CPU_1$;
}
```

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 ET_{i1}[3], the Admission Controller admits T_i if *CPU(t)*+ET_{i1}<1; else rejects T_i.

[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 levels.

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

```
void ACadjust(ΔCPU^o)
  { CPU(t)=CPU(t)-ΔCPU^o; }
```

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 of at most $1-CPU(t)+\Delta CPU^o$, which means that Admission Control will change the requested utilization of the system by ΔCPU^o in the future[4]. For example, suppose the current requested utilization *CPU(t)* = 80%, and ΔCPU^o = -10%, which means that the Admission Controller should reduce the incoming workload by 10% of the 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 $1-CPU(t)+ \Delta CPU^o$ = 10% of the CPU. The adjustment thus, in effect, reduces the requested 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). 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. Due to the complexities in computing systems, it is impossible to model and analyze the scheduling system precisely. In this work, we use a liquid tank model (Figure 3) as an approximate abstraction of the scheduling system. In this model, the accepted tasks are viewed as liquid flowing into the CPU, and the completed tasks are liquid flowing out of CPU. The CPU is modeled as a liquid tank. The height of the liquid level *CPU(t)* corresponds to requested utilization of all current accepted tasks. The requested 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 requirement. 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 finishes. The Admission Controller controls the flow into the system. At any time, the Admission Controller can admit tasks with the total CPU requirements of no more than 1-*CPU(t)*. 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 utilization by admission control and service level control corresponds to the control of the liquid level in the tank. In the ideal case, the EDF scheduler can guarantee the accepted tasks make their deadlines as long as the liquid level is below 1.

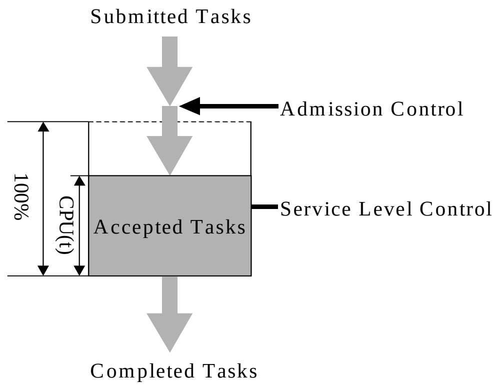

Figure 3 Liquid Tank Model of Scheduling System

The liquid tank emulates the dynamics of a scheduling system, in which tasks are accepted into the system, stay in the system for queuing delay, execution and blocking, and finally leave the system after they are completed. The concept of the liquid level fits well with the concept of CPU utilization in the scheduling system. The liquid tank model is a first-order system that can be analyzed with control theory. Unfortunately, the liquid tank model is only an *approximate* model of the real-world scheduling system. *CPU(t)* is the *estimation* of the requested utilization because the system does not know the precise execution time of the accepted tasks. The use of a fluid model to approximate a discrete scheduling system also introduces some inaccuracies, especially when the workload is composed of a small number of large tasks. However, the liquid tank model is adopted in the analysis of scheduling systems for the following reasons. First, the liquid tank model gives a framework and starting point for the modeling and analysis of the scheduling system. It can be extended to approximate the scheduling system more accurately. For example, the difference between the estimated execution time and the

[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. A more accurate estimation should also consider the estimation of blocking effects.

actual execution time can be modeled as the *disturbance* and *uncertainty* in the system. Second, it is also known in the process control area that the PID control does not require a rigorous analytical model to achieve satisfactory performance. 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 that is used to represent the real-time scheduling system, we have the following mathematical model (the following analysis is presented in the Laplace domain for simplicity of description).

$$y(s) = G(s)u(s) \tag{4}$$

where the output *y(s)* is the liquid level, *u(s)*, the control signal, is the difference between the inflow (i.e., requested 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.

Intuitively, *y(s)* should represent the total requested utilization of the accepted tasks. However, system utilization is not the first-order performance metric of real-time systems. A feedback control scheduler should control the deadline miss ratio of a real-time system instead. Fortunately, there is a correlation between the deadline miss ratio and the system utilization in many situations, which enables us to introduce the deadline miss ratio into the liquid tank model. Suppose *CPU(t)* is the total requested utilization of the current set of accepted tasks, and *MissRatio(t)* is the deadline miss ratio of accepted tasks, then the relationship between the deadline miss ratio and the requested utilization can be modeled as

$$MissRatio(t) = b(t)(CPU(t) - 1)$$

where *b(t)* is the gain from *CPU(t)* to *MissRatio(t)*. When the tasks are scheduled with the EDF policy and under certain assumptions, we know that *b(t)*=0 when *CPU(t)*<1. When *CPU(t)*>1, *b(t)* varies for different values of *CPU(t)* and is much more difficult to determine. *b(t)* is a linkage between the liquid tank model and the deadline miss ratio. A control technique called *gain scheduling* can be applied to adjust *b(t)* and the PID controller parameters based on the current value of *CPU(t)*. The modeling of *b(t)* and the extension of the liquid tank model is part of our on-going research. In this paper, we will assume *b(t)* as the constant *b*. In the following analysis, *y(s)* represents the deadline miss ratio of accepted tasks *MissRatio(s)*.

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

$$D(s) = C_P + \frac{C_I}{s} + C_D s \tag{5}$$

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) = kb/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) = kb/(s(s + a))* for some constant *a*. Moreover, there may be disturbances in the control signal *u(t)*. It is necessary to introduce uncertainty and disturbance terms into the liquid tank model because of the complexities in scheduling systems. For example, in real-time scheduling systems, a form of uncertainty or disturbance is the difference between the *estimated* execution time of tasks and the *actual* execution time.

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 $r(t) = r_0$ (i.e., $r_0=0$ 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) = kb/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

$$y(s) = \frac{kb(C_D s^2 + C_P s + C_I)}{s^2 + kb(C_D s^2 + C_P s + C_I)} \frac{r_0}{s} \tag{6}$$

Feedback control theory tells us that as long as we choose $C_D > 0$, $C_P > 0$, $C_I > 0$ and $kb > 0$, 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) = kb/s(s + a)*. For example, the unmodeled dynamics could be that tasks run longer or shorter than expected. This gives rise to the *(s+a)* term. In this case, the system output is

$$y(s) = \frac{kb(C_D s^2 + C_P s + C_I)}{s^2(s+a) + kb(C_D s^2 + C_P s + C_I)} \frac{r_0}{s} + \frac{kb}{s^2(s+a) + kb(C_D s^2 + C_P s + C_I)} d_0 \tag{7}$$

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

$$(a + kbC_D)C_P > C_I. \quad (8)$$

In other words, we are given guidance on how to choose the coefficients. From this formula one can see that the additional condition (8) 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 liquid 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. When applied to this real-time scheduling control problem, an adaptive algorithm would 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 (8) will be met adaptively, without the knowledge of the system parameters k and a. When the disturbance $d(t)$ is not constant but has some statistical properties, the stochastic control method 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., equation (6)(7)(8)). 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[5]. Third, when the system parameters (e.g., a and k in equations (7)(8)) are unknown or change dynamically, an adaptive algorithm can be applied to tune the PID control parameters on-line. 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).

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 [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 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. *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.

[Gerb95a] presented a *parametric dispatching* algorithm that adjusts the dispatching time of tasks at run-time depending on the execution conditions 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] to derive the timing constraints 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 workload occurs. In comparison, feedback control scheduling in our work is *on-line* scheduling that

[5] This is the most common tuning approach used in industry control systems.

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 [Jehu98] proposed an *automated software meta-controller* to reconfigure 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 [ATM97]). 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 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. Meeting task deadlines is not a focus of this research.

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).

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 multiple 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 as desired by the front-end control system.

6. References

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

[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.

[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 Real-Time Systems Symposium*, 1991.

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

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

[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.

[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 Real-Time Systems Symposium*, December 1996.

[Stan98] J. A. Stankovic, et. al., *Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms*, Kluwer Academic Publishers, 1998.

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

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