Some Results in Rate Monotonic Scheduling with Priority Promotion

Mitra Nasri and Gerhard Fohler
Chair of Real-time Systems,
Technische Universität Kaiserslautern, Germany
{nasri,fohler}@eit.uni-kl.de

***Abstract*—Rate monotonic (RM) scheduling algorithm cannot guarantee schedulability of highly utilized tasks in all cases. In this paper, we increase schedulability of RM by assigning fixed set of priority promotions for each task. We present an algorithm to assign promotion times (PT) and the new priority at each PT. Each priority promotion occurs after a constant relative time from the release of an instance of a task. This technique can be implemented using a container at the operating system's side which sets a timer interrupt for the priority promotion. Our priority promotion algorithm is offline and it promotes priorities only at the latest release of the higher priority tasks before deadline of the first instance of each task. In our solution, the maximum number of promotions is bounded to the number of higher priority tasks with smaller periods. The performance of our solution approach has been evaluated by extensive sets of simulations. According to our experimental results, a) our solution dominates RM, i.e., it guarantees schedulability whenever the task set is RM-feasible while it improves the schedulability in the task sets which are not RM-feasible, b) it has a significantly higher schedulablity ratio than RM, and c) even in cases where it cannot guarantee the schedulability, it has a low ratio of missed jobs.**

I. Introduction

It is known that for task sets with implicit deadline, the rate monotonic (RM) scheduling algorithm is optimal among other fixed-priority (FP) algorithms [1]. However, RM guarantees 100% utilization only in special cases, e.g., in harmonic task sets [2]. Unlike dynamic priority-based scheduling algorithms such as EDF [1], RM has considerably lower run-time overhead, however, it cannot guarantee schedulability in general cases of periodic tasks with high utilization.

The idea of *priority promotion* is introduced in [3] along with the concept of dual-priority tasks where each task is allowed to promote its priority at most once. The system uses a fixed-priority scheduling algorithm. The priority promotion happens after a specified amount of time from the release of each job of the task. In [4], open problems for the existence of a feasible dual-priority assignment has been discussed. It have been shown that for the special case of systems with two tasks, there exists such an assignment which guarantees schedulability up to 100% utilization. Recently, in [5], an upper bound on the priority promotions per job has obtained for EDF. This bound is a function of the largest deadline in the task set. It has been proven that the worst-case response time (WCRT) of tasks with priority promotions scheduled by RM will not necessary happen in their first release, which means that the *critical instant* is no longer at the first release of the tasks. It inherently increases the complexity of finding a feasible priority promotion solution. In [5], no algorithm has been presented to promote priority of the tasks.

In this paper, we propose a method to increase RM schedulability by assigning a fixed set of priority promotions to each task. The priority promotions happen at a fixed relative time from the *release* of each job. For each promotion time (PT), we define a new priority which is higher than the last priority of the task before the latest PT. The basic idea of the algorithm is to set priority promotions for the first release of the low priority task at the times at which high priority tasks have their latest release before the deadline of the low priority task. Using these promotion times, RM and EDF behave almost the same, at least for the first instance of the tasks. Since the maximum number of promotions for each task is limited to the number of higher priority tasks, it is bounded to $n-1$, where n is the number of the tasks. The algorithm itself has $O(n^2 log(n))$ computational complexity.

Priority promotion times and the new priorities are assigned at design-time. To be able to apply them at run-time, each task has a container, implemented on the operating system's level, which governs promotion times (using a timer interrupt). Doing so provides separation of concerns for the application's designers and maintains security at the system level such that the applications cannot modify their own priorities.

The remainder of the paper is organized as follows; the system model is presented in Sect. II. In Sect. III, we introduce our priority promotion algorithm. Efficiency of the algorithm is evaluated in Sect. IV in terms of schedulability ratio and job miss ratio. Some discussions about the algorithm and its schedulability analysis has been provided in Sect. V, and finally, the paper is concluded in Sect. VI.

II. System Model and Problem Statement

We assume a uni-processor system and a hard real-time task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with n independent periodic tasks. Each task τ_i is described by $\tau_i : (c_i, T_i)$, where c_i is the worst-case execution time and T_i is the period of the task. We assume synchronous task sets with no release offsets. Tasks have implicit deadlines, i.e., for each task, the deadline is equal to the period. They are indexed such that $T_1 \leq T_2 \leq \ldots \leq T_n$. The utilization of the task set is denoted by $U = \sum_{i=1}^{n} u_i$ where $u_i = c_i/T_i$. Moreover, the hyperperiod of this task set is the least common multiplier (LCM) of the periods, and is denoted by H.

For each task, we assign a set of time instants at which the priority is promoted. The j^{th} priority promotion of task

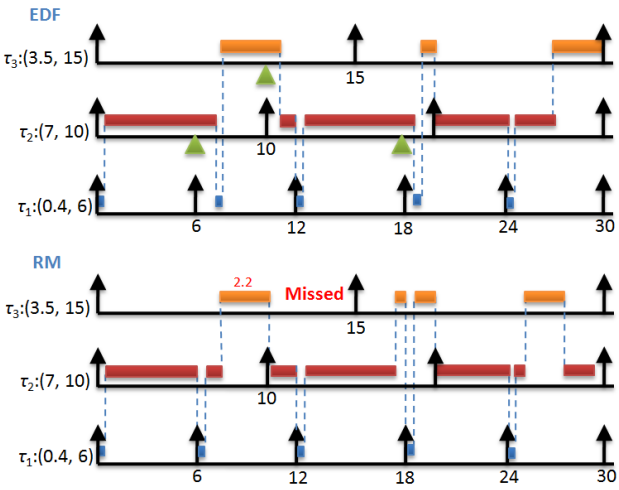

Fig. 1. EDF schedule versus RM schedule in a task set with 100% utilization

τ_i is identified by $P_{i,j} : (\delta_{i,j}, p_{i,j})$ for $1 \leq j \leq m_i$ where $0 \leq \delta_{i,j} < T_i$ is the priority promotion time, $p_{i,j}$ is the new priority indicator, and m_i is the number of priority promotions of task τ_i. Since in RM, the priority indicator is the period of the task, i.e., the smaller the period, the higher the priority, we assume $p_{i,j}$ contains a value which is smaller than the current task's period and it can be used as the promoted priority for the current instant of the task.

III. Priority Promotion Approach

In this section we present a simple idea to derive effective instants of time, i.e., PT, where priority promotion helps RM to behave more like EDF. We introduce our priority promotion assignment algorithm (PPA) which will be used at design-time to find constant relative time instants (w.r.t. the release of each job of the task) at which the priority can be promoted.

For two tasks τ_i and τ_j, $T_j < T_i$, RM priorities and EDF priorities are exactly the same for the jobs of τ_j which have deadlines before the deadline of current instant of τ_i. For those jobs, both EDF and RM prioritize τ_j over τ_i. The only difference happens for the latest job of τ_j which is released before deadline of τ_i and its deadline is after deadline of τ_i. Fig. 1 shows this situation for a task set with 3 tasks at their first releases. As shown in this figure, if we promote priority of τ_3 at 10 at least for the first instance of τ_3, EDF and RM schedule become identical.

Since in a synchronous task set, the first instance of each task encounters the highest workload, we assign priority promotions according to the first instance of the tasks. Our priority promotion assignment algorithm finds the latest releases of the tasks with smaller deadlines than τ_i. Those releases are denoted by r_g for $1 \leq g < i$. In the first step we sort those release times and then, starting from the earliest release, we add a priority promotion to the list of promotions of τ_i. Parameters of this promotion are $\delta_{i,j} = r_g$ and $p_{i,j} = T_i - r_g$ where j is the index of the current priority promotion. Then all releases of the tasks with $T_q \geq T_g$ will be removed from the list because if τ_i gains a priority which is higher than

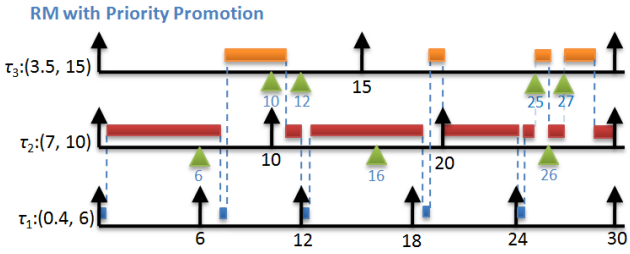

Fig. 2. RM with priority promotion schedule for the task set in Fig. 1

τ_g, it also has a higher priority than τ_q. After removing those items from the list of releases, we continue to the next earliest release and add the next priority promotion time until there is no other item in the list of releases. This algorithm has been shown in Alg. 1. Fig 2 shows the result of RM with the priority promotion for the previous example. Comparing this figure with Fig. 1 we see that these three algorithms produce different schedules.

Algorithm 1 Priority Promotion Assignment Algorithm

Input τ: where τ is the set of tasks
Output P: P is the set of priority promotions

1: $P_1 \leftarrow \emptyset$
2: **for** $i \leftarrow 2$ **to** n **do**
3: $R \leftarrow$ latest releases time of task τ_1 to τ_{i-1} before T_i
4: Sort R in ascending order
5: $P_i \leftarrow \emptyset$
6: **while** (R is not empty) **do**
7: $r^{min} \leftarrow$ the earliest release in R
8: $g \leftarrow$ index of the task which is released at r^{min}
9: $\delta \leftarrow r^{min}$
10: $p \leftarrow T_i - r^{min}$
11: $P_i \leftarrow P_i \cup \{(\delta, p)\}$
12: Remove any r_k, $1 \leq k < i$ with $T_k \geq T_g$ from R
13: **end while**
14: **end for**
15: **return** P

Offline computational complexity of Alg. 1 is $O(n^2 log(n))$ because in Line 3 we must sort R which contains the list of the latest releases, and at most has $n-1$ items. We can use a double linked list during the construction of R to have an efficient implementation. Every item in this list is attached to one of the release values in R and points to the next item in R which has a larger period than the current item. Using this list, Line 12 can be implemented in $O(n)$ while the amortized cost of the while-loop remains $O(1)$ per task, because each task is either used in priority promotions or removed by Line 12. Note that we have at most $n-1$ tasks in the while-loop. As a result, computational complexity of each iteration of the for-loop in Lines 2 to 14 is $O(nlog(n))$ due to the sort operation, and hence, the algorithm runs in $O(n^2 log(n))$.

To apply priority promotion at run-time, each task can have an OS-level container which is equipped with a timer event. Starting from the release of each job, the container sets the next timer interrupt for the next priority promotion time from the given list which is produced by Alg. 1. If the task is finished before the next timer interrupt, the interrupt can be

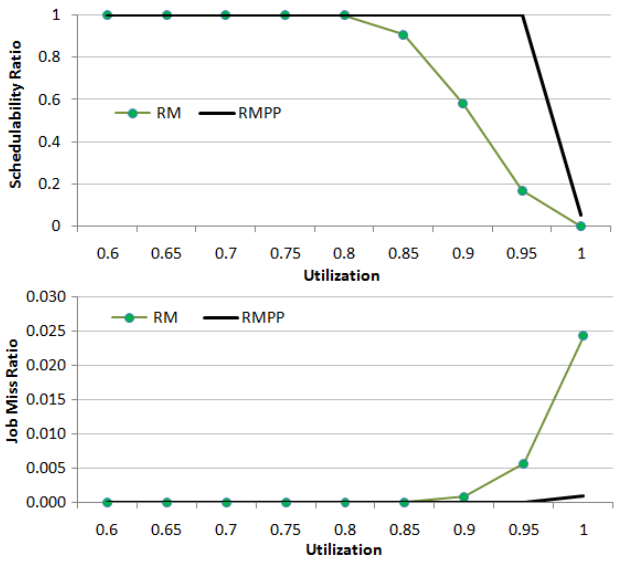

Fig. 3. Schedulability ratio and job miss ratio of the algorithms

canceled. In this implementation, permissions for promoting the priority belong to the operating system. Thus, there will be no issue regarding malicious promotions made by the applications themselves. More discussions about schedulability of our approach are presented in Sect. V.

IV. Experimental Results

In this section, we compare the performance of RM and RM with priority promotion (RMPP) which was described in Sect. III. Performance measures are job miss ratio and schedulability ratio where the former is the number of missed jobs divided by total number of jobs, and the latter is the number of schedulable task sets with no missed jobs divided by total number of generated task sets. Periodic task sets have been constructed according to [6] paper, i.e., in the first step, based on the given utilization U, n random u_i values have been obtained from uUniFast algorithm, and then, T_i has been selected randomly from $[10, 200]$ with uniform distribution. Based on u_i and T_i, c_i is calculated. Due to the space limitation, we have reported only the results for task sets with $n = 7$ tasks.

We use task set utilization U as the parameter of the experiment. It ranges from 0.1 to 1.0 with steps of 0.1. For each utilization value we have generated 200 random task sets as stated before. All resulting data is within a confidence interval ± 0.05 for confidence level 0.95. Fig. 3 shows the results of the experiment. Results of EDF algorithm have not been reported because it has schedulability 1 and miss ratio 0 in this setup.

In the next experiment, we only consider random task sets which are schedulable by RM. We have repeated the experiment for 10000 task sets per utilization value ranging from 0.6 to 1.0. Interestingly, we never found a case where RM guarantees schedulability while RMPP does not. In other words, based on our experiments, RMPP never misses a deadline if the task set is RM-feasible. Later in Sect. V we discuss

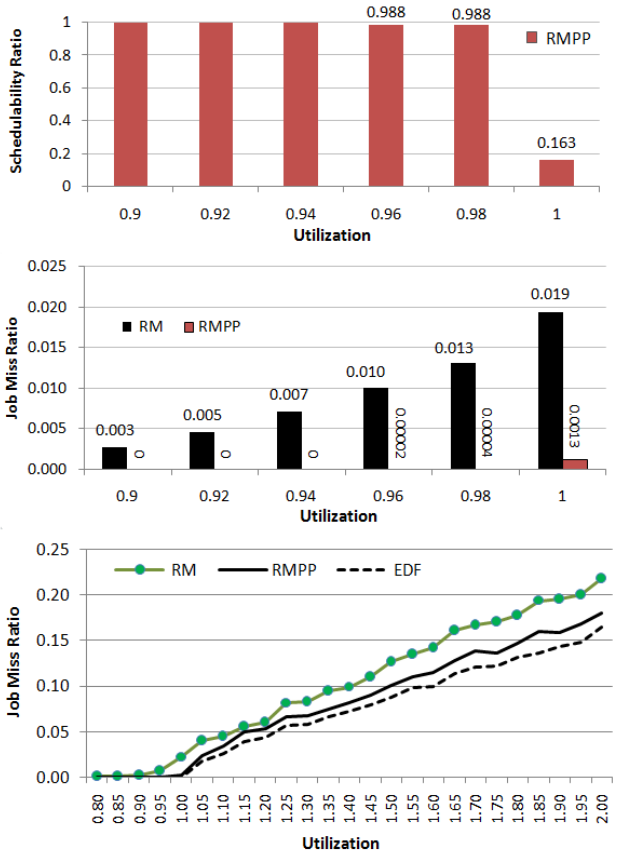

Fig. 4. Schedulability ratio and job miss ratio of RM and RMPP in the task sets which are not RM-feasible. Also for those task sets, we have compared job miss ratio of the algorithms with EDF in cases where $0.8 \leq U \leq 2.0$ which is an overloaded system

this observation with more detail. In the third experiment, we only consider random task sets which are not schedulable by RM algorithm. According to the results shown in Fig. 4, by adding fixed set of priority promotions assigned by Alg. 1 it is possible to efficiently increase RM schedulability. Even in the cases where hard real-time guarantee was not possible for RMPP, it has a very small miss ratio compared to RM.

V. Discussions

As shown by Fig. 3 and 4, our approach for promoting priorities is not optimal, i.e., it cannot guarantee all deadlines. Fig. 5 shows a case where the second instance of a task could not be scheduled even with the promoted priorities. The reason is that to assign promotion times for a task such as τ_i, Alg. 1 only considers set R of the latest releases which are constructed based on T_i. For the case of task sets with two tasks, [4] has shown that the optimal priority promotion assignment must consider the greatest common divisor of two periods. However, the problem remains unsolved for larger task sets with more than two tasks.

As mentioned in the second experiment of Sect. IV, according to our results, if a task set is RM-feasible, it is RMPP-

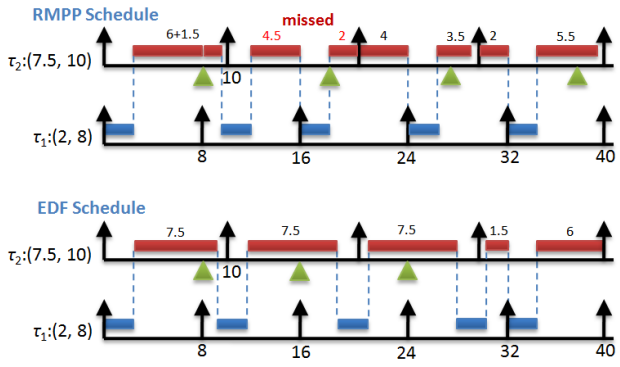

Fig. 5. A counter example to show non-optimality of PPA algorithm

feasible too. For justifying this observation we start with task sets with $WCRT_i \leq r^{min}$ for $1 \leq i \leq n$ where

$$r^{min} = \min \left\{ \left\lfloor \frac{T_i}{T_g} \right\rfloor T_g \right\}_{1 \leq g \leq i} \quad (1)$$

In these task sets, each task will be finished before it is promoted at r^{min} which is the first promotion time. Assume an RM-feasible task set is given which has $r^{min} < WCRT_i \leq r^{max} \leq T_i$ where r^{max} is the smallest release time among all release times in set R (produced in Line 3 of Alg. 1) which is larger than $WCRT_i$. If RM preserves schedulability, it means that within the schedule of higher priority tasks from time instant r^{min} to r^{max}, there will be enough idle time so that τ_i can finish its execution before its deadline. Thus, all higher priority tasks which are released between r^{min} and r^{max} must be finished before r^{max}, otherwise, RM will not start the remaining execution time of τ_i after r^{min}. Note that according to the definition, all of these tasks have deadlines greater than or equal to T_i, otherwise, it would not be their latest release before T_i. In an RMPP and EDF schedule, any of the jobs which are released from r^{min} to T_i will have lower priority than τ_i, consequently, they will be scheduled after τ_i. Due to the fact that the task set is RM-feasible, we know that all of the high priority tasks together with τ_i can be finished before r^{max}, which is smaller than or equal to T_i. Consequently, they can be finished before their deadline as well. As a result, at least for the first instance of the tasks in an RM-feasible task set, priority modifications of RMPP will not cause a deadline miss for any of the higher priority tasks. It is worth mentioning that priority modifications of any of the tasks with smaller period than τ_i has no effect on the schedule of τ_i because still all of those tasks have higher priority than τ_i.

The previous justifications are not complete because we did not consider future releases of the tasks. According to [5], in a fixed-priority algorithm with promoted priorities, the classic notion of critical instant will no longer be valid [5], i.e., the WCRT of the tasks may not appear in their first release. As a result, to provide a valid schedulability test or to prove the dominance of RMPP over RM, we need to consider future releases of the tasks too. As shown in Fig. 5, a deadline miss might happen for the second release of a low priority task because of the fact that in that release (or one of the future releases), promotions must happen earlier than the promotion times which we have assigned according to the first release of the task.

VI. Summary and Conclusion

In this paper we have used priority promotion approach with constant number of promotions per job to improve the schedulability of RM algorithm. Our solution, which is called RM with priority promotion (RMPP) has two phases; in the offline phase we assign promotion times and the new priorities respectively, and in the online phase, an OS-level container applies those priority promotions for each job of a task. Promotion times are obtained based on the latest release of the higher priority tasks under the first release of each task. This algorithm has $O(n^2 log(n))$ computational complexity. Moreover, the maximum number of promotions for each task is bounded to the number of higher priority tasks in the task set.

In our experimental results, our approach dominates RM, i.e., all RM-feasible task sets has been RMPP-feasible too. Besides, in task sets which were not RM-feasible, RMPP had a considerably high schedulability ratio. Also RMPP had a low miss ratio in cases where it cannot guarantee the schedulability. We have tried to justify the intuitive reasons behind the dominance of RMPP over RM, though, the actual proof remains as a future work. We will try to derive necessary or sufficient schedulability conditions for RMPP as well.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] C.-C. Han and H.-Y. Tyan, "A Better Polynomial-time Schedulability Test for Real-time Fixed-priority Scheduling Algorithms," in *IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 1997, pp. 36–45.

[3] A. Burns and A. J. Wellings, "Dual Priority Assignment: A Practical Method for Increasing Processor Utilisation," in *Euromicro Workshop on Real-Time Systems (EWRTS)*, 1993, pp. 48–55.

[4] A. Burns, "Dual Priority Scheduling: Is the Processor Utilisation bound 100% ," in *International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2010, pp. 3–5.

[5] D. Masson, L. George, and J. Goossens, "Dual Priority and EDF: a closer look," in *Work-in-Progress Session of IEEE Real-Time Systems Symposium (WiP-RTSS)*, 2014.

[6] E. Bini and G. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.