UNIVERSITY of CALIFORNIA
SANTA CRUZ

MANAGING THE SOFT REAL-TIME PROCESSES IN RBED

A project report submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Caixue Lin

March 2003

The project report of Caixue Lin is approved:

Professor Scott A. Brandt

Professor Ethan L. Miller

Abstract

Managing the Soft Real-Time Processes in RBED

by

Caixue Lin

This paper describes the implementation of a Resource-rate Based Earliest Deadline First scheduler (RBED). The scheduler consists of two independent components: a resource allocation policy and a scheduling algorithm. It provides uniform scheduling for hard real-time (HRT), soft real-time (SRT), and non-real-time processes.

Weighted-proportional Resource Allocation Policy (WRAP) in RBED ensures the total actual resource usage of all processes is no more than 1 under all situations. Each HRT process reserves a fraction of the resources. SRT processes are managed to receive the resource with a rate (fraction) weighted proportional to their target resource usage. WRAP automatically adjusts the resource share among the SRT processes when the load changes. Non-real-time processes receive a fair share of the leftover resources by HRT and SRT processes, but they never starve under any situation in RBED.

With resource usage no more than 1, the EDF-based scheduling algorithm in RBED always finds a feasible schedule for any set of processes. Each real-time process (including HRT and SRT) is scheduled according to its deadline, which may be dynamically adjusted because of the load change. Non-real-time processes are scheduled as SRT processes with their pseudo deadlines assigned by the scheduler.

The performance results demonstrate RBED's feasibility (in underloaded situations) and

flexibility (in overloaded situations) for fully integrated scheduling of HRT, SRT, and non-real-time processes.

Contents

List of Figures

List of Tables

Acknowledgements

Professor Scott Brandt created the initial concepts of RBED and provided the initial RBED simulator. I am especially grateful to him for his suggestion of this work and insightful discussions on problems. I would like to thank Scott Banachowski and Timothy Bisson for their valuable suggestions on some problems related to this work. I would also like to give thanks to the proofreaders and all the other members in the Computer System Lab at University of California, Santa Cruz, for their comments and help. This work is, in part, supported by Intel Corporation.

To

Xiao Zhou

for her love and encouragement.

To my family for their love.

1 Introduction

Real-time applications are widely used in industrial control systems and multimedia systems. The difference between real-time and non real-time systems is that they have different time constraints. Real-time systems must (or should) meet time constraints, while non real-time systems do not necessarily have time constraints. According to the different requirements of time constraints, real-time systems can be classified into hard real-time and soft real-time systems.

Hard real-time systems must meet their time constraints all of the time. Failure to do this leads to catastrophic consequences. Soft real-time systems are more concerned with average performance, and in most cases, they only require the response time falling within the requirements. In other words, a soft real-time system can occasionally tolerate deadline miss. For instance, time constraints in multimedia systems are softer and more flexible than those in industrial control systems. A small amount delay of video frames may not be noticeable by users. In this case, flexible resource management or quality of service (QOS) may be employed to control multimedia applications when systems cannot provide the best services.

Modern general purpose operating system schedulers lack the ability to schedule multi-class processes, such as hard real-time (HRT), soft real-time (SRT) and non-real-time processes. Most of the non-real-time schedulers do not differentiate resource allocation (how much resources to give) and scheduling (when to provide the resources) for processes. They confuse the importance and urgency of real-time (RT) applications. For example, some schedulers, such as the Linux scheduler, use a round-robin or time sharing scheduling algorithm, which provides fairness to the processes. But these schedulers may fail to satisfy the time constraints of the RT applications,

such as multimedia, because RT processes and non real-time processes are managed with the same importance and urgency level.

On the other hand, specialized real-time operating system schedulers are capable of integrated scheduling of HRT, SRT and non-real-time processes. However, they lack the flexibility to automatically relax the time constraints of SRT processes under fully loaded or overloaded situations. These schedulers also cannot guarantee the performance of non-real-time processes; Such processes may have extremely poor response time under heavy load, and also may starve under overloaded situations.

Schedulers should seamlessly manage RT applications (including HRT and SRT) with conventional non-real-time applications so that the specific integrated scheduling is transparent to the users. In this work, a Resource-Rate Based Earliest Deadline First scheduler (RBED) is implemented and ported into Linux 2.4.20. RBED provides a uniform framework by fully integrated scheduling of HRT, SRT and non real-time processes. Especially, it provides dynamic and flexible resource allocation and scheduling for SRT processes, which may tolerate graceful degraded performance because of insufficient resources.

1.1 RBED

The Resource Allocation and Scheduling (RA/S) model [2] separates resource allocation and scheduling. It treats the amount and the time to allocate the resources to the processes in a totally different way. RBED is a prototype of a complete RA/S scheduler with both independent components, resource allocation and scheduling.

In RBED, each RT process (including HRT and SRT processes) is a periodic process characterized by start time, period (*P*), resource rate (*R, also called usage*) and relative deadline

(d)[1]. Normally, the relative deadline of a periodic process is equal to its period. A periodic process releases its jobs at their deadlines (except the first job, which is released at its start time).

Resource allocation in RBED is accomplished with a rate-based allocation model, in which the available resources are allocated such that the total is less than or equal to 100% of the CPU. Resource rate of a periodic process is the ratio of its worst case execution time (*wcet*) over its period. The *wcet* of a process can be specified by the user or measured by the scheduler [1]. HRT processes have a guaranteed fixed resource rate equal to their required resource usage. SRT processes have a resource rate varied with the load of the system. In underloaded situations, SRT processes are treated as HRT processes. In overloaded situations, however, they may receive actual resources less than they require. A Weighted-proportional Resource Allocation Policy (WRAP) is employed to distribute the available resources among the SRT processes. Though non-real-time processes (called BE processes) are usually aperiodic or non-periodic processes, they are also given resource rates, which are determined by the leftover resources.

Scheduling in RBED, which delivers the resource allocations granted by WRAP, is done by Earliest Deadline First (EDF) scheduling algorithm. EDF is a dynamic online scheduling algorithm, which has the desirable property that given any set of real-time processes whose total resource usage is less than or equal to 100% of the CPU, it will find a feasible schedule [13]. So EDF is optimal for scheduling processes with total resource usage less than or equal to 1. However EDF does not work well when the resource usage is more than 1. In this case, an extended period (relative deadline) is assigned to each SRT process with the actual resource rate adjusted by WRAP so that the total actual resource rate is never more than 1.

The rest of the paper is organized as follows. The following section discusses resource

[1] Correspondingly, D is used to represent the absolute deadline of an RT process in RBED.

Table 2.1: Notations used in this paper

P	Period	R	Resource Usage (Rate)
P_h	HRT Period	R_h	HRT Resource Rate
P_s	SRT Period	R_s	SRT Resource Rate
P_b	BE Period	R_b	BE Resource Rate
D	Deadline	N	Number of Processes
D_h	HRT Deadline	N_h	Number of HRT
D_s	SRT Deadline	N_s	Number of SRT
D_b	BE Deadline	N_b	Number of BE
R_{HRT}	$\sum_{i=1}^{N_h} R_{hi}$	R_{BE}	$\sum_{i=1}^{N_b} R_{bi}$
R_{SRT}	$\sum_{i=1}^{N_s} R_{si}$	R_{sys}	$R_{HRT} + R_{SRT} + R_{BE}$

allocation policy for SRT processes in RBED. Section 3 details the EDF-based RBED scheduling algorithm. Section 4 evaluates the performance of RBED. Section 5 discusses related work of the real-time schedulers. Section 6 concludes this work and proposes future work for RBED.

2 Resource Allocation in RBED

2.1 Resource Distribution and Admission Control in RBED

A real-time system, which supports integrated scheduling of HRT, SRT, and non-real-time (called as Best Effort) processes, has to decide when and how to accept a new process. This is called process admission control. In RBED, an HRT process is accepted into the system if and only if the total resource rate of all the HRT processes is less than or equal to 1 ($R_{HRT} \leq 1$, see table 2.1). This is to ensure an HRT process always receive the resource usage as it requires.

On the other hand, a Best Effort (BE) process always enters the system without rejection. This is because RBED is designed to flexibly schedule SRT processes, but also not starve BE processes. So a minimum resource usage (β) has to be reserved for BE processes when they contends

resource with SRT processes. The value of β should be proportional to the number of the BE processes. This is certainly true because for a given β, more BE processes in the system means higher probability they may starve. A heuristic formula is used to determine β:

$$\beta = \frac{N_b}{N_b + N_s \times \gamma \times (1 + R_{SRT})^2} \times (1 - R_{HRT} - \alpha) \tag{0.1}$$

In this equation, γ is a scale factor, which is determined by the relative importance (or benefit) of SRT processes and BE processes in the system. γ can be specified by the users according to the different benefit values of SRT processes and BE processes. In real systems, some processes in the operating system kernel, such as the scheduler itself and other daemons, always occupy a small amount of CPU time (α). α can be measured approximately by monitoring the system for a long run.

Usually, if an SRT process *i* with period (P_{si}), deadline (D_{si}), and resource rate (R_{si}) is entering the system, then it is subject to the following admission control:

$$SRT\ admission = \begin{cases} accepted, & \text{if } R_{si} \leq (1 - R_{HRT} - R_{SRT} - \beta - \alpha) \\ rejected, & \text{otherwise} \end{cases} \tag{0.2}$$

,where R_{HRT} and R_{SRT} are the total resource rate of the HRT and SRT processes respectively before the SRT process enters the system.

In RBED, however, overloaded usage is allowed. Any SRT process can enter the system at all situations. In this case, graceful performance degradation is accepted if an SRT process misses deadlines. When the system is already in heavy overloaded situation, the user may not like the worse performance because of more accepted SRT processes. A maximum total usage ($R_o > 1$) of the SRT processes can be defined to avoid the problem. Now SRT process *i* has the following admission control:

$$SRT\ admission = \begin{cases} accepted, & \text{if } R_{si} \leq (1 - R_{HRT} - R_{SRT} - \beta - \alpha) \text{ or} \\ & \text{if Overload Allowed \&\& } R_{si} \leq (R_o - R_{HRT} - R_{SRT} - \beta - \alpha) \\ rejected, & \text{otherwise} \end{cases} \quad (0.3)$$

In RBED, the default value of R_o is positive infinite, which allows any SRT process to enter the system.

In an overloaded system, the actual resource usage (rate) for soft real-time processes is $AR_{SRT} = min\{R_{SRT}, 1 - R_{HRT} - \beta - \alpha\}$. With this, the actual resource usage for Best-Effort processes can be computed as follows: $AR_{BE} = 1 - R_{HRT} - AR_{SRT} - \alpha$.

2.2 Resource Allocation for SRT processes

As stated in section 1, in an underloaded system each SRT process *i* receives its desired resource usage, which is equal to the actual resource usage:$R_{si} = AR_{si}$. In this case, an SRT process works exactly like an HRT process. However in an overloaded system, some SRT processes may receive less resource usage than they require and thus miss their deadlines. In order to satisfy the necessary condition for RBED, which is EDF-based scheduler, flexible resource allocation policy needs to be employed to distribute the available resources among SRT processes. For SRT systems, there are many existing resource allocation policies, such as *priority* based and *benefit value* based resource allocation, which are described as follows. Resource can be allocated based on the *importance values* (or *priorities*) [14] specified by the users or applications. The SRT processes with higher priority are allocated required resources first (They will be scheduled like HRT processes). SRT processes with lower priority may receive part of their resources demand or may not receive any resource at all. Based on this policy, the schedulers always favor resource allocation for SRT processes with higher priority; but SRT processes with lower priority may starve.

An alternative is to use *benefit value* to solve the resource contention problem for SRT processes. *Benefit value* indicates how much benefit a process can receive when it completes its execution. If the process completes before its deadline, it usually achieves a constant benefit value. If the process completes after its deadline, its achieved *benefit value* drops dramatically with the amount of the delay increasing. The goal of such a system is to get maximum total *benefit value* of all the processes. However this resource allocation policy requires the user to specify the format of the *benefit value*, which is usually not a constant value but a function of time [10], for the applications. Furthermore, it is hard to get an optimal maximum total *benefit value*.

With no *importance value* (priority) or *benefit value* specified, all the SRT processes have the same priority to get their required resources. In RBED, the way to allocate the resources to the SRT processes is fair proportional allocation [7, 9]. Fair proportional allocation does not necessarily mean equal allocation. In RBED, the SRT processes receives a proportion of available resources proportional to their desired resource rate. Given $R_{SRT} = \sum_{i=1}^{N_s} R_{si}$ and $AR_{SRT} = min\{R_{SRT}, 1 - R_{HRT} - \beta - \alpha\}$, the new adjusted resource rate (wR_{si}) for SRT process *i* is computed as follows:

$$wR_{si} = \frac{R_{si}}{R_{SRT}} \times AR_{SRT} \tag{0.4}$$

By this, the target resource rate of each SRT process has been lowered to a new actual resource rate. As a result, the total actual resource usage of all the processes is still less than or equal to 1. This fair proportional resource allocation policy distributes available resources in a relatively fair way among SRT processes. However it does not guarantee any SRT process meet its time constraints since none of them receives its required resource rate.

A weight, which is a scale factor indicating the flexible lower bound of the resource

rate, can be associated with a SRT process when the time constraints may not be met because of an overloaded system. For instance, an SRT process (rate=45%) with a weight=0.8 means that it would be much better to schedule it with a rate$\geq 36\%(45\% \times 8)$ rather than a rate$< 36\%$. RBED allows the users to specify different weights (W) for different SRT processes so that it flexibly manages and adjusts the share of the SRT processes under overloaded situations. Suppose W_{si} ($W_{si} > 0$) stands for the weight specified by the user for SRT process *i*. The new weighted-proportional resource allocation is as follows:

$$wR_{si} = \frac{W_{si} \times R_{si}}{\sum_{i=1}^{N_s}(W_{si} \times R_{si})} \times AR_{SRT} \tag{0.5}$$

This is called Weighted-proportional Resource Allocation Policy (WRAP). From the equation, the wR_{si} may be bigger than the target rate (R_{si}) of the SRT process. In this case, the extra resource rate ($wR_{si} - R_{si}$) is fairly distributed among all the other SRT processes. This procedure is also done recursively until no extra resources is left. From observation, equation 4 holds for any set of SRT processes in underloaded situations. In addition, equation 4 is a special case of equation 5 with $W_{si} = 1(1 \leq i \leq N_s)$.

The resource allocation action in WRAP will be triggered by any change of system status, which occurs when a new HRT or SRT process enters the system, an HRT or SRT process leaves the system, or any activity of BE process causes the β to change. WRAP dynamically adjusts the actual resource rate of all the SRT processes with change of system load.

WRAP integrates the requirement of the users and the actual time constraints of the applications by combining a new weight with these two components. With this policy, the SRT processes with bigger weights achieve higher resource rates. On the other hand, the SRT processes with smaller weights receive lower resource rates, but never starve. WRAP is totally different from the *benefit value* or *priority* based resource allocation policy. First, the different weights in WRAP

may favor some SRT processes, but also never starves any of them. Second, WRAP dynamically maintains the actual resource rate of the SRT processes to ensure the total resource rate is no more than 1.

3 EDF Scheduling in RBED

Scheduling in RBED is achieved based on EDF algorithm. EDF is a dynamic fixed-priority scheduling algorithm, which is optimal of scheduling processes on one non-idle and preemptable CPU with load less than or equal to 1 [13]. In RBED, the EDF-based scheduling algorithm always picks the process with the earliest deadline to run on the CPU and controls the length of the run according to the worst case execution time (*wcet*) of the process. If the process cannot complete its execution in time (before its deadline), it will be preempted from the CPU immediately and another process with the earliest deadline will be selected to run. During a process's execution time, it may be preempted by a new process with an earlier deadline.

EDF based schedulers are completely different from *priority* based schedulers. The deadlines in RBED are used only to notify the EDF scheduler which process should run on the CPU. In contrast, the priorities in *priority* based schedulers determine both key roles of the scheduling:how much of the resources and when the resources should be allocated to the process.

3.1 Pseudo Deadline-based EDF Scheduling for SRT processes

In underloaded situations, all the SRT processes are considered as HRT processes and are feasibly scheduled by the EDF-based RBED scheduler with other processes.

In overloaded situations, RBED scheduler provides graceful degraded performance for

SRT processes by lowering the desired (target) resource rate of them (section 2) so that the total resource usage is less than or equal to 1 (This is ensured by WRAP in RBED). The RBED scheduling algorithm automatically adjusts the relative deadline of each SRT processes after the resource allocation action in WRAP has been triggered. The basic idea is to extend the deadline (period)[2] of each SRT process, but keeping its *wcet* per period constant. After deadline extension, the actual deadline is not the target deadline of the SRT process. In RBED, the extended deadline of an SRT process is called virtual deadline. In underloaded situations, the virtual deadline of a SRT process is always equal to its target deadline.

In RBED, a deadline miss (process cannot completes before its deadline) should never occur since EDF always can find a feasible scheduling when the total resource usage is less than or equal to 1. One exception is users specify inaccurate estimation of *wcet*. The solution is to use a one-shot timer to bound the amount of time that a process will run on the CPU. Once a process is selected by RBED scheduler to run on the CPU, a one-shot timer is also set associated with it. The one-shot timer will expire after *wcet* of the process. The timer handler associated with the one-shot timer is programmed to preempt the process immediately. In this case, a notification can be sent to the application that it has missed its deadline and the user or the application may choose to terminate the process or shed load. RBED, however, automatically adjusts the *wcet* of the SRT process until it can be feasibly scheduled. This triggers the resource re-allocation action in WRAP, but also ensures no deadline miss occurs with the extended deadline. RBED scheduling algorithm provides fine time granularity because of using the one-shot timer instead of the conventional interval timer.

Since EDF uses the deadline as the unique criterion to schedule processes, the deadline of a process is a critical element in RBED. Periodic processes, such as HRT and SRT processes, always

[2]Deadline extending or shortening is used for period or relative deadline; Deadline advancing or moving is used for absolute deadline

Table 3.1: Some extended notations for SRT

pD	previous deadline	nD	next deadline
cD	current deadline	ecD	extended current deadline
cP	current period	*wcet*	worst case execution time
eP	extended period	ecP	extended current period
cT	current time	aT	accumulated CPU time

have deadlines (section 1). However BE processes usually have no deadline associated with them. The basic idea is to assign a virtual (pseudo) deadline to each BE process (details are described in next subsection). In this paper, a uniform name, pseudo deadline, is used for virtual deadline (for SRT processes) and pseudo deadline (for BE process). For an HRT process, its pseudo deadline equals to its target deadline. As a result, RBED is an earliest pseudo deadline first scheduler.

According to WRAP in RBED, the pseudo period (deadline) cannot only be extended with increasing load, but can also be shortened with decreasing load. Two pseudo period change mechanisms are employed in RBED.

In order to clearly show the two pseudo period change mechanisms, some extended notations are defined for SRT processes in table 3.1 (All the periods and deadlines are pseudo periods and deadlines).

Figure 3.1 shows the two mechanisms of pseudo period (deadline) extension. The blue lines denote the old actual rate and the red lines denote the new actual rate (wR). The two mechanisms work in a same way for the SRT processes which have finished using *wcet* in the current pseudo period. They both extend the pseudo period of such a process *i* to a new pseudo period (eP), which does not take effect until next release time of the process.

$$eP_{si} = \frac{wcet_{si}}{wR_{si}} \tag{0.6}$$

So the next absolute deadline of the SRT process has also been pushed forward:

$$nD_{si} = cD_{si} + eP_{si} \tag{0.7}$$

For the SRT processes which still need to be scheduled in the current pseudo period, the two mechanisms also do the same (equation 6) pseudo period change for the periods after the current one. But they differ in dealing with the change to the current pseudo period of these SRT process. Mechanism 1 computes the current pseudo period starting from the release time (previous pseudo deadline):

$$ecD_{si} = max\{cT + wcet_{si} - aT_{si}, pD_{si} + eP_{si}\} \tag{0.8}$$

$$ecP_{si} = ecD_{si} - pD_{si} \tag{0.9}$$

In this case, idle time may be generated for the SRT process since it still takes the previous pseudo deadline as its release time, which is actually far behind the current time. But this mechanism ensures the process uses a new extended pseudo deadline immediately.

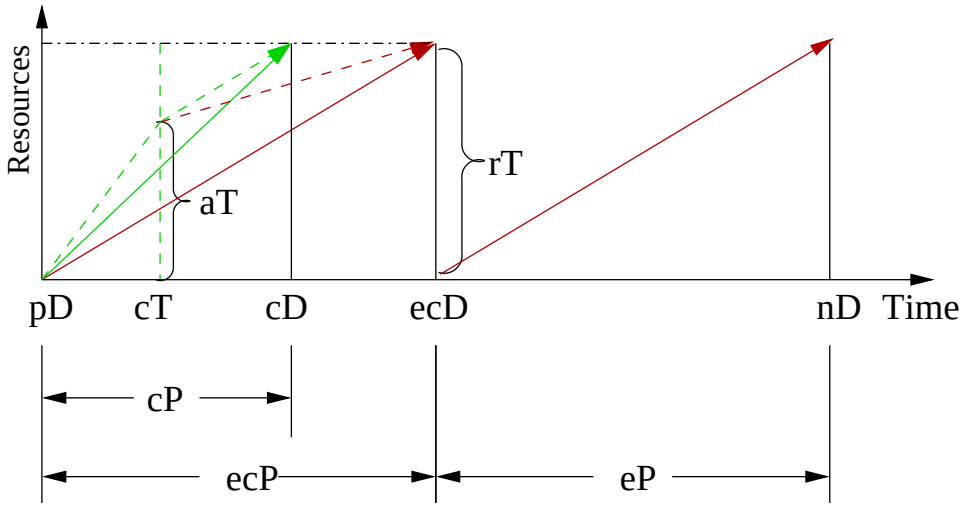
Mechanism 2, however, computes the current pseudo period starting from current time, but not the release time:

$$ecD_{si} = max\{cT + \frac{wcet_{si} - aT_{si}}{wR_{si}}, pD_{si} + eP_{si}\} \tag{0.10}$$

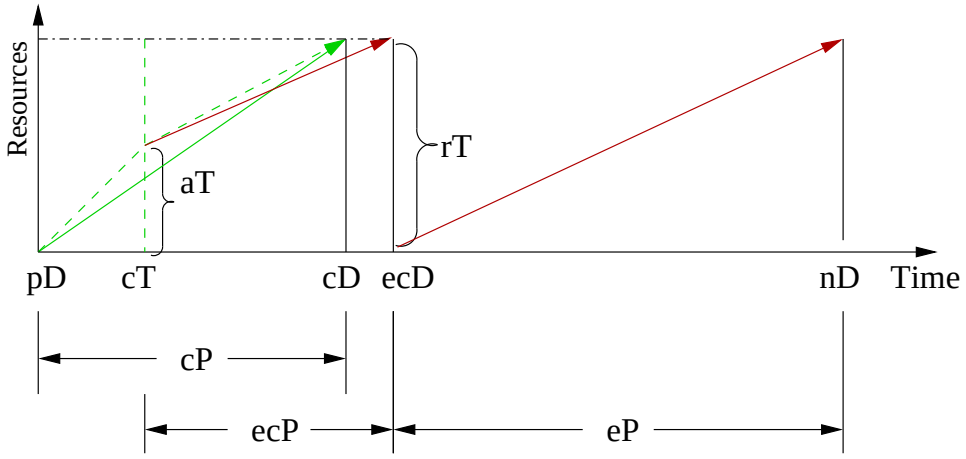
$$ecP_{si} = ecD_{si} - cT \tag{0.11}$$

In this case, there is no idle time generated for the SRT process. But the SRT process suffers a pseudo deadline jitter problem: it uses the old pseudo period, then a temporary one and finally a new extended pseudo period.

It is easy to show the equation 6, 7, 8 and 9 also hold for pseudo period (deadline) shortening of SRT processes. After the pseudo period (deadline) change in both mechanisms, the SRT



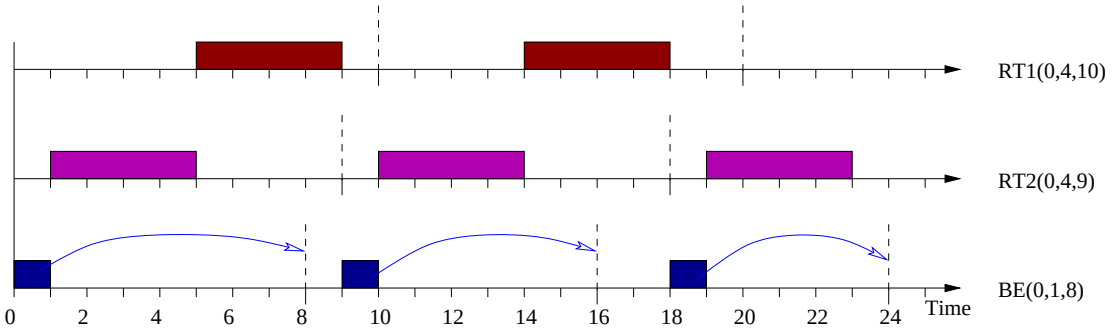
(a)



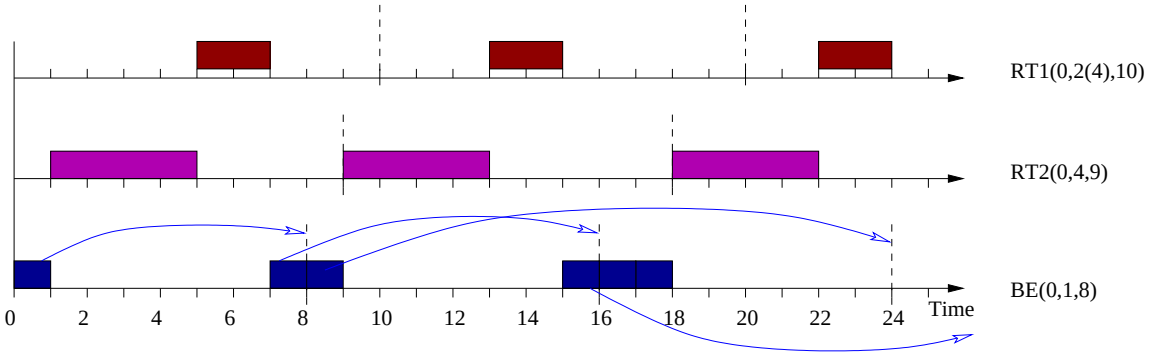
(b)

Figure 3.1: SRT Pseudo Period (Deadline) Extension. (a) Mechanism 1. (b) Mechanism 2

processes can still meet all the time constraints at any pseudo deadlines (including the current one and others). Thus, RBED dynamically maintains the feasibility of scheduling the processes by using the two components: WRAP and pseudo deadline earliest first scheduling.



(a)



(b)

Figure 3.2: Scheduling BE processes in RBED. The blue arrows indicate the pseudo deadlines, by which EDF schedules the BE process in the corresponding periods. (a) Scheduling BE process as periodic process. (b) BE process uses up the slack time

3.2 Scheduling BE Processes as Periodic Processes in RBED

All BE processes share the leftover resources by HRT and SRT processes. The weighted resource rate of BE process *i* is computed as follows:

$$wR_{bi} = \frac{max\{\beta, 1 - R_{HRT} - R_{SRT} - \alpha\}}{N_b} \tag{0.12}$$

HRT and SRT processes can be normally scheduled by EDF-based algorithm because

each of them has periods and deadlines. However, BE processes lack these time properties. In order to use EDF-based scheduling seamlessly for multi-class processes, RBED needs to assign pseudo period (deadline) to BE processes. This is achieved by assigning a pseudo *wcet* to each BE process:

$$eP_{bi} = \frac{wcet_{bi}}{wR_{bi}} \tag{0.13}$$

A good pseudo *wcet* should achieve fast response time for interactive BE processes. RBED tunes pseudo *wcet* by monitoring the actual response time of some interactive BE processes, such as character typing within an editor (vi, emacs). In the implementation of the RBED scheduler, the pseudo deadline (relative) of a BE process is maintained to be a fixed value at run time. The *wcet* of a BE process then will be adjusted according to the new resource rate. This avoids huge overhead of the resource re-allocation actions triggered in WRAP since many BE processes enter and leave the system frequently.

An SRT process usually sleeps until the next release time (current pseudo deadline) after it completes the current execution in the current period. But a BE process may not sleep after it completes its current execution (pseudo *wcet*) because it is usually not a periodic process and may require more resources than the default pseudo *wcet*. That is to say, the BE process is automatically released immediately after it finishes using the given pseudo *wcet*. In this case, in order to ensure each BE process receive no more resource than pseudo *wcet* in one pseudo period, the BE process is punished by advancing the current pseudo deadline to the next pseudo deadline. As a result, its resource rate is always less than or equal to wR_{bi} at all pseudo deadlines.

Furthermore, by using pseudo deadline advancing, BE processes also use up all slack time left by SRT processes. A BE process continues to receive resources during the slack time after it has already finished using the resources in the current pseudo period.

Figure 3.2 shows the scheduling of one BE processes with two other real-time processes in RBED. In Figure 3.2a, the BE process has pseudo $wcet = 1$ and pseudo $period = 8$ and is released at time 0. It is scheduled exactly as a periodic process, which advances its current pseudo deadline to the next deadline after it finishes the current pseudo *wcet* in the current pseudo period. Figure 3.2b shows the BE process uses up the slack time left by SRT processes, which is given an inaccurate *wcet*. RT1 is given a *wcet* of 4, but its actual execution time is 2. So there is slack time of 2 left during each period of RT1. Fortunately,the BE process uses up all the slack time without causing any of the two real-time processes to miss a deadline. For instance, the interval [7,9], which is the slack time left by RT1, is used by the BE process. The pseudo deadline of the BE process is advanced to 24 and 32 respectively after it finishes the slack time [7,8] and [8,9]. At time 9, the BE process stops its execution until time 15 since its new pseudo deadline npD = 32 is larger than the current deadlines of both RT1 (D=18) and RT2 (D=20). So the BE receives total time of 3 for the corresponding pseudo deadline at 24. The actual resource rate of the BE process is constant (R=1/8=2/16=3/24) at all pseudo deadlines.

More importantly, the pseudo deadline advancing mechanism for BE processes matches the expected behaviors of CPU bound and I/O bound BE processes: the I/O bound processes get boosted after a long period of blocking since its new release time is reset to current time, while the CPU bound processes will receive relatively less resources since their pseudo deadlines have been advanced far forward.

4 Performance Measurement

4.1 Performance Metric

Though there is no unique and optimal quality metric to measure the performance of an SRT scheduler, the basic metrics should demonstrate its feasibility under underloaded situations and its flexibility under overloaded situations. These metrics include resource allocation fairness, the number of deadline misses, the distribution of deadline miss,and the jitter of the response time, etc. The behavior of the BE processes is also important when process starvation is not allowed in the system.

1. Resource allocation fairness: the fairness of resource allocation among SRT processes, and between RT processes and non-real-time processes is an important metric for measuring the performance of a real-time scheduler that supports seamlessly scheduling of multi-class processes.

2. Deadline miss: The number and frequency of deadline misses, the amount by which they are missed, and the deadline miss distribution among SRT processes are all important characterizations of SRT scheduler performance measurements. Two methods can be employed to count the deadline misses for the SRT processes:

    - The scheduler knows whether and when an SRT process misses its deadlines. It traces all the SRT processes by recording their status (deadline misses or not) at their deadlines. Applications can retrieve the trace data by system calls.

    - The applications also know whether and when they miss deadlines. This is a better method than the previous one in measuring the performance on different schedulers

because no kernel instrument is needed. However, the measurements done by scheduler have better time resolution.

3. Jitter of the response time (completion time): For multimedia applications, people usually pay more attention to the jitter than the delay since human vision and hearing system are more sensitive to the variance of the response time. In most cases, smaller jitter is more important than fewer deadline misses. Jitter measurement can also be done by scheduler or by application.

4. Performance of BE processes: For a system integrated with HRT, SRT and BE processes, not only the performance of HRT and SRT processes is important, but also BE processes starvation should be avoided. Typically, BE processes are CPU bound processes, such as compiler and bursty math calculation, or I/O bound interactive processes. These BE processes do not have time constraints. However, a faster response time always gives better performance.

5. Overhead of the scheduler: The overhead of the scheduler and context switch are also important performance aspects.

4.2 Simulation and Experimental Results

The RBED simulator is an event-driven Java application with visualization interface support. In this work, SRT process management and QOS management were also implemented in the RBED simulator. So far, the RBED simulator provides fully integrated scheduling of HRT, SRT and BE processes.

In addition, RBED was ported into Linux (called RBED scheduler) by using an one-shot timer. Linux programs the regular programmable interval timer (PIT) to issue timer interrupts at a

frequency of 100-HZ. This means that Linux only provides very low timer resolution, which is not enough to support the refined resource allocation granularity in RBED.

In order to solve this problem, two versions of the RBED scheduler were implemented on Linux: RBED based on the PIT and RBED based on the Advanced Programmable Interrupt Controller (APIC). The former uses the one-shot mode instead of the periodic mode of the PIT. The process running on the CPU is associated with a one-shot timer, which bounds its maximum execution time. Also, the PIT is used to maintain the regular interval timer interrupts for system time service. The idea is to issue the earliest interrupt, which is either the one-shot timer of the running process or the regular interval timer. The APIC-based RBED scheduler uses the APIC timer [16], which can achieve μ-second precision, to issue interrupt for the running process. In both cases, the process is preempted as soon as the one-shot timer expires. The one-shot mechanisms in RBED ensure no process overrun its resource reservation and provide processes with a faster response time.

An EDF-based scheduler (called edf/linux scheduler) was also implemented on Linux to compare the efficiency of the EDF algorithm in RBED. This edf/linux scheduler schedules real-time processes with EDF, but schedules BE processes with the general Linux round-robin scheduling algorithm.

The experiments were performed on a standard PC machine with a single 200 MHz Pentium Pro processor, 96 MB of memory, and 8GB of hard disk space. The system used for testing is RedHat 8.0 and the Linux kernel version is 2.4.16. In both the RBED simulator and RBED scheduler in Linux, α is set to 1% and γ is set to 2. Also the basic pseudo relative period of BE processes in RBED scheduler is set to 100ms. These parameters can be tuned by running many different system workloads. Table 4.1 and 4.2 are the workloads used both in the RBED simulation and RBED scheduler. In the experiments, all process related data is traced in kernel every time there

Table 4.1: workload1 (underloaded)

Process	P_i	R_i	description
RT	0.2s	75%	Periodic SRT process
BE	No	100%	Endless loop

Table 4.2: workload2 (overloaded, weight=1 for all the three SRT processes)

Process	P_i	R_i	description
SRT1	0.2s	45%	Periodic SRT process
SRT2	0.5s	45%	Periodic SRT process
SRT3	1.0s	45%	Periodic SRT process
BE	No	100%	Endless loop

is a context switch. The data is then periodically dumped from the kernel to user space through a system call (*rbed_tracedump*).

Figure 4.1 shows the results with workload1 in Linux round-robin scheduler, the RBED simulator, edf/linux scheduler and RBED scheduler (PIT-based and APIC-based RBED schedulers have almost the same results) respectively. Linux uses round-robin policy to distribute fairly the resources between the two processes regardless what resource requirements of them. Each of them received about 50% of the CPU time. In this case, the SRT process missed all the deadlines. The processes showed almost identical behaviors in the RBED simulator, edf/linux scheduler and RBED scheduler. Each SRT process received its required resources (75%) and met all the deadlines. The BE received the leftover resources (about $1 - 75\% - \alpha = 24\%$). These results demonstrate the edf/linux scheduler and RBED ensure all the time constraints for real-time processes are met when the total resource usage is less than 1. Also RBED provides performance for BE processes at least as good as that in edf/linux scheduler.

Figure 4.2 shows the results with workload2 in Linux round-robin scheduler, the RBED

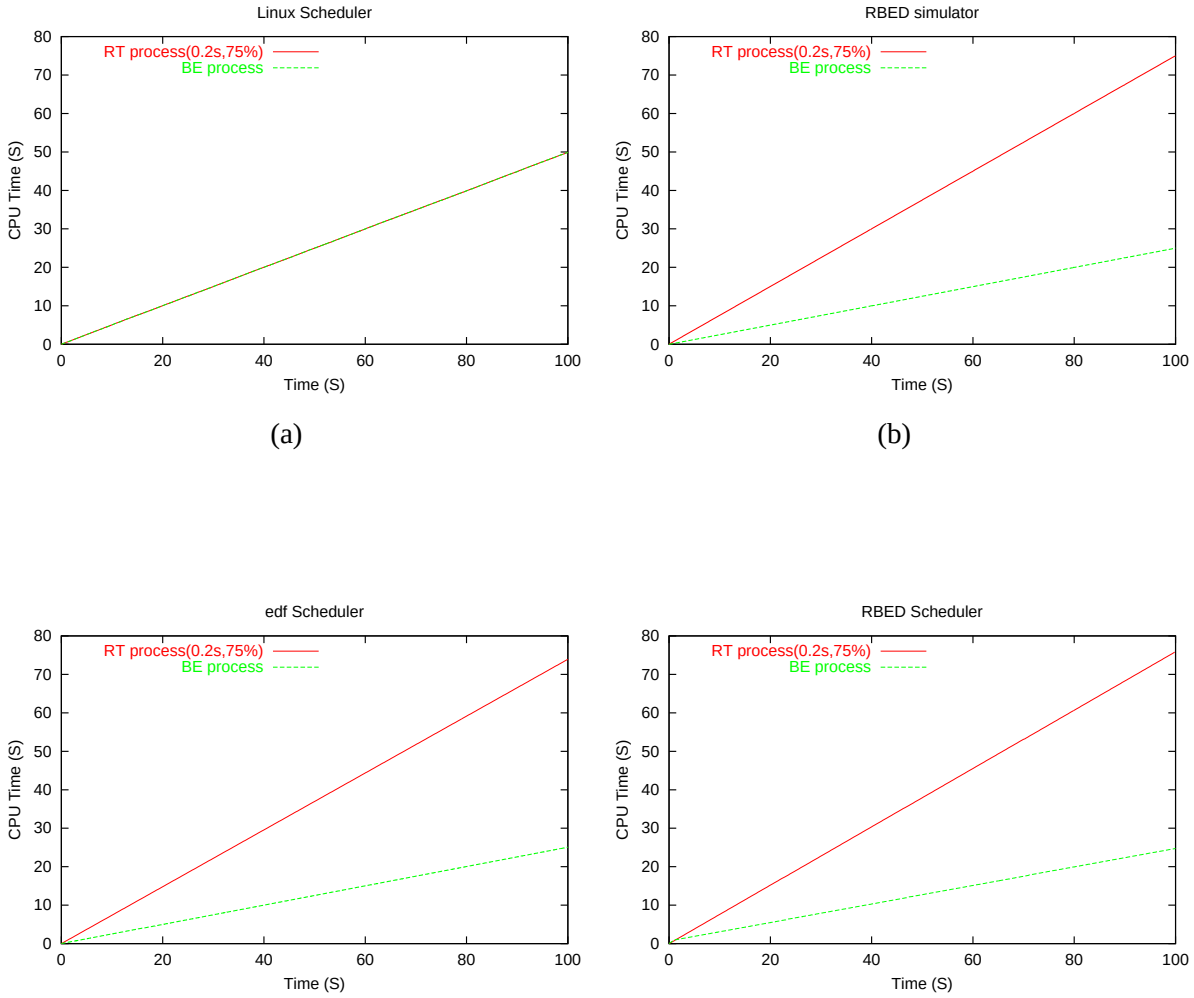


Figure 4.1: The performance of different schedulers with workload1.

simulator, RBED scheduler with PIT, and RBED scheduler with APIC respectively. Again, the four processes shared fairly the resources in Linux scheduler, *i.e.* each of them received about 25% of the CPU time. The three SRT processes missed $100\%, 100\%, 85\%$ of their deadlines respectively. In the RBED simulator, the SRT processes received resources proportional to their resource rates

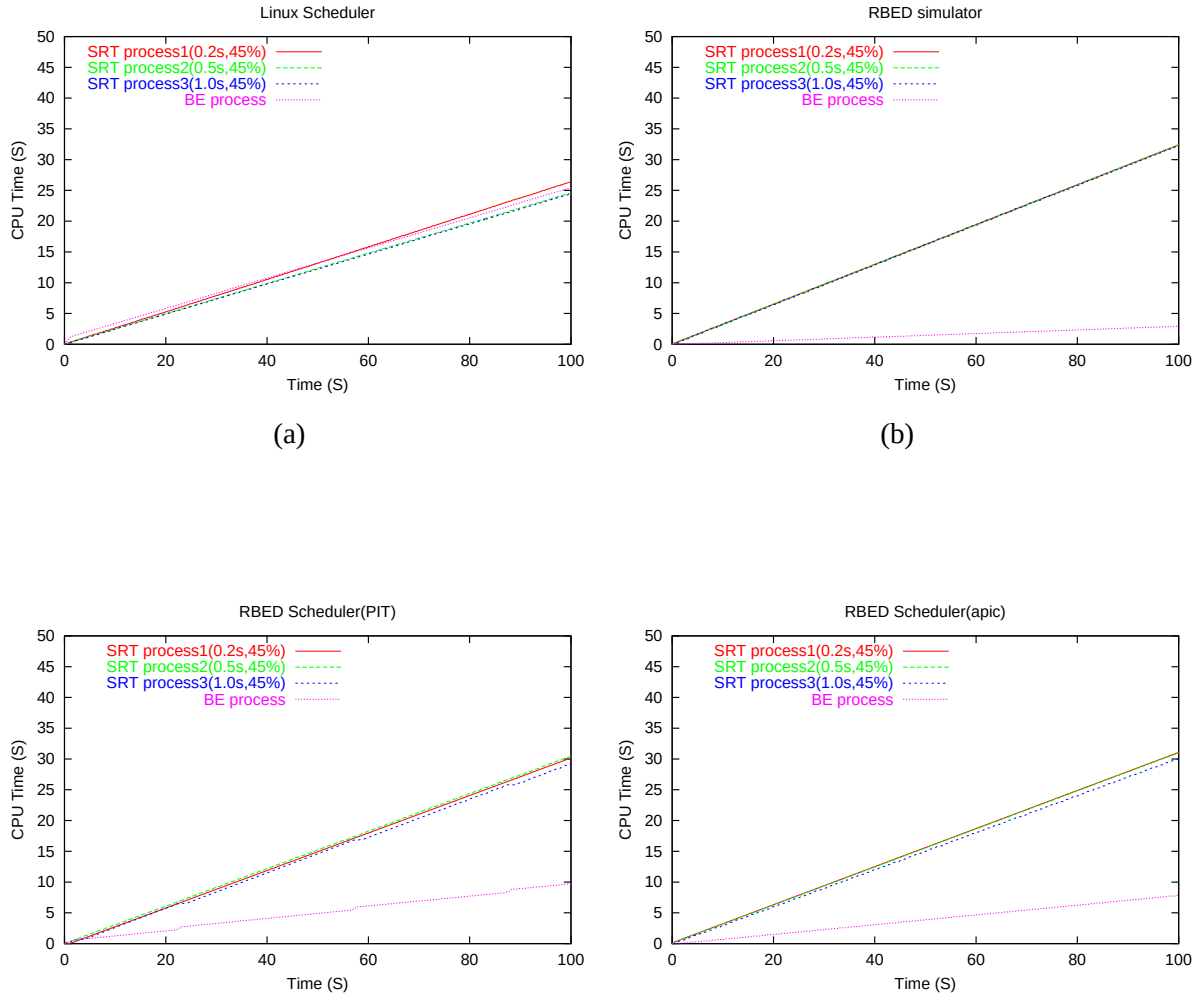
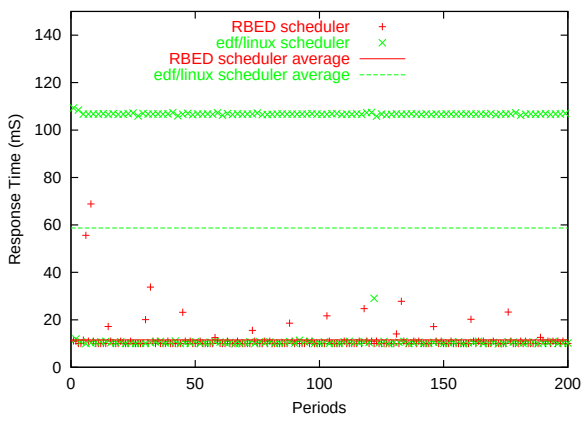


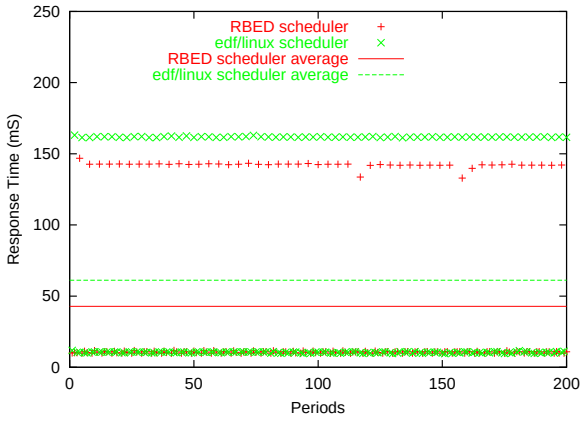
Figure 4.2: The performance of different schedulers with workload2.

(45%). Since the BE process received the minimum reserved resources β, which is about 3% by computation from equation 0.1 for the given α and γ. Thus each of the three SRT processes received about 32% of the resources.

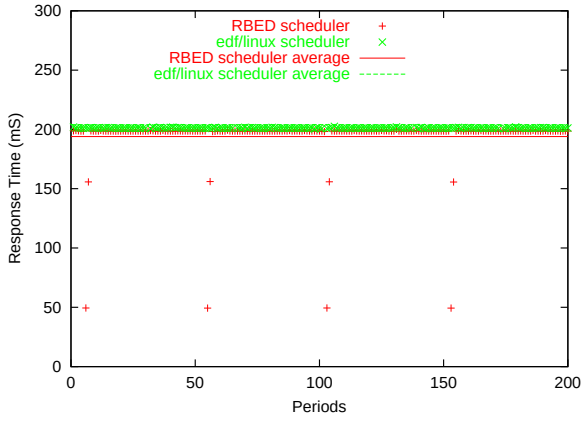
However, in RBED scheduler with PIT, SRT1, SRT2 and SRT3 received about 30%, 30.4%, 29.6%



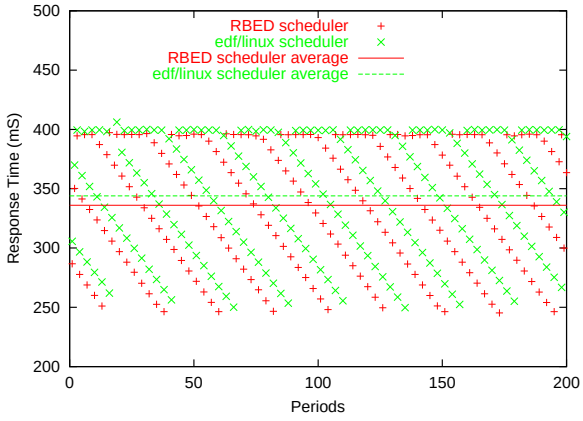
(a).BE(period,wcet)=(93ms,10ms)



(b).BE(period,wcet)=(19ms,10ms)



(c).BE(period,wcet)=(93ms,50ms)



(d).BE(period,wcet)=(935ms,100ms)

Figure 4.3: Response time of periodic BE processes running with an SRT (period,wcet)= (200ms,150ms)

respectively; In RBED scheduler with APIC, SRT1, SRT2 and SRT3 received about $30.7\%, 30.7\%, 30\%$ respectively. In both RBED schedulers, none of the SRT processes received full resource (32%) as they desired. This was caused by users' inaccurate estimation of the worst case execution time of

the SRT processes. The BE process received the leftover resource, which is much more than its share (3%). This is because it used up the slack time left by the SRT processes.

These results show the flexibility of RBED to allocate the resource to the SRT processes with a proportional share under overloaded situations. Each of the SRT processes received more resources than it did in the Linux scheduler, though none of them meets any deadline.

In order to show the performance of interactive BE processes (I/O bound processes) running with RT processes, synthetic periodic BE processes with different periods and *wcet* were used to simulate normal interactive BE processes. Figure 4.3 shows the response time (completion time) of periodic BE processes running with an SRT process (with period=200ms and *wcet*=150ms) on RBED scheduler and edf/linux scheduler respectively. The period and *wcet* of the BE process in figure 4.3a, 4.3b, 4.3c and 4.3d, is (100ms,10ms), (20ms,10ms), (100ms,50ms) and (1000ms,100ms) respectively. In all four cases, RBED scheduler ensured the SRT process meet all its deadlines and still achieved better response time for the BE process than what edf/linux scheduler did. In figure 4.3a and 4.3b, the average response time of the BE process in edf/linux scheduler is about 2.5 and 3 times as much as that in RBED scheduler respectively. In figure 4.3a, all the response times of the BE process in RBED scheduler are within 100ms, and most of them fall within 10ms. The reasons are: one, the total utilization was less than 1 (10/93 + 150/200 = 0.86), which ensured the BE process meet pseudo deadlines (100ms); two, the BE process had a relatively shorter deadline and *wcet* than those of the SRT process, which normally enforced the RBED scheduler to pick the BE process to run first. On the other hand, the BE process in edf/linux scheduler often could not start to execute until the SRT process completed its execution (150ms). It has many response times (about 50%) bigger than 103ms. In a 200ms period, one arrival of the BE process fell within the execution time (0ms-150ms) of the SRT process; the other arrival fell within the sleeping time

(150ms-200ms) of the SRT process. So on average, the response time of the BE process is 60ms. Similarly, in figure 4.3b, the BE process in RBED scheduler has most of its response times fall within 10ms. However, about 25% of the response times are close to 150ms, which is larger than the pseudo deadline (100ms). This is because the BE process couldn't meet all the actual deadlines with utilization more than 1 (10/19 + 150/200 = 1.28). In this case, the BE process was released immediately after it completed the previous execution and its current pseudo deadline(100ms) was advanced to the next pseudo deadline(200ms) at the same time. Thus, the BE process might delay its next execution until the SRT process completed its current execution (150ms). In edf/linux scheduler, the BE process in figure 4.3b has many response times bigger than 150ms (about 160ms). This is because its actual resource rate was 10ms/19ms=0.53 (> 0.25) and it could not start to execute until the SRT process completed at 150ms.

In figure 4.3c, the actual rate of the BE process in RBED scheduler was decreased from 50ms/93ms=0.54 to 0.25 (the leftover resource rate), which was equivalent to extending the deadline from 93ms to 50ms/0.25=200ms. So most of the response times of the BE process are about 200ms. In edf/linux scheduler, the BE process (figure 4.3c) could not start to execute until the SRT process completed at 150ms. This led to a similar behavior of the BE process in both schedulers (figure 4.3c). Likewise, in figure 4.3d, the BE process had similar behavior in both RBED scheduler and edf/linux scheduler. But the RBED scheduler still achieved better performance with the better and uniform EDF-based scheduler.

4.3 Dynamic SRT QOS Management in RBED

SRT processing can be incorporated with Quality of Services (QOS) management to optimize resource allocation. In the previous DQM project [3], a Dynamic QOS manager (DQM) was

developed as a middleware mechanism that operates on the collective QOS level specification. The QOS levels for each application are characterized by the application resource usage (rate), benefit provided by each level, and the period for that level. The user specifies the QOS levels, the resource usage, the benefit and the period for each QOS level of the application. Each application provides a benefit table specifying its QOS level information in the form of a set of quadruples (level, Resource usage, Benefit, Period). Level 1 represents the highest level and provides the maximum benefit using the maximum amount of resources, and lower QOS Levels are represented with larger numbers. The period of the application can remain the same, increase, or even decrease when the QOS level changes.

Resource Allocation for SRT QOS Management

A proportional algorithm [3] is used as QOS manager for SRT processes in RBED. The proportional algorithm is a centralized algorithm in which each application is allocated a percentage of the available resources proportional to the benefit/resource usage ratio. It raises or lowers the level of the application with the highest or lowest benefit/resource usage ratio.

The proportional algorithm is an adaptive algorithm, which dynamically changes the QOS levels of the SRT processes when the available resource to SRT processes changes. Two variables, cAR_{SRT} and nAR_{SRT}, are defined to represent respectively the current total actual resource rate and the new total available resource rate of all SRT processes with their current QOS levels. The variable, *raise_limit*, defines the minimum resource usage that can activate a QOS level change. The detail of the algorithm is described as follows:

1. Get the new total available resource rate (nAR_{SRT}) for the SRT processes. Compute the

Table 4.3: Benefit tables for SRT1(a), SRT2(b) and SRT3(c)

Number of QOS Levels: 4			
Level	Rate	Benefit	Period(μs)
1	0.04	0.04	100000
2	0.03	0.02	100000
3	0.02	0.01	100000
4	0.01	0.005	100000

(a)

Number of QOS Levels: 4			
Level	Rate	Benefit	Period(μs)
1	0.34	0.4	100000
2	0.23	0.35	100000
3	0.22	0.3	100000
4	0.11	0.1	100000

(b)

Number of QOS Levels: 4			
Level	Rate	Benefit	Period(μs)
1	0.33	0.3	100000
2	0.22	0.15	100000
3	0.21	0.10	100000
4	0.105	0.06	100000

(c)

difference between nAR_{SRT} and cAR_{SRT} for the SRT processes at every schedule. i.e.

$$\mathrm{d}iff = nAR_{SRT} - cAR_{SRT} \tag{0.14}$$

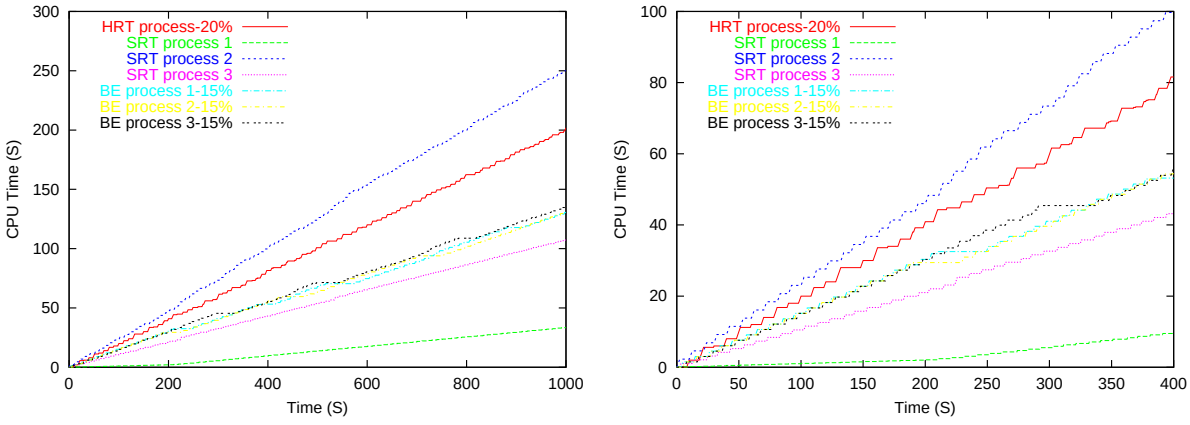
2. If $diff \geq raise\_limit$, then raise the level of the SRT process with the highest benefit/resource usage ratio. Decrease *diff* by the amount of the raised resource usage at the same time. Repeat this step until $diff < raise\_limit$.

3. If $diff < 0$, then lower the level of the SRT process with the lowest benefit/resource usage ratio. Increase *diff* by the amount of the lowered resource usage at the same time. Repeat this step until $diff \geq 0$.

4. If $0 \leq diff < raise\_limit$, then there is not enough resource to change QOS level. So keep the current value of *diff* until it changes at next schedule.

SRT QOS Level Change in the RBED Simulator

Figure 4.4 shows the example of QOS level change by using the proportional algorithm in the RBED simulator. Figure 4.4b is a close-up view of figure 4.4a. The QOS configuration for the SRT processes in figure 4.4 are described in table 4.3. The BE processes can be blocked with random delays in the RBED simulator to simulate the behavior of I/O bound processes.



(a) (b)

Figure 4.4: (a) SRT QOS management. (b) close-up view of (a)

In order to show the QOS level change of the SRT processes, the initial resource rates of the three SRT processes were set to the rates in their lowest QOS level ($1\%, 11\%, 10.5\%$ for SRT1, SRT2 and SRT3 respectively). Also, the rates of all the three BE processes were set to 15%. At the beginning of the simulation, the three SRT processes tried to adjust their QOS levels respectively according to the proportional (benefit density) QOS management mechanism. However, only SRT2 successfully changed its QOS level from 4 (11%) to 2 (23%). This is because the benefits density of SRT2 are always larger than those of SRT1 or SRT3 at their current QOS levels. After SRT2 completed its adjustment, there was no more resource left for SRT1 or SRT3 to increase their QOS levels.

SRT1 increased its level from 1 to 4 (at about 210 seconds) when the BE processes (BE5 and BE6) lowered their rates from 15% to 13% after they woke up from blocked status. Once the

available resource left by the BE processes is enough to fit a higher QOS level of any SRT process, the QOS level change occurs according to the proportional algorithm.

5 Related Work

The Proportional Share (PS) scheduler in [9] supports integrated scheduling of user applications and operating system kernel activities, such as network protocol processing. Like general proportional share algorithms, the PS scheduler associates a weight with each process. The weight determines the relative share of the resource that the process should receive. A share represents reserved resource for a process. A *lag*, which is the difference between the target service time a process should receive and the service time it has received in any interval, is used to evaluate the performance of the PS scheduling algorithm. So this PS scheduler cannot guarantee time constraints even in underloaded situations. Furthermore, the PS scheduler does not allow overloaded situations.

The elastic model introduced by Buttazzo, *et al.* [5] supports flexible workload management for SRT processes by providing varied process utilization. The model can also be extended to handle overloaded situations. An elastic parameter specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration. A real-time process, however, will be still rejected if it is not elastic enough to keep the system underloaded.

SMART [14] supports the execution of real-time (multi-media) applications in conjunction with traditional non-real-time applications. It uses priority-based and weighted fair share resource allocation mechanism. More interestingly, it also uses EDF scheduler. So it is similar to RBED. SMART uses a bias on conventional batch tasks to account for their ability to tolerate more varied service latencies. This ensures resource sharing across both real-time and conventional appli-

cations, but also implies preferentially dispatching the real-time processes. RBED assigns pseudo deadlines to conventional processes and schedules them as real-time processes seamlessly. Furthermore, SMART's time quantum-based scheduling uses 10ms clock interrupts. So it produces a large time granularity for scheduling real-time processes. Unlike SMART, RBED scheduler uses one-shot timer to achieve a better time resolution and a faster response time for RT processes since the periods and deadlines can be specified with microseconds granularity.

Rialto [11] provides a low-latency real-time scheduling with dynamic time constraints, activity-based reservation and extensible resource management support. It tries to balance meeting more time constraints of real-time processes and preventing starvation of non-real-time processes. The basic idea is to allow a thread to specify its resource reservation. A thread is accepted into the system and given the reservation only if the new set of the processes is still schedulable. Proportional and fair-share scheduling is also used for conventional processes without resource reservation. The reservation mechanism is similar to RBED. Rialto, however, does not allow for long-run overloaded situation because it does not do dynamic load shedding or adaptive scheduling.

In the RBE task model [8], a task with rate specification (x,y) expects to receive and process, on average, x events in every interval of length y. They also use EDF scheduling for the RBE task model. For the resource allocation and scheduling algorithm, this task model is similar to our RBED scheduler. However, the RBE task model only allows underloaded situations so that some SRT processes may be rejected to enter the system.

R-EDF [17] is another close example to RBED. It is a resource utilization (rate) based task model for scheduling SRT processes and BE processes. Particularly, it provides starvation protection for BE tasks and overrun protection for RT processes. But it does not adjust the deadline adaptively when a deadline miss occurs. Some other existing schedulers do not let the SRT processes run until

an underloaded system status is detected [4, 15, 6]. The scheduler in [12] even may abort a process at any time before or at the deadline miss.

RBED provides flexible scheduling for SRT processes by weighted sharing the resource in overloaded conditions. It never rejects a new SRT or BE process, while it still ensures no deadline misses occur by extending the actual deadline of a process to a new pseudo deadline. Though not all the time constraints can be met, the SRT processes always achieve graceful degraded performance in RBED. In addition to this, BE processes always receive resources when contending resources with other real-time processes.

6 Conclusions and Future Work

RBED is a prototype of a complete RA/S [2] scheduler that provides fully integrated scheduling of HRT, SRT and non real-time processes. Rate-based Weighted-proportional Resource Allocation Policy and EDF scheduling were employed in RBED to manage and schedule SRT processes with other classes of processes. Furthermore, SRT processes management were integrated in both the RBED simulator and RBED scheduler on Linux. QOS management for SRT processes was also introduced in the RBED simulator. The simulation results and results from RBED scheduler on Linux demonstrate the flexibility and feasibility of RBED to dynamically adjust the actual resource rate of SRT processes and schedule them with a graceful degraded performance under overloaded situations. RBED never starves BE processes, which are scheduled as SRT processes with pseudo deadlines assigned by the scheduler. Furthermore, in RBED, BE processes use up the slack time left by real-time processes.

In future work, using the slack time by SRT or BE processes is an important issue. Ex-

ploiting good metrics for measuring the performance of SRT processes is also important. Furthermore, the implementation and improvement of QOS management for SRT processes in RBED scheduler is an interesting topic. In addition, a QOS management user interface should be provided so that the users can specify QOS properties for the applications and input feedback information based on the scheduler notification.

References

[1] Scott Banachowski and Scott Brandt. The BEST scheduler for integrated processing of best-effort and soft real-time processes. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, January 2002.

[2] Scott Brandt. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. Technical report, Computer Science Department, UCSC, 2002.

[3] Scott Brandt and Gary Nutt. Flexible soft real-time processing in middleware. *Real-Time Systems*, pages 77–118, 2002.

[4] Giorgio Buttazzo, Marco Spuri, and Fabrizio Sensini. Value vs. deadline scheduling in overload conditions. In *Proceedings of the 16th IEEE Real-Time Systems Symposium(RTSS95)*, Pisa, Italy, 1995. IEEE.

[5] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, March 2002.

[6] Jan Carlson, Tomas Lennvall, and Gerhard Fohler. Value based overload handling of aperiodic tasks in offline scheduled real-time systems. In *13th Euromicro Conference on Real-Time Systems*, Delft, Netherlands, 2001.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review*, 19(4):1–12, September 1989.

[8] Kevin Jeffay and Steve Goddard. A theory of rate-based execution. In *Real-Time Systems Symposium(RTSS99)*, pages 304–314. IEEE, 1999.

[9] Kevin Jeffay, F. Donelson Smith, Arun Moorthy, and James Anderson. Proportional share scheduling of operating system services for real-time applications. In *Proceedings of the 19th IEEE Real-Time Systems Symposium(RTSS98)*, pages 480–491. IEEE, December 1998.

[10] E. Douglas Jensen, C. Douglass Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the 6th IEEE Real-Time Systems Symposium (RTSS 1985)*, December 1985.

[11] Michael B. Jones, Joseph S. Barbera III, Allesandro Forin, Paul J. Leach, Daniela Roşu, and Marcel-Cătălin Roşu. An overview of the Rialto real-time architecture. In *Proceedings of the 7th ACM SIGOPS European Workshop*, pages 249–256, September 1996.

[12] Gilad Koren and Dennis Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the 16th IEEE Real-Time Systems Symposium(RTSS95)*, Pisa, Italy, 1995. IEEE.

[13] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery 20*, pages 46–61, January 1973.

[14] Jason Nieh and Monica S. Lam. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 184–197. ACM, 1997.

[15] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*, chapter 5, Planning-based scheduling. Kluwer Academic Publishers, October 1998.

[16] Uwe Walter and Vincent Oberle. μ-second precision timer support for the linux kernel. http://www.tm.uka.de/~ walter/papers/oberle_walter_ibm_linux_challenge2001.pdf, 2001.

[17] Wanghong Yuan, Klara Nahrstedt, and Kihun Kim. R-EDF: A reservation-based EDF scheduling algorithm for multiple multimedia task classes. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, May 2001. IEEE.