Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor *

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

Abstract

In this paper we study GRMS, a global scheduling scheme for rate monotonic tasks on a multiprocessor. Several admission control algorithms for GRMS are presented, both for hard and soft real-time tasks. The average performance of these admission control algorithms is compared with the performance of known partitioning schemes. The result of these comparisons outlines some situations where one scheme is preferable over the other. Partitioning schemes are better suited for hard real-time systems, while a global scheme is preferable for soft real-time systems.

1 Introduction

A real-time system is one in which 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 task 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. The *utilization* of a task is defined to be C_i/T_i.

Given this task model, a *hard 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*[1] for the real-time system. The admission control is an algorithm that ensures that only tasks that will meet their deadlines are accepted in the system. A task set that does not miss any deadline is called *schedulable*. The scheduling policy determines which task instance is to dispatched next. In contrast to hard real-time, in a *soft real-time* system tasks can sometime miss a deadline. An admission control for a soft real-time system aims at admitting as many tasks as possible while minimizing the number of instances that miss a deadline. Since multiprocessor systems are becoming more common for real-time applications [2], scheduling real-time tasks on multiprocessor systems is also an important problem. Tasks can be scheduled on multiprocessor systems by doing *global scheduling* or *partitioning*[4]. In a global scheduling scheme, tasks can execute on any processor and, after being preempted, can be resumed on a different processor. In a partitioning scheme, each task is assigned to a processor and is only executed on this processor.

One of the most widely used scheduling policies on uniprocessors for preemptive periodic real-time tasks is *rate-monotonic scheduling* (RMS)[8]. RMS associates each task τ_i with a fixed priority $p_i = 1/T_i$. At any time, the available instance with the highest priority is being processed, where an available instance is one that has been released and has not yet completed. It is assumed that a task can be preempted by a higher priority task in negligible time. This work focuses on GRMS, a global scheduling scheme where tasks are assigned priorities according to the rate-monotonic policy (that is $p_i = 1/T_i$). GRMS enforces that at any moment in time, all p processors execute the p available tasks with the highest priority.

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

[1] In this paper scheduling policy refers to short-term scheduling or dispatching.

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. 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 [7]. 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.

The rest of the paper is organized as follows. Section 2 motivates the interest in GRMS. Section 3 focuses on hard real-time systems: a negative result for the critical instant of GRMS is first presented, then two admission control algorithms for GRMS are described and their performance is compared with the performance obtained by admission control algorithms based on a partitioning scheme. Section 3 concludes that a GRMS is not suitable for hard real-time systems. Section 4 considers soft real-time systems and analyzes the performance of GRMS when worst case computation times can be inaccurate. It is shown that in this case, GRMS outperforms the best partitioning scheme. The conclusion is presented in Section 5.

2 Motivation

Most multiprocessor scheduling techniques proposed in the literature are based on a partitioning scheme [4, 3, 1, 9]. Because a global scheduling scheme relaxes the constraint that a task must always execute on the same processor, one would expect global schemes to achieve a higher processor utilization than partitioning schemes. However, this is not the case. In [4], Dhall and Liu showed that a multiprocessor running GRMS can fail to schedule a task set with an arbitrary low utilization. This task set consists of $m-1$ tasks with computation 2ϵ, for some small ϵ, and period 1, and a task with computation 1 and period $1+\epsilon$. This task set cannot be scheduled on less than m processors. Since the $m-1$ first tasks have higher priority, when $m \to \infty$, the utilization of the system is arbitrarily low. Keeping in mind this impossibility result, it is legitimate to wonder if, for most task sets, a global scheme also achieves a lower processor utilization than a partitioning scheme. Therefore, our evaluation of global and partitioning schemes does not consider worst case situations, but rather the average behavior of these two schemes.

As mentioned above, global scheduling allows a task to be preempted on one processor and to resume on another processor. This preemption across processors has a greater overhead than preemption within the same processor. On some systems this preemption overhead can be too large to make global scheduling attractive. As example, on a cluster of workstations connected by a slow network, the preemption of tasks between processors is too slow and complex to be used for a real-time system. As a consequence, a partitioning scheme in which each task is assigned to a single processor is better suited for loosely connected computing systems. However, shared memory multiprocessor systems offer a fast and easy preemption across processors. Because all processors can access the shared memory, there is no need to relocate a task when it is preempted across processors. It is therefore interesting to investigate if a global scheme can take advantage of the efficient preemption across processors on a shared memory machine.

Task	Period	Computation	Utilization
τ_1	5	3	0.60
τ_2	7	4	0.57
τ_3	10	2	0.20
τ_4	15	7	0.46

Table 1. Example of task set

There are cases in which a global scheme performs better than a partitioning scheme. As an example, consider the task set given by Table 1. Any partitioning scheme requires at least 3 processors to schedule these tasks. This is because any partitioning of these tasks in two subgroups will create a subgroup of tasks with a total utilization greater than 1. However, this task set can be scheduled on 2 processors using GRMS if task τ_1 starts at time 0 and the other tasks start at time 2. This example further motivates the search for conditions in which global schemes outperform partitioning schemes. If such situations can be identified, this can help to determine when a global scheme should be used or when a partitioning scheme is preferable.

3 Global scheduling for hard real-time

When scheduling hard real-time tasks on a multiprocessor system, it is important to ensure that no task admitted in the system misses a deadline. This implies that a stringent admission control is required for hard real-time systems. In this section, we present two admission control algorithms for GRMS and discuss their performance.

3.1 Critical instant

From [8], the first step in designing an admission control for RMS on a uniprocessor is to determine the *critical instant*. A critical instant is defined to be the instant at which

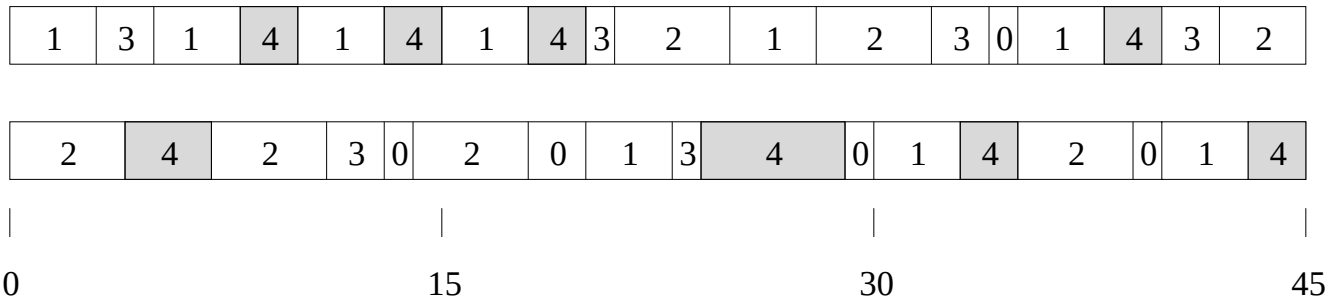

Figure 1. Critical instant of a task set

a request for a task will have the largest response time. The critical instant for a uniprocessor occurs when all tasks are submitted simultaneously [8]. If each task meets its first deadline when all tasks are submitted simultaneously, the task set is schedulable. We show below that, unlike for RMS in a uniprocessor, the critical instant for GRMS is not when all tasks arrive at the same time. For example, when the tasks from Table 1 are all submitted at time 0 on a system with 2 processors running GRMS, the first deadline of each task is met, but task τ_4 misses a deadline at time 45. Figure 1 shows the behavior of this task set. Numbers in each box correspond to the number of the task executing during the corresponding interval, and 0 means that the processor is idle because there are no tasks eligible for execution. Shaded boxes highlight the usage obtained by task τ_4. During its first two periods, task τ_4 can obtain the 7 time units needed to perform its computation. During its third period (time 30 to 45), τ_4 can only obtain 6 time units and misses its deadline at time 45.

The implication of this observation is that it is not enough to guarantee that all tasks do meet their first deadlines when submitted simultaneously, since later deadlines are not always guaranteed. Indeed in the worst case, it may be necessary to check that no task misses it deadlines for the first $LCM(T_i) + \max(T_i)$ time units, where $LCM(T_i)$ is the least common multiplier of the periods of all the tasks. Checking the first $LCM(T_i) + \max(T_i)$ time units would add undue overhead to the GRMS admission control, and thus, a stronger condition is needed.

In order to ensure that if the first deadline of a task is met all future deadlines will be met, we need to find the maximum amount of computation that a task can request regardless of its phasing with the other tasks. Theorem 1 gives an upper bound on this quantity.

Theorem 1 *Given a task τ with a computation of C and a period of T, define $Comp(\tau, t, t')$ to be the amount of computation that τ can request between time t and t'. Then,*

$$Comp(\tau, t, t') \leq (\lfloor \frac{t'-t}{T} \rfloor + 2)C \quad (1)$$

PROOF: Between time t and t' at most $\lfloor \frac{t'-t}{T} \rfloor$ complete instances of τ can execute. Before the first complete instance of τ executes, τ can request at most C time units for its computation. Similarly, after the last complete instance of τ finishes, at most C time units can be requested for computation. Therefore,

$$Comp(\tau, t, t') \leq (\lfloor \frac{t'-t}{T} \rfloor)C + C + C$$

□

3.2 Analytical bound

In this section, the following problem is considered: given a task set $\mathcal{T}$ of m periodic tasks and p processors, will any of the tasks miss a deadline if $\mathcal{T}$ is scheduled on the p processors using GRMS?

Lemma 1 shows that it is possible to consider the schedulability of one task at a time if the tasks are ordered by increasing period.

Lemma 1 *Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_i\}$ be a task set of i periodic tasks ordered by increasing period. If tasks $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ are schedulable using GRMS on p processors, scheduling task τ_i with the first $i-1$ tasks will not cause any of the first $i-1$ tasks to miss a deadline.*

PROOF: By assumption, the $i-1$ first tasks are schedulable (that is, they do not miss any deadline). Since task τ_i has a lower priority than the $i-1$ first tasks, τ_i cannot preempt (and therefore delay) any of the $i-1$ first tasks. The $i-1$ first tasks will execute exactly as they would if τ_i was not scheduled.

□

By using Lemma 1, the problem of admitting tasks τ_1, τ_2, ..., τ_i is now reduced to the following problem: will τ_i miss a deadline if tasks τ_1, τ_2, ..., τ_{i-1} have already been admitted?

Theorem 2 gives a necessary condition for task τ_i to be schedulable if tasks τ_1, τ_2, ..., τ_{i-1} have already been admitted. The idea is to compute how much available time

there is to execute task τ_i within its period, considering that tasks τ_1, τ_2, ..., τ_{i-1} have higher priority.

Theorem 2 *If tasks $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ are schedulable using GRMS on p processors, task τ_i is schedulable with the other $i-1$ tasks if*

$$C_i \leq (pT_i - \sum_{j=1}^{i-1}(\lfloor \frac{T_i}{T_j} \rfloor + 2)C_j)/p \quad (2)$$

PROOF: Without loss of generality we can assume that task τ_i is available at time 0. Note that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ can arrive before time 0, they will only interact with task τ_i after time 0. The amount of time available to τ_i between time 0 and time T_i is

$$AT(\tau_i) = pT_i - \sum_{j=1}^{i-1} Comp(\tau_j, 0, T_i)$$

where $Comp(\tau_j, 0, T_i)$ is defined in Theorem 1.

From Equation (1), we know that

$$Comp(\tau_j, 0, T_i) \leq (\lfloor \frac{T_i}{T_j} \rfloor + 2)C_j$$

Hence,

$$AT(\tau_i) \geq pT_i - \sum_{j=1}^{i-1}(\lfloor \frac{T_i}{T_j} \rfloor + 2)C_j$$

However, τ_i may not be able to use all $AT(\tau_i)$ time units to execute. The available time computed by $AT(\tau_i)$ can be scattered across p processors and different segments of available time may overlap. In the worst case, $AT(\tau_i)$ is divided in p segments of $\frac{AT(\tau_i)}{p}$ time units that completely overlap. Because task τ_i cannot be executed in parallel, only $\frac{AT(\tau_i)}{p}$ time units will be available for processing.

Therefore, if

$$C_i \leq (pT_i - \sum_{j=1}^{i-1}(\lfloor \frac{T_i}{T_j} \rfloor + 2)C_j)/p$$

task τ_i will be guaranteed to always complete before its deadline. □

We call GRMS-A the admission control for GRMS based on Equation (2) and Theorem 2. This admission control first sorts the task set by increasing period, considers each task in order, and uses the condition given by Theorem 2 to check that the task will not miss its deadlines. GRMS-A guarantees that no admitted task misses a deadline. However, this admission control is very pessimistic (as will be discussed in Section 3.4) and tasks sets that are schedulable can be rejected.

3.3 Optimal admission control

Since the schedulability condition given in Section 3.2 is not tight, we have developed another admission control, GRMS-OPT. GRMS-OPT is based on the same principle as the exact characterization of RMS for uniprocessors presented in [6]. This admission control checks at each point in time that no instance misses a deadline. In [6] the admission control only needs to check from time 0 to the maximum period in the task set, since the critical instant for uniprocessors is known. However, as explained in Section 3.1, the critical instant for uniprocessors does not apply to multiprocessors. As a consequence, this admission control needs to check that no task misses a deadline between time 0 and time $LCM(T_i) + \max(T_i)$, for $1 \leq i \leq m$.

Because GRMS-OPT checks the deadline of every instance of every task, it is an optimal admission control, that is, GRMS-OPT accepts a task set if and only if no deadline is missed. The drawback of GRMS-OPT is its large computational complexity that can make it unattractive for real life systems. In this paper we use GRMS-OPT in two ways. First, it is used to evaluate how pessimistic GRMS-A is. Second, the performance of GRMS-OPT is compared against the performance of known partitioning schemes.

3.4 Performance evaluation

The metric for the performance evaluation of the different admission control algorithms is the average processor utilization achieved.

3.4.1 Experimental setup

A task set is randomly generated according to the following input parameters.

- U_{tot} is the sum of all the tasks utilizations,
- T_{max} is the maximum period of a task,
- U_{min} is the minimum utilization of a task,
- U_{max} is the maximum utilization of a task,

A task set is generated as follows: a new task is randomly generated until the sum of the utilization of the tasks generated exceeds U_{tot}. Each task has a period T that is randomly drawn from $[1, T_{max}]$ with a uniform distribution and a worst case computation time C, which is randomly drawn from $[T \cdot U_{min}, T \cdot U_{max}]$ with a uniform distribution.

This task set is given to different admission control algorithms and each algorithm outputs p_{min}, the minimum number of processors needed to schedule this task set. The processor utilization for the experiment is defined to be

U_{tot}/p_{min}. 1000 experiments were conducted and the results presented are the averages obtained from the 1000 experiments.

The performance of the following algorithms are examined. The first three are based on a partitioning scheme, the last two are based on global scheduling.

- FF-LL: This scheduling algorithm uses a first fit packing technique with the bound developed in [8].
- FF-RB: This scheduling algorithm uses a first fit packing technique with a bound developed in [5]. This bound takes into account the ratio between tasks period and is tighter than the one from [8].
- BETA: This scheduling algorithm is presented in [1]. It based on a next fit with a bound tighter than the one from [8].
- GRMS-OPT: This admission control presented in Section 3.3 is used to determine how many processors are required by GRMS.
- GRMS-A: This method is based on the schedulability analysis presented in Theorem 2.

3.4.2 Average processor utilization

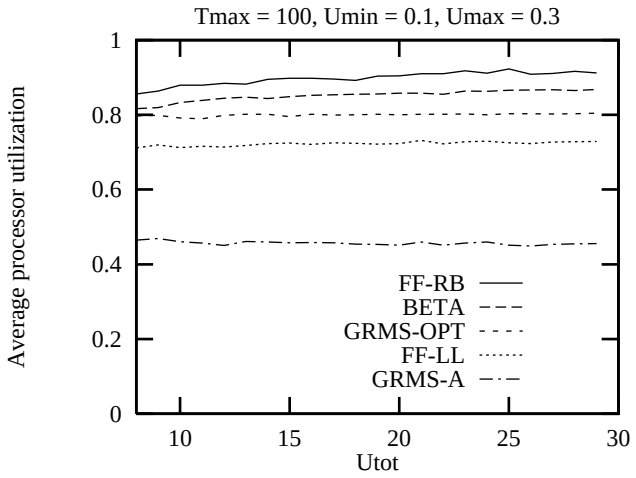

Figure 2. Performance of some scheduling algorithms for increasingly large task sets.

Figure 2 shows the performance of the different scheduling techniques when the total utilization of the task set varies. The most efficient algorithm is FF-RB, which achieves an average processor utilization of 90%, while BETA is around 85% and GRMS-OPT is around 80%. These results show that global scheduling performs typically worse when compared to good partitioning algorithms. The schedulability analysis presented in Section 3.2, GRMS-A, yields a very low processor utilization (less than 50%). Even if a better schedulability analysis is developed, the processor utilization it will achieve cannot exceed the one of GRMS-OPT, which is lower than the processor utilization achieved by good partitioning techniques.

3.4.3 Influence of the task characteristics

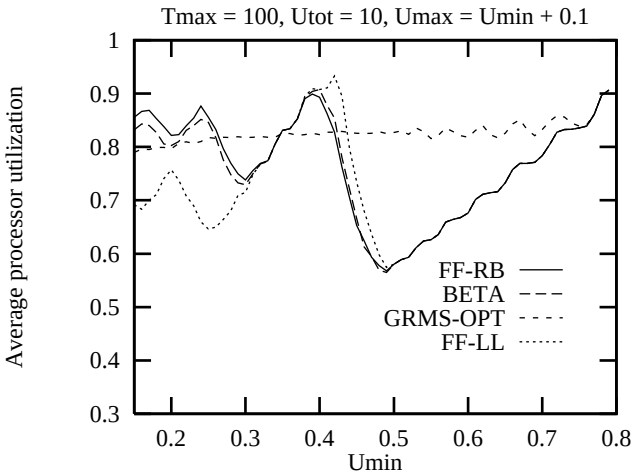

Figure 3. Influence of task utilization on the performance of some scheduling algorithms

We also ran a set of experiments to assess how the performance of scheduling algorithms varies when the characteristics of the tasks change. The first set of experiments evaluates the influence of the utilizations of individual tasks. In these experiments, U_{tot} is fixed and each task is generated so that its utilization is in the window [U_{min}, U_{min} + 0.1] (i.e, U_{max} = U_{min} + 0.1). Figure 3 shows that the performance of partitioning algorithms has large variation when the utilization of the tasks changes, whereas the performance of GRMS remains stable, around 80%.

The variation in the processor utilization achieved by partitioning schemes can be explained as follows. When a partitioning scheme is based on an admission control with a bound that achieves a per-processor utilization of U_b, the processor utilization increases when the task utilization approaches U_b/k ($k = 2, 3, \ldots$) and decreases after that. This is because when U_{min} is close to U_b/k, the sum of the utilizations of those k tasks approaches U_b and can be successfully scheduled. When U_{min} becomes greater than U_b/k, the utilization bound is exceeded and only $k-1$ tasks can be scheduled on the processor. These peaks appear first for less efficient admission control algorithms since their U_b, and therefore U_b/k, is smaller.

Figure 4 shows the performance of the different scheduling schemes when the maximum period of the tasks T_{max}

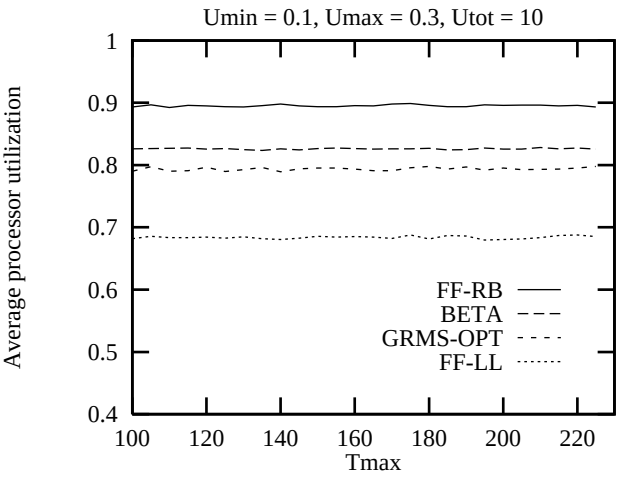

Figure 4. Performance of some scheduling algorithms for increasingly large task periods.

varies. The processor utilization of the different scheduling schemes is not affected by the tasks periods.

3.5 Discussion

GRMS exhibits some interesting properties for scheduling periodic real-time tasks on a multiprocessor system. The processor utilization GRMS achieves is constant for a large range of tasks periods and tasks utilizations, unlike partitioning schemes (Figure 3). However, if processor utilization is to be used as a performance criteria, global scheduling should not be used for hard real-time systems for several reasons. First, GRMS has complex behavior that makes it difficult to design an admission control that is not too pessimistic. Second, even if one is willing to use the complex optimal admission control for GRMS, the processor utilization achieved by GRMS-OPT is lower both in the worst-case [4] and in the average-case (Figure 2) than the processor utilization achieved by partitioning schemes. Therefore, partitioning schemes should be used for hard real-time tasks on a multiprocessor system.

4 Global scheduling for soft real-time

Section 3 has shown that global scheduling is not suitable for hard-real time system. A hard real-time system assumes that the worst case execution time of each task is properly computed. However, determining the worst case computation time of a task is a complex and sometimes inaccurate procedure. This section examines how global scheduling performs when the worst-case computation time is inaccurate (i.e, can be over or under-estimated) and some deadlines can be missed. This should typically increases the utilization of the processors [10, 11].

4.1 Admission control

Although some tasks may sometimes miss a deadline in a soft real-time system, an admission control is still useful. This admission control should reject tasks that are likely to miss their deadlines. The experiments from Section 3 have shown that the processor utilization achieved by GRMS is not affected by the characteristics of the task set and is always close to 80%. Therefore, an admission control that enforces a processor utilization below 80% should perform well in most cases. GRMS-S is the admission control used in this section for GRMS on soft real-time systems. GRMS-S keeps track of U, the total utilization of the tasks admitted in the system. When a new task τ_i with utilization U_i arrives, GRMS-S accepts τ_i if

$$U + U_i \leq 0.8p \tag{3}$$

where p is the number of processors in the system.

4.2 Experimental setup

The tasks characteristics T and C are generated as described in Section 3. A new task characteristic is introduced: C_{actual} which is drawn with a uniform distribution from the interval $[C(1-\Delta), C(1+\Delta)]$, where Δ is a parameter of the experiment. Tasks are added to the task set until the sum of their utilizations exceeds U_{tot}. The number of processors available in the system, p, is set to U_{tot}, that is each processor has a load of 1 if all the tasks can be scheduled.

The task set generated is given to the partitioning algorithm that performs the best for hard real-time tasks, FF-RB. FF-RB partitions the tasks according to C, since C_{actual} is unknown. Note that, on average, C_{actual} is equal to C. If FF-RB cannot find a processor to which a task is assigned, the task is rejected. Once the partitioning algorithm assigns tasks to processors, the admitted tasks are run on each processor and the percentage of instances that miss a deadline is measured. The same task set is also given to GRMS-S. GRMS-S rejects tasks according to Equation 3. The tasks admitted are run on U_{tot} processors using GRMS and the percentage of instances that miss a deadline is measured.

In this soft real-time system, the performance of the scheduling schemes is analyzed in two cases. In the first case, an instance that misses its deadline is terminated by the scheduler. In the second case, an instance that misses a deadline is allowed to run to completion.

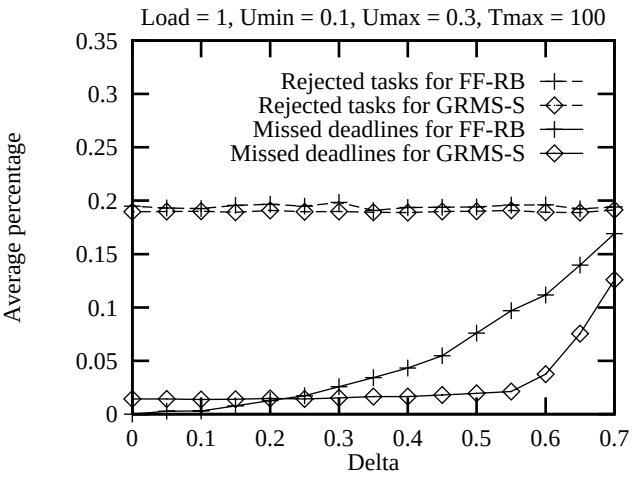

Figure 5. Performance of GRMS and FF-RB for soft real-time when late tasks are allowed to complete.

4.3 Performance analysis

Figures 5 and 6 show the percentage of tasks rejected and the percentage of instances that miss a deadline for GRMS and FF-RB as a function of Δ. Figure 5 was obtained when instances that miss a deadline are allowed to run to completion. Figure 6 was obtained when instances that miss a deadline are aborted. Note that both cases exhibit the same behavior, but a few more deadlines are missed if late tasks are allowed to complete, since this may delay other tasks.

The percentage of rejected tasks is similar for both FF-RB and GRMS and is around 19%, with GRMS being slightly better. The percentage of admitted tasks that miss a deadline has wider variation between FF-RB and GRMS. When Δ is close to 0, FF-RB misses less deadlines, this situation is close to the one examined in Section 3 where $\Delta = 0$. However, when Δ increases, GRMS misses less deadlines than FF-RB. This is because GRMS can reclaim the unused CPU time when a task does not conform to its timing specifications. This resource reclamation can be done among all the tasks in the task set. FF-RB can only perform this resource reclamation among the tasks assigned to the same processor. When partitioning the tasks, it may be the case that FF-RB assigns tasks with an underestimated worst case computation time to the same processor. In this case tasks will miss deadlines and will not be able to use the spare processing power of another less loaded processor (i.e, a processor that is assigned tasks with an over-estimated worst case computation time.)

It is often the case that tasks do not execute for their worst case execution time. When this happens, resources are reclaimed and can be used for other tasks. Because GRMS can perform this resource reclamation among all the tasks in the system, we suspect that GRMS does also outperform FF-RB in this case.

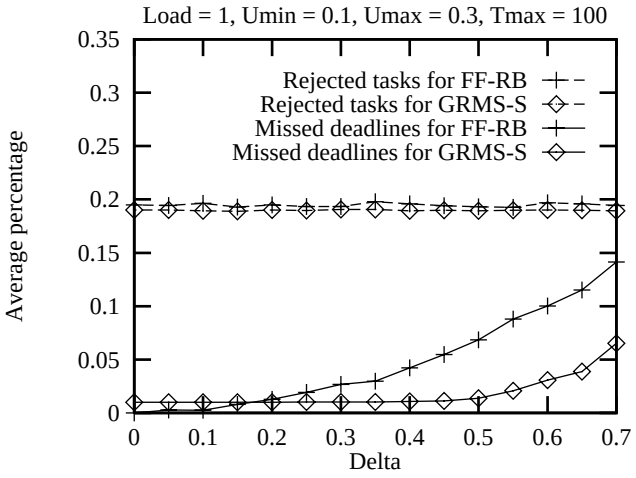

Figure 6. Performance of GRMS and FF-RB for soft real-time when late tasks are aborted.

5 Conclusion

This work focuses on a global scheduling scheme for real-time tasks where priorities are assigned according to a rate-monotonic policy. Several admission control algorithms are presented and their performance analyzed. Experimental results indicate that a global scheduling scheme performs poorly on average for hard-real time systems and that partitioning schemes should be preferred in this case. However, for soft real-time systems, a global scheduling scheme outperforms good partitioning schemes when the worst-case computation time of a task is inaccurate. This is because in a global scheme, overloaded processors can take advantage of the spare processing power from underloaded processors, while in a partitioning scheme a task assignment is fixed. Our future research will investigate other dispatching algorithms and different priority assignments for global scheduling which will, consequently, lead to different bounds.

References

[1] 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.

[2] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. ARINC659 acheduling: Problem definition. In *Pro-*

ceedings of the Real Time Systems Symposium, pages 165–169, December 1994.

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

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

[5] S. Lauzac, R. Melhem, and D. Mossé. An efficient RMS admission control and its application to multiprocessor scheduling. Technical Report CS-TR-97-11, University of Pittsburgh, 1997.

[6] 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.

[7] 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.

[8] 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.

[9] Y. Oh and S. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems Journal*, 9:207–239, 1995.

[10] C. Shen, K. Ramamritham, and J. Stankovic. Resource Reclaiming in Multiprocessor Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):382–397, April 1993.

[11] P. Thambidurai and K.S. Trivedi. Transient overloads in fault-tolerant real-time systems. In *Real-Time Systems Symposium*, pages 126–133, Dec 1989.