In: *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003, pp. 131-140.

Resource Sharing in an Enhanced Rate-Based Execution Model *

Xin Liu Steve Goddard
Department of Computer Science and Engineering
University of Nebraska — Lincoln
Lincoln, NE 68588-0115
{lxin, goddard}@cse.unl.edu

Abstract

A theory of resource sharing in a mixed system with hard real-time and non-real-time processing requirements is presented. The real-time processing is modeled as rate-based execution (RBE) tasks whose resource needs are known in advance. The non-real-time processing is modeled as aperiodic requests and dynamically mapped to weight-based variable rate execution tasks. The resource sharing requirements of the aperiodic requests are unknown a priori and only become known when a critical section is reached. A scheduling algorithm that supports resource sharing without deadlock in such a mixed system is presented with a sufficient off-line schedulability condition.

1 Introduction

Many systems have both real-time and non-real-time processing requirements. When the real-time processing has hard deadlines (i.e., deadlines that cannot be missed), non-real-time processing is often modelled as aperiodic request processing within a real-time system. The canonical approach to supporting aperiodic requests in a uniprocessor real-time system has been to add a server that processes the aperiodic requests [14, 19, 20, 9, 21, 7, 8, 1, 4, 5, 6, 13]. A common feature of these algorithms is that they support a fixed number of aperiodic servers and a constant bandwidth for each server. Although some algorithms, such as GRUB (Greedy Reclamation of Unused Bandwidth) [16], allow bandwidth reclamation, the bandwidth initially allocated to each server is fixed.

Increasingly, mixed computing environments host a dynamic number of adaptive execution applications, typically multimedia applications. These applications negotiate with the system to decide their execution rates and service qualities. If the system load changes, they adjust their execution rates and service qualities accordingly. The traditional aperiodic server algorithms are unable to handle adaptive executions.

Instead of adding traditional aperiodic servers, an enhanced Rate-Based Execution (RBE) model was created in [10]. The RBE model schedules tasks at the average rate they are expected to execute. In the enhanced RBE model, aperiodic tasks (or request servers) are modeled as weight-based variable-rate execution tasks, which execute with a dynamic rate depending on the system workload. The rate changes result in adjusting the deadline assignment and the pending deadlines of aperiodic jobs. As in a multiprogramming system, tasks may require mutually exclusive access to a shared resource. In this work, we extend the enhanced RBE model of [10] to support resource sharing between real-time (RBE tasks) and non-real-time tasks (aperiodic requests).

Most systems use a lock-based method for accessing shared resources. That is, a mutex is usually used to ensure mutually exclusive access to a shared resource. If a higher priority task is blocked by a lower priority task, then a task, with priority lower than the blocked task but higher than the resource owner can be executed before the blocked task with higher priority. This is called priority inversion. The basic idea in preventing priority inversion is to increase the priority of the job that locked resources. Most resource sharing algorithms are designed for static real-time task sets and follow the priority inheritance principle [18, 11, 2]. Recent work also supports resource sharing among real-time tasks and non-real-time tasks [13, 6, 4]. The conventional algorithms, however, require fixed relative deadlines or priorities set before runtime. Thus, the dynamic rate and deadline of aperiodic tasks in the enhanced RBE model lead to the inappropriateness of conventional resource sharing algorithms.

Our research differs from previous work by relaxing the constraint of fixed deadlines. Instead of setting a static preemptive level for each task, each resource is associated with a (dynamic) resource-sharing task set and a dynamic relative deadline ceiling is defined for that task set. When a job

*Supported, in part, by a grant from the National Science Foundation (CCR-0208619).

In: *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003, pp. 131-140.

accesses the resource, it gets its deadline changed based on the resource deadline ceiling, which is equivalent to inheriting the highest priority in the resource-sharing task set; its deadline is changed back to its original value when it leaves the critical section of the resource. If the workload changes, the resource deadline ceiling may also change. A sufficient off-line schedulability condition is given in this work to simplify schedulability checking. A necessary condition cannot be derived with the limited knowledge assumed about aperiodic requests.

The rest of this paper is organized as follows. Section 2 discusses canonical resource sharing algorithms and related work in resource sharing among real-time and non-real-time tasks. Section 3 introduces the processing model assumed in this work. Section 4 presents the theoretical correctness and a schedulability condition. Section 5 discusses the overhead. We conclude with a summary in Section 6.

2 Related Work

The basic strategy dealing with priority inversion is to raise the priority of the resource owner. The Priority Inheritance Protocol (PIP) [18] lets the resource owner inherit the highest priority of blocked tasks. Thus, tasks with lower priorities are unable to preempt the resource owner.

Although PIP accelerates the release of shared resources, it does not avoid deadlocks. Also proposed in [18], the *Priority Ceiling protocol* (PCP) avoids deadlock by allocating resources in a specified order, which is the resource ceiling in practice. In PCP, each resource is assigned a priority ceiling equal to the highest priority of tasks sharing that resource. A job J is allowed to lock a resource S only if $p(J) > c(S')$, where $p(J)$ is the priority of J, and $c(S')$ is the highest priority ceiling among all resources currently locked by jobs other than J. If J cannot lock S, J is blocked and the job J', which holds the lock on S', temporarily inherits the priority $p(J)$ until S' is unlocked. Each task is assigned a fixed priority based on a static priority scheduling algorithm in PCP.

The Dynamic Priority Ceiling Protocol (DPCP) [3] extends the PCP protocol to dynamic priority scheduling algorithms, typically the EDF algorithm. The DPCP assigns a variable priority to each task; the priority is dynamically adjusted based on the emergence of the task's earliest deadline. The DPCP protocol updates the priority ceilings of all the resources, as well as the system priority ceiling. This creates more overhead than the method presented here though it could be extended to the task model assumed in this work. The Stack Resource Protocol (SRP) algorithm [2] aims at stack resource sharing, where tasks share a single stack. Each resource also maintains a ceiling in terms of the preemption level. A task is scheduled only if its preemption level is greater than the ceiling.

The *Dynamic Deadline Modification (DDM)* algorithm [11] considers a task as a sequence of resource sharing phases. Each resource is marked with a "shortest period," which is equivalent to a relative deadline. When a task is inside a critical section, the task's deadline is dynamically modified to prevent preemption by other tasks that share the resource.

Resource sharing among aperiodic tasks is a little different. Each server is associated with a bandwidth which might be mapped into a priority in real-time systems. So there is essentially no priority inversion among aperiodic tasks. In single server models such as TBS [20], there is at most one aperiodic request executing at any time; Thus, no resource is shared among aperiodic tasks in TBS. In multiple server methods like CBS [5], resources can be shared among different servers. In this case, bandwidth is considered as the replacement of priority. The *Bandwidth Inheritance Protocol* [13] was aimed at extending PIP to CBS. Instead of inheriting priority, BIP lets the resource owner inherit the higher bandwidth of blocked jobs. By inheriting bandwidth, the release of a resource is accelerated.

Coccamo and Sha studied resource sharing between real-time tasks and aperiodic tasks based on the CBS+SRP model [6]. The authors developed a theory of dynamic preemption levels. Rather than static preemption level for real-time tasks, each aperiodic task (more precisely, aperiodic server) is assigned a dynamic preemption level inversely proportional to its dynamic relative deadline. Then resource sharing in the mixed task set is handled by SRP. To reach an off-line schedulability condition, the CBS rules are modified to give a bound on the maximum preemption level.

The work presented here extends the DDM/EDF [11] algorithm to the enhanced RBE model [10] by relaxing the constraint of fixed deadlines. In the enhanced RBE model, each aperiodic request is modeled as a weight-based variable rate task, which has its rate and deadline adjusted dynamically based on system workload. In this work, each resource is associated with a dynamic deadline ceiling. When a job enters its critical section, its deadline is modified based on the deadline ceiling. The modified deadline actually provides a ceiling such that the critical section will not be preempted by other tasks sharing the resource or other tasks with relative deadlines longer than the ceiling.

3 The Model

The previous work on resource sharing requires fixed priorities, deadlines or bandwidth. In the enhanced RBE model [10], neither the execution rate nor the deadline of an aperiodic task is fixed. Thus, the previous resource sharing algorithms will not work in the enhanced RBE model.

Our approach is an extension to the DDM/EDF algorithm [11] called *Earliest-Deadline-First with Deadline-Ceiling-*

Inheritance (*EDF-DCI*). Instead of maintaining a ceiling for each resource as in PCP [18], each resource is associated with a task set whose members share that resource. Each resource-sharing task set has a deadline ceiling which is defined as the minimum relative deadline of all its member tasks. If a job gains access to a resource, it can take the current time plus the deadline ceiling as its new deadline during the critical section, which is equivalent to inheriting the highest priority in the resource-sharing task set; And the original deadline is restored when the job leaves the critical section. A final property of the EDF-DCI algorithm is that if two jobs have identical deadlines, deadline ties are broken in favor of the job that is currently executing or was previously executing and was preempted.

We assume *RBE tasks have their resource requests known in advance; The resource requests of aperiodic tasks are unknown a priori, but known when a critical section is reached*. Thus aperiodic tasks are considered to join a resource-sharing task set only when they reach their critical sections. When an aperiodic task $\hat{T}$ reaches its critical section, its quantum size is adjusted to tightly cover the critical section and registered with the corresponding resource-sharing task set. When $\hat{T}$ leaves the critical section, it is unregistered from the resource-sharing task set.

The remainder of this section first introduces the enhanced RBE model. Then we show how the quantum size is adjusted and how the deadline ceiling is constructed.

3.1 Introduction to the Enhanced RBE Model

The RBE task model was developed to support the real-time execution of event-driven tasks in which no *a priori* characterization of the *actual* arrival rates of events is known; only the *expected* arrival rates of events is known [12]. A RBE task is parameterized by a 4-tuple (x_i, y_i, c_i, d_i), making progress at the rate of processing x_i events every y_i time units, each event takes no longer than c_i time units and should be processed within d_i time units. Rate is achieved by deadline assignment. The jth job of a RBE task T_i, J_{ij}, is assigned a deadline as follows:

$$D_i(j) = \begin{cases} t_{ij} + d_i & \text{if } 1 \le j \le x_i \\ \max(t_{ij} + d_i, D_i(j - x_i) + y_i) & \text{if } j > x_i \end{cases} \quad (1)$$

where t_{ij} is the release time of J_{ij}.

In the enhanced RBE model presented in [10], the execution times of aperiodic tasks are unknown. Thus the execution has to be decomposed into a sequence of time slices. Each time slice is modeled as a job; The sequence of time slices can be viewed as a task, which releases its next job right after the previous job finishes. We called it a weighted-based variable rate execution task because its execution rate changes when the system workload changes.

Usually, a weight $w_i > 0$ is associated with each aperiodic request A_i. Let $\hat{F}$ denote the fraction of the CPU capacity allocated to processing aperiodic requests. This fraction will be shared by the aperiodic tasks in proportion to their respective weights. Thus, if $\mathcal{A}(t)$ denotes the set of aperiodic requests at time t, the fraction $f_i(t)$ of the CPU each aperiodic request $A_i \in \mathcal{A}(t)$ should receive can be computed as

$$f_i(t) = \begin{cases} 0 & \text{if } A_i \notin \mathcal{A}(t) \\ \frac{w_i}{\sum_{j \in \mathcal{A}(t)} w_j} \hat{F} & \text{otherwise.} \end{cases} \quad (2)$$

The execution of an aperiodic task is decomposed into a sequence of time slices, each of size q_i. Thus, an aperiodic task A_i can be mapped to a RBE task by a function $\psi(A_i)$ as follows:

$$\begin{aligned} \psi(A_i) : A_i \rightarrow \hat{T}_i &= (x_i, y_i(t), d_i(t), c_i) \\ &= (1, \frac{q_i}{f_i(t)}, \frac{q_i}{f_i(t)}, q_i) \end{aligned} \quad (3)$$

Note $y_i(t)$, which determines the rate of aperiodic tasks together with q_i, is a function of time t that is affected by the system workload ($\sum_{j \in A(t)} w_j$). Thus, parameter y_i in Equation (1) is replaced by a variable $y_i(t)$, which achieves the variable rate.

Workload change is caused by the arrival or termination of aperiodic tasks. The arrival or termination of aperiodic tasks will cause the update of $y_i(t)$. In an implementation, the current deadlines of aperiodic tasks can be maintained in virtual time; Thus the rate change can be done by simply adjusting the speed of virtual time. For theoretical correctness, the virtual deadlines have to be mapped into real time, which change with the system workload. Moreover, once dispatched, the deadline must be in real-time units to efficiently implement the protocol. The choice of maintaining the deadlines of pending aperiodic requests in real time or virtual time is dependent on system work load. In this work we assume the deadlines of pending aperiodic requests are maintained in real time. Let f_i denote the fraction of an aperiodic task $\hat{T}_i$ before the workload changes and f_i' the fraction after the workload changes, the current deadline $D_i(j)$ is adjusted using Equation (4).

$$D_i(j) = \begin{cases} t_x + (D_i(j) - t_x)\frac{f_i}{f_i'} & \text{if } A_x \text{ arrives at } t_x \\ D_x(l) + (D_i(j) - D_x(l))\frac{f_i}{f_i'} & \text{if } A_x \text{ terminates at } t_x^f \end{cases} \quad (4)$$

See [10] for more details on the enhanced RBE model.

3.2 Quantum Expansion at Critical Section

As described in Section 3.1, an aperiodic request is modeled as a weight-based variable rate task, which has its time

slices modeled as separate jobs. At the release of each aperiodic job, a count-down counter of size q_i is set. When the job is executed, its counter decreases. If the counter reaches 0, then the job terminates, a new job is immediately released and the scheduler is invoked. This is similar to the budget mechanism in server methods like TBS and CBS.

Since aperiodic tasks are unknown a priori, they may reach their critical section at any time. The counter may expire within a critical section; Thus the critical section may require more than one time slice to finish, which will delay the release of the critical section, as shown in Figure 1.

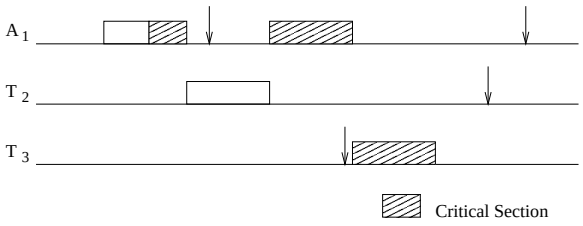

Figure 1. Counter-down timer expires in a critical section. The count-down counter expires for A_1 within a critical section; The remaining execution of the critical section is delayed to the next time slice. Task T_2 executes earlier than T_3 since the resource release is delayed. The result is a missed deadline by T_3.

This problem was addressed in [6] by recharging the budget with multiple quanta until it is large enough to cover the whole critical section. Rather than recharging the budget by quanta, we terminate the job at its current progress and readjust the quantum size to exactly cover the critical section as shown in Figure 2.

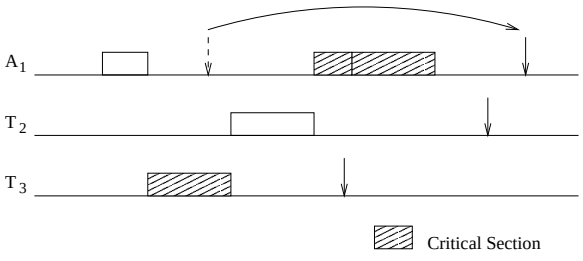

Figure 2. Quantum size recharged before a critical section. The quantum size of A_1 was expanded to exactly cover the critical section; The deadline of A_1 is moved to a later time. Thus, T_3 executes its critical section before A_1 and all tasks make their deadlines (as opposed to the example shown in Figure 1).

For simplicity, we set a bound for the minimum value of relative deadlines. Let Y_j denote the minimum relative deadline (which is assumed to be equal to the respective y parameter in this work) for resource r_j and c_{ij} be the size of the critical section, then the quantum size is set using Equation (5) when an aperiodic task $\hat{T}_i$ reaches its critical section for resource r_j, where f_i is the fraction of $\hat{T}_i$ computed by Equation (2).

$$q_i^{rj} = max(c_{ij}, \lceil Y_j \cdot f_i \rceil) \tag{5}$$

Let the fraction of the CPU allocated to aperiodic request processing be $\hat{F}$ and the weight summation of aperiodic tasks be W. Assume an aperiodic task $\hat{T}_i$ has an original quantum size of q_i time units, current deadline $D_i(x)$, and R_i time units remaining in its counter. Then it is registered with the resource-sharing task set using the following RBE description when it reaches the critical section (i.e., when it attempts to lock the resource):

$$(1, y_i'(t) = \frac{q_i^{rj}}{f_i}, q_i^{rj}, d_i'(t) = y_i'(t)) \tag{6}$$

At the same time, its counter is reset to q_i^{rj} and its current deadline $D_i(x)$ is updated using Equation (7).

$$D_i'(x) = D_i(x) + \frac{q_i^{rj} - R_i}{w_i \hat{F}} \tag{7}$$

In Equations (6) and (7), if $q_i^{rj} = R_i$, then nothing needs to be done; If $q_i^{rj} > R_i$ then the quantum size is expanded to q_i^{rj} and the deadline is moved to a later time; If $q_i^{rj} < R_i$ then the quantum size is shortened to q_i^{rj} and the deadline is moved to an earlier time.

When the aperiodic task leaves its critical section, it is unregistered from the task set and its quantum size is restored to q_i. The new job is then assigned a deadline using Equation (1).

3.3 Dynamic Deadline Ceiling

In deadlock free algorithms such as PCP [18] and SRP [2], a priority (preemption level) ceiling is maintained for each resource. When a job is blocked or gains access to a resource, the ceiling is increased. A job is scheduled only if its priority (or preemption level) is greater than the ceiling. Thus, resources are allocated in a specific order, which prevents deadlock.

In this work, we prevent deadlock in a different way. We assume

> *The resource requests of RBE tasks are known in advance; The resource requests of aperiodic tasks are known only when they reach a critical section.*

Thus, each resource is associated with a resource-sharing task set whose elements may request the resource. When

a job gains access to a resource, it is assigned the highest priority (earliest deadline) within the resource-sharing task set. Thus, deadlock is prevented.

Since the resource requests of aperiodic tasks are unknown in advance, the resource-sharing task set is not static; It contains both static members (RBE tasks) and dynamic members (aperiodic tasks). Let $T_{ri}(t)$ denote the resource-sharing task set of resource r_i at time t. Then $T_{ri}(t) = R_{ri} \cup A_{ri}(t)$, where R_{ri} is the static set of real-time tasks that share resource r_i and $A_{ri}(t)$ is the dynamic set of aperiodic tasks that share resource r_i at time t. Aperiodic tasks are dynamically registered with the resource-sharing task set when they reach their critical sections. The construction of $T_{ri}(t)$ is as follows:

- Initially, $T_{ri}(t) = R_{ri}$
- When an aperiodic task A_x reaches its critical section for resource r_i,
  - change A_x's counter to q_x^{ri} and adjust the quantum size and deadline of A_x using Equations (6) and (7), as discussed in Section 3.2
  - register A_x to the resource-sharing task set: $T_{ri}(t) = T_{ri}(t) \cup \{A_x\}$
- When an aperiodic task A_x leaves its critical section of resource r_i,
  - unregister A_x from the resource-sharing task set $T_{ri}(t) = T_{ri}(t) - \{A_x\}$
  - change A_x back to its original quantum size q_x

The deadline ceiling for r_i is defined as the minimum relative deadline of tasks in $T_{ri}(t)$. That is,

$$d_{ri}(t) = min(d_i | T_i \in T_{ri}(t)) \quad (8)$$

where d_i is a relative deadline of task T_i, which may either a RBE task or an aperiodic task.

When a job J_x enters a critical section of resource r_i at time t, it has its deadline changed to

$$D_m = min(D_x, t + d_{ri}(t)) \quad (9)$$

where D_x is the original deadline.

When J_x leaves the critical section, it has its deadline changed back to the original deadline value D_x. If J_x is an aperiodic job, then J_x terminates; J_x's task has its quantum size changed back q_x and releases a new job immediately.

Since $T_{ri}(t)$ is dynamic, the deadline ceiling $d_{ri}(t)$ has to be recomputed each time an aperiodic task registers or unregisters. Even when the membership of $T_{ri}(t)$ does not change, members of $A_{ri}(t)$ will have their relative deadlines changed if the workload changes. In this work, we assume critical sections are short and new aperiodic requests are not accepted when a task is in a critical section. Thus, $T_{ri}(t)$ and $d_{ri}(t)$ will not change within any critical section, and $d_{ri}(t)$ is only computed when needed. That is, $d_{ri}(t)$ is computed only when a job gains access to its critical section for r_i. This means that D_m will not change within any critical section since system workload remains constant in that interval.

To summarize, the dynamic deadline ceiling works as follows:

- Each resource r_i is associated with a resource-sharing task set T_{ri}, which initially only contains RBE tasks that will use resource r_i
- When an aperiodic task A_x reaches its critical section for resource r_i, its quantum size and deadline are adjusted using Equations (6) and (7). Then A_x is registered with T_{ri}
- When a task (either a RBE task or an aperiodic task) begins to execute its critical section for resource r_i at time t, it has its deadline, D_x, changed to $D_m = min(D_x, t + d_{ri})$
- When a task leaves the critical section for r_i, it has its deadline changed back to D_x
  - If the task is an aperiodic task A_x, then it is unregistered from T_{ri}; A_x has its quantum size changed back to q_x and releases a new job.

The EDF-DCI algorithm is an EDF scheduling algorithm that supports the dynamic deadline ceiling and breaks deadline ties in favor of the job that is currently executing or was previously executing and was preempted.

3.4 An Example

The following example illustrates the usage of the dynamic deadline ceiling.

Initially, $T_{ri}(0) = \{T_1, T_2, T_3\}$ has three RBE tasks, as shown in Table 1. Assume job J_{11} is released by RBE task T_1 at time 4 with its original deadline $D_1(1) = 4+15 = 19$. As shown in Figure 3, it immediately gains access to resource r_i and its deadline is changed to the deadline ceiling, $min(19, 4 + 10) = 14$, using Equation (9).

Table 1. $T_{ri}(0)$.

	x	y	c	d
T_1	1	15	3	15
T_2	1	10	1	10
T_3	1	12	2	12
d_{ri}		10		

Table 2. $T_{ri}(6)$.

	x	y	c	d
T_1	1	15	3	15
T_2	1	10	1	10
T_3	1	12	2	12
$\hat{T}_4$	1	6	1	6
d_{ri}		6		

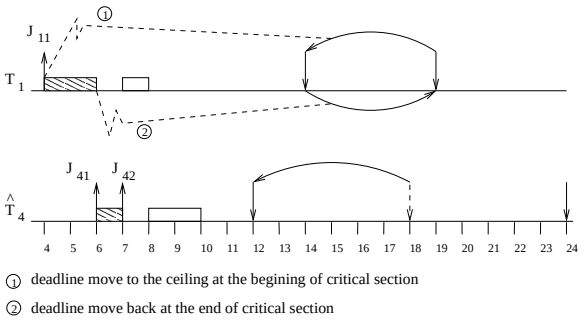

Figure 3. Deadline adjustment at a critical section. Job J_{11} has its deadline changed to 14 at the beginning of critical section and changed back to 19 at the end of critical section; Task $\hat{T}_4$ arrives at time 5 but not accepted until time 6. $\hat{T}_4$ has its quantum size changed to 1 at time 6 and changed back to 2 at time 7.

At time 5, an aperiodic task $\hat{T}_4 = (1, 12, 2, 12)$, with fraction $\frac{1}{6}$, arrives. Since job J_{11} is in its critical section, $\hat{T}_4$ is not accepted immediately.

At time 6, job J_{11}leaves the critical section and its deadline is changed back to the original deadline value of 19. Aperiodic task $\hat{T}_4$ is accepted at time 6 and releases job J_{41}, which is assigned the deadline $6 + 12 = 18$. Since job J_{41} has the earliest deadline (18), it is the next job to execute and immediately reaches its critical section for resource r_i, which has a size of 1 time unit. Thus, aperiodic task $\hat{T}_4$ is registered with task set T_{ri}, as shown in Table 2. Let 6 be the minimum relative deadline in the task set T_{ri} (i.e., $d_{ri}(6) = 6$). Then job J_{41}'s quantum q_4^i is set to $min(1, \lceil \frac{6}{6} \rceil) = 1$. Thus, its rate specification is set to $(1, 6, 1, 6)$; its deadline is changed to 12 using Equations (6) and (7); and its execution counter (budget) is reduced from 2 time units to 1 time unit.

When job J_{41} finishes at time 7, $\hat{T}_4$ is unregistered from task set T_{ri}, $T_{ri}(7) = \{T_1, T_2, T_3\}$; the rate specification for $\hat{T}_4$ is changed back to $(1, 12, 2, 12)$; and its second job is released (at time 7), with the execution counter reset to 2 time units and a deadline of $12 + 12 = 24$. Finally, job J_{11} is the next job to begin execution at time 7.

The execution schedule for this example is shown in Table 3.

4 Theoretical Validation

This section discusses the theoretical correctness of the EDF-DCI algorithm. We first show that the algorithm satisfies the mutual exclusion constraint on access to resources and that it prevents deadlock. Then, an off-line sufficient schedulability condition is given.

Table 3. Schedule.

$time$	Job	t_{arrive}	t_{accept}	d_{ri}	D	D_m
4	J_{11}	4	4	10	19	14
5	J_{11}					
6	J_{41}	5	6	6	12	12
7	J_{11}	4	4	10	19	–
8	J_{42}	7	7	10	24	–
9	J_{42}					

Property 1 states that the EDF-DCI algorithm provides mutually exclusive access to resources. Property 2 states that a job requesting a resource is never blocked from accessing the resource. Property 3 combines the first two properties to show that the EDF-DCI algorithm provides mutually exclusive access to resources and prevents deadlocks.

Property 1 The EDF-DCI scheduling algorithm satisfies the mutual exclusion constraint on access to resources.

Proof: by contradiction.
Assume job J_i begins to execute its critical section for resource r_x at time t'_i with deadline D'_i, but is preempted by some job before it completes its critical section. At the time it entered the critical section, its deadline was the earliest eligible deadline and this deadline was retained or moved earlier using Equation (9) at time t'_i. Thus $D'_i = min(D_i, t'_i + d_{rx}(t'_i))$.

Assume job J_j shares resource r_x, begins to execute at time t_j and enters its critical section for resource r_x at time t'_j before job J_i leaves its critical section. Let $D'_j = min(D_j, t'_j + d_{rx}(t'_j))$. Then it must be the case that that $D'_j \leq D_j < D'_i$ since earliest deadlines are given priority and ties are broken in favor of job J_i under EDF-DCI scheduling. Moreover, since the workload cannot change once a job enters its critical section, it must also be the case that $d_j \geq d_{rx}(t'_j) = d_{rx}(t_j) = d_{rx}(t'_i)$ by the definition of d_{rx}. Thus, $t_j \leq t'_j < t'_i$ since $D_j = t_j + d_j \leq t_j + d_{rx}(t'_j) < D'_i \leq t'_i + d_{rx}(t'_i) \leq t'_i + d_j$. This, however, contradicts the assumption that job J_i had the earliest deadline when it entered its critical section. Thus, the EDF-DCI scheduling algorithm satisfies the mutual exclusion constraint on access to resources. □

Property 2 If a job J_i requests resource r_x, it will not be blocked by another task that shares resource r_x.

Proof: by contradiction.
Assume job J_i requests resource r_x, but is blocked by job J_j of another task that shares resource r_x. That is, assume job J_i cannot access the resource, even though it has the earliest deadline, because job J_j is in its critical section.

Let time t_i be the release time of job J_i and D_i be its deadline. Let t'_j be the time job J_j enters its critical section, D_j be the original deadline of job J_j, and D'_j be the modified deadline of job J_j set using Equation (9).

Then $D_j \geq D'_j > D_i$ based on Equation (9) and the assumption that J_i is blocked by J_j under the EDF-DCI scheduling algorithm. Thus, $t'_j < t_i$; otherwise, J_j will not be able to enter its critical section at time t'_j since $D_i < D_j$. Moreover, since the workload did not change within the critical section, $d_{rx}(t'_j) = d_{rx}(t_i) \leq d_i$. Therefore,

$$D'_j \leq t'_j + d_{rx}(t'_j) < t_i + d_{rx}(t'_j) \leq t_i + d_i = D_i$$

because $D'_j = min(D_j, t'_j + d_{rx}(t'_j)) \leq t'_j + d_{rx}(t'_j)$ and $D_i = t_i + d_i$. This, however, contradicts the fact that D_i is the earliest deadline.

Thus, if a job J_i requests resource r_x, it will not be blocked by another task that shares resource r_x. □

Property 3 The EDF-DCI algorithm provides mutually exclusive access to resources and prevents deadlocks.

Proof: This is straightforward from Properties 1 and 2. □

Jeffay presented an off-line feasible condition in [11], which is extended to this work. Lemma 4.1 bounds the demand of a RBE task, which was already presented in [12]. We present it here since it is used in the sufficient schedulability condition for the enhanced RBE model considered in this work.

Lemma 4.1. *For a RBE task $T_i = (x_i, y_i, d_i, c_i)$,*

$$\forall t > 0,\ \ dbf_i(t) = \begin{cases} 0 & \text{if } t \in [0, d_i) \\ \lfloor \frac{t - d_i + y_i}{y_i} \rfloor x_i c_i & \text{if } t \in [d_i, \infty] \end{cases} \quad (10)$$

is a least upper bound on the number of units of processor time required to be available in the interval $[0, L]$ to ensure that no job of T_i misses a deadline in $[0, L]$.

Proof: See [12]. □

Theorem 4.2 gives a sufficient schedulability condition for the RBE task model. The RBE model is a special case of the enhanced RBE model in which there exists no aperiodic requests. For space considerations, only a short proof sketch is presented. The full proof follows the proof of Theorem 4.4.

Theorem 4.2. *Let τ be a task system with n RBE tasks, sorted in non-decreasing order by y parameter, that share a set of serially reusable resources $r_1, r_2, ..., r_m$. Assume $d_i = y_i$ for all tasks, τ will be schedulable on a uniprocessor under EDF-DCI if:*

$$\forall L, L > 0, L \geq \sum_{i=1}^{n} \lfloor \frac{L}{y_i} \rfloor x_i c_i \quad (11)$$

and
$\forall i, 1 \leq i \leq n, \forall k, 1 \leq k \leq m \wedge r_{ik} \neq 0, \forall L, d_{rk}(0) \leq L \leq y_i$

$$L \geq c_{ik} + \sum_{j=1}^{i-1} \lfloor \frac{L-1}{y_i} \rfloor x_i c_i \quad (12)$$

where:

- $r_{ik} \neq 0$ *if task T_i uses resource r_k,*
- c_{ik} *is the size of the critical section of task T_i for resource r_k,*
- $d_{rk}(0)$ *is defined by Equation* (8) *in Section 3.3.*

Proof sketch: This theorem is a special case of Theorem 4.4 in which there are no aperiodic tasks and $\hat{F} = 0$. Under these conditions, Theorem 4.4 reduces to Theorem 4.2 and the proof follows the proof of Theorem 4.4 in which the third condition holds vacuously. □

Because aperiodic tasks are modeled as weight-based variable rate tasks, an enhanced RBE system can be treated as a RBE system in any interval where the system workload does not change. Thus, Theorem 4.2 can also be used to check the schedulability in such intervals. But the conditions have to be rechecked if the system workload changes, which can cause great overhead.

Aperiodic tasks are initially considered as critical section free; They are registered with the resource-sharing task set only when they reach their critical sections. Thus, RBE tasks and aperiodic tasks can be viewed as running on two virtual processors with partial power of the real processor (fraction F for RBE tasks and fraction $\hat{F}$ for aperiodic tasks). They interfere only when aperiodic tasks reach their critical sections.

Lemma 4.3 bounds the demand of an aperiodic task, which is proved in [10]. Theorem 4.4 gives a sufficient schedulability condition for the enhanced RBE model under EDF-DCI scheduling.

Lemma 4.3. *Let $\hat{T}_i = \psi(A_i)$ represent the aperiodic request $A_i \in \mathcal{A}(t)$. If no job of $\hat{T}_i$ released before time $t_0 \geq 0$ requires processor time in the interval $[t_0, l]$ to meet a deadline in the interval $[t_0, l]$, then*

$$\forall l > t_0,\ \ \widehat{dbf}_i([t_0, l]) = \int_{t_0}^{l} f_i(t)dt \quad (13)$$

is an upper bound on the processor demand in the interval $[t_0, l]$ created by $\hat{T}_i$ where $\psi(A_i)$ is defined by Equation (3) *and $f_i(t)$ is defined by Equation* (2)*.*

Proof: See [10]. □

When $\hat{F}$ denotes the fraction allocated to aperiodic tasks, the demand of all aperiodic tasks is bounded by $L\hat{F}$ because $\sum_{i\in\hat{T}} f_i = \hat{F}$, as discussed in [10].

Although the resource requests of aperiodic tasks are unknown in advance, the minimum relative deadline is assumed to be bounded by Y_{rk} in Equation (5); the maximum relative deadline is assumed to be Y^{rk}; the critical sections for resource r_k are assumed to be less than a maximum bound C_{rk}. These are reasonable assumptions for the mixed systems we work with that are dominated by real-time tasks.

Theorem 4.4. *Let $\tau = R \cup \mathcal{A}$ that share a set of serially reusable, single unit resources $\{r_1, ..., r_m\}$, R consists of n RBE tasks sorted in non-decreasing order by y parameter and $\mathcal{A}$ is the dynamic aperiodic task set. Assume $d_i = y_i$, τ can be scheduled if the following conditions hold:*

1) $\sum_{i\in R}^{n} \frac{x_i \cdot c_i}{y_i} + \hat{F} \leq 1$

2) $\forall i, 1 < i \leq n, \forall k, 1 \leq k \leq m \wedge r_{ik} \neq 0, \forall L, d_{r_k}(0) < L < y_i$*:*

$$(1 - \hat{F})L \geq c_{ik} + \sum_{j=1}^{i-1} \lfloor \frac{L-1}{y_j} \rfloor x_j \cdot c_j$$

3) $\forall k, 1 \leq k \leq m, \forall L, Y_{rk} < L < Y^{rk}$*:*

$$L(1 - \hat{F}) \geq C_{rk} + \sum_{i\in R} \lfloor \frac{L-1}{y_i} \rfloor x_i \cdot c_i$$

where:

- $r_{ik} \neq 0$ *if task* T_i *uses resource* r_k*,*
- Y_{rk} *is a lower bound of y parameters of all tasks sharing resource* r_k*,*
- Y^{rk} *is a upper bound of y parameters of all tasks sharing resource* r_k*,*
- C_{rk} *is an upper bound of the size of critical section for resource* r_k*,*
- $\hat{F}$ *is the processor fraction allocated to aperiodic requests.*

Condition (1) is a utilization test that guarantees the system is not overloaded; Condition (2) is a generalized form of Theorem 4.2 that guarantees the processor is able to handle resource sharing among RBE tasks when no aperiodic gains access to any resource; if an aperiodic task gains access to a resource, then Condition (3) is used to check the correctness.

Proof: by contradiction.

Suppose job J_l is the first job that misses its deadline D_l. All released jobs are divided into two subsets: $A = \{Jobs\ with\ deadline\ equal\ or\ less\ than\ D_l\}$, $B = \{Jobs\ with\ deadline\ greater\ than\ D_l\}$.

Choose t_0 as the later of the last idle point and the last scheduling point of any task in B. t_0 is set to 0 if no such point exists. Then the problem consists of two cases:

Case 1: t_0 is 0 or an idle point

From Lemma 4.1 and Lemma 4.3, we know the processor demand bound $Demand_{[t_0,D_l]}$ is bounded as follows:

$$(F + \hat{F})(D_l - t_0) \geq Demand_{[t_0,D_l]} > D_l - t_0$$

which contradicts Condition (1), $\sum_{i\in R}^{n} \frac{x_i \cdot c_i}{y_i} + \hat{F} \leq 1$.

Case 2: t_0 is the last scheduling point of a job J_b in B

If t_0 is not in J_b's critical section or t_0 is in a critical section but the modified deadline of $J_b$is greater than D_l, then any job in $\mathcal{A}$ can preempt J_b. The analysis in *Case 1* applies in this case.

If t_0 is in J_b's critical section for resource r_k with the modified deadline less than D_l, then job J_l is unable to preempt J_b before J_b leaves its critical section. There are two subcases:

Case 2a: J_b is a RBE task

In this case, the demand of RBE tasks $Demand^{R}_{[t_0,D_l]}$ is bounded by $\sum_{i=1}^{i=b} \lfloor \frac{L-1}{y_i} \rfloor x_i c_i$; the demand of aperiodic tasks $Demand^{A}_{[t_0,D_l]}$ is bounded by $L\hat{F}$; and $Demand_{[t_0,D_l]} \leq Demand^{R}_{[t_0,D_l]} + Demand^{A}_{[t_0,D_l]} + c_{bk}$, where c_{bk} is the size of the critical section. Thus,

$$\sum_{i=1}^{i=b} \lfloor \frac{L-1}{y_i} \rfloor x_i c_i + L\hat{F} + c_{bk} \geq Demand_{[t_0,D_l]} > (D_l - t_0)$$

which contradicts Condition (2).

Case 2b: J_b is an aperiodic task

Similar to Case 2a, the demand of RBE tasks $Demand^{R}_{[t_0,D_l]}$ is bounded by $\sum_{i\in R} \lfloor \frac{L-1}{y_i} \rfloor x_i c_i$; the demand of aperiodic tasks $Demand^{A}_{[t_0,D_l]}$ is bounded by $L\hat{F}$; the critical section size is bounded by C_{bk}. Thus,

$$\sum_{T_i\in R} \lfloor \frac{L-1}{y_i} \rfloor x_i c_i + L\hat{F} + C_{bk} \geq Demand_{[t_0,D_l]} > (D_l - t_0)$$

which contradicts Condition (3).

□

In practice, the system utilization $F + \hat{F}$ has to be less than 1. Otherwise, Condition (3) in Theorem 4.4 is unlikely to be satisfied. More simply, a task set is schedulable if its utilization is below a threshold as shown in Theorem 4.5.

Theorem 4.5. *Let $\tau = R \cup A$, be a set of rate-based execution tasks and aperiodic tasks, that share a set of serially reusable, single unit resources $\{R_1, ..., R_m\}$. τ can be scheduled if its utilization $\hat{F} + F \leq 1 - \frac{r}{y}$.*
where:

- *$\hat{F}$ is the processor fraction allocated to aperiodic requests*
- *F is the processor fraction allocated to RBE tasks*
- *r is an upper bound of critical sections*
- *y is a lower bound of y parameters*

Proof: $\hat{F} + F \leq 1 - \frac{r}{y}$ implies the three conditions in Theorem 4.4. □

When $r << y$, Theorem 4.5 is more useful than Theorem 4.4 in that it can be easily checked and is not so loose that it becomes meaningless.

5 Performance Analysis

Before implementing the algorithm, we simulated the operations on resource-sharing sets and measured the introduced overhead. The simulation is made on a 2GHz Pentium 4-M IBM Thinkpad T30. Our measurement employs the *"rdtsc"* instruction on the i386 architecture; the *"rdtsc"* instruction reads the time stamp counter which records the processor cycles since power-on. The resource-sharing set is implemented as a sorted linked list. Totally 100 tasks are simulated. Each task is randomly assigned a relative deadline.

Since our main interest is on supporting mixed task set in general-purpose operating systems, we did our measurement on Redhat 8.0. Figure 4 shows the overhead to insert a task in a resource-sharing set with increasing size.

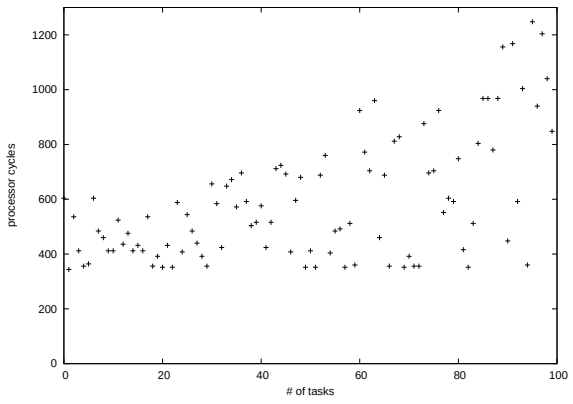

Figure 4. Overhead on Redhat-8.0.

As we can see from Figure 4, with 100 tasks, the number of processor cycles required to insert a task into the sharing task set is below 1300, and the average is 591. The overhead to remove a task from the resource-sharing set is relatively constant, about 200 cycles, because the task being removed is always the first task in the resource-sharing list.

On the same hardware, the Redhat *mutex* operations, *"pthread_mutex_lock"* and *"pthread_mutex_unlock"*, consume about 7000 and 3000 processor cycles respectively. Thus the overhead introduced by *EDF-DCI* is competitive with the Redhat *mutex* operations. Moreover, the *EDF-DCI* prevents priority inversions and deadlock.

6 Summary

The enhanced Rate-Based Execution (RBE) model presented in [10] was extended to support resource sharing between real-time and non-real-time applications. As in [10] the real-time applications were modeled as RBE tasks and the non-real-time applications were modeled as weight-based, variable-rate execution tasks.

To support resource sharing in the enhanced RBE model, a *Earliest-Deadline-First with Deadline-Ceiling-Inheritance* (*EDF-DCI*) algorithm was developed. Instead of maintaining a ceiling for each resource as in PCP [18] or DPCP [3], each resource is associated with a dynamic task set whose members share that resource. Each resource-sharing task set has a dynamic deadline ceiling which is defined as the minimum relative deadline of all its current member tasks. The EDF-DCI algorithm uses the dynamic deadline ceiling to assign deadlines such that the mutual exclusion constraint on shared resources is maintained while preventing deadlock.

To verify temporal correctness of a task set scheduled with the EDF-DCI algorithm, sufficient off-line schedulability conditions were provided.

References

[1] Abeni, L., Buttazzo, G., "Integrating Multimedia Applications in Hard Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.*, Madrid, Spain, Dec. 1998.

[2] Baker, T.P., "Stack-Based Scheduling of Real-Time Processes," *The Journal of Real-Time Systems 3(1)*, pp. 67-100, 1991. IEEE Computer Society Press.

[3] Chen, M, Lin, K, "Dynamic priority ceilings: A concurrency control protocol for real-time systems", *Journal of Real-Time Systems*, 2:325–346, 1990.

[4] Caccamo, M., Lipari, G., Buttazzo, G., "Sharing Resource among Periodic and Aperiodic Tasks with Dynamic Deadlines," *Proc. IEEE Real-Time Systems Symp.*, Phoenix, AZ, Dec. 1999.

[5] Caccamo, M., Buttazzo, G., Sha, L., "Capacity Sharing for Overrun Control," *Proc. IEEE Real-Time Systems Symp.*, Orlando, FL, Dec. 2000.

[6] Caccamo, M., Sha, L., "Aperiodic Servers with Resource Constraints," *Proc. IEEE Real-Time Systems Symp.*, London, England, Dec. 2001.

[7] Deng, Z., Liu, J.W.S., Sun, J., "A Scheme For Scheduling Hard Real-Time Applications in Open System Environment," In *Proceedings of the Ninth Euromicro Workshop on Real-Time System*, Toledo, Spain, June 1997, pp. 191-199.

[8] Deng, Z., Liu, J.W.S., "Scheduling Real-Time Applications in an Open Environment," *Real-Time Systems Journal*, vol. 16, no. 2/3, pp.155-186, May 1999.

[9] Ghazalie, T. M., Baker, T. P., Aperiodic Servers in Deadline Scheduling Environment, *Real-Time Systems Journal*, vol. 9, no. 1, pp. 31-68, 1995.

[10] Goddard, S., Liu, X., "Scheduling Aperiodic Requests under Rate-Based Execution model" *Proceesings of IEEE Real-Time System Symposium*, December 2002, pp. 15-25.

[11] Jeffay, K., "Scheduling Sporadic Tasks with Shared Resources in Hard Real-Time Systems," *Proceesings of IEEE Real-Time System Symposium*, pp. 89-99, December 1992.

[12] Jeffay, K., Goddard, S., "A Theory of Rate-Based Execution," *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999, pp. 304-314.

[13] Lamastra, G., Lipari, G., Abeni, L., "A Bandwidth Inheritance Algorithm for Real-Time Task Synchronization in Open Systems," *Proc. IEEE Real-Time Systems Symp.*, London, England, Dec. 2001.

[14] Lehoczky, J.P., Sha, L., and Strosnider, J.K., "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proceedings of IEEE Real-Time Systems Symposium*, pp. 261-270, Dec. 1987.

[15] Lipari, G., Buttazzo, G., "Schedulability Analysis of Periodic and Aperiodic Tasks with Resource Constrains", *Journal of Systems Architecture*, Vol. 46, No.4, pp. 327-338, January 2000.

[16] Lipari, G., Baruah, S., "Greedy reclamation of unused bandwidth in constant-bandwidth servers", *Proceedings of the EuroMicro Conferences on Real-Time Systems*, pp. 193-200, Stockholm, Sweden. June 2000.

[17] Liu, C., Layland, J., "Scheduling Algorithms for multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, Vol 30., Jan. 1973, pp. 46-61.

[18] Sha, L., Rajkumar, R., Lehoczky,J.P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, September 1990.

[19] Sprunt, B., Sha, L., Lehoczky, J.P., "Aperiodic Task Scheduling for Hard Real-time Systems," *Real-Time Systems Journal*, vol 1, no. 1, pp. 27-60, 1989.

[20] Spuri, M., Buttazzo, G., "Efficient Aperiodic Service Under the Earliest Deadline Scheduling," *Proc. of the IEEE Symposium on Real-Time Systems*, December 1994.

[21] Spuri, M., Buttazzo, G., Sensini, F., "Robust Aperiodic Scheduling Under Dynamic Priority Systems," *Proc. of the IEEE Symposium on Real-Time Systems*, December 1995.