An Efficient RMS Admission Control and its Application to Multiprocessor Scheduling [*]

Sylvain Lauzac, Rami Melhem, Daniel Mossé
University of Pittsburgh
Department of Computer Science
Pittsburgh, PA 15260
(lauzac, melhem, mosse)@cs.pitt.edu

Abstract

A real-time system must execute functionally correct computations in a timely manner. In order to guarantee that all tasks accepted in the system will meet their timing requirements, an admission control algorithm must be used to only accept tasks whose requirements can be satisfied. Rate-monotonic scheduling (RMS) is arguably the best known scheduling policy for periodic real-time tasks on uniprocessors. We propose a new and efficient admission control for rate-monotonic scheduling on a uniprocessor and analyze its performance. This admission control is then modified to provide an admission control mechanism for multiprocessor systems. Experimental results indicate that this new admission control for multiprocessor systems achieves a processor utilization of up to 96% for a large number of tasks and has a low computational complexity. The proposed admission control is also used to derive a new and better multiprocessor schedulability bound for RMS with provisions for periodic servers and for RMS with tolerance to transient faults.

1 Introduction

A real-time system processes tasks that must produce functionally correct results in a timely manner. This implies that the tasks submitted to the system have known timing requirements. These tasks are called *real-time tasks*. Many of these real-time tasks (such as in process control) are periodic and modeled as follows. At the beginning of each period a new instance of the task is generated and is immediately available for processing. The processing of each task instance must be completed by the end of the task's period, called the *deadline* of the instance. Traditionally, the requirements of a periodic real-time task τ_i are characterized by a period T_i and a worst-case computation time C_i.

Given this task model, a real-time system must ensure that each task instance will complete before its deadline. This is done by using an *admission control* and a *scheduling policy* for the real-time system. The admission control is an algorithm that depends on the scheduling policy and ensures that only tasks that will meet their deadlines are accepted into the system. A task set that does not miss any deadline is called *schedulable*. The scheduling policy determines which task instance is to be processed next.

One of the most widely used scheduling policies for preemptive periodic real-time tasks is called *rate-monotonic scheduling* (RMS)[13]. RMS associates each task τ_i with a fixed priority $p_i = 1/T_i$. At any time, the available task with the highest priority is being processed. It is assumed that preemption takes negligible time.

An admission control for RMS based on processor utilization is also given in [13]. The *utilization* of task τ_i is defined to be C_i/T_i and the utilization of a task set is the sum of the utilizations of all the tasks in the task set. This admission control compares the utilization of the task set to a bound that depends only on the number of tasks in the set and shows that a set of m tasks will not miss any deadline if

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq m(2^{1/m} - 1) \tag{1}$$

Such an admission control has the advantage of being computationally simple. However, Equation (1) is a sufficient but not necessary condition for schedulability. As a

[*]This work has been supported by the Defense Advanced Research Projects Agency (Contract DABT63-96-C-0044).

consequence, the processor utilization achieved by this admission control may be lower than necessary, causing under utilization of computing resources. This observation leads to the first problem addressed by this paper: **is there a computationally simple admission control for RMS that yields a processor utilization better than the classical bound given by Equation (1)?**

The RMS policy was defined only for uniprocessors. However, since multiprocessor systems are becoming more common for real-time applications [3], admission control for multiprocessor systems is also an important problem. RMS can be used on multiprocessor systems by doing *global scheduling* or *partitioning*[7]. In a global scheduling scheme, all instances are stored in a single queue and, at any given time, the processors run the instances with the highest priority. A partitioning scheme adds the constraint that all instances of a task must execute on the same processor. Partitioning has the advantage of reducing the multiprocessor scheduling problem to scheduling problems on individual processors for which many results are known. Partitioning may seem less powerful than global scheduling because it has more constraints, but it has been proven [7] that global scheduling with RMS can give an arbitrarily low processor utilization. Therefore, this paper focuses only on partitioning schemes for multiprocessor systems.

When a partitioning scheme is used, the admission control must not only decide which tasks can be accepted, but also create an assignment of tasks to processors. Finding an optimal assignment of tasks to processors is known to be NP-hard [12], therefore it is important that the complexity of the admission control remains low. Partitioning schemes are usually based on a bin-packing algorithm and a schedulability bound. The bin-packing algorithm assigns tasks to processors and uses the schedulability bound to determine if a processor can accept a task. This defines the second problem this paper addresses: **what partitioning admission control for a multiprocessor gives a near optimal processor utilization and has a low computational complexity?**

Many extensions to RMS have been proposed in the literature, including periodic servers [16], tolerance to transient faults [8] and task synchronization [15]. In order to be practical, a new admission control should easily adapt to these extensions of RMS. This defines the third problem addressed by this paper: **what admission control can be easily adapted to and improve the processor utilization of these RMS extensions?**

The rest of the paper is organized as follows. Section 2 gives an overview of some related work. Section 3 describes two new admission control algorithms for a uniprocessor, T-BOUND and R-BOUND. In Section 4, we extend the R-BOUND to multiprocessor systems and compare its performance with other admission control algorithms for multiprocessors. Sections 5 and 6 develop new schedulability bounds based on the R-BOUND for periodic servers and RMS with tolerance to transient faults. These bounds perform best when used on a multiprocessor. The conclusion is presented in Section 7.

2 Related work

2.1 Admission control for uniprocessors

As mentioned above, an admission control based on Equation (1) usually under-utilizes the processor because this schedulability condition is sufficient but not necessary. A necessary and sufficient condition for rate-monotonic schedulability is presented in [10]. The drawback of this admission control is its computational complexity. A similar admission control can be based on the conditions given in [1] and can achieve very high processor utilization.

Other work takes advantage of the knowledge of some characteristics of the task set being scheduled in order to get a more precise schedulability condition. In [2], knowledge of the tasks periods is used to get a tighter bound and increase the processor utilization.

2.2 Admission control for multiprocessors

Schedulability conditions for uniprocessors can be used to partition tasks on a multiprocessor. An early admission control for multiprocessors using partitioning was proposed in [7]. This admission control uses two bin-packing algorithms (*Next Fit* and *First Fit*) to assign tasks to processors. For each processor, Equation (1) is used for admission control.

The processor utilization obtained by a partitioning scheme can be increased in three ways. First, one can use a *better bin-packing* algorithm. In [6] an improved bin packing algorithm is used, where tasks are grouped in classes according to their utilizations. In [14] the performance of the *Best Fit* bin-packing algorithm is analyzed and it is shown that its worst case performance is not better than the worst case performance of *First Fit*.

Second, one can use a *tighter bound* for the admission control on each processor. The schedulability conditions given in [10] and [1] can be used to create an admission control that yields a high utilization, but these admission control algorithms are computationally expensive. The bound for multiprocessor admission control presented in [2] yields a higher processor utilization than partitioning with Equation (1).

Third, the tasks can be ordered so that *compatible tasks* are assigned to the same processor. Compatible tasks are tasks that achieve a high processor utilization when scheduled on the same processor. For example, tasks that have the same period are compatible and have a utilization bound of 100%. In [2] tasks that have periods closest to a power of two are considered to be compatible. In general, the order in which the tasks are presented to the bin packing algorithm can also yield significant improvements. In [5] tasks are sorted by decreasing utilization factor and then given to a *First Fit* bin packing algorithm. The partitioning technique proposed in this paper uses all three techniques to achieve a high processor utilization.

3 Admission control for uniprocessors

3.1 T-BOUND

A more optimistic admission control bound than Equation (1) can be obtained by taking advantage of the knowledge of the task characteristics [2, 8]. Similarly, we will use knowledge of the tasks periods to get a tighter bound.

The T-BOUND admission control first transforms the original task set, then applies an admission control to this transformed task set. We will show in Lemma 2 that if this transformed task set is schedulable, the original task set is also schedulable. To prove Lemma 2, we first paraphrase a Lemma which is proven in [2].

Lemma 1 *Given a task set $\mathcal{T} = \{(C_1, T_1), (C_2, T_2), \ldots, (C_m, T_m)\}$, if $\mathcal{T}$ cannot be scheduled on one processor with RMS, the task set $\mathcal{T}' = \{(2C_1, 2T_1), (C_2, T_2), \ldots, (C_m, T_m)\}$ cannot be scheduled on one processor with RMS either.*

```
1 ScaleTaskSet (In: $\mathcal{T}$, Out: $\mathcal{T}'$)
2 begin
3   for $i = 1$ to $m - 1$ do
4     $T'_i = T_i 2^{\log\lfloor \frac{T_m}{T_i} \rfloor}$
5     $C'_i = C_i 2^{\log\lfloor \frac{T_m}{T_i} \rfloor}$
6   endfor
7   sort the tasks in $\mathcal{T}'$ by increasing period
8   return ($\mathcal{T}'$)
9 end
```

Figure 1. Scale Task Set

Lemma 2 *Given a task set $\mathcal{T}$, let $\mathcal{T}'$ be the task set resulting from the application of the algorithm Scale Task Set to $\mathcal{T}$. If $\mathcal{T}'$ is schedulable on one processor using RMS, then $\mathcal{T}$ is schedulable on one processor with RMS.*

PROOF: We first prove that each time a period and a computation time are doubled (lines 4 and 5), the schedulability property is conserved, that is, if the modified task set is schedulable then the input task set is also schedulable. Each time a period is doubled, the task set is transformed from $\{(C_1, T_1), (C_2, T_2), \ldots, (C_m, T_m)\}$ into $\{(2C_1, 2T_1), (C_2, T_2), \ldots, (C_m, T_m)\}$. The application of the converse of Lemma 1 for each period transformation ensures that Lemma 2 holds. □

After performing the task set transformation, we apply our admission control to the transformed task set. This admission control is based on the bound given by Lemma 3. The proof of Theorem 4 in [13] shows that the computation times that minimize the utilization are

$$C_i = T_{i+1} - T_i \ \ (i = 1, \ldots, m-1) \ \text{ and } \ C_m = 2T_1 - T_m$$

It is easy to see that rewriting the task set utilization with these computation times gives the bound of Lemma 3.

Lemma 3 *Given a set $\mathcal{T}$ of m tasks ordered by increasing periods, and the restriction that the ratio between any task periods is less than 2, $\mathcal{T}$ is schedulable if*

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq \sum_{i=1}^{m-1} [\frac{T_{i+1}}{T_i}] + 2\frac{T_1}{T_m} - m \qquad (2)$$

Our goal is to use Lemma 3 to test the schedulability of a task set. Since the bound of Lemma 3 requires all period ratios to be less than 2, this bound can be used after the task set is transformed by *Scale Task Set.* This gives the following theorem.

Theorem 1 *Given a task set $\mathcal{T}$, let $\mathcal{T}' = \{(C'_1, T'_1), (C'_2, T'_2), \ldots, (C'_m, T'_m)\}$, be the task set resulting from the application of the algorithm Scale Task Set to $\mathcal{T}$. If Equation (2) holds for $\mathcal{T}'$, then $\mathcal{T}$ is schedulable on one processor with RMS.*

PROOF: For each task τ'_i ($1 \leq i < m$) in the transformed task set $\mathcal{T}'$, we have $T'_m/2 < T'_i \leq T'_m$, and therefore the ratio between any two periods is less than 2. This implies that we can use Lemma 3 to test the schedulability of $\mathcal{T}'$. From Lemma 2, we know that the schedulability of $\mathcal{T}'$ implies the schedulability of $\mathcal{T}$. □

The admission control based on *Scale Task Set* and Equation (2) is later referred to as T-BOUND. It should be noted that if it is impossible to schedule $\mathcal{T}'$ with T-BOUND, we cannot infer anything about the schedulability of $\mathcal{T}$.

3.2 R-BOUND

This section derives a least upper bound on the processor utilization for T-BOUND. Given a task set of m tasks, let r be T_m/T_1, where T_1 and T_m are the minimum and maximum periods and $r < 2$. Let $B(r, m)$ be the utilization re-written from Equation (2).

$$B(r, m) = \sum_{i=1}^{m-1} [\frac{T_{i+1}}{T_i}] + \frac{2}{r} - m \qquad (3)$$

$B(r,m)$ depends on r, m and the periods $T_1,\ldots,T_m$. We define $B_{min}(r,m)$ to be the minimum of $B(r,m)$ with respect to the periods, that is, the least upper bound on the processor utilization when using the T-BOUND. Theorem 2 shows what are the periods that minimize $B(r,m)$ and the expression of $B_{min}(r,m)$ in this case.

Theorem 2 *Consider a set of m tasks ordered by increasing periods, where the minimum and maximum periods are fixed and known and the ratio of any two periods is less than 2. The processor utilization for this task set is minimum when the ratio between the periods of two consecutive tasks is constant. In this case, the utilization $B_{min}(r,m)$ is*

$$B_{min}(r,m) = (m-1)(r^{1/(m-1)}-1) + 2/r - 1 \qquad (4)$$

PROOF: $B(r,m)$ is minimum when its derivative with respect to T_i is null for $i = 2,\ldots,m-1$.

$$\begin{aligned} \forall 1 < i < m, \qquad & dB(r,m)/dT_i = 0 \\ \Rightarrow \quad & -\frac{T_{i+1}}{T_i^2} + \frac{1}{T_{i-1}} = 0 \\ \Rightarrow \quad & T_i^2 = T_{i+1}T_{i-1} \end{aligned}$$

This system of equations gives

$$\frac{T_2}{T_1} = \frac{T_3}{T_2} = \ldots = \frac{T_m}{T_{m-1}} = K$$

This implies

$$r = \frac{T_m}{T_1} = K^{m-1} \qquad (5)$$

Combining (3) and (5), we obtain (4) to complete the proof. $\square$

Corollary 1 *When $m \to \infty$ the processor utilization $B_{min}(r,m)$ approaches* $\ln r + 2/r - 1$.

Figure 2 shows the processor utilization as a function of r. When r is close to 1, the processor utilization is also close to 1. We will use this observation in Section 4 to schedule RMS tasks on a multiprocessor system. For a given m the utilization given by Equation (1) is the minimum of the curve $B_{min}(r,m)$ in Figure 2. An admission control based on the bound (4) is from now on referred to as R-BOUND.

3.3 Comparison of T-BOUND and R-BOUND

We now wish to compare R-BOUND to T-BOUND and determine the conditions for which R-BOUND is a good approximation of T-BOUND. We define $B_{max}(r,m)$ to be the maximum of $B(r,m)$ with respect to the periods. If $B_{max}(r,m)$ is close to $B_{min}(r,m)$ then R-BOUND is a good approximation of T-BOUND, since the utilization achieved by T-BOUND is between $B_{min}(r,m)$ and $B_{max}(r,m)$. The periods that maximize $B(r,m)$ and the resulting $B_{max}(r,m)$ are given by Theorem 3.

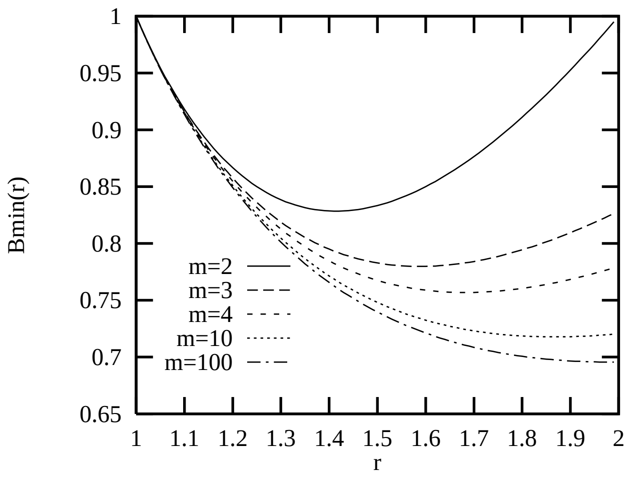

Figure 2. Minimum utilization of R-BOUND for different values of m.

Theorem 3 *Consider a set of m tasks ordered by increasing periods, where the minimum and maximum periods are fixed and known and the ratio of any two periods is less than 2. The utilization for this task set is maximum when, for any $k = 1,\ldots,m$*

$$T_i = T_1 \;\; (i = 1,\ldots,k) \;\text{ and }\; T_j = T_m \;\; (j = k+1,\ldots,m) \qquad (6)$$

In this case,

$$B_{max}(r,m) = r + 2/r - 2$$

.

PROOF: When the periods are given by (6), we have from (3)

$$\begin{aligned} B_{max}(r,m) &= \sum_{i=1}^{k-1}[\frac{T_1}{T_1}] + \frac{T_m}{T_1} + \sum_{j=k+1}^{m-1}[\frac{T_m}{T_m}] + 2\frac{T_1}{T_m} - m \\ &= r + 2/r - 2 \qquad (7) \end{aligned}$$

We now show that the utilization for any task set is less or equal to (7). From (3),

$$\begin{aligned} B(r,m) &= \sum_{i=1}^{m-1}[\frac{T_{i+1}}{T_i}] + \frac{2}{r} - (m-1) - 1 \\ &= \sum_{i=1}^{m-1}[\frac{T_{i+1}-T_i}{T_i}] + \frac{2}{r} - 1 \qquad (8) \end{aligned}$$

Since the tasks are ordered by increasing periods, we have $T_1 \le T_i$ ($\forall 1 \le i \le m$), which implies from (8)

$$B(r,m) \;\; \le \;\; \sum_{i=1}^{m-1}[\frac{T_{i+1}-T_i}{T_1}] + \frac{2}{r} - 1$$

$$
\begin{aligned}
&= \frac{\sum_{i=1}^{m-1}[T_{i+1}-T_i]}{T_1} + \frac{2}{r} - 1 \\
&= \frac{T_m - T_1}{T_1} + \frac{2}{r} - 1 \\
&= r + 2/r - 2 = B_{max}(r,m) \qquad (9)
\end{aligned}
$$

$\square$

Two facts worth noting from (9) are that the maximum utilization is independent of the number of tasks and that for $m = 2$, $B_{max}(r, 2) = B_{min}(r, 2)$. The processor utilization obtained by T-BOUND is always between $B_{max}(r, m)$ and $B_{min}(r, m)$. From Figure 3, we see that when r is close to 1, the difference between $B_{max}(r, m)$ and $B_{min}(r, m)$ is very small and therefore R-BOUND is a good approximation of T-BOUND. Section 4 shows how it is possible to obtain small values of r on each processor of a multiprocessor system when we use the R-BOUND for admission control.

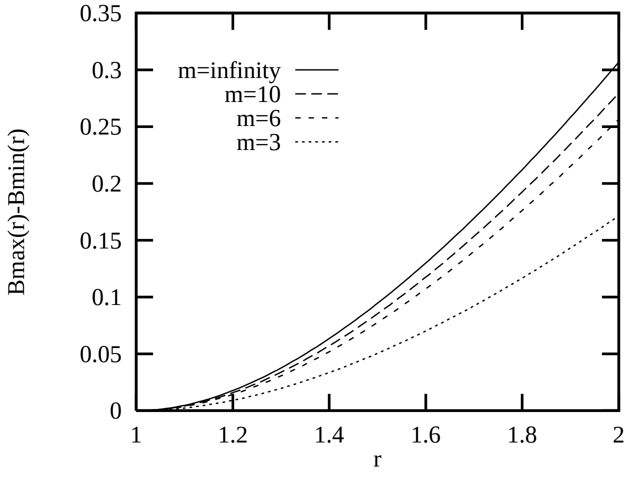

Figure 3. Difference between upper and lower bounds for T-BOUND

4 Admission control for multiprocessors

4.1 Admission control based on partitioning

In order to create an admission control for multiprocessor systems based on partitioning, we need to answer the three following questions.

- What uniprocessor admission control should be used? Clearly, a uniprocessor admission control with a high processor utilization should be preferred for partitioning. Further, because the uniprocessor bound is evaluated multiple times per processor, the complexity of the multiprocessor admission control increases very rapidly when the uniprocessor bound is complex. This implies that complexity of the uniprocessor admission control should be kept low. R-BOUND meets both criteria and is selected as the uniprocessor admission control.
- What is a criteria for compatible tasks? From Figure 2 we have observed that task sets with r close to one yield high processor utilization when assigned to the same processor. This gives us the criteria needed to isolate compatible tasks: we should try to schedule tasks with transformed periods close to each other on the same processor.
- What bin-packing algorithm should be used? Because the schedulability test given in Lemma 3 requires the tasks to be ordered by increasing periods, the bin-packing algorithm used in the multiprocessor admission control cannot reorder the tasks given as input. This implies that bin-packing algorithms like *Decreasing First Fit* [9] cannot be used. On the other hand, *Next Fit* is not only a suitable algorithm, but also has the advantage of working well with the compatibility criteria. Since the input tasks are sorted by increasing periods, tasks with periods close to each other are usually assigned to the same processor and yield high utilization. The performance can be improved further by using the *First Fit* algorithm. *First Fit* also respects the compatibility criteria and improves upon *Next Fit* by considering a processor P for a future task even after P rejected a task in the past. As consequence, *First Fit* is selected as the bin packing for partitioning tasks.

R-BOUND-MP, the multiprocessor admission control algorithm uses R-BOUND as follows. First, algorithm *Scale Task Set* is used to transform the original task set. Second, transformed tasks are assigned to processors with a *First Fit* bin packing algorithm. The bin packing algorithm uses the R-BOUND to determine if a processor can accept a new task. Finally, each original task is assigned to the processor that its transformed task was assigned to.

4.2 Performance analysis

This section compares the performance of different partitioning algorithms for multiprocessor systems. The performance is measured in terms of average processor utilization and complexity of the partitioning algorithm.

4.2.1 Experimental setup

Tasks are randomly generated according four parameters, minimum and maximum task period (T_{min} and T_{max}), minimum and maximum task utilization (U_{min} and U_{max}). The worst-case computation time C is randomly drawn from [1, T_{min}] with a uniform distribution and the period T is randomly drawn from [T_{min}, T_{max}] with a uniform distribution. For each task, $U_{min} \leq \frac{C}{T} \leq U_{max}$ must hold.

A fifth parameter, U_{tot}, is used to generate a task set that spans several processors. Tasks are generated and added to the task set until the sum of their utilization exceeds U_{tot}. This task set is given to the partitioning algorithms and each algorithm returns the number of processors needed to accept this task set. If an admission control requires P processors to schedule a task set with total utilization U_{tot}, then we define the average processor utilization of this admission control to be U_{tot}/P. Experiments are repeated 1,000 times per algorithm and the achieved processor utilizations are averaged.

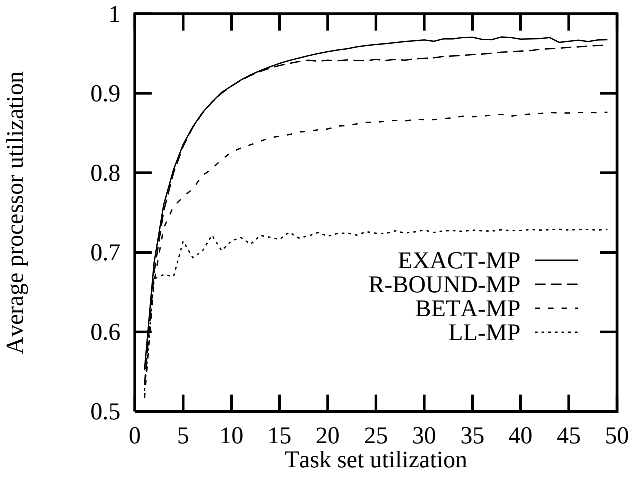

Figure 4. Average processor utilization for different multiprocessor admission control algorithms

4.2.2 Average processor utilization

Figure 4 shows the average processor utilization obtained with the experimental setup of section 4.2.1. The first thing to note is that as the load increases, the average processor utilization also increases. This is because when the number of processors is large, the bin packing algorithm has more freedom to assign compatible tasks to the same processor.

The algorithms compared are shown in Table 1. For high loads, the best processor utilization is obtained by EXACT-MP with a *FirstFit* bin-packing (97%), closely followed by R-BOUND-MP (96%). BETA-MP achieves a processor utilization of 87% and LL-MP with a *FirstFit* bin-packing has the lowest processor utilization (72%). The reason why R-BOUND-MP achieves such a high processor utilization when the load is high is that when many processors are needed, it is possible to assign tasks on a processor such that r is close to 1 on each processor. In this case, the utilization bound is close to 1. As a consequence, for high loads, R-BOUND-MP yields a processor utilization comparable to EXACT-MP. For these two admission control algorithms, the schedulability bound is not the bottleneck anymore, but the bin packing algorithm is.

4.2.3 Algorithms complexity

Admission control	Uniprocessor bound	Complexity
EXACT-MP	EXACT [10]	$O(m^3pN)$
R-BOUND-MP	R-BOUND	$O(m(p+\log m))$
BETA-MP	BETA [2]	$O(m\log m)$
LL-MP	LL [13]	$O(mp)$

Table 1. Admission control algorithms for multiprocessors

Table 1 also shows the complexity of the algorithms from the previous section, where m is the number of tasks, p the number of processors and $N = \sum_{i=1}^{m} \lfloor \frac{T_m}{T_i} \rfloor$. R-BOUND-MP has a complexity of $O(m(p+\log m))$ explained as follows. First, *Scale Task Set* takes $O(m \log m)$, then the *First Fit* bin-packing takes $O(mp)$ to assign transformed tasks to processors, and finally it takes $O(m)$ to map back to the original task set.

5 Periodic servers with priority exchange

Several solutions to service aperiodic tasks with a good response time have been proposed for RMS on a uniprocessor [11, 16]. One of them is based on *priority exchange* [11], in addition to the periodic tasks $\tau_1, \ldots, \tau_m$, a special task τ_s services the aperiodic requests as they arrive. τ_s has the highest priority and executes when an aperiodic task is being serviced. When there is no aperiodic task to service, τ_s exchanges its priority with the task of next highest priority to allow it to execute. If the periodic server τ_s has a utilization of U_s, when $m \to \infty$ the processor utilization approaches [11]

$$U = U_s + \ln \frac{2}{U_s + 1} \tag{10}$$

Theorem 4 shows that the R-BOUND improves upon the processor utilization of priority exchange given by Equation (10).

Theorem 4 *A set of m tasks can be scheduled with RMS on one processor with a priority exchange server of utilization*

U_s if the processor utilization U is such that

$$U \leq U_s + ((m-1)(r^{1/(m-1)} - 1) + 2/r - 1)(1 - U_s) \quad (11)$$

where r is the ratio between the largest and the smallest transformed periods.

PROOF: One can consider the processor P with a processing power of 1 as being composed of two smaller virtual processors P_1 and P_2. When a task set with a utilization bound U_b is scheduled on a virtual processor with processing power p, the resulting processor utilization U_p is

$$U_p = pU_b \quad (12)$$

We assign P_1 a processing power U_s and P_2 a processing power $1 - U_s$. If task τ_s is scheduled on P_1, its utilization bound is 1 (since this is the only task on this virtual processor) and from (12) yields a processor utilization U_{P_1}

$$U_{P_1} = U_s \quad (13)$$

The remaining tasks $\tau_1, \ldots, \tau_m$ are assigned to virtual processor P_2. From (4) and (12) we derive the utilization U_{P_2} of the virtual processor P_2 to be

$$U_{P_2} = ((m-1)(r^{1/(m-1)} - 1) + 2/r - 1)(1 - U_s) \quad (14)$$

Summing the utilization of the two virtual processors (13) and (14) gives (11). □

Corollary 2 *When $m \to \infty$ the processor utilization approaches*

$$U \to U_s + (\ln r + 2/r - 1)(1 - U_s) \quad (15)$$

Figure 5 plots the processor utilization obtained by Equation (15) (curves R-BOUND-PE) versus the one obtained from Equation (10) (curve PE). R-BOUND-PE achieves a better processor utilization for all values of r and U_s. When the tasks are assigned to a multiprocessor system as described in Section 4, compatible tasks are assigned to the same processor which implies that r is close to 1. When $r < 1.1$ on each processor, the processor utilization of the whole system is always above 90%.

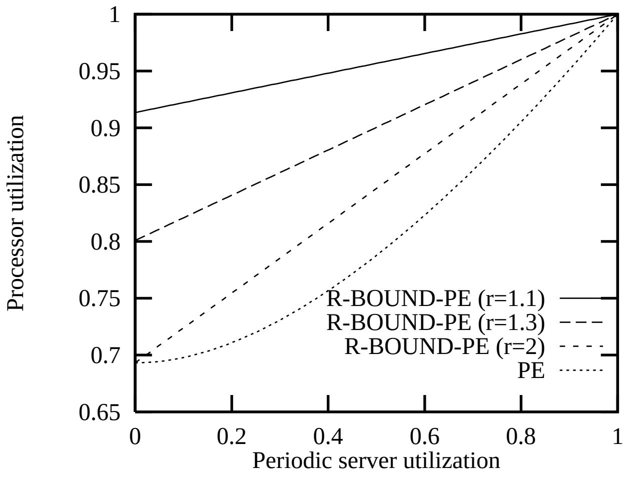

Figure 5. Processor utilization as a function of the server utilization.

6 Tolerance to transient faults

When a transient fault occurs, a common recovery technique is to re-execute the faulty task. However, in a real-time system, re-executing a task may delay other tasks and cause them to miss their deadlines. The conditions for recovery by task re-execution for RMS scheduling on a uniprocessor are given in [8]. By using slack of utilization U_b to re-execute faulty tasks, when at most one transient fault occurs within $T_m + T_{m-1}$, the processor utilization $U_{G-FT-RMS}$ for a set of m tasks has been shown [8] to be

$$U_{G-FT-RMS} = m(2^{1/m} - 1)(1 - U_b) \quad (16)$$

Theorem 5 shows that using the R-BOUND increases the processor utilization when transient faults are to be tolerated.

Theorem 5 *A set of m tasks can be scheduled with RMS on one processor with a backup utilization U_b if the processor utilization U is such that*

$$U \leq ((m-1)(r^{1/(m-1)} - 1) + 2/r - 1)(1 - U_b) \quad (17)$$

where r is the ratio between the largest and the smallest transformed periods.

PROOF: The first step is to transform the original task set $\mathcal{T}$ with the algorithm *ScaleTaskSet*. If the schedulability condition derived for the transformed task set $\mathcal{T}'$ holds, then $\mathcal{T}$ is schedulable. The same analysis as the one in [8] shows that the processor utilization U is minimum when

$$\text{for } 1 \leq i \leq m-1 \quad C'_i = (T'_{i+1} - T'_i)(1 - U_b) \quad (18)$$

$$\text{and} \quad C'_m = (2T'_1 - T'_m)(1 - U_b) \quad (19)$$

From (18) and (19) the resulting processor utilization is

$$U = (\sum_{i=1}^{m-1} [\frac{T'_{i+1}}{T'_i}] + 2\frac{T'_1}{T'_m} - m)(1 - U_b) \quad (20)$$

Following the proof of Theorem 2 with (20) instead of (2), the least upper bound for the processor utilization is (17). □

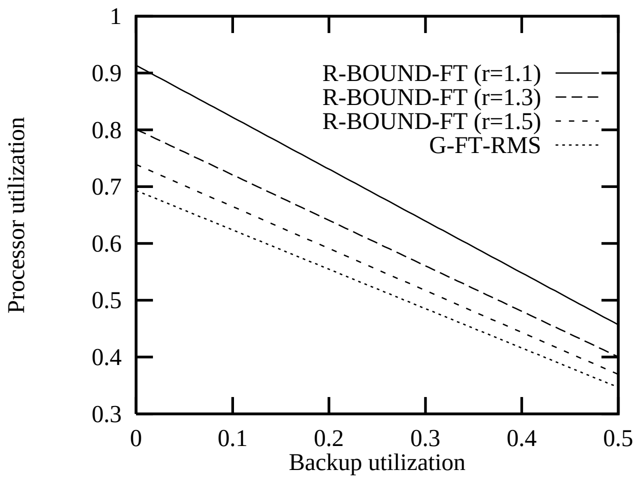

Figure 6. Processor utilization as a function of the backup utilization.

Corollary 3 *When $m \to \infty$ the processor utilization approaches*

$$U \to (\ln r + 2/r - 1)(1 - U_b) \tag{21}$$

Figure 6 shows the improvement on the processor utilization when Equation (21) is used (curves R-BOUND-FT) instead of Equation (16) (curve G-FT-RMS) when $m \to \infty$. The improvement of the processor utilization is larger (up to 20%) when r is close to 1.

7 Conclusion and further research

Rate-monotonic admission control for a uniprocessor exhibits a tradeoff between its computational complexity and the processor utilization it can achieve. Our work proposes two new RMS admission control algorithms for uniprocessors, T-BOUND and R-BOUND. When R-BOUND is used in a partitioning scheme to provide an RMS admission control for multiprocessor systems, experimental results show that this multiprocessor admission control yields very good processor utilization (up to 96% for a large number of tasks) and has low computational complexity. R-BOUND is also used to derive new and higher multiprocessor schedulability bounds when RMS is used with a periodic server or when transient faults need to be tolerated.

We are currently examining how R-BOUND can be modified to be used for on-line scheduling with dynamically changing task sets. We are also investigating how R-BOUND can be extended to deadline-monotonic and preemptive tasks with critical sections [15, 4].

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[2] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. on Computers*, 44(12):1429–1442, 1995.

[3] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. ARINC659 acheduling: Problem definition. In *Proceedings of the Real Time Systems Symposium*, pages 165–169, December 1994.

[4] M. Chen and K. Lin. Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems. *Journal of Real-Time Systems*, 2(4):325–346, 1990.

[5] S. Davari and S. K. Dhall. On a periodic real-time task allocation problem. In *19th Annual International Conference on System Sciences*, pages 133–141, 1986.

[6] S. Davari and S. K. Dhall. An on line algorithm for real-time tasks allocation. *IEEE Real-time Systems Symposium*, pages 194–200, 1986.

[7] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[8] S. Ghosh, D. Mossé, and R. Melhem. Fault-Tolerant Rate-Monotonic Scheduling. *Journal of Real-Time Systems*, 1998. to appear.

[9] D. S. Johnson, A. Demers, et al. Worst case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.

[10] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling: Exact characterization and average case behavior. *IEEE Real-time Systems Symposium*, pages 166–171, 1989.

[11] J. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. *IEEE Real-time Systems Symp.*, pages 261–270, 1987.

[12] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[13] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):47–61, 1973.

[14] Y. Oh and S. H. Son. Tight performance bounds of heuristics for a real-time scheduling problem. Technical Report CS-93-24, University of Virginia, 1993.

[15] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[16] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *Journal of Real-Time Systems*, pages 27–60, 1989.