Scientia Iranica D (2014) 21(6), 2127–2137

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
www.scientiairanica.com

Performability guarantee for periodic tasks in real-time systems

M. Bashiri and S.G. Miremadi*

Department of Computer Engineering, Dependable Systems Laboratory (DSL), Sharif University of Technology, Tehran, Iran.

Received 17 March 2013; received in revised form 6 April 2014; accepted 8 December 2014

KEYWORDS
Real-time systems;
Schedulability conditions;
Performability;
Reliability.

Abstract. Performability is an important parameter in safety-critical real-time systems. This parameter is defined as the joint consideration of two other important parameters, i.e., reliability and performance. This paper proposes a schedulability condition that guarantees a desired level of performability under various working conditions for real-time systems. The basic idea underlining this condition is to select a subset of schedulable tasks and manage their slack times to satisfy a desired performability level. The proposed condition is evaluated on a hard real-time system that employs the Rate-Monotonic (RM) scheduling algorithm and uses the re-execution mechanism to improve the reliability. Evaluation results reveal that by employing the condition, the level of performability of the system is always greater than the desired performability. In addition, it yields, on average, 1% improvement in system performability in comparison with traditional schedulability conditions, while the actual failure rate is greater than the expected rate. This amount of performability improvement is significant for safety-critical real-time systems.

1. Introduction

Systems, such as brake controllers in vehicles or controllers of medical embedded devices, are categorized as safety-critical systems [1]. This means that malfunctions in such systems may result in disastrous consequences to human life. In such systems, the logical accuracy of the result is not usually sufficient and it is necessary for tasks to fulfill several timing constraints. These kinds of systems are categorized as hard real-time systems.

Three attributes are frequently used in this paper; reliability, performance and performability. Reliability is defined as the conditional probability of the correct execution of a task, provided that, at the beginning of the task run, the system is in a fault-free state [10,19]; here, the task deadline is not considered. Performance is defined as the probability of a task execution within its deadline, no matter if the task delivers a correct or an incorrect result. In a hard real-time system, both correct execution of a task in the presence of faults and task execution within its deadline are important. In this regard, performability is defined as the joint consideration of performance and reliability [9,14,21,22]. For example, in [22], performability is defined as the probability of completing an application correctly within its deadline in the presence of faults.

Task scheduling determines the sequence of task execution in a real-time system. The ways of assigning priorities to tasks are named scheduling algorithms [6,16]. One class of these algorithms is fixed-priority scheduling algorithms. In fixed-priority scheduling algorithms, the number of tasks is fixed and the tasks are periodic; here, the priorities of the tasks are defined before their execution. The Rate-Monotonic (RM) algorithm is an optimal fixed-priority algorithm for simply periodic tasks [2,13]. It is noteworthy that the fixed-priority scheduling algorithms

*. *Corresponding author.*
E-mail addresses: m_bashiri@ce.sharif.edu (M. Bashiri); miremadi@sharif.edu (S.G. Miremadi)

are usually used forscheduling tasks in hard real-time applications [17].

It is worth mentioning that all scheduling algorithms assign only priorities to tasks, without determining if they meet their deadlines or not. This means that if there are copious numbers of tasks that are waiting to be run on a processor, it is probable that some tasks will miss their deadlines. To remove this shortcoming, the schedulability conditions are used to determine whether the assignment of a new task to the processor is feasible or not; if it is not feasible, the task will not be assigned to the processor at all. The first schedulability condition, called the LL condition, proposed by Liu and Layland [13], is a condition with the worst-utilization bound that is introduced for the RM scheduling algorithm. Several attempts have been made to improve the utilization bound of the LL schedulability condition [3-5,7,12,15,20]. Under the improved conditions, the issue of performance guarantee is the main concern, while the issue of performability guarantee is not addressed at all.

In [8], the performability of a safety-critical real-time embedded system is evaluated by employing five different schedulability conditions for the RM algorithm. To the best of our knowledge, the issue of guaranteeing the performability of a real-time system, by employing schedulability conditions, has not been addressed in previous work.

The present paper extends the work in [8] by introducing an innovative schedulability condition that guarantees performability. This condition utilizes the task re-execution mechanism [23] to realize the guarantee. The basic idea underlining the proposed condition is to manage the slack times of tasks in order to re-execute a task when it fails. This condition selects a proper schedulable task-set according to an expected failure rate and a desired performability level. To evaluate this condition, as well as the five traditional schedulability conditions, a software tool is developed to extract the performability of a multi-processor with m schedulable periodic tasks. Here, the tasks are scheduled by the RM scheduling algorithm.

The rest of the paper is organized as follows: Section 2 discusses the schedulability conditions for the RM algorithm. The proposed schedulability condition is introduced and discussed in Section 3; this section also includes proof of the proposed condition. The simulation method and the simulation results are presented in Sections 4 and 5, respectively. Finally, Section 6 concludes the paper.

2. Methods of task scheduling and schedulability checking in real-time systems

The process of assigning priorities to tasks to determine the sequence of their execution in a real-time system is accomplished by scheduling algorithms [6,16]. These algorithms are classified into two classes:

1. dynamic-priority scheduling algorithms, such as EDF;
2. fixed-priority scheduling algorithms, such as RM. The fixed-priority scheduling algorithms may have four main characteristics:
   (a) The priority of each task is assigned before its execution;
   (b) The number of tasks is fixed;
   (c) The tasks are always periodic;
   (d) These algorithms are usually employed in hard real-time applications [17].

As mentioned above, all scheduling algorithms assign only priorities to tasks, however, they do not check if the tasks meet their deadlines or not. This means that some tasks may miss their deadlines due to the processor overrun. To prevent deadline misses, schedulability conditions are utilized. These conditions examine the deadlines of existing tasks to determine if the assignment of a new task to the processor is feasible or not. If it is not feasible, the task will not be assigned to the processor at all [2]. The first condition, called the LL condition, proposed by Liu and Layland [13], is a worst-case condition in terms of schedulable processor utilization bound; this condition is employed with the RM scheduling algorithm. Several attempts have been made to improve the LL utilization bound [3-5,7,12,15,20].

It is worth mentioning that the schedulability conditions can also assist the task distributor in a multiprocessor system to determine the number of processors required to run a set of periodic tasks. In the following, we will introduce the five most important and extensively used schedulability conditions.

2.1. Schedulability conditions for RM algorithm

Five important schedulability conditions for RM scheduling algorithms are explained hereafter. These conditions examine the schedulability of a new task on a processor. If the employed condition is satisfied, the new task is scheduled.

1. LL condition [13]: If a set of n tasks is scheduled according to the RM algorithm, then, the minimum achievable CPU utilization is:

$$n\left(2^{1/n}-1\right). \tag{1}$$

In Eq. (1), if $n \to \infty$, then $n(2^{1/n}-1) \to \ln 2$.

The schedulability of tasks can be checked by Eq. (1). If the sum of utilizations of the newly arrived task and previously accepted tasks is less

than the value of Eq. (1), then, the newly arrived task is accepted for scheduling. This condition is a worst-case condition because it leads to the minimum achievable utilization bound for the processor. The next four conditions attempt to improve the utilization bound.

2. UO condition [20]: Assume that $X = \{\tau_i = (C_i, T_i) | i = 1, 2, ..., n-1\}$ is a set of $n-1$ tasks and it can be scheduled by the RM algorithm. The scheduling of a newly arrived task (n-th task) may only be possible together with other $n-1$ tasks, if:

$$C_n/T_n \leq 2\left[\Pi_{i=1}^{n-1}(1+u_i)\right]^{-1} - 1. \quad (2)$$

3. IP condition [3]: Let $X = \{\tau_i = (C_i, T_i) | i = 1, 2, ..., n-1\}$ be a set of periodic tasks that are sorted by their decreasing periods (highest to the lowest period). If $u = \sum_{i=1}^{n-1} C_i/T_i$, then, X is schedulable by the RM algorithm if both Eqs. (3) and (4) are true:

$$u \leq (n-1)\left(2^{1/(n-1)} - 1\right), \quad (3)$$

$$C_n/T_n \leq 2\left(1 + u/(n-1)\right)^{-(n-1)} - 1. \quad (4)$$

4. PO condition [12]: Let $X = \{\tau_i = (C_i, T_i) | i = 1, 2, ..., n-1\}$ be a set of periodic tasks that are sorted by their decreasing periods (highest to the lowest period). Tasks are schedulable by the RM algorithm if Eq. (5) is true:

$$\Pi_{i=1}^{n}(u_i + 1) \leq 2. \quad (5)$$

5. LC condition [15]: Let $X = \{\tau_i = (C_i, T_i) | i = 1, 2, ..., n-1\}$ be a set of periodic tasks, which are sorted by their increasing period. These tasks are schedulable on the processor if Eq. (6) is true (here, $z = T_1/T_n$):

$$\sum_{i=1}^{n} C_i/T_i \leq 2z - 1 + (n-1)\left((1/z)^{1/n-1} - 1\right). \quad (6)$$

3. The proposed schedulability condition

This section proposes a new condition to guarantee the performability of hard real-time systems. This condition examines if a newly arrived task can be accepted or rejected, with respect to a desired performability level. The main idea underlining this condition is to re-execute a task when the task fails; the re-execution is repeated until the first failure-free run appears. It is important to mention that this condition calculates the number of re-executions that a task can execute within the slack time. Task re-execution may improve task performability, which, in turn, improves system performability to a desired level. The proposed schedulability condition is a function of four parameters: 1) The current task characterization, i.e. the release time, the execution time, and the period; 2) The characterizations of previously accepted tasks; 3) The failure rate; and 4) A desired level of performability. To show how this condition is extracted, the condition is studied in three steps. In the first step, the condition for the first task is derived. In the second step, the condition for the second task is also derived. Based on the achievements in the above two steps, in the third step, the generalized condition for the i-th task is derived. The proof of each step is shown in the following. The steps are based on using the RM algorithm. It is worth noting that the first task has the highest priority and the last one has the lowest priority.

First step. In this step, there is only one task in the system. If this task is accepted and scheduled, the value of system performability is equal to the performability of this task. This means that, if the performability of this task is greater than or equal to a desired performability level, system performability will also be greater than or equal to L. Theorem 1 shows the schedulability condition for the task.

Theorem 1. In a real-time system, the task with the highest priority is acceptable if the task characterizations satisfy Eq. (7). The lefthand side of Eq. (7) is the performability of this task:

$$1 - q_1^{\left\lfloor \frac{T_1}{C_1} \right\rfloor} \geq L. \quad (7)$$

In Eq. (7), $\lfloor T_1/C_1 \rfloor$ indicates the maximum possible number of task executions (including the original run and its re-executions) from the release time until the absolute deadline of the task. Figure 1 illustrates the tasks lack time, which can be used for task re-executions until the first correct run, upon detection of a failure.

Proof 1. To show task performability, we list, in the following, all possible situations under which the task can be executed; each situation has a certain probability:

- The correct run of the task at the first attempt with probability p_1;

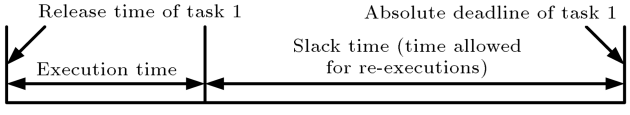


Figure 1. The execution and re-execution times of task 1.

- The correct run of the task at the second attempt (the first attempt fails) with probability $p_1 \times q_1$;
- The correct run of the task at the third attempt (the first and second attempts fail) with probability $p_1 \times q_1^2$;
- .....
- The correct run of the task at the i-th attempt (all the previous $(i-1)$-th attempts fail) with probability $p_1 \times q_1^{i-1}$;
- .....
- The last chance for the correct run of the task before the deadline (all the previous attempts fail) is probability $p_1 \times q_1^{\left\lfloor \frac{T_1}{C_1} \right\rfloor - 1}$.

Since the above situations are independent, the probability of the correct run of the task is equal to the sum of the above probabilities, i.e.:

$$\text{performability}\,(\tau_1) = p_1 + p_1 \times q_1 + p_1 \times q_1^2 + \ldots + p_1 \times q_1^{i-1} + \ldots + p_1 \times q_1^{\left\lfloor \frac{T_1}{C_1} \right\rfloor - 1}.$$

Substituting p_1 by $(1 - q_1)$, the performability of the task is calculated as $1 - q_1^{\left\lfloor \frac{T_1}{C_1} \right\rfloor}$.

Corollary 1. Assume that E is the maximum number of executions. Substituting $\lfloor T_1/C_1 \rfloor$ by E, the performability of the task with a given failure probability (q) is defined as $1 - q^E$.□

Second step. Assume that the first task is scheduled in the first step. The possible scheduling of the second task should not degrade the system performability, which is the performability of both tasks, i.e. the performability of each task should be greater than L.

Theorem 2. In a real-time system, the second task (the task with the second highest priority) is acceptable if the task characterizations satisfy Eq. (8). Note that the first task was previously accepted in the first step.

$$P(\tau_2) = \sum_n P\left(\tau_2 | N_1 = n\right) P(N_1 = n). \tag{8}$$

Here, Eq. (8) is the probability of the correct run of the second task, which is derived by employing the total probability theorem presented in [18], with the condition of the number of execution/re-executions of the first task in the first step.

In Eq. (8), N_1 is the number of execution/re-executions of the first task until the correct run within the deadline. It is obvious that the number of re-executions of the first task reduces the slack time of the second task. Figure 2 shows the relationship between execution/re-executions of both tasks with the slack time of the second task. It is possible that a correct run of the first task needs several re-executions that may reduce the slack time of the second task.

The two components of the right hand side of Eq. (8) can be rewritten to Eqs. (9) and (10), respectively:

$$P\left(\tau_2 | N_1 = n\right) = \begin{cases} 1 - q_2^{\left\lfloor \frac{T_2 - n \times C_1}{C_2} \right\rfloor} & T_2 - n \times C_1 \geq C_2 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Eq. (9) defines the probability of the correct run of the second task when the number of executions of the first task is known and is equal to n. Eq. (10) defines the probability of the correct run of the first task within n executions:

$$P(N_1 = n) = \underbrace{p_1^{\left\lceil \frac{T_2}{T_1} \right\rceil} \times q_1^{n - \left\lceil \frac{T_2}{T_1} \right\rceil}}_{\text{Part } A} \times \sum_{t=0}^{\left\lceil \frac{T_2}{T_1} \right\rceil - 1} \underbrace{\left[(-1)^t \times \binom{\left\lceil \frac{T_2}{T_1} \right\rceil}{t} \times \binom{n - t \times \left\lfloor \frac{T_1}{C_1} \right\rfloor - 1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1} \right]}_{\text{Part } B}. \tag{10}$$

Inserting Eqs. (9) and (10) into Eq. (8), Expression (11) is derived:

$$P(\tau_2) = \sum_{n=1}^{\left\lfloor \frac{T_1}{C_1} \right\rfloor} \left(1 - q_2^{\left\lfloor \frac{T_2 - n \times C_1}{C_2} \right\rfloor}\right) \times \left(p_1^{\left\lceil \frac{T_2}{T_1} \right\rceil} \times q_1^{n - \left\lceil \frac{T_2}{T_1} \right\rceil} \times \sum_{t=0}^{\left\lceil \frac{T_2}{T_1} \right\rceil - 1} \left[(-1)^t \times \binom{\left\lceil \frac{T_2}{T_1} \right\rceil}{t} \times \binom{n - t \times \left\lfloor \frac{T_1}{C_1} \right\rfloor - 1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1} \right] \right). \tag{11}$$

Proof 2. According to Corollary 1, $E = \lfloor (T_2 - n \times C_1)/C_2 \rfloor$ is the maximum number of allowable

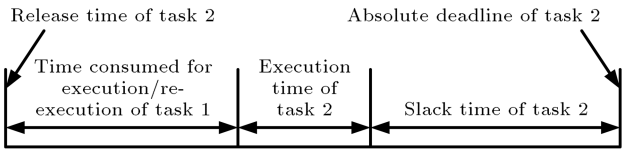


Figure 2. The relationship between execution/re-executions of both tasks.

execution/re-executions of the second task. $n \times C_1$ is the effect of the number of execution/re-executions for the first task. So, the performability of the second task, when the number of execution/re-executions of the first task is known, can be calculated by Eq. (9).

In Eq. (10), Part A is the product of the probabilities of n times run of the first task. T_2 is the time in which the second task has the chance to finish its work to avoid deadline miss. However, according to Eq. (9), this time is affected by the number of execution/re-executions of the first task. It is obvious that with n times execution/re-executions of the first task during T_2, the first task is correctly executed $\left\lceil \frac{T_2}{T_1} \right\rceil$ times and is erroneously executed $\left(n - \left\lceil \frac{T_2}{T_1} \right\rceil\right)$ times. So, Part A in Eq. (10) can be written as $p_1^{\left\lceil \frac{T_2}{T_1} \right\rceil} \times q_1^{n-\left\lceil \frac{T_2}{T_1} \right\rceil}$.

Part B in Eq. (10) calculates the number of distributing these n executions into $\left\lceil \frac{T_2}{T_1} \right\rceil$ sections. Here, each section contains at least one execution (with correct run) and, at most, $\left\lfloor \frac{T_1}{C_1} \right\rfloor$ execution/re-executions ($\left\lfloor \frac{T_1}{C_1} \right\rfloor - 1$ re-executions plus the correct run). To derive Part B, the inclusion-exclusion principle [11] is used. Suppose that there are n executions separated by $n-1$ delimiters (see Figure 3). Using stage 1 of the above mentioned principle, the number of distributing the n executions into $\left\lceil \frac{T_2}{T_1} \right\rceil$ sections, with at least one execution in each part, is calculated. In this case, there are $\left\lceil \frac{T_2}{T_1} \right\rceil - 1$ delimiters to perform this partitioning. This means that there are $\binom{n-1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1}$ possible ways to distribute the n executions into $\left\lceil \frac{T_2}{T_1} \right\rceil$ sections.

Using stage 2 of the inclusion-exclusion principle, the number of sections in which each contains more than $\left\lfloor \frac{T_1}{C_1} \right\rfloor$ executions must be excluded from the result in stage 1. Suppose that $\left\lfloor \frac{T_1}{C_1} \right\rfloor$ executions of the total executions are dedicated to a section, and the remaining executions are distributed among all $\left\lceil \frac{T_2}{T_1} \right\rceil$ sections to ensure having at least one section with more than $\left\lfloor \frac{T_1}{C_1} \right\rfloor$ executions. Similar to stage 1, there are $\binom{n - \lfloor \frac{T_1}{C_1} \rfloor - 1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1}$ possible ways to do this distribution. So, $\left\lceil \frac{T_2}{T_1} \right\rceil \times \binom{n - \lfloor \frac{T_1}{C_1} \rfloor - 1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1}$ distributions must be excluded from the result of stage 1.

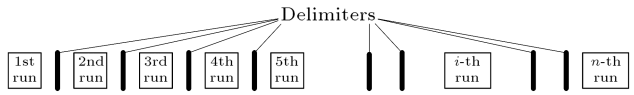


Figure 3. There are $n-1$ total delimiters for selecting $\left\lceil \frac{T_2}{T_1} \right\rceil - 1$ of them to create $\left\lceil \frac{T_2}{T_1} \right\rceil$ sections.

In stage 3 of the inclusion-exclusion principle, the number of distributions of each pair of sections that have more than $2 \times \left\lfloor \frac{T_1}{C_1} \right\rfloor$ executions in each pair must be included in the result of the previous stages. Suppose that for each pair of sections, $2 \times \left\lfloor \frac{T_1}{C_1} \right\rfloor$ executions from the total number of executions are dedicated, and the remaining executions are distributed among all $\left\lceil \frac{T_2}{T_1} \right\rceil$ sections to ensure having at least two sections with more than $\left\lfloor \frac{T_1}{C_1} \right\rfloor$ executions in each section. Like the two aforementioned stages, there are $\binom{n - 2 \times \lfloor \frac{T_1}{C_1} \rfloor - 1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1}$ possible ways to do this distribution in $\binom{\left\lceil \frac{T_2}{T_1} \right\rceil}{2}$ pairs of sections, So, $\binom{\left\lceil \frac{T_2}{t_1} \right\rceil}{2} \times \binom{n - 2 \times \lfloor \frac{T_1}{C_1} \rfloor - 1}{\left\lceil \frac{T_2}{T_1} \right\rceil - 1}$ distributions must be included in the results of the two preceding stages.

By continuing these stages, multiplying $\binom{\left\lceil \frac{T_2}{T_1} \right\rceil}{0}$ to the result of the first stage and multiplying the result of each stage by $(-1)^{\text{stage number}-1}$ to show the inclusion/exclusion, Eq. (10) is obtained.□

The value of Eq. (11) determines the second task performability. If it is greater than the desired performability level, the second task will be accepted. Otherwise, it is rejected and should be sent to a redundant processor.

Third step. This step tries to generalize the schedulability condition for the i-th task. To guarantee the system performability, the performability of each task should be greater than L.

Theorem 3. In a real-time system, the i-th task (i.e., the task with the i-th highest priority in the task list) is acceptable if the task characterizations satisfy Eq. (12).

Like the second step, by using the total probability theorem and conditioning on the number of executions of already accepted periodic tasks, the probability of the correct run of the i-th task is calculated. The result of this calculation is shown in Eq. (12):

$$P(\tau_i) = \sum_{n_1,n_2,\ldots,n_{i-1}} P\big[\tau_i | N_1 = n_1, N_2 = n_2, \ldots, N_{i-1} = n_{i-1}\big] \times P(N_1 = n_1, N_2 = n_2, \ldots, N_{i-1} = n_{i-1}). \quad (12)$$

Eq. (12) is constituted of two parts that are calculated in Eqs. (13) and (14), respectively. Eq. (13) calculates the probability of the correct run of the i-th task when the number of executions of previously scheduled tasks are known. The probability of the occurrence of a specific number of task re-executions for the previously scheduled tasks is calculated in Eq. (14):

$$P\left[\tau_i | N_1 = n_1, N_2 = n_2, ..., N_{i-1} = n_{i-1}\right] = \begin{cases} 1 - q_i^{\left\lfloor \frac{T_i - \sum_{j=1}^{i-1} n_j \times C_j}{C_i} \right\rfloor} & T_i - \sum_{j=1}^{i-1} n_j \times C_j \geq C_i \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

$$P\left(N_1 = n_1, N_2 = n_2, ..., N_{i-1} = n_{i-1}\right) = P(N_1 = n_1) \times P(N_2 = n_2) \times ... \times P(N_{i-1} = n_{i-1}) = \Pi_{j=1}^{i-1} \left(p_j^{\left\lceil \frac{T_i}{T_j} \right\rceil} \times q_j^{n_j - \left\lceil \frac{T_i}{T_j} \right\rceil} \times \sum_{t=0}^{\left\lceil \frac{T_i}{T_j} \right\rceil - 1} \left[(-1)^t \times \binom{\left\lceil \frac{T_i}{T_j} \right\rceil}{t} \times \binom{n_j - t\left\lfloor \frac{T_j}{C_j} \right\rfloor - 1}{\left\lceil \frac{T_i}{T_j} \right\rceil - 1} \right] \right). \quad (14)$$

Proof 3. According to Corollary 1, $E = \lfloor (T_i - \sum_{j=1}^{i-1} n_j \times C_j)/C_i \rfloor$ is the maximum allowable number of task execution/re-executions (including the original run and its corresponding re-executions). Here, $\sum_{j=1}^{i-1} n_j \times C_j$ is the effect of the number of execution/re-executions for the already $i-1$ accepted tasks (see Figure 4). So, the performability of the i-th task, while the number of execution/re-executions of the previously accepted tasks is known, can be calculated by Eq. (13).

To show the correctness of Eq. (14), it is assumed that the executions of tasks are independent. This means that the probability of occurring an exact number of executions of tasks is equal to the product of the probability of executing each task individually. For each task, this probability is the same as the probability shown in Eq. (11).□

Based on Theorem 2, Eq. (12) shows the value of performability of the i-th task when $i-1$ tasks, with a performability of not less than L, are previously accepted. If the performability of the i-th task is also not less than L, this task is able to be scheduled in the system. This statement is shown in Eq. (15):

$$P(\tau_i) \geq L. \quad (15)$$

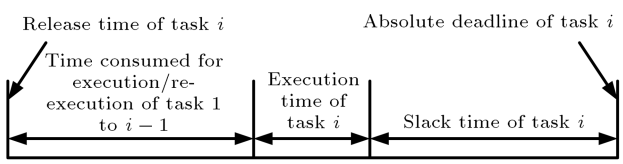


Figure 4. The execution and re-execution times of tasks 1 to i.

A case study. To clear the above condition, three examples are presented.

Example 1. The schedulability of a sample task set, X, is checked by a traditional schedulability checking condition. Suppose that $X = \{\tau_i = (C_i, T_i) | i = 1, 2, 3, 4, 5\}$ is a set of periodic tasks, where $\tau_1 = (C_1 = 3, T_1 = 10)$, $\tau_2 = (C_2 = 3, T_2 = 12)$, $\tau_3 = (C_3 = 3, T_3 = 25)$, $\tau_4 = (C_4 = 3, T_4 = 58)$ and $\tau_5 = (C_5 = 3, T_5 = 70)$. According to the LL condition, the schedulability check for task set X is shown in Table 1.

As mentioned in Section 2, the LL condition is known as the worst-case condition because it provides the lowest achievable utilization bound for the processor. This may result in rejecting tasks, such as the 5th task (τ_5). However, the 5th task is schedulable by using a relative optimistic condition for the RM algorithm, or using the EDF scheduling algorithm. For example, the schedulability check for task set X when the EDF scheduling algorithm is used is shown in Table 2. Based on the optimality of the EDF scheduling algorithm, the achievable schedulable utilization for this algorithm is 1.

Example 2. The performability of the system using the LL condition and running the four accepted tasks, as discussed in Table 1, is calculated as follows:

1. Performability of the 1st task=0.999999000000000;
2. Performability of the 2nd task=0.999996030000000;
3. Performability of the 3rd task=0.999996029994119;
4. Performability of the 4th task=0.999996029994120.

The above values are calculated by employing the task re-execution mechanism, and the probability of failure occurrence in the tasks during their run is set to $q = 0.01$. In addition, the "iomanip.h" library is added to the simulator code to increase the results accuracy. Here, the provided accuracy is 10^{-15}. As a result, the average system performability is 0.999996772497059.

Example 3. Suppose that the desired level of performability of the system running the task set X is $L = 0.99999999$, and the probability of failure occurrence in the tasks is $q = 0.01$. The performabilities of tasks using the proposed condition (Eq. (12)) are extracted as follows:

1- Performability of the 1st task = 0.999999000000000

Table 1. Schedulability checking for the task set X by employing LL condition.

Schedulability of the i-th task	LL schedulability checking condition	Result
1st task	$1(2^{1/1}-1)=1.000>u_1=0.300$	$\surd$ Task is schedulable
2nd task	$2(2^{1/2}-1)=0.828>u_1+u_2=0.550$	$\surd$ Task is schedulable
3rd task	$3(2^{1/3}-1)=0.780>u_1+u_2+u_3=0.670$	$\surd$ Task is schedulable
4th task	$4(2^{1/4}-1)=0.757>u_1+u_2+u_3+u_4=0.722$	$\surd$ Task is schedulable
5th task	$5(2^{1/5}-1)=0.743>u_1+u_2+u_3+u_4+u_5=0.765$	$\star$ Task is not schedulable

Table 2. Schedulability checking for the task set X by employing EDF algorithm (condition).

Schedulability of the i-th task	EDF schedulability checking algorithm	Result
1st task	$1>u_1=0.300$	$\surd$ Task is schedulable
2nd task	$1>u_1+u_2=0.550$	$\surd$ Task is schedulable
3rd task	$1>u_1+u_2+u_3=0.670$	$\surd$ Task is schedulable
4th task	$1>u_1+u_2+u_3+u_4=0.722$	$\surd$ Task is schedulable
5th task	$1>u_1+u_2+u_3+u_4+u_5=0.765$	$\surd$ Task is schedulable

$\star$ Task is not schedulable;
2- Performability of the 2nd task = 0.999999990000000
$\surd$ Task is schedulable;
3- Performability of the 3rd task = 0.999999989999960
$\star$ Task is not schedulable;
4- Performability of the 4th task = 0.999999990000000
$\surd$ Task is schedulable;
5- Performability of the 5th task = 0.999999990000000
$\surd$ Task is schedulable.

This means that the tasks $\{\tau_2, \tau_4, \tau_5\}$ are definitely accepted using the proposed condition. Hence, the proposed condition guarantees the system performability above the desired level when tasks $\{\tau_2, \tau_4, \tau_5\}$ are accepted.

4. Simulation method

To evaluate the proposed condition, two experiments are carried out. In the first experiment, a single processor is simulated to reveal the effects of using different schedulability conditions on system performability. To perform this, the five schedulability conditions discussed in Section 2 are employed with the RM algorithm.

For the sake of simplicity, the following notations and definitions are used in this paper:

n	Number of tasks;
τ_i	The i-th task;
C_i	Worst case execution time of the i-th task;
T_i	Period of the i-th task;
r_i	Release time of the i-th task;
d_i	Absolute deadline of the i-th task;
u_i	Utilization of the i-th task that is equal to c_i/T_i;
λ	Failure rate;
p_i	The probability of the i-th task correct run within a single execution, that is a function of λ;
q_i	The probability of a failure occurrence during the run time of the i-th task, that is equal to $1-p_i$;
$P(\tau_i)$	The probability of the i-th task first correct run;
L	The desired level of performability. It is a value between 0 and 1.

The following steps are implemented for this simulation:

1. An initial task set is generated randomly. In this set, the generated tasks have sufficient slack time for re-execution upon detecting a probable occurred failure. Moreover, with lower task utilization, the schedulability conditions enter their saturated mode and reveal their utilization bounds.
2. The generated schedulable task set is simulated for over one million cycles and random failures are injected into the tasks. In this step, the simulator re-executes the failed tasks. The log of system runs and the probability of task completion before the deadline are checked to measure the performability.

```
1- Performability calculation procedure
2- BEGIN
3-     If the task performability is not calculated yet
4-     BEGIN
5-         temp= 1 − q_1^{X(1)}                    'X(i) is defined as ⌊(T_i − Σ_{j=1}^{i−1} n_j * C_j) / C_i⌋
6-         if (temp>= L and the task is not already assigned to the processor)
7-             add the task to the scheduled task set and mark it as assigned task
8-         RETURN temp                             'Temp is the task performability
9-     END
10-    Initialize the current scheduled queue      'For calculating the performability of the newly arrived periodic task which may be scheduled with the previously scheduled tasks.
11-    temp=calculate the value of Eq. (12)
12-    if (temp>= L and the task is not already assigned to the processor)
13-        add the task to the scheduled task set and mark it as assigned task
14-    RETURN temp
15- END
```

'Let L be the desired performability level

Figure 5. Pseudo-code for the implemented algorithm.

In the second experiment, the proposed condition is applied to the simulator to evaluate its effects on performability. Figure 5 demonstrates the algorithm of the second experiment.

The following models are considered in the two aforementioned experiments:

1) Task model. a set of n tasks $\sum = \{\tau_1, \tau_2, ..., \tau_n\}$ are given, where $\{\tau_i = (C_i, r_i, d_i, T_i) | i = 1, 2, 3, ..., n\}$ and c_i is the required execution time of task τ_i. In addition, r_i, d_i and T_i are designated as the release time, deadline and period of τ_i, respectively.

The following assumptions are used in the task model:

a) The arrivals of all tasks are periodic with constant intervals between arrivals.

b) It is assumed that the relative deadline and period of each task are equal ($d_i = T_i$).

c) It is assumed that the tasks are independent. They are preemptable as well.

d) The tasks execution times are constant.

e) There are sufficient resources in the system to avoid resource conflict between tasks.

f) It has been proven that the release times of tasks do not affect the schedulability of the tasks [13]. Therefore, the tasks release times are ignored.

2) System model. Figure 6 shows the system model. In this system, the tasks distributer utilizes the proposed condition for assigning the maximum available tasks to a processor and then assigns the rejected tasks to other processors.

3) Scheduling model. The processor has a queue for scheduled tasks and the RM scheduling algorithm

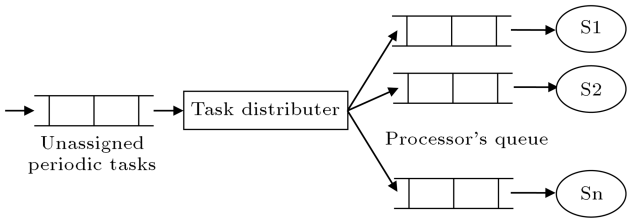


Figure 6. The system model (Si is the name of i-th processor).

works on the accepted tasks in this queue. The acceptance or rejection of tasks is performed by one of the five aforementioned traditional conditions in the first experiment and by the proposed condition in the second experiment.

In these experiments, task assignment overheads and context switching overheads for the RM scheduling algorithm are ignored.

Failure model. The following assumptions are used in the experiments:

a) In the current experiments, the system malfunctions are considered as processor failure or task failure. Thus, the causes of failures are not important in this paper.

b) The failures are independent, i.e. the correlated failures are not considered.

c) The failures do not occur in the scheduler, therefore, the scheduler is failure free.

d) The failure-detection coverage is 100% and the failure detection latency is assumed as zero.

5. Simulation results

In this section, the simulation results of studying five schedulability conditions for the RM scheduling

algorithm are presented. The results are derived from two experiments: Experiment A and Experiment B.

In Experiment A, the performability of a real-time single processor system is evaluated. Table 3 shows the effects of different schedulability conditions on system performability. It is important to note that a small difference between the performabilities of the conditions may have significant effects on overall system performability and reliability [22]. This happens in a safety-critical real-time system, such as the controller of an avionic system, when system reliability and performability are calculated serially by multiplying the performabilities of system components.

According to the simulation results, the UO and the PO conditions have the same and the highest performability among the five simulated conditions. This can be motivated by the maximum utilization bound in both UO and PO conditions. For example, the LC condition, which is an improved version of the LL condition, provides a higher utilization bound; therefore, the re-execution method has less chance to re-execute the failed tasks using the LC condition.

In Figure 7(a) and (b), performabilities of the LL and the LC conditions are evaluated, with respect to the failure rate. These figures show that performability decreases rapidly when the failure rates increase. This means that the conditions are not aware of the environmental failure rates. In this case, the conditions cannot guarantee the performability of the system at the desired level. These figures also demonstrate that these two conditions have an upper bound of performability for each failure rate. Experiment A is also applied to UO, IP, and PO conditions; the simulation results are similar to the results shown in Figure 7(a) and (b).

Experiment B evaluates the proposed condition. In this Experiment, the system model shown in Figure 6 is evaluated to determine the effects of the proposed condition on system performability. In this experiment, the performability of this condition is compared with the highest performability that was achieved in Experiment A, i.e. the performability of the PO condition (see Table 3). The evaluation results of Experiment B are shown in Figure 8(a) and (b). These figures show the ability of the proposed condition to guarantee or improve the performability of the system in comparison with the PO condition.

In Figure 8(a), the proposed condition is compared with the PO condition. Here, system performability is measured with respect to failure rate, when the performability level (L) is 0.9999 and the probability of the correct run of task (P_i) is 0.9. In Figure 8(b), this experiment is repeated when $L = 0.999999999$ and $P_i = 0.99$. These figures demonstrate that by employing the proposed condition, the level of system performability is at a higher level than the PO condition. These figures also show that the proposed condition provides a significant improvement in system performability, even when the failure rate is higher than the expected failure rate. On average, 1% improvement in system performability is achieved in comparison with the PO condition. This amount of performability improvement is important for safety-critical real-time systems.

Experiment B also shows that when actual failure

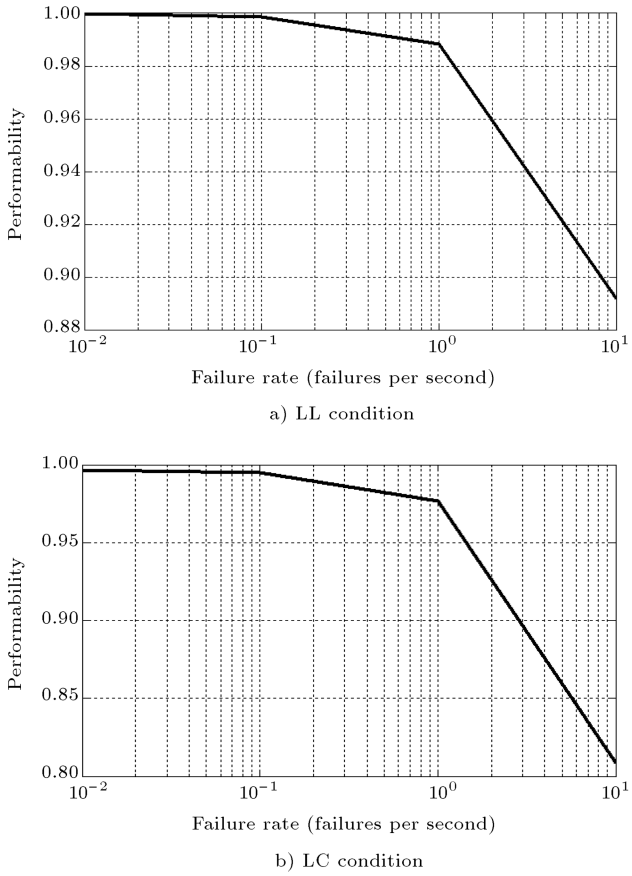
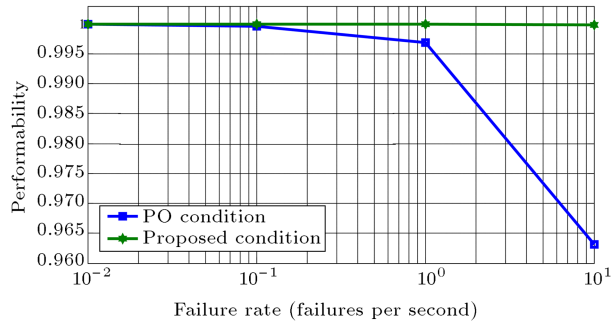


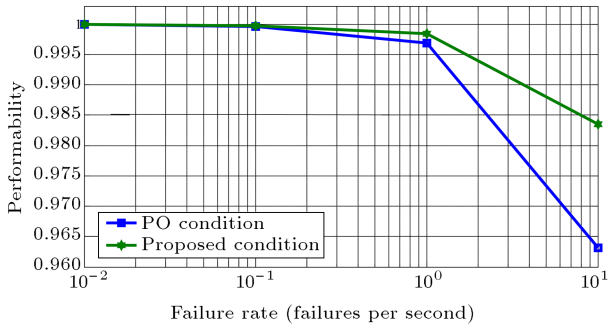
Figure 7. The upper bound of performability for two conditions in the presence of failures.

Table 3. System performability in case of using five different schedulability conditions.

	Schedulability checking conditions				
	LL condition [13]	**UO condition [20]**	**IP condition [3]**	**PO condition [12]**	**LC condition [15]**
Measured performability	0.892789	0.892803	0.892422	0.892803	0.881547



a) $L = 0.9999$ and $P_i = 0.9$



b) $L = 0.999999999$ and $P_i = 0.99$

Figure 8. The performability of the proposed condition in comparison with the PO condition in different situations.

rate is less than or equal to an expected rate, the performability of the system, which employs the proposed condition, is greater than the desired level. This means that the condition can guarantee system performability at a desired level specified by the system designers.

6. Conclusions

In this study, the performabilities of five schedulability conditions in the presence of different failure rates have been measured by Experiment A. The measurement results show that the upper bound of performability of each condition is limited, and the conditions are not aware of the environmental failure rates. Based on these results, a new schedulability condition for real-time systems has been proposed that guarantees a desired level of performability under various working situations. This condition gets three user-defined parameters, i.e. a set of tasks, a failure rate, and a performability level, and then gives a subset of the task set, which are schedulable on the processor. This condition has been proved mathematically in Section 3. To confirm the proof, Experiment B has been done by simulation of a typical hard real-time system, by employing the RM algorithm and the re-execution mechanism. In Experiment B, the proposed condition is compared to the PO condition, because the PO condition showed the highest performability in Experiment A. The simulation results using the proposed condition showed that the level of system performability was constantly greater than the user-defined performability level, when the actual failure rate was less than or equal to the given failure rate. In addition, the results of Experiment B show that the proposed condition gives, on average, 1% improvement in system performability, in comparison to traditional conditions. This improvement has been achieved, while the actual failure rate was greater than the initially given failure rate. In conclusion, based on the results of Experiment B, the proposed condition works better than other conditions in various working situations.

References

1. Marwedel, P., *Embedded System Design*, 2nd Edition, Netherlands, Springer, ISBN: 9400702566 (2011).

2. Cottet, F., Delacroix, J., Kaiser, C. and Mammeri, Z., *Scheduling in Real-Time Systems*, John Wiley & Sons, ISBN: 9780470856345 (2002).

3. Dhall, S.K. and Liu, C.L. "On a real-time scheduling problem", *Operations Research, Informs*, **26**(1), pp. 127-140, January-February (1978).

4. Burchard, A., Liebeherr, J., Oh, Y. and Son, S.H. "New strategies for assigning real-time tasks to multiprocessor systems", *IEEE Transactions on Computers*, **44**(12), pp. 1429-1442 (December 1995).

5. Park, M. and Cho, Y. "Feasibility analysis of hard real-time periodic tasks", *Journal of Systems and Software*, **73**(1), pp. 89-100 (September 2004).

6. Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J. and Baruah, S. "A categorization of real-time multiprocessor scheduling problems and algorithms", In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Joseph Y-T Leung, Ed., Chapman Hall/CRC Press (2003).

7. Lu, W.C., Lin, K.J., Wei, H.W. and Shin, W.K. "Rate monotonic schedulability tests using period-dependent conditions", *Real-Time Systems*, **37**, pp. 123-138 (2007).

8. Bashiri, M. and Miremadi, S.G. "Performability comparison of schedulability conditions in real-time embedded systems", In *Proceedings of the 3rd International Conference on Dependability (DEPEND 2010)*, pp. 70-75 (July 2010).

9. Kavi, K.M., Hee Yong Youn, Shirazi, B. and Hurson, A.R. "A performability model for soft real-time systems", In *Proceedings of the 27th Hawaii International Conference on System Sciences (HICSS)*, pp. 571-579 (1994).

10. Avizienis, A., Laprie, J., Randell, B. and Landwehr, C. "Basic concepts and taxonomy of dependable and secure computing", *IEEE Transactions on Dependable and Secure Computing*, **1**(1), pp. 11-33 (January 2004).

11. Jukna, S., *Extremal Combinatorics with Applications in Computer Science*, Springer, ISBN: 9783642085598 (2011).

12. Bini, E., Buttazzo, G.C. and Buttazzo, G.M. "Rate monotonic analysis: The hyperbolic bound", *IEEE Transactions on Computers*, **52**(7), pp. 933-942 (July 2003).

13. Liu, C. and Layland, J. "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM*, **20**(1), pp. 46-61 (January 1973).

14. Smith, R.M., Trivedi, K.S. and Ramesh, A.V. "Performability analysis: Measures, an algorithm, and a case study", *IEEE Transactions on Computers*, **37**(4), pp. 406-417 (April 1988).

15. Lauzac, S., Melhem, R. and Mosse, D. "An improved rate-monotonic admission control and its application", *IEEE Transactions on Computers*, **52**(3), pp. 337-350 (March 2003).

16. Manimaran, M., Manikutty, A. and Murthy, C.S.R. "A tool for evaluating dynamic scheduling algorithms for real-time multiprocessor systems", *Journal of Systems and Software*, **50**, pp. 131-149 (June 1998).

17. Liu, J.W.S., *Real-Time Systems*, Prentice Hall, ISBN: 0130996513 (2000).

18. Papoulis, A. and Pillai, S.U., *Probability, Random Variables, and Stochastic Processes*, 4th Edition, McGraw-Hill, ISBN: 0071226613 (2002).

19. Johnson, B.W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley Longman Publishing Co., ISBN: 0201075709 (1988).

20. Oh, Y. and Son, S.H. "Allocating fixed-priority periodic tasks on multiprocessor systems", *Real-Time Systems*, **9**(3), pp. 207-239 (November 1995).

21. Meyer, J.F. "On evaluating the performability of degradable computing systems", In *Proceedings of 8th IEEE International Symposium on Fault-Tolerant Computing (FTCS-8)*, pp. 44-49 (June 1978).

22. Ejlali, A., Al-Hashimi, B.M., Schmitz, M.T., Rosinger, P. and Miremadi, S.G. "Combined time and information redundancy for SEU-tolerance in energy-efficient real-time systems", *IEEE Transactions on Very Large Scale Integrated Systems*, **14**(4), pp. 323-327 (April 2006).

23. Ghosh, S., Melhem, R., Mosse, D. and Sarma, J.S. "Fault-tolerant rate-monotonic scheduling", *Real-Time Systems*, Springer, **15**(2), pp. 149-181 (September 1998).

Biographies

Mohsen Bashiri received his BS and MSc degrees in Computer Engineering from Shahid Beheshti University and Sharif University of Technology, respectively, Tehran, Iran. He is currently a Ph.D. Student at the Department of Computer Engineering, Sharif University of Technology. His research interests include real-time embedded systems (RTESs), dependability enhancement in RTESs, and dependability evaluation of RTESs.

Seyed Ghassem Miremadi is a Professor of Computer Engineering at Sharif University of Technology. As fault-tolerant computing is his specialty, he initiated the "Dependable Systems Laboratory" at Sharif University in 1996 and has chaired the Laboratory since then. The research laboratory has participated in several research projects which have led to several scientific articles and conference papers. Dr. Miremadi and his group have done research in Physical, Simulation-Based and Software-Implemented Fault Injection, Dependability Evaluation Using HDL Models, Fault-Tolerant Embedded Systems, Fault-Tolerant NoCs, and Fault-Tolerant Real-Time Systems. He was the Education Director (1997-1998), the Head (1998-2002), the Research Director (2002-2006), and the Director of the Hardware Group (2009-2010) of Computer Engineering Department at Sharif University. During 2003 to 2010, he was the Director of the Information Technology Program at Sharif International Campus in Kish Island. From 2010 to 2012, Dr. Miremadi was the Vice-Chairman of Academic Affairs of Sharif University. He served as the general co-chair of the 13th Int'l CSI Computer Conference (CSICC 2008) and the Executive Chair of the 3rd Engineering Education, 2013. He is currently the Editor of the Journal of Scientia on Computer Science and Engineering, and the Vice-Chairman of Academic Affairs of Sharif University. Dr. Miremadi got his M.Sc. in Applied Physics and Electrical Engineering from Linköping Institute of Technology and his Ph.D. in Computer Engineering from Chalmers University of Technology, Sweden, in 1984 and 1995, respectively. He is a senior member of the IEEE Computer Society, IEEE Reliability Society.