Int. J. Appl. Math. Comput. Sci., 2015, Vol. 25, No. 4, 709–722
DOI: 10.1515/amcs-2015-0051

MAINTAINING THE FEASIBILITY OF HARD REAL–TIME SYSTEMS WITH A REDUCED NUMBER OF PRIORITY LEVELS

MUHAMMAD BILAL QURESHI [a,*], SALEH ALRASHED [b], NASRO MIN-ALLAH [a,b], JOANNA KOŁODZIEJ [c], PIOTR ARABAS [d]

[a]Department of Computer Sciences
COMSATS Institute of Information Technology, Islamabad 46000, Pakistan
e-mail: muhdbilal.qureshi@gmail.com

[b]College of Computer Science and Information Technology
University of Dammam, Saudi Arabia
e-mail: saalrashed@ud.edu.sa, nasar@mit.edu

[c]Institute of Computer Science
Cracow University of Technology, ul. Warszawska 24, 31-155 Cracow, Poland
e-mail: jokolodziej@pk.edu.pl

[d]Institute of Control and Computation Engineering
Warsaw University of Technology, Wąwozowa 18, 02-796 Warsaw, Poland
e-mail: piotr.arabas@nask.pl

When there is a mismatch between the cardinality of a periodic task set and the priority levels supported by the underlying hardware systems, multiple tasks are grouped into one class so as to maintain a specific level of confidence in their accuracy. However, such a transformation is achieved at the expense of the loss of schedulability of the original task set. We further investigate the aforementioned problem and report the following contributions: (i) a novel technique for mapping unlimited priority tasks into a reduced number of classes that do not violate the schedulability of the original task set and (ii) an efficient feasibility test that eliminates insufficient points during the feasibility analysis. The theoretical correctness of both contributions is checked through formal verifications. Moreover, the experimental results reveal the superiority of our work over the existing feasibility tests by reducing the number of scheduling points that are needed otherwise.

Keywords: real-time systems, feasibility analysis, fixed-priority scheduling, rate monotonic algorithm, online scheduling.

1. Introduction

Real-time systems are usually defined as those where no deadlines for the scheduled tasks may be missing. The effectiveness of such systems depends on the logical results of the computations and also on the physical instant (deadline) at which these results are produced. Those systems can be classified according to various criteria specified as external (hard real-time versus soft real-time, and fail-safe versus fail-operational) and internal (guaranteed-timeliness versus best-effort, resource-adequate versus resource-inadequate, and event-triggered versus time-triggered) computing constraints and conditions (Kopetz, 1997). Based on the general characteristics of the application submitted to the system, real-time systems can be divided into two main categories, namely, *soft real-time systems* and *hard real-time systems*.

In soft real-time systems, the applications try to meet deadlines as much as possible, but lagging to meet the deadline does not cause any severe damages in the system. Typical examples of soft real-time systems are video and audio streaming and voice over Internet protocols (Hong and Leung, 1992). In hard real-time systems, all tasks *must* meet their deadlines (Burns and Wellings, 2009).

*Corresponding author

The examples of hard systems include air traffic control systems, nuclear power plants, various operating systems, etc.

Proper design of effective real-time scheduling algorithms can guarantee that each job meets its deadline, which is a crucial issue in hard real-time systems. A simple taxonomy of scheduling techniques in hard real-time systems is presented in Fig. 1.

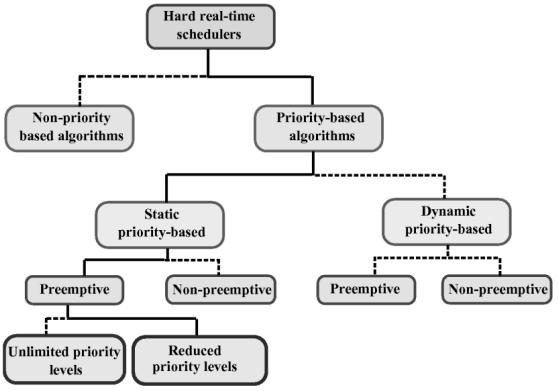


Fig. 1. Simple taxonomy of hard real-time systems.

A wide class of real-time scheduling algorithms in hard systems can be divided into two main categories, namely, *non-priority* methods, where no priority relations among tasks are specified, and *priority-based* ones (Lee *et al.*, 2003). The priority scheduling criteria of tasks can be saved and do not change over time. This scheduling scenario is called *static priority* scheduling (Xu and Parnas, 1990). In this case, changes in priorities are impossible when the system is running. The main goal of the *dynamic priority* scheduler is to maximize the CPU usage in the system. To achieve efficient system performance, the scheduler can change the priorities of the assigned tasks during system execution.

Each static and dynamic priority class of scheduling algorithms can be divided into two additional subcategories, namely, *preemptive* and *non-preemptive* methods. In preemptive scheduling, the currently executing task will be preempted upon the arrival of a higher priority task. In non-preemptive scheduling, the currently executing task will not be preempted until completion.

The most popular static priority-based preemptive algorithm for periodic tasks is *rate monotonic* (RM), developed by Liu and Layland (1973). In this algorithm, the priority of tasks are related to their periods. The main concept of the model proposed by Liu and Layland, along with the ideas behind many dynamic priority-based techniques, relies on the availability of a possible unlimited number of priority levels for the tasks, which rapidly increases the cost of scheduling and task executions. The reduction of the priority levels is therefore an important aspect for the system designer due to hardware cost and management considerations.

Most of the constraints imposed by the conventional RM algorithm have been successively weakened by adopting the RM framework to aperiodic tasks (Sha *et al.*, 1989), by introducing the synchronization procedure of the tasks based on semaphores (Sha and Goodenough, 1998), or by the implementation of deadline-monotonic algorithms, where the deadlines of the tasks are usually shorter than the tasks' periods (Leung and Whitehead, 1982). However, none of such techniques can effectively reduce the number of priority levels in the case of large batches of tasks submitted to the system.

Let us assume that n tasks in the system are classified according to their priority levels into n categories $\Omega_1, \ldots, \Omega_n$. If the number of tasks n is large, the complexity of the RM scheduling algorithm (in exact form) may be also very high. The reduction of the n tasks priority-based classes into the smaller set of k classes $\widehat{\Omega_1}, \ldots, \widehat{\Omega_k}$ $(k \leq n)$ requires the verification of the $(n-1)!/(k-1)!(n-k)!$ possible methods of the classification n tasks to be arranged into k priority classes. All of these issues will necessitate the development of effective mapping procedures for reducing the priority levels in the system and the complexity of the RM procedures.

The well-known illustrating examples of the realistic systems that offer a limited number of priority levels include IEEE 802.6 (with four priority levels), IEEE 802.5 (with eight priority levels), the MSI-C851 controller (with two priority levels (Bini *et al.*, 2001)), and other low cost devices, such as toys and wrist watches. In contrast to unlimited priority levels, this limited priority class has received little attention from researchers (Orozco *et al.*, 1998; Xuelian *et al.*, 2003; Sheng *et al.*, 2007). We further investigate limited priority levels (the classification is shown as a solid line in Fig. 1) and develop a generic framework that allows

- mapping the unlimited priority levels into a reduced number of classes,
- ensuring that the timing constraints of the new classes are in agreement with the original presented set of tasks,
- extending the exact feasibility analysis for a reduced number of priority levels,
- obtaining a faster RM feasibility test for determining the schedulability of the system with a reduced number of priority levels.

Our proposed solution has been theoretically and experimentally evaluated at various system utilization levels. Additionally, the associated schedulability

test has been provided. The complexity and the execution time of the proposed feasibility tests are much lower compared with the conventional feasibility tests for limited priority levels. Our model extends significantly the limited priority-levels methodologies developed by Ketcher *et al.* (1993) and Orozco *et al.* (1998) by taking RM-schedulable task sets into account for transformation into a lower number of priority levels. Though a lot of work has been done on reducing the feasibility analysis time of real-time systems with unlimited priority levels (Audsley *et al.*, 1993; Bini and Buttazzo, 2004; Lehoczky *et al.*, 1989; Bini *et al.*, 2001), to the best of our knowledge, we are the first to propose a faster feasibility tests based on scheduling points for a limited priority counterpart. The rest of the paper is organized as follows. Section 2 introduces notation and highlights related to the work. In Section 3, we define the essentials of the conventional RM scheduling. Section 4 presents the proposed reduction methodology for task priority levels. The exact feasibility test for a limited number of task classes is detailed in Section 5. The experimental results are presented in Sections 6. The paper is concluded in Section 7.

Table 1. Notation.

Notation	Meaning
Γ	Set of periodic tasks
τ_i	i-th task, $\tau_i \in \Gamma$
c_i	Worst case execution time of τ_i
p_i	Period of τ_i
d_i	Deadline of τ_i
u_i	Utilization of τ_i
n	Number of elements in Γ
$u(n)$	Utilization of Γ
T_i	Subset of tasks set having a priority higher than τ_i
k	Reduced number of priority levels
Ω_i	i-th class of tasks of the same priority level
$\widehat{\Omega_i}$	i-th class of metatasks
$J_{i,l}$	l-th job of τ_i
$r_{i,l}$	Release time of the l-th job of τ_i
t	Time instant
$W_i(t)$	Workload of τ_i at time t
f_i	Free slots in interval $[0, p_i]$
L_i	Cumulative workload of τ_i
L	Cumulative workload of the entire task set Γ
Z_i	Finite number of reference points of τ_i
P	Least common multiple of all periods $(p_1, \ldots, p_n)$
$\widehat{P_{i_s}^s}$	Period of the class
$\widehat{\Gamma}$	Set of metatasks
$\widehat{\tau_s}$	s-metatask
$Card(\widehat{\Omega_s})$	Cardinality of class $\widehat{\Omega}_s$
e^s	Execution demands of all tasks in class $\widehat{\Omega}_s$
X_s	Set of insufficient points set generated by $\widehat{\Omega}_s$

2. Background and related work

Before proceeding to a more detailed description, we first introduce notation (in Table 1) and shed some light on the working of RM scheduling.

Liu and Layland (1973) derived a sufficient feasibility condition for RM scheduling on a uniprocessor based on minimum achievable utilization. Necessary and sufficient (exact) feasibility conditions for uniprocessor RM scheduling were also studied by Lehoczky *et al.* (1989) as well as Joseph and Pandya (1986).

Lehoczky and Sha (1986) proposed a mapping procedure that consists of a logarithmic grid $g_1, g_2, \ldots, g_m$ where the ratio of two adjacent grid lines is kept constant. Tasks are assigned priorities based on their periods; i.e., τ_i with period p_i is assigned a priority g_i, when $g_{i-1} < p_i \leq g_i$. Orozco *et al.* (1998) showed that there exist certain partitionable sets that are declared unpartitionable by the aforementioned logarithmic grid methodology.

Another key issue is the problem of the identification of the best partition that would have the minimum loss in schedulability. Xuelian *et al.* (2003) and Sheng *et al.* (2007) provide a brute force technique to scan the entire search space for the best partition. Orozco *et al.* (1998) define the groups of tasks in order to generate multiple schedulable partitions (each partition has the same number of classes). Based on such partitions, according to the performance criteria proposed by Lehoczky and Sha (1986), the best partition is identified. Audsley (2001) addressed the problem of a minimum number of priority levels and proved that the feasibility order in the problem is the dominant part of the complexity, not the assignment order. The results were established for asynchronous tasks, and optimal priority assignment is made with response time analysis. Audsley (2001) pointed out that the work holds for the synchronous case, too. We further explore the work done by Audsley (2001) and how the number of priority levels can be reduced for the synchronous case with a scheduling points test.

In the literature, as observed by Audsley *et al.* (1993), Bini and Buttazzo (2004), Davis *et al.* (2008), Min-Allah *et al.* (2011) and Lehoczky *et al.* (1989), one can find sufficient work on scheduling tasks with unlimited priority levels. Davis *et al.* (2008) discussed feasibility tests, and a new direction of observing the feasibility analysis from the lowest priority first approach was explored. The existing literature can be divided into scheduling points tests and utilization based tests.

From the scheduling points domain, interesting results are derived (Audsley *et al.*, 1993; Bini and Buttazzo, 2004; Lehoczky *et al.*, 1989). Audsley *et al.* (1993) focused on assigning static priority to tasks and derived an efficient feasibility test. Bini and Buttazzo (2004) further extended the classic work done by Lehoczky *et al.* (1989) by restricting the scheduling point test to a subset of points. Recently, Min-Allah *et al.* (2007) derived another mechanism which avoids testing the schedulability of a task at insufficient points: the point where the feasibility of a higher priority task is negative. From utilization based perspective, some bounds have been derived recently. Bini *et al.* (2001) presented a hyperbolic bound which has a higher acceptance ratio than the LL-bound (Liu and Layland, 1973).

In contrast, only few researchers (Katcher *et al.* 1995; 1993; Orozzo *et al.*, 1998; Xuelian *et al.*, 2003) have reported work on scheduling tasks with limited priority levels. Ketcher *et al.* (1995) addressed the scheduling of a fixed priority system by restricting priority levels to a limited number. Orozco *et al.* (1998) discusses how to achieve a minimum level of priority levels. The aforementioned results are further extended by Xuelian *et al.* (2003) for a general task model.

3. Rate monotonic and deadline monotonic scheduling essentials

Rate-monotonic (RM) scheduling is a static priority based mechanism (Liu and Layland, 1973), where priorities of tasks are inversely proportional to the lengths of the task periods (task activation rates). This means that the task with the shortest period is assigned the highest priority. All tasks are executed in a preemptive manner.

Another variation of RM is deadline monotonic (DM), which is optimal for the case when the task deadlines are less than or equal to the task periods ($d_i \leq p_i$). The DM algorithm assigns priorities to tasks based on their deadlines: the shorter the deadline, the higher the priority, and vice versa. The RM and DM scheduling policies are identical when tasks deadlines are equal to their periods. In the rest of the paper, RM and DM can be used interchangeably.

The general characteristics of tasks and RM schedulability conditions are discussed in the remainder of this section.

3.1. General characteristics of tasks. Let us denote by $\Gamma = \{\tau_1, \ldots, \tau_n\}$ the set of independent periodic tasks. Each task τ_i $(i = 1, \ldots, n)$ is characterized by p_i (the value of the period of the task, which is expressed as a difference in time between any two consecutive instances or jobs of τ_i), c_i (the worst-case execution/computation time), d_i (the task relative deadline), and Ω_i (the class of tasks of the same priority level).

It is assumed that the deadlines of the tasks determine the periods of tasks. In the conventional RM scheduling method, it is assumed that there are n priority level classes $\Omega_1, \ldots, \Omega_n$ and each class contains just one task τ_i. Each task τ_i generates a job $J_{i,l}$ $(l = 1, 2, \ldots)$, where l is the number of periods of this task. The job becomes ready for execution on a uniprocessor system as soon as the job $J_{i,l}$ (l-th job of τ_i) is released at time $r_{i,l}$. It is also assumed that the related overhead, such as task swapping times, etc. are subsumed into c_i.

3.2. Schedulability condition. The schedulability criterion for the conventional RM algorithm is based upon the *critical instant* concept. *Critical instances* are defined as times at which all tasks are released simultaneously (Liu and Layland, 1973; Liu, 2000).

Definition 1. (*Critical instant*) In fixed priority, the scheduling of any task τ_i occurs when a job $J_{i,c}$ generated by the task τ_i is released simultaneously with the jobs generated by the tasks of higher priorities, that is, $r_{i,c} = r_{m,s(m)}$, where $m = 1, 2, \ldots, i-1$ and $s(m)$ is an indicator of the job $J_{m,s(m)}$ generated by the task τ_m.

It can be noted that the first critical instant occurs at time 0. At all other points, the workload is a non-increasing function at any time t as the task periods are not the same and hence subsequent jobs arrive after gaps of sizes equal to the task periods. Similarly, the interval $[0, p_1]$ is smaller thanor equal to the interval $[0, p_2]$, and $[0, p_2]$ is smaller than or equal to $[0, p_3]$, and so on. Let us denote by u_i an *utilization (load)* of the task τ_i that is expressed as $u_i = c_i/p_i$.

The cumulative utilization $u_{(tot)}$ of periodic tasks set Γ can be defined as the sum of u_i over all tasks:

$$u_{(tot)} = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{c_i}{p_i}. \tag{1}$$

The feasibility test *LL-bound* for the RM algorithm presented by Liu and Layland (1973) is defined as follows: A periodic tasks set is static-priority *feasible* if the condition below is satisfied:

$$u_{(tot)} \leq n(2^{1/n} - 1). \tag{2}$$

where n denotes the total number of tasks in Γ.

In Eqn. (2), the expression $n(2^{1/n} - 1)$ decreases monotonically from 0.83 (when $n = 2$) to $\ln(2)$ (as $n \to \infty$). It can be observed that if the task has utilization less than 69.3% it is guaranteed to be scheduled by the RM algorithm.

The condition defined in Eqn. (2) is a *sufficient* condition (SC) of the schedulability test, but it is not a *necessary* one. For an illustrating example, let as assume that there are two tasks τ_1 and τ_2, and $c_1 = 3$, $c_2 = 6$, $p_1 = 6$, $p_2 = 12$ stand for the worst-case execution

times and periods of these tasks, respectively. All times are expressed in arbitrary time units. The utilizations calculated for tasks τ_1 and τ_2 are $u_1 = u_2 = 0.5$, so the cumulative utilization is 1 (100%), which is greater than the allowable upper bound of 83%. However, when run, both tasks will meet their deadlines.

The theoretical work of Liu and Layland (1973) on RM schedulability was further extended by Han and Tyan (1997) as well as Min-Allah *et al.* (2010). However, in all those papers, the authors just tried to modify the sufficient condition presented in Eqn. (2). Numerous variants of RM schedulability constraints were proposed by Bini and Buttazzo (2004), Bini *et al.* (2008), Min-Allah *et al.* (2007), Lehoczky *et al.* (1989), Tindell *et al.* (1994), Audsley *et al.* (1993), Sjodin and Hansson (1998), and Davis *et al.* (2008) for determining RM feasibility analysis that generally falls into major classes: scheduling points tests (Lehoczky *et al.*, 1989; Bini *et al.*, 2008; Min-Allah *et al.*, 2007) and response time based tests (Audsley *et al.*, 1993; Joseph and Pandya, 1986; Davis *et al.*, 2008). The latter are superior over the former from analysis time perspectives, because in scheduling points tests feasibility is tested at all scheduling points for all tasks in the set, while iterative techniques have the advantage of making larger jumps in t that result in skipping a large number of scheduling points and hence the feasibility of a task is determined much early; however, scheduling point tests are considered to be the fundamental ones and can be used at the system design stage (Bini *et al.*, 2008), and hence are the focus of this work.

Let us denote by $T_i \in \Gamma$ the set of all priority tasks higher than τ_i. Let us also denote by J_i the first job generated by the task τ_i. It is assumed in this work that each task must generate at least one job in a given time slot, and J_i has the maximal response time compared with all other jobs generated by τ_i (Lehoczky *et al.*, 1989; Tindell *et al.*, 1994; Sjodin and Hansson, 1998)[1].

Let us assume that the critical instant occurs at time $t = 0$, and let W_i denote the maximum workload of the task τ_i at this critical instance. This means that all tasks can meet their deadlines if they are initialized (for execution) at the critical instant; then all these deadlines will be met during the lifetime of the system. The workload of τ_i at time t can be expressed as the sum of the task execution time demand c_i and all interferences and delays caused by the executions (and completion) of the higher priority tasks $\tau_{i-1}, \ldots, \tau_1$. Formally, the workload $W_i(t)$ can be defined as follows:

$$W_i(t) = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil c_j, \tag{3}$$

where p_j and c_j are the periods and worst-case execution times for the tasks $\tau_{i-1}, \ldots, \tau_1$, and $\lceil x \rceil$ signifies the smallest integer no less than x.

Definition 2. (*Sufficient and necessary condition of RM schedulability*) A periodic task τ_i is *feasible* if and only if the condition

$$L_i = \min_{0<t\leq p_i} (W_i(t) \leq t) \tag{4}$$

is satisfied for $t \in [0, p_i]$.

The main difference between the SC and SNC schedulability tests presented in Eqns. (2) and (4) lies in their computational complexities. The SC test is an $O(n)$ procedure in the number of tasks. The complexity of the SNC test is data dependent. This is because the number of calculations required is entirely dependent on the values of the task periods. Additionally, t is a continuous variable, so there are infinite numbers of reference points to be tested. Lehoczky *et al.* (1989) showed that $W_i(t)$ varies just at a finite number of such points[2]. Therefore, in order to determine whether τ_i is schedulable, we need to compute $W_i(t)$ only in the finite number of periods of τ_i of priority lower than τ_j $(1 \leq j \leq i)$, that is, on the set Z_i of parameters, where $Z_i = \left\{ a \cdot p_b \;\middle|\; b = 1, \ldots, i;\ a = 1, \ldots, \lfloor p_i/p_b \rfloor \right\}$ and $\lfloor x \rfloor$ evaluates to the largest integer no greater than x.

The SNC of RM schedulability can be defined in the following way (Lehoczky *et al.*, 1989; Orozco *et al.*, 1998).

Theorem 1. (Lehoczky *et al.*, 1989) *Each task $\tau_i \in \Gamma$ $(\Gamma = \{\tau_1, \ldots, \tau_n\})$ can be feasibly scheduled by using the RM scheduling algorithm if and only if the following condition is satisfied:*

$$L_i^* = \min_{t \in Z_i} \frac{W_i(t)}{t} \leq 1. \tag{5}$$

Theorem 2. (Lehoczky *et al.*, 1989) *The entire task set Γ is RM-schedulable if and only if*

$$L = \max_{1\leq i\leq n} L_i^* \leq 1. \tag{6}$$

4. Reduction of the priority levels in the system (mapping Ω_n into Ω_k)

In this section, we propose a solution for partitioning RM-schedulable task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with n priority levels into a lower number k $(k \leq n)$ of priority levels. The main aim of our method is to reduce the

[1]Maximal response time is defined as the maximal time needed for finalizing/finishing the job on a processor.

[2]These reference points are called *rate-monotonic scheduling* points (Lehoczky *et al.*, 1989).

number of priority classes of the tasks by using a simple uni-partitioning mapping. The tasks from the different priority classes are ordered by using the standard RM scheme, while the tasks inside the same class are ordered by using the FIFO method.

It is assumed that all tasks are initialized for the execution at the critical instant with the maximum workload. It is also assumed that each task τ_i is characterized by (i) its release time r_i and (ii) its deadline d_i ($d_i = p_i$).

Let us denote by $t_{\min}$ the minimal period (deadline) for the tasks in the set Γ. The following relation can be observed.

Observation 1. For the set of RM-ordered tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with periods $p_1, \ldots, p_n$, the relation $t_{\min} = p_1 = t_1$ holds.

Proof. The tasks from the set Γ are sorted in descending order by their priority levels. This means that task τ_1 has the highest priority (magnitude) over the rest of the tasks and it is executed first. For RM-ordered tasks, the following relation is satisfied (see Section 3):

$$[0, p_1] \subseteq [0, p_2], \cdots, \subseteq [0, p_n], \tag{7}$$

and $t_0 = 0$, which means that $p_1 \leq \ldots \leq p_n$ and therefore $t_{\min} = p_1 = t_1$. This completes the proof. ■

It follows from the above relation that, for the analysis of the RM schedulability of tasks in the interval $[0, t_1]$, we must ensure that either the release times of the tasks are no greater than t_1 or their respective deadlines are within $[0, t_1]$.

It can be also noted that for RM-ordered tasks the cumulative utilization $u_{(tot)}$ cannot be greater than 1, otherwise no scheduling algorithm can produce a feasible schedule with a uniprocessor system. The following remark can be then formulated.

Remark 1. Let $u_{(0,p_i)}$ represent system utilization in the period $[0, p_i]$. The whole system would be deemed infeasible when $u_{(0,p_i)} > 1$. At critical instant $t = 0$, for any RM-feasible schedule, the following relation holds:

$$u_{(0,p_1)} \geq u_{(0,p_2)}, \cdots, \geq u_{(0,p_n)}, \tag{8}$$

where $u_{(0,p_1)}$ denotes system utilization in interval $[0, p_1]$, $u_{(0,p_2)}$ denotes system utilization in interval $[0, p_2]$, and so on.

Remark 1 is a consequence of the fact that the RM algorithm does not leave any processor idle when there is a task that waits for execution. The relations (7) and (8) will be used for the adaptation of the empty-slot method to RM scheduling in the following section.

4.1. Empty-slot method adapted to RM scheduling.

One of the most popular methodologies of scheduling in real-time systems is the empty-slot method (Santos *et al.*, 1991). It is assumed in this model that any given period of time can be divided into some time units called *slots*. The estimation of the remaining free time slots after the scheduling of a given batch of periodic tasks is needed for the re-balancing of workload of the processors in the system and further re-classification of the tasks into a lower number of priority level groups.

Let us assume that each task $\tau_i \in \Gamma$ has priority i, period p_i, is released at $t_0 = 0$, and is RM-schedulable within the period $[0, p_i]$. The following proposition defines the amount of free time slots for scheduling the remaining lower priorities tasks.

Proposition 1. *There are at least*

$$f_i = p_i - \sum_{j=1}^{i} \left\lceil \frac{p_i}{p_j} \right\rceil c_j$$

free slots that can be utilized by tasks of a priority lower than τ_i.

Proof. All tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have priorities higher than τ_i and are RM-schedulable within the interval $[0, p_i]$. Task τ_i can only be scheduled when there are no $i - 1$ pending higher priority tasks within the system. Due to the RM-schedulability feature, the periods of all tasks of a priority higher than τ_i are no greater than p_i, that is, $p_1 \leq p_2 \leq, \ldots, \leq p_i$. Therefore, in the worst case, for a given time window $[0, p_i]$, there could be $\sum_{j=1}^{i-1} \lceil p_i/p_j \rceil$ instants of higher priority tasks that are executed in $\sum_{j=1}^{i-1} \lceil p_i/p_j \rceil c_j$ time intervals, where c_j denotes the worst execution time for the task τ_j. Therefore, the intervals left for task τ_i could be at least $f_i = p_i - \sum_{j=1}^{i-1} \lceil p_i/p_j \rceil c_j$. ■

The above proposition can be concluded as follows.

Corollary 1. *The task τ_i is RM-schedulable along with all of the higher priority tasks when*

$$c_i = p_i - \sum_{j=1}^{i-1} \left\lceil \frac{p_i}{p_j} \right\rceil c_j. \tag{9}$$

In the case of $d_i = p_i$, most of the workload generated by the task τ_i is realized in the interval $[0, p_i]$. When $d_i \geq p_i$, task τ_i can be executed for more than one period p_i. In such a case, the following remark can be formulated.

Remark 2. The cumulative workload constituted by τ_i in any interval $[(n-1)p_i, np_i]$ is always less than (when $p_i \neq p_j | i \neq j, \forall i, j : 1 \leq i, j \leq n$) or equal to (when $p_i = p_j | i \neq j, \forall i, j : 1 \leq i, j \leq n$) the cumulative workload constituted in $[0, p_i]$.

It can be observed that for the n time intervals $[0, p_1], [p_1, 2p_1], \ldots, [(n-1)p_1, np_1]$, in the worst case, for the tasks initialized at $t = 0$ each task encounters the maximum workload in first interval.

For the whole system, the aforementioned proposition and remark can be generalized in the following way. Let us denote by P the least common multiple of all periods $p_1, \ldots, p_n$. Following the notation and terminology introduced by Orozco *et al.* (1998), the system is said to be *non-saturated* if $W_n(P) \leq P$, where $W_n(P)$ denotes the cumulative workload of task τ_n at time P (see Eqn. (3)). This means that in the period $[0, P]$ there are $P - W_n(P)$ empty slots in the system of n tasks. In terms of system saturation, the sufficient and necessary condition for RM schedulability is the non-saturation condition for the system of $n-1$ tasks, that is, $\{\tau_1, \ldots, \tau_{n-1}\}$, and the period of task τ_n must be no shorter than the empty slot c_n calculated as in Eq. (9). Remarks 1 and 2 allow us to group tasks into a single class in the following section. A dummy period is associated with each class and all tasks whose timing constraints are satisfied with the dummy period are eligible to be part of the same class.

4.2. Priority levels reduction procedure. In this section we define the procedure of the reduction of n priority tasks $\tau_1, \ldots, \tau_n$ into a lower number of k classes $\widehat{\Omega}_1, \ldots, \widehat{\Omega}_k$ ($k \leq n$). Usually, in the conventional RM scheduling, it is assumed that each class Ω_i contains just one task τ_i ($i = 1, \ldots, n$), and that is the reason for the number n of priority levels needed. In the case of the reduction of the number of priority levels, at least one of the classes $\widehat{\Omega}_s$ ($s \in \{1, \ldots, k\}$) must contain more than one task. A sequence of tasks in such a class is defined as a *metatask*. Let us denote by $\widehat{\Gamma} = \widehat{\tau}_1, \ldots, \widehat{\tau}_k$ the set of k metatasks in the system, which is a result of the reduction of priority levels. We will call this system *a partition of the tasks,* and the mapping

$$\Theta : \Omega_1 \times \ldots \Omega_n \ni (\tau_1, \ldots, \tau_n) \longrightarrow (\widehat{\tau}_1, \ldots, \widehat{\tau}_k) \in \widehat{\Omega}_1 \times \ldots \times \widehat{\Omega}_k. \quad (10)$$

denotes the *priority levels reduction mapping* in the $\Gamma = \{\tau_1, \ldots, \tau_n\}$ system.

Each metatask $\widehat{\tau}_s$ ($s = 1, \ldots, k$) represents a sequence of tasks $\tau^s_{i_1}, \ldots, \tau^s_{i_s}$ from the original set Γ that are elements of the class $\widehat{\Omega}_s$. The following two scheduling procedures should be considered for the whole system:

- *'external' scheduling*: the conventional RM scheduling of the metatasks $\widehat{\tau}_1, \ldots, \widehat{\tau}_k$ means that the priorities of all tasks from $\widehat{\Omega}_1$ are higher than the priority of any task from $\widehat{\Omega}_2$, the priorities of all tasks from $\widehat{\Omega}_2$ are higher than the priority of any task from $\widehat{\Omega}_3$, and so on; and
- *'internal' scheduling*: this is the scheduling method inside each class $\widehat{\Omega}_s$ ($s = 1, \ldots, k$); we propose the FIFO (first-in first-out) methodology for this scheduling.

Let as assume that metatask $\widehat{\tau}_s$ in $\widehat{\Omega}_s$ is defined as a sequence of tasks $\tau^s_{i_1}, \ldots, \tau^s_{i_s}$. The priority of the metatask $\widehat{\tau}_s$ is defined as follows:

$$priority\,(\widehat{\tau}_s) = \max_{l=i_1}^{i_s} priority\,(\tau^s_l). \quad (11)$$

The assumption that the 'external' scheduling procedure is realized according to the conventional RM scenario is equivalent to the relation $priority\,(\widehat{\tau}_i) > \ldots > priority\,(\widehat{\tau}_k)$ of the priorities of metatasks. The priority level reduction (partitioning) procedure is therefore based on simple clustering of the tasks in the initial set Γ.

It should be noted that the reduction in the priority levels does not change the total number of tasks in the system, i.e.,

$$n = \sum_{s=1}^{k} card\left(\widehat{\Omega}_s\right), \quad (12)$$

where $card\left(\widehat{\Omega}_s\right)$ denotes the cardinality (the number of tasks) of the class $\widehat{\Omega}_s$.

The tasks $\tau^s_{i_1}, \ldots, \tau^s_{i_s}$ can be selected for the class $\widehat{\Omega}_s$ according to the following criterion:

$$\widehat{P^s}_{i_s} \leq c_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil c_j, \quad (13)$$

where $\widehat{P^s}_{i_s}$ denotes *the period of the class* parameter, which is interpreted as the largest period of tasks τ_i in the class $\widehat{\Omega_s}$. In Eqn. (13), t comes from the set of scheduling points constituted by all higher priority tasks that are being grouped and never exceeds the largest period of the class. Equation (13) defines a cumulative workload of tasks with priority i or higher that are already grouped into class $\widehat{\Omega}_s$. This demand must be fulfilled at or before $\widehat{P^s}_{i_s}$. The procedure of fulfilling the classes of metatasks starts with the selection of the task τ_1 of the highest priority and accommodates as many tasks as possible in class $\widehat{\Omega}_1$ till the saturation level is achieved. Equation (13) shows that any task with the period not greater than $\widehat{P}_s$ with all its iterations completed at or before $\widehat{P}_s$ is eligible to be a member of the class $\widehat{\Omega}_s$. Therefore, τ_i is interpreted as the lowest priority task that is schedulable in class $\widehat{\Omega}_s$. In other words, tasks τ_{i+1} to τ_n are now candidates to be grouped into class $\widehat{\Omega}_{s+1}$, and so on. This process is executed till all tasks are mapped onto qualifying classes.

The aforementioned discussion may lead to the following observation: Arranging i-tasks $(\tau_1, \cdots, \tau_i)$

with $\widehat{\tau}_i$ would mean that the total $\widehat{u}_i$ is very high and approaching 1 in the interval $[0, \widehat{p}_i]$. An interesting scenario develops here: How could the lower priority tasks $\tau_{i+1}, \ldots, \tau_n$ be scheduled in the presence of $\widehat{\tau}_i$?

In order to explain the scenarios further, we discuss the system from the critical instant point of view (the worst-case scenario). Let the metatask $\widehat{\tau}_1$ cover i tasks. Therefore, the sufficient and necessary condition must be $c_1 + c_2 + \cdots + c_i \leq \widehat{p}_1$. Moreover, at $t = 0$, the cumulative computation demand must be equal to $c_1 + c_2 + \cdots + c_n$. Since the first instants of jobs of tasks from τ_1 to τ_i are executed in interval $[0, \widehat{p}_1]$, their load needs to be subtracted from the total workload presented at $t = 0$. However, another job of task τ_1 is also available at point $\widehat{p}_1$ as it is the task period of $\widehat{\tau}_1$, and hence care should be taken of another c_1, i.e., $2c_1$ in total. In short, out of cumulative computational demands, the term $c_1 + c_2 + \ldots + c_i$ must be executed in $[0, \widehat{p}_1]$. Therefore, the pending workload at $t = \widehat{p}_1$ becomes

$$(c_1 + c_2 +, \cdots + c_n) - (c_1 + c_2 + \ldots + c_i) + c_1 = (c_1 + c_{i+1} + \cdots + c_n). \quad (14)$$

Eventually, all of the i-tasks are scheduled and $W_i(t) \leq t$ is satisfied at or before its deadline. Similarly, from the utilization perspective, the original task set is feasible if the achievable utilization in $[0, p_i]$ is either $i(2^{1/i} - 1)$ (as reported by Liu and Layland (1973)) or at the maximum utilization of 1.0 (as reported by Han and Tyan (1997)). The utilization of such systems can be represented by

$$\frac{c_1}{p_1} + \frac{c_2}{p_2} + \cdots + \frac{c_i}{p_i} \leq i\,(2^{1/i} - 1) \leq 1.$$

Multiplying this equation by $p_1 \times p_2 \cdots \times p_i$, we get

$$\sum_{j=1}^{i} c_j \prod_{l=1}^{i} \frac{p_l}{p_j} \leq\ i\,(2^{1/i} - 1) \prod_{j=1}^{i} p_j \leq \prod_{j=1}^{i} p_j. \quad (15)$$

Such arrangements always keep the total utilization of tasks unchanged and accommodate lower tasks $\tau_{i+1}, \cdots, \tau_n$ to be executed in free intervals, i.e., when there are no pending higher priority tasks. All those higher priority tasks that are waiting for CPU slots at a particular instant of time are termed pending tasks. Equation (15) is purely based on the utilization bounds; however, we are specifically interested in the development of an exact test.

With above formulation, when the priority levels are less than the available ones ($\Omega_n > \widehat{\Omega}_k$), then Γ can easily be divided into groups $\widehat{\Omega}_1, \widehat{\Omega}_2, \cdots, \widehat{\Omega}_k$. Let $card(\widehat{\Omega}_s)$ denote the number of tasks grouped into class s, such that every group has at least one task, i.e., $\widehat{\Omega}_s \neq \emptyset, \forall s : 1 \leq s \leq k$. Consequently, we can write $\sum_{s=1}^{k} card(\widehat{\Omega}_s)$. The terms "sum of tasks belong to an individual group" and "actual number of tasks in a group" represent the same concept. Therefore, these terms as well as class and "metatask" can be used interchangeably in the rest of this paper.

In order to find out $\widehat{\Omega}_k$, all the tasks in Γ need to be analyzed that are keyed by priority and listed in descending order. Therefore, there are $(n - i + 1)$ such potential candidates to be grouped into classes. As already known for the unlimited priority case, a task τ_1 is always feasible when $c_1 \leq \widehat{p^1}_1$. Similarly, it can be derived that $\widehat{\Omega}_s(s = 1)$ is also schedulable when $\sum_{h=1}^{i} e_h^1 \leq \widehat{p^1}_1$, where e_h^1 is the execution demand of all of the tasks of the priorities in class $\widehat{\Omega}_1$.

The mapping (partitioning) technique can be implemented as an iterative procedure that can proportionate load into different classes. In order to illustrate this phenomenon, let us examine the first task $\tau_1 \in \Gamma$. Because $s = 1$, the term e_i^s will be equal to c_1.

As mentioned above, the term e_h^1 denotes the execution demands of all the tasks in $\widehat{\Omega}_1$ that have a priority lower than or equal to τ_h. If τ_2 is also schedulable under $\widehat{\Omega}_s(s = 1)$, then the term e_i^s becomes $c_1 + c_2$. As more and more tasks are added to $\widehat{\Omega}_s$, the term e_i^s increases proportionally. Eventually, when the maximum number of tasks is grouped into $\widehat{\Omega}_s$, the proposed mapping technique will accommodate the remaining tasks into $\widehat{\Omega}_{s+1}$.

Let τ_{i+1} be the first task to be arranged in $\widehat{\Omega}_{s+1}$. The timing constraints of the current task τ_{i+1} and any task having a priority higher than τ_{i+1} must be fulfilled prior to $\widehat{P}_{i_{s+1}}^{s+1}$. Therefore, the following relation must hold:

$$W_{|\widehat{\Omega}_s|}(t) + e_{i+1}^{s+1} \leq \widehat{P}_{i_{s+1}}^{s+1}, \quad (16)$$

where $W_{|\widehat{\Omega}_s|}(t)$ is the maximum workload presented by the last task $\tau_i \in \widehat{\Omega}_s$ at time t. The workload is the maximum because τ_i has the lowest natural priority among all of the tasks grouped into Ω_s. On the other hand, any other arrangement will result in feasibility loss. Moreover, the aforementioned technique identifies a reduced number of priority levels $|\widehat{\Omega}_k| \leq |\Omega_n|$.

Once the groups of tasks are identified by the proposed mapping technique to be treated as one class, we need to ensure the timing constraints of all the tasks in the class in which the tasks are grouped. To keep the timing constraints of the original task set intact, tasks in a class are ordered according to RM priority; i.e, $priority(\widehat{\Omega}_1)$ is higher than $priority(\widehat{\Omega}_2)$ and $priority(\widehat{\Omega}_2)$ is higher than $priority(\widehat{\Omega}_3)$, and so on. Each group $\widehat{\Omega}_h = \{\tau_i, \tau_j\}$ covers at most $|\widehat{\Omega_h}|$ number of tasks. These tasks can be scheduled on an FIFO basis. It is understood that under FIFO implementation a task τ_i of a priority greater than τ_j may be delayed by a task τ_j that arrives before τ_i, though the priority of τ_i is higher as per the original task set. However, this anomaly is rectified by the fact that task

τ_i also avoids preemption due to the higher priority tasks (τ_i); i.e., all tasks within a group have the same priority. It is worth mentioning that FIFO has an associated run-time overhead but for simplicity we assume this overhead to be subsumed into task computation times. The pseudo-code of the procedure of mapping the n priority classes $\widehat{\Omega}_n, \ldots, \widehat{\Omega}_1$ of tasks $\tau_n, \ldots, \tau_1$ into k classes $\widehat{\Omega}_k, \ldots \widehat{\Omega}_1$ ($n \geq k$) is presented as Algorithm 1.

5. Schedulability test for reduced priority levels

The reduction of task priority levels in the system requires updating the RM schedulability tests specified in Theorems 1 and 2 (see Section 3.2). For systems with the k levels $k \leq n$, Theorems 1 and 2 can be reformulated and merged into the following criterion (Orozco *et al.*, 1998).

Theorem 3. *Let us consider a periodic task set* $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *, where tasks are ordered based on their respective priority levels. Let us assume that the tasks from the set* Γ *are classified (by using some priority scheduling algorithm) into the following* k *priority categories (*k *priority classes):* $\widehat{\Omega}_1, \widehat{\Omega}_2, \ldots, \widehat{\Omega}_k$*. Each task* $\tau_i \in \widehat{\Omega}_s$ $(s \in \{1, \ldots, k\})$*, for which the deadline* $d_i \leq p_i$*, will be executed within its deadline if and only if*

$$\min_{0<t\leq d_i} \frac{W_i(t)}{t} \leq 1,$$

where

$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j + \sum_{l \in \widehat{\Omega}_s} c_l.$$

To test the feasibility of τ_i, the workload $W_i(t)$ must be evaluated for all of the tasks grouped into $\widehat{\Omega}_s$. It can be seen that the number of points for which the condition defined in Theorem 3 must be verified depends on the number of elements of set Z_i ($t \in Z_i$). This set may be very large for big values of the ratio p_n/p_1.

To improve the feasibility test defined in Theorem 3, we introduce the *insufficient point scheduling* mechanism that can be specified as follows.

Definition 3. (*Insufficient point*) The RM scheduling point t is called *an insufficient point* for any class $\widehat{\Omega}_s$ of tasks with the priority s if for any task $\tau_i \in \widehat{\Omega}_s$ the inequality constraint $W_i(t) > t$ holds (Min-Allah *et al.*, 2007).

Definition 4. (*Insufficient points set*) An insufficient point t for the class $\widehat{\Omega}_s$ is also an insufficient point for all classes of tasks of priorities lower than s. The set of all insufficient points for the classes $\widehat{\Omega}_s, \widehat{\Omega}_{s+1}, \ldots \widehat{\Omega}_k$ is called *the insufficient points set generated by* $\widehat{\Omega}_s$ and is denoted by X_s.

Algorithm 1. Reduction of n priority classes into k priority classes ($n \geq k$).

```
1: procedure reduced-priority-levels (τ_n)
2: m := 1
3: Ω̂ := ∅
4: current − length := 0 {Mechanism for placing tasks into classes}
5: for all s := 1 to n do
6:    e^s := 0
7:    for all τ_i := τ_m to τ_n do
8:       e^s := e^s + c_i
9:       schedulable:= schedulability-test (Ω̂_0,τ_i,group − period−Ω̂_s,current − length,e^s) {Place a task in the class if it does not hurt the feasibility of the class}
10:      if (schedulable is TRUE) then
11:         Ω̂_s. add(τ_i)
12:         m := m + 1 {Shift task to another class}
13:      else
14:         current − length := current − length + |Ω̂_s| + 1
15:         |Ω̂_s| := 0
16:         break
17:      end if
18:   end for
19:   Terminate if τ_i is the last task
20: end for
21: end procedure
```

```
1: Boolean function schedulability test(Ω̂_s,τ_i,group − period − Ω̂_s,current − length,e^s)
2: if (Ω̂_s := 1) && (group − period − Ω̂_s ≥ e^s) then
3:    τ_i is schedulable in group Ω̂_s
4: end if
5: compute Z_i = { a · p_b | b = 1, ..., i;  a = 1, ..., ⌊p_i/p_b⌋ }
6: for all t ∈ Z_i do
7:    if ∃ t ≥ (W_i(t) + e^s) then
8:       τ_i is schedulable in Ω̂_s
9:       return TRUE
10:   end if
11: end for
12: if t < (W_i(t) + c^{Ω̂_s}, ∀t ∈ Z_i) then
13:   τ_i is infeasible in Ω̂_s
14:   return FALSE
15: end if
16: end function
```

Based on the concepts of an insufficient point and insufficient points set, the following theorems define

efficient schedulability tests for a reduced number of priority levels of tasks in Γ.

Theorem 4. *Let us assume that n periodic tasks from the set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ are classified into k priority level classes ($k \leq n$). Each insufficient point for $\widehat{\Omega}_s$ ($s \leq k$) must also be the insufficient point for $\widehat{\Omega}_{s+1}$.*

Proof. Let $\tau_{i+1} \in \widehat{\Omega}_{s+1}$ and t be the insufficient point for $\widehat{\Omega}_s$. The workflow of τ_{i+1} at the time t can be calculated as follows:

$$\begin{aligned} W_{i+1}(t) &= \sum_{j=1}^{i+1} \left\lceil \frac{t}{p_j} \right\rceil c_j + \sum_{l \in \widehat{\Omega}_{s+1}} c_l \\ &= \sum_{j=1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j + \sum_{l} c_j + \sum_{l \in \widehat{\Omega}_s} c_l \\ &\quad + \left\lceil \frac{t}{p_{i+1}} \right\rceil c_{i+1} + c_{l+1} \\ &= W_i(t) + \left\lceil \frac{t}{p_{i+1}} \right\rceil c_{i+1} + c_{l+1}. \end{aligned} \tag{17}$$

It can be noted that the values of $\lceil t/p_{i+1} \rceil \, c_{i+1}$ and c_{l+1} are positive. Therefore, $W_{i+1}(t) > W_i(t)$, and thus t is also an insufficient point for $\tau_{i+1} \in \widehat{\Omega}_{s+1}$. ■

Theorem 5. *Consider an RM-ordered group of priority level classes $\widehat{\Omega}_1, \ldots, \widehat{\Omega}_k$. Task $\tau_i \in \widehat{\Omega}_s$ ($s \leq k$),*

$$W_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j + \sum_{\forall l \in \widehat{\Omega_s}} c_l,$$

$$L_i = \min_t \frac{W_i(t)}{t} \leq 1$$

where $t \in Z_i \backslash X_{s-1} = \{t : t \in Z_i \text{ and } t \notin X_{s-1}\}$. By extension, $X_0 = \emptyset$.

Proof. It follows directly from Theorem 4; i.e., if t is an insufficient point for $\widehat{\Omega}_{s-1}$, then t is also an insufficient point for $\widehat{\Omega}_s$. ■

In the RM schedulability test defined in Theorem 5, the search space is reduced to the set $Z_i \backslash X_{s-1}$, which is the main contribution of this section.

6. Results and analysis

In this section we present the results of experimental evaluation of the proposed priority level reduction procedure and the improved feasibility test. The platform used for these experiments consisted of 12 cores Intel Xeon X5650 2.67 GHz CPU (12MB L3-cache) and 48 GB memory.

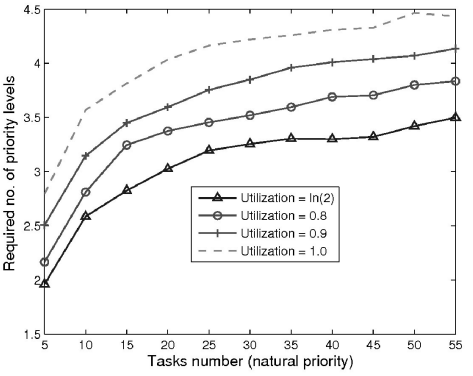


Fig. 2. Required number of priority levels for tasks from the set Γ for various system utilizations.

6.1. Priority tansformation process under various system utilization. First, we study the effect of cumulative system utilization on the mapping (or conversion) procedure. The tasks for the set Γ have been generated randomly. The periods of tasks were generated by using the uniform probability distribution over the interval of $[10; 10000]$. The corresponding task execution demands were randomly generated from the intervals $[1, p_i]$. For the task set, the task priorities were assigned according to the RM scheduling algorithm, though a workload generation procedure was also suggested by Bini and Buttazzo (2004) or Min-Allah *et al.* (2010).

A series of periodic tasks were generated by varying the range from 5 to 55 with an increment of 5 tasks. Results were established by taking the average after the first 300 iterations. Figure 2 depicts the required (minimal) number of priority levels for the tasks from the set Γ in the cases of various cumulative utilizations of the system.

We considered four utilization values, namely, $\ln(2)$, 0.8, 0.9, and 1. The number of the required priority levels increases as $n \to \infty$. From Fig. 2, the trend in the required priority levels is fast as long as the number of tasks is less than 15 and stabilizes (is very slow) for the number of tasks greater than 25. This trend is aligned with the existing literature (Liu, 2000). In such cases, more tasks must be classified into a single class with the inequality (13). Another simple observation is that the computation demands of individual tasks increase with higher utilization and fewer tasks can be grouped into a single meta task. It can be seen that utilization plays a very important role in the priority levels needed, as reflected by term computation times c in the inequality (13), where time periods p remain fixed but c do change/increase with higher system utilization, and vice-versa. For utilization $\ln(2)$ it needs 2 levels of are needed for 5 tasks, while 2.8 priority levels for 5 tasks when utilization is 1, i.e.,

100%. This is due to the fact that c_i of a task is higher when system utilization is 1, and hence it is difficult to group many tasks. Task periods for the tasks are obtained with the aforementioned task set criteria, irrespective of utilization, so accommodating few tasks in the case of higher system utilization is understandable. In other words, tasks require more priority levels when utilization is higher.

In all the scenarios considered, the required number of priority levels varies from 60% (in the case of 5 tasks) to 80% (in the case of 55 tasks) of the total number of tasks in the system. This confirms that the proposed methodology significantly reduces the complexity of the RM scheduling algorithm.

6.2. Evaluation of the improved feasibility test. The main aim of the analysis presented in this subsection is to compare the effectiveness of the *improved feasibility test* (IFT) defined in Section 5 and the *traditional approach* (TA) (see Theorem 3 of Orozco *et al.* (1998)). The general settings of this experiment are the same as discussed in Section 6.1. Both techniques, the IFT and TA, are compared under the required number of reference scheduling points criteria.

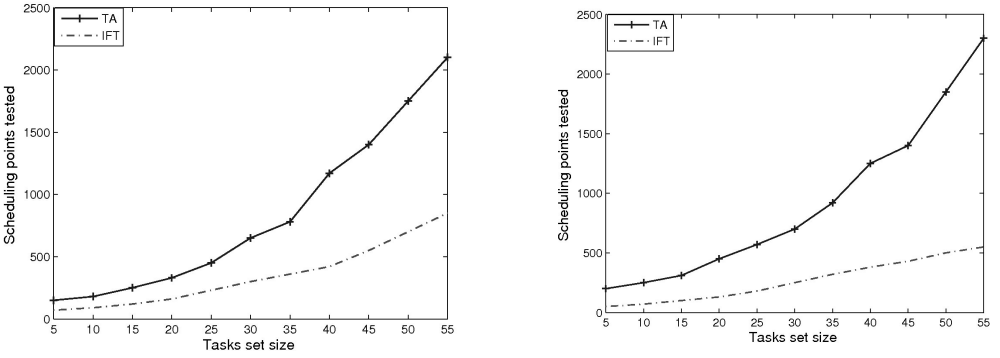
In this experiment, a series of periodic tasks were generated by varying the range from 5 to 55 with an increment of 5 tasks, and results were drawn by taking the average after the first 300 iterations. It is worth mentioning that, when $u_{(tot)} \leq \ln(2)$, there is no point in applying exact tests, as a solution does exist (Liu and Layland, 1973; Bini *et al.*, 2001) that answers the feasibility of such a system much earlier because of its $O(n)$ nature. Similarly, at a higher utilization, the system becomes infeasible (Laplante *et al.*, 2004). In the light of the aforementioned reasons, we keep the cumulative utilization of the system in the range of $\ln(2)$ to 0.85. The reason for taking such system utilization is to make sure that only feasible task sets are generated and hence all tasks' feasibility is determined.

It can be seen from Figs. 3(a) and (b) that more points are needed by both the TA and IFT. A rationale behind this trend is that system utilization is low and hence, under given task set generation criteria, the computation times of the individual tasks are low while the periods are in a fixed range of $[10, 10000]$. Consequently, many tasks get scheduled by the first scheduling point. For example, the starting point for testing the feasibility of task τ_1 is p_1 and c_1 is always less than p_1, and hence τ_1 is schedulable. The same scheduling point p_1 is also the first point to be tested in the feasibility of τ_2, and so on. Here p_1 remains the first scheduling point at which the schedulability of the remaining low priority tasks will be analyzed. Similarly, when p_1 does not satisfy the requirements of a task, say τ_{i+1}, which misses the deadline at p_1, p_1 is identified as an insufficient point (see Definition 3 for details) and the feasibility test is run on the subsequent points of the scheduling points set. However, for the same task sets, the number of points needed for determining schedulability is low in Fig. 3(b) for the IFT. The reason behind this behavior is that the workload is high at a utilization of 0.85 and hence c_i is high, which means that investigation of more points is needs. The trend for the *TA* is high as now it needs to test more points, while testing feasibility at more points helps in declaring more insufficient points for the *IFT* and hence these points can be skipped during the analysis of lower priority tasks.

The number of testing points increases as $n \to \infty$. This is because of the larger values of the ratio p_n/p_1 for large n. However, in both the cases of small and large cumulative utilization levels, the increase in the number of the scheduling points requested by the IFT procedure is very small compared with the TA procedure. This is because of the small accumulation of c_i for the task τ_i in the IFT test. As a consequence, the total utilization $u_{(tot)}$ is a smaller and many tasks can be accommodated into a single class, as computation demands c possesses a smaller value with lower utilization, while the rest of the parameters are fixed (see Theorem 4). Therefore, a sufficiently small number of points is needed to be tested for $\widehat{\Omega}_s$. The advantage of IFT over the TA is significant when more tasks are deemed feasible within the system. In Figs. 3(a) and (b), the IFT technique supersedes the TA (in performance) when the total utilization is equal to 0.85. In this case, the computation demand of the individual task τ_i is quite high. Therefore, more points are scanned before the feasibility is concluded, which is again in the interest of the IFT; i.e., the set of insufficient points for $\widehat{\Omega}_{s-1}$ is large.

7. Conclusion and future work

This paper investigated the effectiveness of the RM scheduling algorithm in the reduction of the priority levels of tasks scheduled under RM in preemptive mode. The transformation procedure from the set of n classes to the reduced set of k classes ($k \leq n$) was formally defined and explained. This theoretical model was also experimentally evaluated at four different levels of cumulative utilization of the system. In addition, the unnecessary testing of the feasibility of a limited priority system was avoided by identifying insufficient points. The results achieved in the experiments confirm those of the theoretical analysis and show high effectiveness of the proposed method in reducing the complexity of the conventional RM scheduling algorithm for limited priority levels. A comparison of the improved feasibility test procedure with the most effective known conventional feasibility technique (based on scheduling points techniques) revealed the advantage of the proposed methodology over the existing counterpart.



(a) (b)

Fig. 3. Advantage of the IFT over the TA in terms of reducing the number of scheduling points tested: $u_{(tot)} \leq \ln(2)$ (a), $u_{(tot)} \leq 0.85$ (b).

As a future work, the proposed technique can be extended to different task parameters using various scheduling algorithms with more relaxed assumptions.

Acknowledgment

The authors would like to thank the members of the Supercomputing Technologies Group at the CSAIL, Massachusetts Institute of Technology (MIT), USA, for their helpful discussions.

References

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

Audsley, N.C. (2001). On priority assignment in fixed priority scheduling, *Information Processing Letters* **79**(1): 39–44.

Bini, E., Buttazzo, G.C. and Buttazzo, G. (2001). A hyperbolic bound for the rate monotonic algorithm, *Proceedings of the 13th Euromicro Conference on Real-Time Systems, Washington, DC, USA*, pp.59–66.

Bini, E. and Buttazzo, G.C. (2004). Schedulability analysis of periodic fixed priority systems, *IEEE Transactions on Computers* **53**(11): 1462–1473.

Bini, E., Natale, M.D. and Buttazzo, G. (2008). Sensitivity analysis for fixed-priority real-time systems, *Real-Time Systems* **39**(1–3): 5–30.

Burns, A. and Wellings, A.J. (2009). *Real-Time Systems and Programming Languages*, 4th Edn., Addison Wesley, Longmain.

Davis, R.I., Zabos, A. and Burns, A. (2008). Efficient exact schedulability tests for fixed priority real-time systems, *IEEE Transactions on Computers* **57**(9): 1261–1276.

Han, C.C. and Tyan, H.Y. (1997). A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms, *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), Washington, DC, USA*, p. 36.

Hong, K.S. and Leung, J.Y.-T. (1992). On-line scheduling of real-time tasks, *IEEE Transactions on Computers* **41**(10):1326–1331.

Joseph, M. and Pandya, P.K. (1986). Finding response times in a real-time system, *The Computer Journal* **29**(5): 390–395.

Katcher, D.I., Arakawa, H. and Strosnider, J.K. (1993). Engineering and analysis of fixed priority schedulers, *IEEE Transactions on Software Engineering* **19**(9): 920–934.

Katcher, D.I., Sathaye, S.S. and Strosnider, J.K. (1995). Fixed priority scheduling with limited priority levels, *IEEE Transactions on Computers* **44**(9): 1140–1144.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications,* Kluwer Academic Publishers, Norwell, MA.

Laplante, P.A., Kartalopoulos, S.V., Akay,M., El-Hawary, M.E., Periera, F. M.B., Anderson, J. B., Leonardi, R., Singh, C., Baker,R.J., Montrose, M., Tewksbury, S., Brewer, J.E., Newman, M.S. and Zobrist, G. (2004). *Real-Time Systems Design and Analysis, 3rd Edition*, John Wiley and Sons, Hoboken, NJ.

Lee, W. Y., Hong, S. J. and Kim, J. (2003). On-line scheduling of scalable real-time tasks on multiprocessor systems, *Journal of Parallel and Distributed Computing* **63**(12): 1315–1324.

Lehoczky, J.P. and Sha, L. (1986). Performance of real-time bus scheduling algorithms, *'86 ACM SIGMETRICS Joint International Conference on Computer Performance Modeling, Measurement and Evaluation, New York, NY, USA*, pp. 44–53.

Lehoczky, J.P., Sha, L. and Ding, Y. (1989). The rate monotonic scheduling algorithm: Exact characterization and average case behavior, *Proceedings of the IEEE Real-Time Systems Symposium, Santa Monica, CA, USA*, pp. 166–171.

Leung, J.Y.T. and Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, *Performance Evaluation* **2**(4): 237–250.

Liu, C.L. and Layland, J.W. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment, *Journal of the ACM* **20**(1): 40–61.

Liu, J.W.S. (2000). *Real Time Systems*, Prentice Hall, New York, NY.

Min-Allah, N., Yong-Ji, W., Jian-Sheng, X. and Jiu-Xiang, L. (2007). Revisiting fixed priority techniques, *in* T.W. Kuo *et al.* (Eds.), *Proceedings of Embedded and Ubiquitous Computing*, Lecture Notes in Computer Science, Vol. 4808, Springer, Berlin/Heidelberg, pp. 134–145.

Min-Allah, N., Khan, S.U. and Yongji, W. (2010). Optimal task execution times for periodic tasks using nonlinear constrained optimization, *Journal of Supercomputing* **59**(3): 1–19.

Min-Allah, N. and Khan, S.U. (2011). A hybrid test for faster feasibility analysis of periodic tasks, *International Journal of Innovative Computing, Information and Control* **7**(10): 5689–5698.

Orozco, J., Cayssials, R., Santos, J. and Santos, R. (1998). On the minimum number of priority levels required for the rate monotonic scheduling of real-time systems, *Proceedings of the 10th Euromicro Workshop on Real Time Systems, Berlin, Germany*.

Patan, M. (2012). Distributed scheduling of sensor networks for identification of spatio-temporal processes, *International Journal of Applied Mathematics and Computer Science* **22**(2): 299–311, DOI: 10.2478/v10006-012-0022-9.

Santos, J., Gastaminza, M. L., Orozco, J., Picardi, D. and Alimenti, O. (1991). Priorities and protocols in real-time LANs, *Computer Communications* **14**(9): 507–514.

Sha, L. and Goodenough, J.B. (1988). Real-time scheduling theory and ADA, *CMU/SEI-88-TR-33*, Software Engineering Institute, Carnegie-Mellon University, Piittsburgh, PA.

Sha, L., Sprunt, B. and Lehoczky, J.P. (1989). Aperiodic task scheduling for hard real-time systems, *Journal of Real-Time Systems* **1**(1): 27–69.

Sheng, J., Wang, Y., Liu, J., Zeng,H. and Min-Allah, N. (2007). A static priority assignment algorithm with least number of priority levels, *Journal of Software* **18**(7): 1844–1854.

Sjodin, M. and Hansson, H. (1998). Improved response-time analysis calculations, *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain*, pp. 399–409.

Tindell, K.W., Bums, A. and Wellings, A.J. (1994). An extendible approach for analyzing fixed priority hard real-time tasks, *Real-Time Systems Journal* **6**(2):133–151.

Xuelian, B., Yuhai, Y. and Shiyao, J. (2003). Optimal fixed priority assignment with limited priority levels, *Proceedings of the Advanced Parallel Programming Technologies, Xiamen, China*, pp. 194–203.

Xu, J. and Parnas, D. (1990). Scheduling processes with release times, deadlines, precedence, and exclusion relations, *IEEE Transactions on Software Engineering* **16**(3): 360–369.

Muhammad Bilal Qureshi is a Ph.D. student of computer science at the COMSATS Institute of Information Technology, Islamabad, Pakistan. He received his B.S. with distinction (Gold Medal) in information technology and his M.S. in computer science in Pakistan. He is the recipient of various awards for his outstanding academic performance, including an HEC Pakistan Indigenous Scholarship for Master's and Doctoral Studies. He is the author and co-author of many research articles published in high ranking journals in grid computing, parallel computing, parallel and distributed computing, and so on. His research interests span the areas of real-time systems, scheduling theory and resource allocation problems in HPC systems.

Saleh Alrashed has been an assistant professor in the College of Computer Science and Information Technology at the University of Dammam since September 2012. He received his Ph.D. degree in knowledge base systems and his M.S. in information systems from Cardiff University, UK, in 2004 and 2001, respectively. He did his B.Sc. in information systems at King Saud University, Riyadh, in 1995. At the beginning of his carrier, Dr. Alrashed was a lecturer at King Faisal Air Academy, and then a research assistant at Royal Air Force Research Labs. In 2004, Dr. Alrashed led research and IT operations for establishing the Electronic Warfare Center for Royal Saudi Air Force. In 2008, he directed a UK-based team to design, develop, and integrate a cutting-edge knowledge base system for electronic warfare equipment. Dr. Alrashed is the dean of the College of Computer Science and Information System, and the vice-dean for e-learning at the College of Applied Studies at the University of Dammam.

Nasro Min-Allah has been an associate professor at the College of Computer Science and Information Technology, University of Dammam, KSA, since September 2014. He worked at the SuperTech group of the MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) from September 2012 to June 2014 as a visiting scientist, and taught at the Electrical Engineering and Computer Science Department from January 2013 to May 2014. He has a distinguished career in education, research and administration. He was an associate professor and the head of the Department of Computer Science at the COMSATS Institute of Information Technology (CIIT), Pakistan, from 2002 to 2012, and served the Green Computing and Communication Lab as the director. He is the recipient of three prestigious awards: the CIIT Golden Medallion for Innovation (CIMI-2009), Best Mobile Innovation in Pakistan (BMIP-2010), and a Best University Teacher Award, Pakistan (BUTA-2011).

Joanna Kołodziej graduated in mathematics from Jagiellonian University in Cracow in 1992, where she also received a Ph.D. in computer science in 2004. She works as an assistant professor at the Cracow University of Technology. She has been serving as a PC co-chair, a general co-chair and an IPC member of several international conferences and workshops, including *PPSN 2010, ECMS 2011, CISIS 2011, 3PGCIC 2011, CISSE 2006, CEC 2008, IACS 2008-2009, ICAART 2009-2010*. Dr. Kolodziej is an editorial board member and a guest editor of several peer-reviewed international journals, and an author and a co-author of many publications in high quality peer reviewed international journals. For more information, please visit `http://www.joannakolodziej.org/`.

Piotr Arabas received his Ph.D. in computer science from the Warsaw University of Technology, Poland, in 2004. Currently he is an assistant professor at the Institute of Control and Computation Engineering at the Warsaw University of Technology. Since 2002 he has also been with the Research and Academic Computer Network (NASK). His research area focuses on modeling computer networks, predictive control and hierarchical systems.

Received: 11 October 2014
Revised: 10 December 2014
Re-revised: 14 February 2015