JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 8, No. 1, March 2014, pp. 43-56

SS-DRM: Semi-Partitioned Scheduling Based on Delayed Rate Monotonic on Multiprocessor Platforms

Saeed Senobary*
Imam Reza International University, Mashhad, Iran
s.senobary@imamreza.ac.ir

Mahmoud Naghibzadeh
Dept. of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
naghibzadeh@um.ac.ir

Abstract
Semi-partitioned scheduling is a new approach for allocating tasks on multiprocessor platforms. By splitting some tasks between processors, semi-partitioned scheduling is used to improve processor utilization. In this paper, a new semi-partitioned scheduling algorithm called SS-DRM is proposed for multiprocessor platforms. The scheduling policy used in SS-DRM is based on the delayed rate monotonic algorithm, which is a modified version of the rate monotonic algorithm that can achieve higher processor utilization. This algorithm can safely schedule any system composed of two tasks with total utilization less than or equal to that on a single processor. First, it is formally proven that any task which is feasible under the rate monotonic algorithm will be feasible under the delayed rate monotonic algorithm as well. Then, the existing allocation method is extended to the delayed rate monotonic algorithm. After that, two improvements are proposed to achieve more processor utilization with the SS-DRM algorithm than with the rate monotonic algorithm. According to the simulation results, SS-DRM improves the scheduling performance compared with previous work in terms of processor utilization, the number of required processors, and the number of created subtasks.

Category: Embedded computing

Keywords: Real-time systems; Scheduling algorithms; Delayed rate monotonic; Semi-partitioned technique

I. INTRODUCTION

One of the significant issues in multiprocessor scheduling is how to better utilize processors. The scheduling algorithms are classified as global scheduling and partitioned scheduling. In global scheduling, only one queue exists for the entire system, and tasks can run on different processors. This scheduling algorithm has high overhead. On the other hand, in partitioned scheduling, each processor has a separate queue, and each task may only run on one processor.

The problem of partitioned scheduling is very similar to bin-packing, which is known to be NP-hard [1]. In recent years, a new approach called semi-partitioned scheduling has been proposed. In this approach, most tasks are assigned to a single processor, while a few tasks are allowed to be split into several subtasks and assigned to different processors to improve utilization.

Open Access http://dx.doi.org/10.5626/JCSE.2014.8.1.43 http://jcse.kiise.org

Received 15 January 2014; **Revised** 11 February 2014; **Accepted** 17 February
*Corresponding Author

The delayed rate monotonic algorithm is a modified version of the rate monotonic algorithm, which can achieve higher processor utilization than the rate monotonic algorithm [2, 3]. In this algorithm, a new state called the delayed state is added to the system. All tasks except for the one with lowest priority enter this state as they make a request. It is proven that the delayed rate monotonic can safely schedule any system composed of two tasks with total utilization less than or equal to that on a single processor.

In this paper, the problem of scheduling tasks with fixed priority on multiprocessor platforms is studied. The proposed approach can increase the utilization of some processors to 100% based on the delayed rate monotonic policy. Our novel contributions are as follows:

- First, it is formally proven that any task which is feasible under the rate monotonic algorithm will be feasible under the delayed rate monotonic algorithm as well. Then, we propose a new semi-partitioned scheduling algorithm based on the delayed rate monotonic algorithm called SS-DRM, which adapts a previous allocation method [4] with the delayed rate monotonic algorithm. The scheduling policy [4] is based on the rate monotonic algorithm. The use of the delayed rate monotonic algorithm makes SS-DRM perform better against possible overload compared to the rate monotonic algorithm.
- After that, two improvements are proposed to change the allocation method of SS-DRM in order to improve the processor utilization by using a new pre-assignment function and a new splitting function. Based on these two improvements, SS-DRM can achieve full processor utilization in some special cases. Another advantage of the new pre-assignment function is the number of created subtasks that can be safely scheduled. The simulation results demonstrate that SS-DRM improves the scheduling performance compared with previous work in terms of processor utilization, the number of required processors, and the number of created subtasks.

The rest of this paper is organized as follows. In Section II, related works are discussed, while in Section III, the system model is introduced. Sections IV and V are related to the allocation method and SS-DRM. Finally, an evaluation and conclusions are discussed in Sections VI and VII.

II. RELATED WORKS

The first semi-partitioned technique was used with the EDF algorithm for soft real-time systems [5]. Then, some studies worked on periodic tasks [4, 6-8]. An approach called PDMS_HPTS selects and splits the highest priority task in each processor [8]. The scheduler of processors in this paper is based on the deadline monotonic algorithm, with which the utilization is 60%.

In some other research, the processor utilization is raised up to the well-known L&L bound ($L\&L\ bound = n*(2^{1/n} - 1)$) [7, 9, 10]. The distinctive points of these papers are the allocation method and definition of tasks. For instance, a semi-partitioned scheduling algorithm called SPA2 was proposed [7] for periodic tasks, with which the processor utilization reaches the L&L bound under a rate monotonic policy. The allocation method used in this paper is a worst-fit method, which selects a processor with the least total utilization of tasks assigned to it so far.

Increasing the processor utilization beyond the L&L bound has also been studied in some papers. An approach called pCOMPATS was proposed [6] for multicore platforms. As the number of cores increases, this approach can achieve 100% utilization. When the utilization of each task does not exceed 0.5, this processor utilization is achieved. Therefore, tasks are divided into two groups: light tasks with utilization of lower than 0.5 and heavy tasks with utilization more than or equal to 0.5. Light tasks are assigning by a first-fit method. First, tasks are sorted in ascending order of their periods, and then allocation is done from the front of the queue towards the end. Heavy tasks are assigned like PDMS_HPTS. It means that the highest priority task is selected and then split. The least upper bound to the processor utilization for heavy tasks is 72%.

The scheduler of processors is based on the rate monotonic algorithm, and the schedule-ability test used in this paper is R-bound [11]. This schedule-ability test was proposed by Lauzac et al. [11] in 2003. Their approach is sufficient for the schedule-ability test with good approximation.

The RM-TS approach increases the processor utilization beyond the L&L bound [4]. As in another study [7], the worst-fit method is used in the allocation of RM-TS, but admission control in RM-TS is based on response time analysis, so the processor utilization can be improved. There are also some other works which studied semi-partitioned scheduling based on the earliest deadline first algorithm [12, 13].

The approach of this paper can also increase the processor utilization more than the L&L bound based on delayed rate monotonic policy. The overhead of task splitting is the same as in previous approaches [6, 8]. Based on these studies, the overhead of context switching due to task splitting is negligible. There are also some other approaches that are similar to delayed rate monotonic approach which improves the schedule-ability of the rate monotonic algorithm. This group of scheduling algorithms studied the problem of fixed priority scheduling with a limited-preemptive approach. For instance, an approach was proposed that improves the feasibility of tasks, so the processor utilization could be enhanced [14].

III. SYSTEM MODEL

In the proposed system, tasks are periodic, and their deadline parameters (i.e., relative deadline) are assumed to be equal to their periods. A request of task τ_i ($i = \{1, \dots, n\}$) is called a job. Every task τ_i is modeled by two parameters:

C_i = the worst-case execution time required by task τ_i on each job.

T_i = the time interval between two jobs of task τ_i.

Tasks are considered hard real-time (as opposed to soft real-time [3]). Every job of task τ_i should be completed before the next job of the same task arrives. The response time of a job is the time span from the job arriving up to its execution completion.

Liu and Layland [15] proved that the worst response time of task τ_i occurs when it simultaneously requests with all higher priority tasks. Task τ_i is feasible if its worst response time is lower than or equal to its period. The utilization of task τ_i is defined by $U_i = \frac{C_i}{T_i}$. In this system, we want to assign task set $\Gamma = \{(C_1, T_1), (C_2, T_2), \dots, (C_n, T_n)\}$ to m processors, $\{M_1, \dots, M_m\}$.

Definition 1. *The total utilization of task set Γ is defined by:*

$$U(\Gamma) = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1}$$

Definition 2. *The processor utilization of M_i, $i = \{1, \dots, m\}$ is the total utilization of task set Γ', which is assigned to M_i.*

Definition 3. *The average processor utilization of task set Γ with m processors is*:

$$U_m(\Gamma) = \frac{U(\Gamma)}{m} \tag{2}$$

A semi-partitioned scheduling includes two phases: partitioning and scheduling. In the partitioning phase, all tasks are assigned to the processors. If a task cannot be entirely accommodated by a processor, it is split into two subtasks. These tasks are called split-tasks. The other tasks which are entirely assigned to a processor are called non-split tasks. Non-split tasks always run on a single processor.

The second phase of a semi-partitioned scheduling is the policy to determine how to schedule assigned tasks on each processor. The scheduler used within each processor is based on the delayed rate monotonic algorithm, which is an improved version of the rate monotonic algorithm [15]. In the rate monotonic algorithm, every task with a lower period has higher priority. Further, this algorithm is preemptive. So, the lowest priority task has the highest probability for overrun. In the delayed rate monotonic algorithm, all tasks have a secure delay except for the task with the lowest priority. Therefore, in this scheduler, these two queues are defined for tasks:

- Ready: a task with the lowest priority directly enters this queue, and other tasks enter when their delays are over.
- Delay: all tasks except for the lowest priority one enter this queue as they make a request.

A task in the ready queue has higher priority over all tasks in the delay queue. If no tasks exist in the ready queue, then a task is selected from the delay queue to run next. Tasks in both ready and delay queues are scheduled based on the rate monotonic algorithm. Suppose tasks are sorted in ascending order of their periods. Now, Formula (3) calculates the delay of each task.

For τ_1, with the shortest period, the delay is equal to:

$$T_1 - C_1$$

For $\tau_i\ \forall\ 2 \le i \le n-1$, the delay is equal to:

$$Delay_i = \alpha * T_i * \sum_{j=i+1}^{n} \frac{C_j}{T_j} \tag{3}$$

Parameter α is a constant used to adjust the amount of delay imposed on tasks. It can be a value varying between [0, 1]. The delay of tasks is the maximum if α is equal to one.

It should be mentioned that the concept of delay used by the delayed rate monotonic algorithm is different from zero laxity [16]. If $T_{i,k}$ is the period of the k-th job of task τ_i and $C_{i,k}(t)$ is the remaining worst-case execution time of task τ_i in time t, then the laxity of task τ_i in time t is defined as $T_{i,k} - (t + C_{i,k}(t))$. When this laxity is equal to zero, it is called zero laxity. A task with zero delay (i.e., a task whose state has to change from delay to ready) is not necessarily a zero-laxity task.

IV. ALLOCATION METHOD

An approach called RM-TS was introduced in another study [4] for allocating tasks on multiprocessor platforms with a semi-partitioned technique and the rate monotonic algorithm. In this paper, the schedule-ability test is based on the response time analysis (RTA) [17]. The worst response time of a task under the rate monotonic algorithm can be calculated by Formula (4). Suppose tasks are sorted in ascending order of their periods. Now, the worst response time of task τ_i is:

$$R_i^1 = C_1$$

$$R_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^k}{T_j} \right\rceil * C_j \tag{4}$$

This iteration is terminated when $R_i^{k+1} = R_i^k$. If $R_i^k \le T_i$, then task $\tau_i(C_i, T_i)$ is feasible under the rate monotonic algorithm. In the following, to check the feasibility of tasks, we demonstrate an example.

Example 1. Task set Γ includes three tasks with the parameters shown in Table 1. Now, we want to check whether this task set is schedulable under the rate monotonic algorithm or not.

• First, τ_1 (30, 125).

$$R_1^1 = 30$$
$$R_1^2 = 30\ ;\ R_1^2 = R_1^1$$
$$R_1 \le T_1$$

• Second, τ_2 (48, 130).

$$R_2^1 = 48$$
$$R_2^2 = 48 + \left\lceil \frac{48}{125} \right\rceil * 30 = 78$$
$$R_2^3 = 48 + \left\lceil \frac{78}{125} \right\rceil * 30 = 78\ ;\ R_2^3 = R_2^2$$
$$R_2 \le T_2$$

• Third, τ_3 (92, 275).

$$R_3^1 = 92$$
$$R_3^2 = 92 + \left\lceil \frac{92}{125} \right\rceil * 30 + \left\lceil \frac{92}{130} \right\rceil * 48 = 170$$
$$R_3^3 = 92 + \left\lceil \frac{170}{125} \right\rceil * 30 + \left\lceil \frac{170}{130} \right\rceil * 48 = 248\ ;$$
$$R_3^4 = 92 + \left\lceil \frac{248}{125} \right\rceil * 30 + \left\lceil \frac{248}{130} \right\rceil * 48 = 248\ ;$$
$$R_3^4 = R_3^3$$
$$R_3 \le T_3$$

The worst response time of all tasks is lower than their periods. Therefore, task set Γ is schedulable under the rate monotonic algorithm.

Tasks in RM-TS are classified into two groups.

• Light: task τ_i is light when $U_i \le \frac{\Theta(\tau)}{\Theta(\tau)+1}$

• Heavy: task τ_i is heavy when $U_i > \frac{\Theta(\tau)}{\Theta(\tau)+1}$

$\Theta(\tau)$ is the L&L bound for task set τ.

Table 1. Period and worst-case execution time of tasks

No. task	C_i	T_i
τ_1	30	125
τ_2	48	130
τ_3	92	275

A. Pre-assignment Function for Heavy Tasks

First, tasks are sorted in descending order of their periods, and then a pre-assignment function is done from the end of the queue towards the front. In the pre-assignment function, the heavy tasks which satisfy Condition (5) should be separated and assigned to processors in the pre-assignment phase. These processors are called pre-assign processors. Suppose $\tau_i(C_i, T_i)$ is a heavy task.

$$\Sigma_{j<i} U_j \le (|normal\ processors| - 1) * \Lambda(\tau) \qquad (5)$$

$\Lambda(\tau)$ in Condition (5) determines the parametric utilization bound (e.g., L&L is one of the parametric utilization bounds for rate monotonic algorithm). Condition (5) mentions that task $\tau_i(C_i, T_i)$ can be pre-assigned if all lower priority tasks can be accommodated by all normal processors. Normal processors are processors which do not have a pre-assign task.

B. Assigning Normal Tasks

After the pre-assignment phase, it is time for the remaining tasks. These tasks are called normal tasks. Based on the sorting method, the lowest priority task is in front of the queue. Now, a processor is chosen among normal processors with the worst-fit method. It means the processor with the lowest load factor is chosen for assigning the current task. If all tasks (including the current task) in this processor can be feasible and all can meet their deadlines, we can add the entire task to this processor. As mentioned, such a task is called a non-split task. If an infeasible task exists, then the current task should be split. This task is called a split-task. The first part is added to the processor, and the second part is put back in front of the queue. After adding the first part to a processor, no other task or subtask is assigned to this processor.

If no normal processor exists, then a pre-assign processor is chosen. In this case, the pre-assign processor which has a task with the largest period is chosen for assigning.

This process continues. If the queue of tasks is empty, we can say that all tasks are successfully assigned to processors. Algorithm 1 shows the pseudo-code of the allocation method of RM-TS.

It is easy to derive the following property from Algorithm 1.

Lemma 1. *Suppose task τ_i is split into several subtasks $(\{\tau_i^1, \tau_i^2, \ldots, \tau_i^q\}$. Now, each subtask τ_i^j, $j=\{1,\ldots, q-1\}$ is the highest priority task in its own processor.*

Proof. The proof follows from the method of splitting tasks. □

Now, under the rate monotonic algorithm, the first subtask of a split-task will start its execution as soon as it makes a request, and it will run to the end without any interruption. Other subtasks of a split-task should wait until the prior subtasks are completed before starting to execute. For this concept, the term of release time is used [4]. It means that the release time of each subtask of a split-task is equal to the total worst response time of prior subtasks. This release time is calculated as shown in Formula (6).

Suppose task $\tau_i\,(C_i,\ T_i)$ is split into several subtasks $\{\tau_i^1(C_i^1,\ T_i),\ \ldots,\ \tau_i^q(C_i^q,\ T_i)\}$ and each subtask is assigned to a different processor. Now, the release time of subtask τ_i^j is:

$$\text{Release time}\ (\tau_i^j) = \textstyle\sum_{v=1}^{j-1} R_i^v \tag{6}$$

These subtasks need to be synchronized to execute correctly. For example, subtask τ_i^j cannot start its execution until prior subtasks, $\tau_i^1,\ \tau_i^2,\ \ldots,\ \tau_i^{j-1}$, are finished. According to Lemma 1, the worst response time of a subtask is equal to its worst-case execution time. However, for the last subtask, this may not hold.

A binary search can be done for finding the maximum value for the worst-case execution time of the first subtask of a split-task. With this allocation method, it was proven that all tasks including non-split tasks and split-tasks are feasible under the rate monotonic algorithm [4].

Algorithm 1 Allocation method of RM-TS [4]

```
Input: task set $\Gamma=\{\tau_1,\ldots,\tau_n\}$ and number of processors m.
    // Sort all tasks in descending order of their periods
1. For each task $\tau_i$ in $\Gamma$ do
2.    Delay$_i$ $\leftarrow$ 0
3. End for
    // Phase 1: pre-assign heavy tasks
4. For each task $\tau_i$ in $\Gamma$ do //start from the end of queue
5.    If $\tau_i$ is a heavy task then
6.       If $\sum_{j<i} U_j \le (|\text{normal processors}| - 1) * \Lambda(\tau)$ then
7.          Pick a normal processor ($M_a$)
8.          Assign $\tau_i$ to $M_a$
9.          Mark $M_a$ as `pre-assign processor`
10.      End if
11.   End if
12. End for
    // Phase 2: assigning normal tasks
13. For each task $\tau_i$ in $\Gamma$ do //start from the front of queue
14.   If $\tau_i$ is a normal task then
15.      If exist a normal processor then
16.         Choose a normal processor ($M_a$) with minimum
            load factor.
17.      Else if exist a pre-assign processor then
18.         Choose a pre-assign processor ($M_a$) with the largest
            period of heavy tasks which are assigned to it.
19.      Else
20.         Return "Can't assign task set $\Gamma$ on m processors."
21.      End if
22.      If all tasks in $M_a$ including $\tau_i$, are feasible under rate
         monotonic algorithm, then
23.         Assign $\tau_i$ to $M_a$.
24.      Else
25.         Split $\tau_i$ into two subtasks $\tau^1_i$ and $\tau^2_i$.
26.         Find maximum value for worst case execution time
            of $\tau^1_i$ so that all tasks in $M_a$ including $\tau^1_i$ are
            feasible under rate monotonic algorithm.
27.         Assign $\tau^1_i$ to $M_a$.
28.         Put back $\tau^2_i$ in front of the queue.
29.         Delay$_{i+1}$ $\leftarrow$ Delay$_i$ + $C^1_i$
30.         Mark $M_a$ as `Full`.
31.      End if
32.   End if
33. End for
34. Return "Task set $\Gamma$ is successfully assigned to m
    processors."
```

V. SS-DRM

In this section, we propose a new semi-partitioned scheduling algorithm based on a delayed rate monotonic algorithm called SS-DRM, which adapts the allocation method of RM-TS to the delayed rate monotonic algorithm. In the delayed rate monotonic algorithm, all tasks except for the lowest priority one enter the delay queue before moving to the ready queue. Each task stays for a different time interval in the delay queue. To prove that the feasibility of tasks is exactly the same as in the rate monotonic algorithm, the delay of tasks has been changed in this proposal.

As mentioned, tasks in RM-TS are divided into several groups. In a general classification, we have non-split tasks and split-tasks.

- Non-split tasks: these tasks are each entirely assigned to a processor.
- Split-tasks: these tasks are split into several subtasks and each assigned to a different processor. Considering Formula (6) and Lemma 1, the release time of a subtask is equal to the total worst-case execution time of prior subtasks.

Both heavy tasks and light tasks belong to the same groups, i.e., non-split tasks. It means a non-split task can be heavy or light.

Now, three types of delay are introduced in SS-DRM.

- For τ_n, with the largest period, delay is equal to zero.
- For $\tau_i\ \forall\ 1 \le i \le n-1$

If τ_i is a non-split task, then delay is equal to:

$$Delay_i = T_i - R_i \tag{7}$$

where R_i is the worst response time of task τ_i and is calculated by Formula (4). Suppose task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ is safely scheduled by the rate monotonic algorithm. Now, the delay of task τ_i, $i = \{2, \ldots, n-1\}$ rarely becomes zero. On the other hand, the worst response time of task τ_i is rarely equal to its period. The delay of task τ_1 with the highest priority is not equal to zero unless $U_1 = 1$.

Otherwise, τ_i is the j-th subtask of a split-task, and the delay is equal to:

$$Delay_i = \Sigma_{l=1}^{j-1} C_i^l \tag{8}$$

Considering Algorithm 1, processors are occupied in parallel, so we can express Lemma 2.

Lemma 2. *The lowest priority task in each processor is a non-split task.*

Proof. Considering Phase 1 of Algorithm 1, a heavy task is entirely assigned to a processor if all lower priority tasks can be accommodated by all normal processors. Therefore, pre-assign tasks are the lowest priority tasks in their own processors. Further, they are non-split tasks.

According to Phase 2 of Algorithm 1, normal processors are chosen by a worst-fit method and accommodated in parallel, so one non-split task exists in each normal processor. The non-split task is the lowest priority task in each normal processor, because of the sorting method. Therefore, a non-split task exists in each processor in which it is the lowest priority task. □

Now, by Lemma 3, we show that non-split tasks which are feasible under the rate monotonic algorithm will be feasible under the SS-DRM algorithm as well.

Lemma 3. *Suppose task τ_i (C_i, T_i) is a non-split task and it is feasible under the rate monotonic algorithm. This task is feasible under the SS-DRM algorithm as well.*

Proof. According to Lemma 2, the lowest priority task in each processor is a non-split task. The lowest priority task in SS-DRM has no delay and directly enters the ready queue. We assumed in Lemma 3 that task τ_i is feasible under the rate monotonic algorithm, and it means that $R_i \le T_i$. In SS-DRM, scheduling in both the ready and delay queues is done based on the rate monotonic algorithm. Therefore, if τ_i is the lowest priority task, then it is feasible under SS-DRM, because it directly enters the ready queue.

For other non-split tasks, the delay is considered as in Formula (7). As mentioned earlier, scheduling in the ready queue is based on the rate monotonic algorithm, so the worst response time of task τ_i in the ready queue is equal to R_i. Therefore, the maximum response time of task τ_i under SS-DRM, $R_{i_{SS-DRM}}$, is equal to $Delay_i + R_i$.

$$\tau_i \textit{ is feasible under rate monotonic} \rightarrow R_i \le T_i$$

$$Delay_i = T_i - R_i$$

$$R_{i_{SS-DRM}} = T_i - R_i + R_i$$

The worst response time of task τ_i is at most equal to its period. So, this task is feasible under the SS-DRM algorithm. □

Therefore, it is proven that all non-split tasks, which are feasible under the rate monotonic algorithm, are feasible under SS-DRM as well. So, the response time analysis can be used for admission control of SS-DRM.

In SS-DRM, based on the delayed rate monotonic algorithm, a non-split task which is in the delay queue can run next if the ready queue is empty. According to Formula (8), the delay of a split-task is equal to the total worst-case execution time of prior subtasks. It means that with the delay of subtasks, we express the concept of release time used in another study [4]. Now, a split-task in SS-DRM can run only if its delay is completely finished and it has entered the ready queue.

This is done to synchronize subtasks of a split-task. In this work, the same behavior for a split-task as it is regarded in RM-TS is considered. More details on the feasibility of the split-tasks are presented in a previous study [4]. In this work, to use the advantages of the delayed rate monotonic algorithm, such as its resilience against possible overload, the allocation method of RM-TS is adapted to this algorithm.

A. Changing Assigning Method

To improve the processor utilization under the SS-DRM algorithm, we express two improvements. First, Theorem 1 is stated.

Theorem 1. *Two tasks, τ_1 and τ_2, where τ_1 has higher priority and τ_2 is not the first subtask of a split-task, are feasible under SS-DRM if and only if $U_1 + U_2 \le 1$.*

Proof. As mentioned earlier, the delayed rate monotonic algorithm can safely schedule any system composed of two tasks with total utilization less than or equal to one on a single processor [2]. The proof is based on the fact that task τ_1 (C_1, T_1) is at most delayed for $T_1 - C_1$. With SS-DRM, three types of systems composed of two tasks exist.

- First, a system composed of two non-split tasks.

Suppose two non-split tasks $\tau_1(C_1, T_1)$ and $\tau_2(C_2, T_2)$ are assigned to a processor, $T_2 > T_1$ and $U_1 + U_2 \le 1$. According to Relation (7), the delay of task τ_1 under SS-DRM is equal to $T_1 - R_1$. Considering Formula (4), the worst response time of the highest priority task is equal to its worst-case execution time, so the delay of task τ_1 is equal to $T_1 - C_1$, and this system is scheduled safely under SS-DRM.

- The second type of system is composed of two tasks and includes a non-split task τ_1 (C_1, T_1) and a split-task

$\tau_2^i(C_2^i, T_2)$, $i \in \{2, ..., q\}$, in which τ_2^q is the last subtask of split-task τ_2.

To illustrate this system, suppose τ_2 is split into two subtasks $\tau_2^1(C_2^1, T_2)$ and $\tau_2^2(C_2^2, T_2)$. The first subtask is assigned to a processor. Now, we try to assign the second subtask to processor M_a. The processor M_a consists of a non-split task $\tau_1(C_1, T_1)$, which is assigned earlier. According to Relation (8), the delay of subtask τ_2^2 in processor M_a is equal to C_2^1, which is less than $T_2 - C_2^2$.

We know that $C_2^1 + C_2^2 = C_2$, and in real-time systems the worst-case execution time of a task is at most equal to its period. Now, we have:

$$T_2 \geq C_2$$
$$T_2 \geq C_2^1 + C_2^2$$
$$T_2 - C_2^2 \geq C_2^1 .$$

Therefore, the delay of subtask τ_2^2 in processor M_a is lower than $T_2 - C_2^2$, which is considered in a previous study [2]. So, based on the delayed rate monotonic algorithm, two tasks τ_1 and τ_2^2 are feasible under SS-DRM, if and only if $\frac{C_1}{T_1} + \frac{C_2^2}{T_2} \leq 1$.□

In Theorem 1, it is expressed that because of the third type of system, the task τ_2 should not be a first subtask of a split-task.

• The third type of a system composed of two tasks is a system which consists of a non-split task $\tau_1(C_1, T_1)$ and the first subtask of a split-task, $\tau_2^1(C_2^1, T_2)$.

The delay of this subtask is equal to zero and it directly enters the ready queue. The fact that their total utilization is less than one does not guarantee the feasibility of these tasks under SS-DRM. Therefore, this state is separated in Theorem 1.

Now, we use Theorem 1 to express two improvements in allocation method of SS-DRM. These improvements make SS-DRM achieve higher processor utilization than RM-TS. The first improvement is a new pre-assignment function, and the second improvement is a new splitting function. In the new pre-assignment function of SS-DRM, we try to fully utilize the existing processors by assigning two tasks with total utilization in a specific bounds to each processor.

For each task τ_i, $i = \{1, ..., n\}$ with $U_i \geq 0.5$, if another task exists, i.e., τ_j, $j = \{1, ..., n\} - i$, which satisfies Condition (9), then tasks τ_i and τ_j are entirely assigned to a processor, and no other task or subtask is assigned to this processor.

$$\delta \leq U_i + U_j \leq 1 \tag{9}$$

Condition (9) determines the specific bounds of processor utilization of pre-assign processors. The lower bound, δ, is $\Theta(2) = 2 * (2^{0.5} - 1) \approx 0.82$. The best value for threshold δ in our evaluations is equal to 0.95. It should be mentioned that if very many tasks τ_j exist in which $\delta \leq U_i + U_j \leq 1$, then the best result which has the largest total utilization is chosen. Algorithm 2 illustrates the pseudo-code of the allocation method of SS-DRM which uses the new pre-assignment function.

First, we try to find some groups of two tasks, each with total utilization satisfying Condition (9). Then, each group is assigned to a single processor. These processors are removed from the queue of processors, and hence, no other task or subtask is assigned to these processors.

After the pre-assignment function, it is time for remaining tasks. These tasks are assigned to the remaining processors, just like in the assignment method used in RM-TS.

Algorithm 2 Allocation method of SS-DRM

```
Input: task set $\Gamma=\{\tau_1,...,\tau_n\}$ and number of processors m
    // Sort all tasks in descending order of their periods
    // Phase 1: pre-assignment function of SS-DRM
1.  $M_a \leftarrow 1$
2.  For each task $\tau_i$ in $\Gamma$ do
3.  If $U_i \geq 0.5$ then
4.    If $M_a$ is equal to m then
5.      Break for
6.    End if
7.    Max $\leftarrow U_i$
8.    Index $\leftarrow$ -1
9.    For each $\tau_j$ in $\Gamma - \{i\}$ do
10.     If $\delta \leq U_i + U_j \leq 1$ then
11.       If Max < $(U_i + U_j)$ then
12.         Max $\leftarrow (U_i + U_j)$
13.         Index $\leftarrow j$
14.       End if
15.     End if
16.   End for
17.   If Index is not equal to -1 then
18.     Assign tasks $\tau_i$ and $\tau_{Index}$ to processor $M_a$
19.     Mark $M_a$ as `Full`
20.     $M_a \leftarrow M_a + 1$
21.   End if
22. End if
23. End for

    // Phase 2: assigning the remaining tasks
24. Call allocation method of RM-TS with the remaining
    tasks and $m - M_a + 1$ processors
```

From Algorithm 2, it is easy to derive the following property.

Lemma 4. *The composition of our new pre-assignment* *function and the allocation method of RM-TS preserves the feasibility of the system, similarly to a previous study [4].*

Therefore, Theorem 2 can be stated.

Theorem 2. *If all tasks in task set Γ are successfully partitioned by the allocation method of SS-DRM on m processors and scheduled using the delayed rate monotonic algorithm policy, then all tasks can meet their deadlines.*

Proof. Theorem 2 can be proven by noting that the feasibility of two tasks with total utilization less than or equal to one are guaranteed based on Theorem 1 and Type 1, since these tasks are entirely assigned to the processors and considered as non-split tasks. The feasibility of other tasks which are assigned using the allocation method of RM-TS are guaranteed based on the exact timing analysis method [4]. □

Therefore, using this pre-assignment function, some processors exist in which the processor utilization is raised up to 100%. Another advantage of the new pre-assignment function is the number of created subtasks. The number of created subtasks is at most m − 1 [4]. Based on the new pre-assignment function, after assigning two tasks to one processor, no other task or subtask is assigned to this processor. Therefore, the number of created subtasks in SS-DRM is smaller than that of RM-TS, and due to the number of created subtasks, the additional overhead of the system is reduced.

The second improvement proposed for the allocation method of SS-DRM in order to improve the processor utilization is a new task splitting function. This function uses the second type of system composed of two tasks, which is expressed in Theorem 1. Two assumptions exist in our splitting function.

Assumption 1. The current task in front of the queue of un-assigned tasks is a subtask, i.e., $\tau_i^j(C_i^j, T_i)$, $j \in \{2, ..., q-1\}$, C_i^q is the last subtask of split-task τ_i, and this subtask cannot be entirely accommodated by the host processor.

Assumption 2. The host processor includes only one non-split task τ_x, which is assigned earlier.

If these two assumptions hold, then tasks τ_x and τ_i^j are not feasible under the rate monotonic algorithm. Therefore, considering Algorithm 1 and line 25, the subtask $\tau_i^j(C_i^j, T_i)$, $j \in \{2, ..., q-1\}$ should be split into two subtask $\tau_i^{j'}(C_i^{j'}, T_i)$ and $\tau_i^{j+1}(C_i^{j+1}, T_i)$.

Now, in the new splitting function of SS-DRM, and such a situation, if the total utilization of two tasks, τ_x and τ_i^j, is larger than one, then the worst-case execution time of subtask $\tau_i^{j'}$ is obtained in which the processor utilization of the host processor is equal to one, $U_x + \frac{C_i^{j'}}{T_i} = 1$. Otherwise, when the total utilization of two tasks, τ_x and τ_i^j, is lower than one, two tasks, τ_x and τ_i^j, are feasible under SS-DRM based on Theorem 1, so C_i^j is returned.

Algorithm 3 demonstrates the pseudo-code of the new splitting function of SS-DRM. Three parameters are inputs of this algorithm: first, the current task; second, a task set Γ' which illustrates all assigned tasks to the host processor; and third, the delay of the current task.

As mentioned earlier, the delay of non-split tasks is calculated by Formula (7). This delay is calculated in the scheduling phase when all tasks are clearly assigned to the processors. Therefore, in the partitioning phase, the delay of non-split tasks is assumed to be zero (line 2 of Algorithm 1). On the other hand, for split-tasks, the delay is calculated in the partitioning phase, so their delay is not equal to zero (line 29 of Algorithm 1).

The longest value for the worst-case execution time of the current task by which all tasks in Γ' are feasible is the output of this algorithm. Based on this function, in some processors, SS-DRM can raise the processor utilization to 100%.

Algorithm 3 Splitting function of SS-DRM

```
Input: current task $\tau_i (C_i, T_i)$, task set $\Gamma'=\{\tau_1,...,\tau_{n'}\}$ and $Delay_i$
Output: worst-case execution time of first subtask, $C^1_i$

1.  If $U(\Gamma') + U_i > 1$ then
2.    Upperbound ← $\left[1 - U\left(\Gamma'\right)\right] * T_i$
3.   Else
4.    Upperbound ← $C_i$
5.   End if
6.   If $n'$ is equal to one and $Delay_i$ is not equal to zero then
7.     Return Upperbound
8.   End if
9.  End if
10. Return binary search (0, Upperbound)
```

In line 6 of Algorithm 3, we check two assumptions. First, only one task exists in the host processor; $n' = 1$. Second, the current task is not the first subtask of a split task; $Delay_i \neq 0$. If these two assumptions are correct and $U(\Gamma') + U_i > 1$, then the worst-case execution time of subtask τ_i^1 is calculated, by which the processor utilization of the host processor is equal to one.

In the following, we illustrate an example to show the performance of our splitting function.

Example 2. There are two processors, M_1 and M_2, and three tasks, as shown in Table 2. All of these tasks are

Table 2. Period and worst-case execution time of tasks

No. tasks	C_i	T_i
τ_1	60	100
τ_2	36	64
τ_3	40	48

heavy, but only two tasks τ_2 (36, 64) and τ_1 (60, 100) are pre-assigned to processors M_1 and M_2, respectively. Now, task τ_3 is in front of the queue of un-assigned tasks. No normal processor exists, so a pre-assign processor is selected to assign the current task. Therefore, a processor which has a task with the largest period is chosen for assigning task τ_3, and hence, processor M_2 is selected.

According to row two of Table 3, the current task, τ_3, cannot be entirely accommodated by processor M_2, and we should split task τ_3 into two subtasks $\tau_3^1(C_3^1, T_3)$ and $\tau_3^2(C_3^2, T_3)$.

For finding a suitable value for C_3^1, the splitting function is called. This subtask is the first subtask of split-task τ_3, and its delay is equal to zero. We express this state as the third type of system in Theorem 1. As mentioned, this state is separated from Theorem 1, and only a simple binary search is done to find the worst-case execution time of subtask τ_3^1. According to the binary search, this value is equal to 18. Therefore, subtask τ_3^1 (18, 48) is assigned to the processor M_2 and no other task or subtask is assigned to this processor. After this step, subtask τ_3^2 (22, 48) is in front of the queue of un-assigned tasks. The processor M_1 is chosen for assigning this subtask. Task τ_1 (36, 64) exists in this processor. Considering Table 3 and row four, this subtask cannot entirely be accommodated by M_1.

Therefore, it should be split into two subtasks $\tau_3^{2'}(C_3^{2'}, T_3)$ and $\tau_3^3(C_3^3, T_3)$, and the splitting function is called again to find the suitable value for $C_3^{2'}$. This subtask is the second subtask of τ_3, and its delay is equal to 18. So, the second type of system in Theorem 1 is obtained here. It means that two tasks τ_1 (36, 64) and $\tau_3^{2'}(C_3^{2'}, T_3)$ are feasible under SS-DRM if and only if $U_1 + \frac{C_3^{2'}}{T_3} \leq 1$. Therefore:

$$\frac{36}{64} + \frac{22}{48} = 1.02 > 1$$

$$Upper\ bound = \left(1 - \frac{36}{64}\right) * 48 = 21$$

The output of the splitting function of SS-DRM is equal to 21. It means the value of the worst-case execution time of $\tau_3^{2'}$ is obtained as 21, and the processor utilization of M_1 is equal to one.

If we use a simple binary search which has to be used for RM-TS, the worst-case execution time of $\tau_3^{2'}$ reduces to 14. In this case, the processor utilization of M_1 increases from 0.854 to 1 using the new splitting function and SS-DRM algorithm.

VI. EVALUATION

In this section, we investigate the performance of SS-DRM compared with three prior works. In our evaluations, three different kinds of task sets are randomly generated. The distribution of our random function is uniform. The details of each task set are shown in Table 4.

The total utilization of each task set, $U(\Gamma)$, is a random value between

$$[L\&L\ bound(m)_{m\to\infty} * V_i,\ V_i]\,.$$

We assume that:

$$L\&L\ bound(m)_{m\to\infty} = 0.7$$

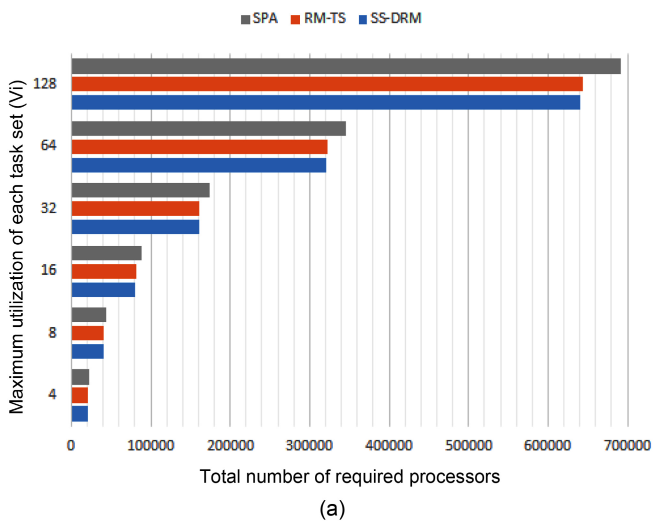
where V_i determines the maximum total utilization of each task set and its value will be one of the following

Table 3. Check feasibility of tasks using response time analysis

Processor M_2 with two tasks τ_1 (60, 100) and τ_3 (40, 48)	
$R_1^1 = 40$ $R_1^2 = 40 \rightarrow$ this task is feasible	$R_2^1 = 60$ $R_2^2 = 60 + \left\lceil \frac{60}{48} \right\rceil * 40 = 140 \rightarrow$ this task is not feasible
Processor M_1 with two tasks τ_2 (36, 64) and τ_2^3 (22, 48)	
$R_1^1 = 22$ $R_1^2 = 22 \rightarrow$ this subtask is feasible	$R_2^1 = 36$ $R_2^2 = 36 + \left\lceil \frac{36}{48} \right\rceil * 22 = 58$ $R_2^3 = 36 + \left\lceil \frac{58}{48} \right\rceil * 22 = 80 \rightarrow$ this task is not feasible

Table 4. Features of each test

No. test	No. task sets	T_i	C_i	$U(\Gamma)$
1	5000	[5, 1000]	[0.01*T_i, T_i]	[0.7*V_i, V_i]
2	5000	[5, 1000]	[0.01*T_i, 0.49*T_i]	[0.7*V_i, V_i]
3	5000	[5, 1000]	[0.5*T_i, T_i]	[0.7*V_i, V_i]



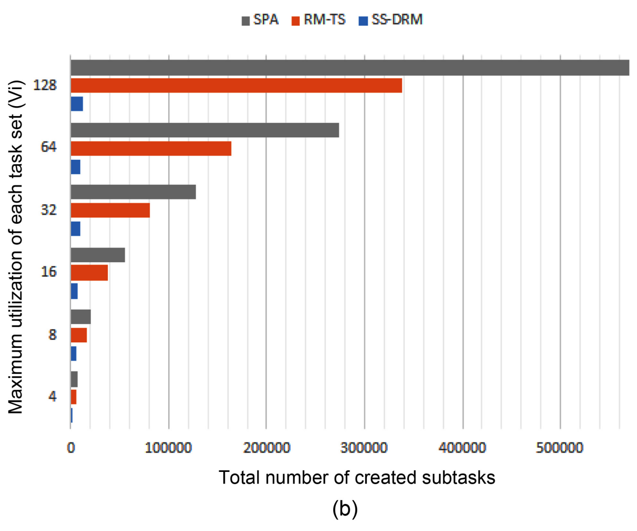


Fig. 1. Experimental result based on variable number of processors for investigating (a) the total number of required processors (b) the total number of created subtasks.

Table 5. Total number of required processors

V_i	SS-DRM	RM-TS	SPA	% extra processors	
				RM-TS	SPA
4	20680	20883	22996	0.97	10
8	40618	40907	44551	0.71	9
16	80601	80990	87861	0.5	8.2
32	160472	161230	173896	0.47	7.7
64	320484	321853	346150	0.43	7.4
128	640603	643263	691877	0.41	7.4

Table 6. Total number of created subtasks

V_i	SS-DRM	RM-TS	SPA	% extra processors	
				RM-TS	SPA
4	1681	6171	6728	72	75
8	5335	16297	20240	68	73
16	7321	38084	54863	80	86
32	9843	80179	127455	87	92
64	9484	164229	274048	94	95
128	12756	337852	569645	96	97

values of the following set: $V_i \in \{4, 8, 16, 32, 64, 128\}$

For instance, when V_i is equal to 4, it means that a task set $\Gamma = \{(C_1, T_1), (C_2, T_2), \ldots, (C_n, T_n)\}$ is generated in which $0.28 \le \sum_{i=1}^{n} \frac{C_i}{T_i} \le 4$. As shown in Table 4, for each V_i, 5000 task sets with random utilization are generated. The number of processors is assumed to be variable so that these task sets can completely be assigned by all scheduling algorithms. Indeed, by assigning these task sets to lower processors, higher utilization can be achieved.

We define another parameter called the average processor utilization to evaluate each algorithm. This parameter is calculated by Formula (10).

$$\text{Average processor utilization} = \frac{\sum_{j=1}^{5000} (U(\Gamma))_j}{\text{total number of required processors}} \quad (10)$$

First, we compare SS-DRM with SPA [7] and RM-TS [4]. In the first row of Table 4, the features of generated task sets are illustrated. For instance, the utilization of each task is randomly selected from 0.01 to 1 and consists of both light tasks and heavy tasks.

The total number of required processors in the system are demonstrated in Fig. 1(a) and Table 5 through the three aforementioned algorithms. Fig. 1(b) and Table 6 show the total number of subtasks which are created by these three algorithms. As shown in Fig. 1(a) and Table 5, by increasing the value of V_i, the difference of the total number of required processors in SS-DRM compared with RM-TS or SPA is decreasing. On the other hand, the difference of created subtasks is increasing as well. For instance, when V_i is equal to 64, then the total number of required processors in SS-DRM is 0.43% lower than that of RM-TS. On the other hand, the total number of created subtasks under SS-DRM, when V_i is equal to 64, is 94% lower than RM-TS.

Considering Fig. 1(b) and Table 6, the difference of the total number of created subtasks under SS-DRM compared with RM-TS or SPA is remarkable. As mentioned earlier, the overhead of task splitting is the same as previous works. Therefore, by creating a lower number of sub-

Table 7. Average processor utilization

V_i	4	8	16	32	64	128
SS-DRM	0.82	0.84	0.85	0.85	0.85	0.85
RM-TS	0.812	0.83	0.84	0.84	0.84	0.845
SPA	0.74	0.76	0.78	0.78	0.78	0.786

Table 8. Average processor utilization

V_i	4	8	16	32	64	128
SS-DRM	0.79	0.82	0.83	0.833	0.83	0.834
pCOMPATS	0.8	0.83	0.837	0.836	0.83	0.83

Table 9. The largest number of required processors

V_i	SS-DRM		pCOMPATS	
	$NT_{Largest}$	$P_{Largest}$	$NT_{Largest}$	$P_{Largest}$
4	1542	5	1276	5
8	4	10	20	10
16	2	19	1	20
32	1	37	12	38
64	2	73	4	76
128	2	145	1	151

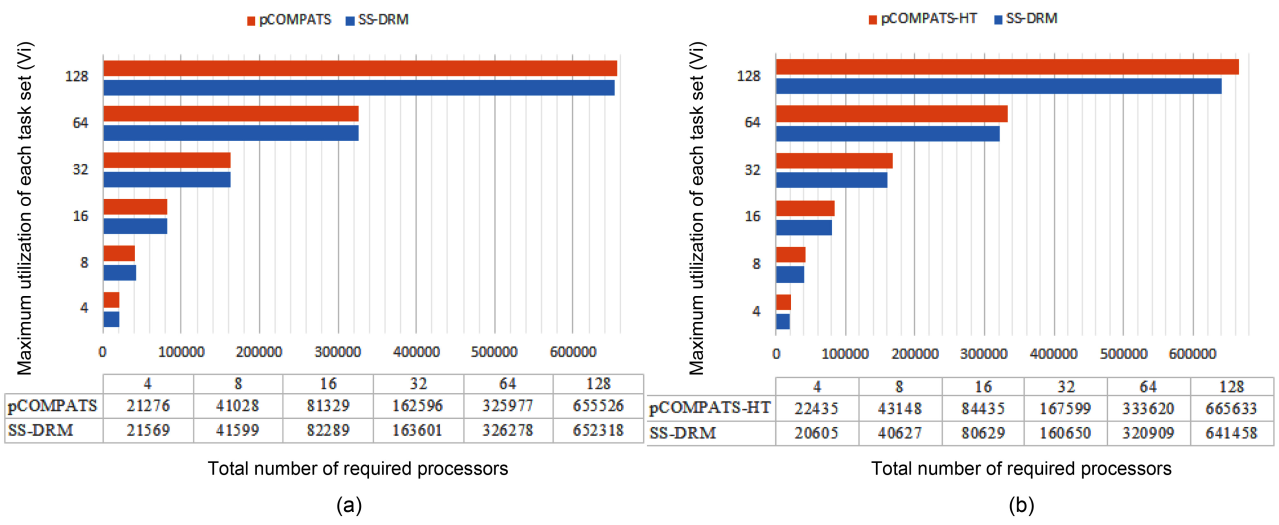
tasks, the overhead of task splitting is reduced in comparison to RM-TS and SPA. This difference occurs because of the pre-assignment function of SS-DRM. As V_i is increasing, the usage of the pre-assignment function of SS-DRM is increasing as well. For instance, when V_i is equal to 8 and 32, then 32% and 55% of existing processors are pre-assigned, respectively. These processors include two non-split tasks in which the processor utilization is between [0.95, 1], and they are scheduled safely with the SS-DRM algorithm. In these evaluations, the threshold δ is assumed to be 0.95.

Table 7 illustrates the average processor utilization of three algorithms. It can be observed that SS-DRM significantly achieves better average processor utilization in comparison to SPA and RM-TS. The average processor utilization of SS-DRM is about 0.58% and 10% more than RM-TS and SPA, respectively.

The average number of callings of our new splitting function in all simulations is about 1.1% of the total number of processors. It means that the processor utilization of 1.1% of the processors is almost equal to one. These processors have two tasks: a non-split task and a split-task. The split-task is not the first part of the task which is split. Their safety is assured by Theorem 1.

The second row in Table 4 shows tasks with random utilization between 0.01 and 0.49 in order for SS-DRM to be comparable with pCOMPATS [6]. As mentioned earlier, tasks in pCOMPATS are divided into two groups: light and heavy. For each group, a separate approach is proposed. A first-fit method is used for allocating light tasks (tasks with utilization lower than 0.5).

In this approach, the lowest priority task is split. pCOMPATS uses R-Bound for the schedule-ability test to cluster compatible tasks together. Table 8 shows the average processor utilization, and Fig. 2(a) shows the total number of required processors. As shown in Fig. 2(a), the performance of pCOMPATS is a little better than SS-DRM. Since the utilization of each task is less than 0.5, it is hard to find a system composed of two tasks in which the total utilization of a system is near one. Therefore, the pre-assignment function of SS-DRM is not used much. But the largest number of required processors in pCOM-



	4	8	16	32	64	128
pCOMPATS	21276	41028	81329	162596	325977	655526
SS-DRM	21569	41599	82289	163601	326278	652318

	4	8	16	32	64	128
pCOMPATS-HT	22435	43148	84435	167599	333620	665633
SS-DRM	20605	40627	80629	160650	320909	641458

Fig. 2. Evaluation of SS-DRM based on the number of required processors in comparison with (a) pCOMPATS (b) pCOMPATS-HT.

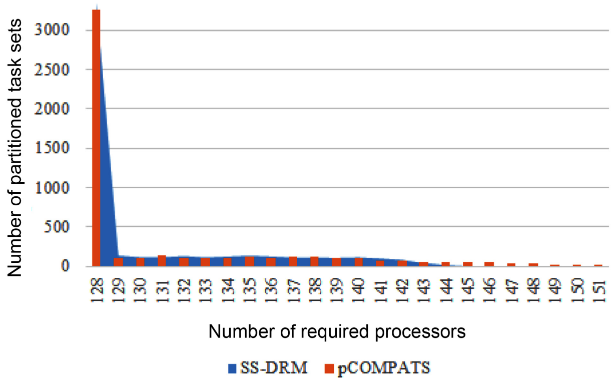


Fig. 3. The range of required processors when V_i is equal to 128.

PATS is more than the largest number of required processors in SS-DRM. Details of the largest number of required processors for each approach are shown in Table 9. $NT_{Largest}$ is the number of task sets which are successfully partitioned on $P_{Largest}$ processors. For instance, the range of required processors when V_i is equal to 128 is illustrated in Fig. 3.

The total number of created subtasks by two approaches is shown in Fig. 4(a). Considering Fig. 4(a), the total number of created subtasks by SS-DRM is more than pCOMPATS, since pCOMPATS uses the first-fit method to allocate compatible tasks to one processor. The disability of pCOMPATS to partition tasks with utilization larger than 0.5 is the main challenge of this approach. The third group of generated task sets is used to compare SS-DRM with pCOMPATS-HT. In [6], pCOMPATS-HT is proposed to assign heavy tasks (tasks with $U_i \geq 0.5$).

In pCOMPATS-HT, tasks are sorted in descending order of their periods. First, one task is assigned to each empty processor. If no empty processor is found, and the current processor is not full, then the current task is split into two subtasks. According to the sorting method, the highest priority task is split. Fig. 2(b) shows the total number of required processors under SS-DRM and pCOMPATS-HT.

Table 10. Average processor utilization

V_i	4	8	16	32	64	128
SS-DRM	0.82	0.84	0.84	0.85	0.85	0.85
pCOMPATS-HT	0.75	0.79	0.8	0.81	0.81	0.815

As shown in Fig. 2(b), the difference of the total number of required processors between two approaches is remarkable. The total number of created subtasks is more proof which indicates the superiority of SS-DRM compared to pCOMPATS-HT. The total number of created subtasks by two approaches is demonstrated in Fig. 4(b).

According to the utilization of each task, most processors have two tasks. The total utilization of two tasks is at least equal to one. Therefore, the pre-assignment function of SS-DRM is not useful here. On the other hand, the splitting function of SS-DRM can be used to split a task in order to fully utilize processors. For instance, when V_i is equal to 32, the processor utilization of almost 20% of the processors is raised up to 100% using the splitting function of SS-DRM. Table 10 illustrates the average processor utilization. Considering Table 10, the average processor utilization is increased by about 11% using SS-DRM.

For evaluation of the delayed rate monotonic algorithm against overload, we randomly generated 1000 task sets for each test. These task sets are generated so that they can be completely assigned to the processors. The allocation method of RM-TS is used for assigning tasks. For producing overload, a task is randomly selected, and its execution time is somewhat increased. Then, the executions of these task sets are simulated under both the rate monotonic algorithm and the delayed rate monotonic algorithm.

For the first evaluation, a task is randomly selected, and its execution time is enlarged so that its overload factor is increased by 0.1, 0.2, and 0.3 for three different

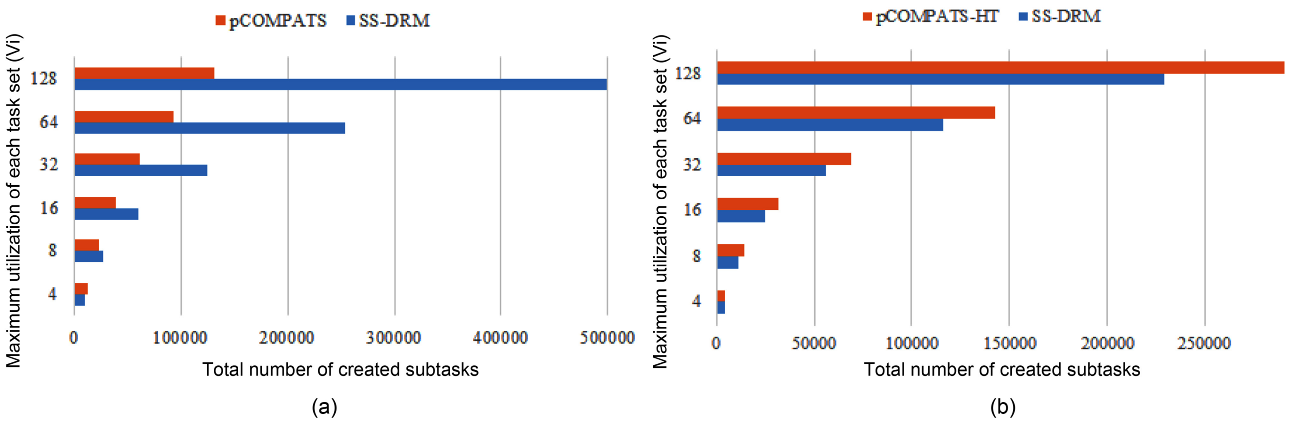


Fig. 4. Evaluation of SS-DRM based on the number of created subtasks in comparison with (a) pCOMPATS (b) pCOMPATS-HT.

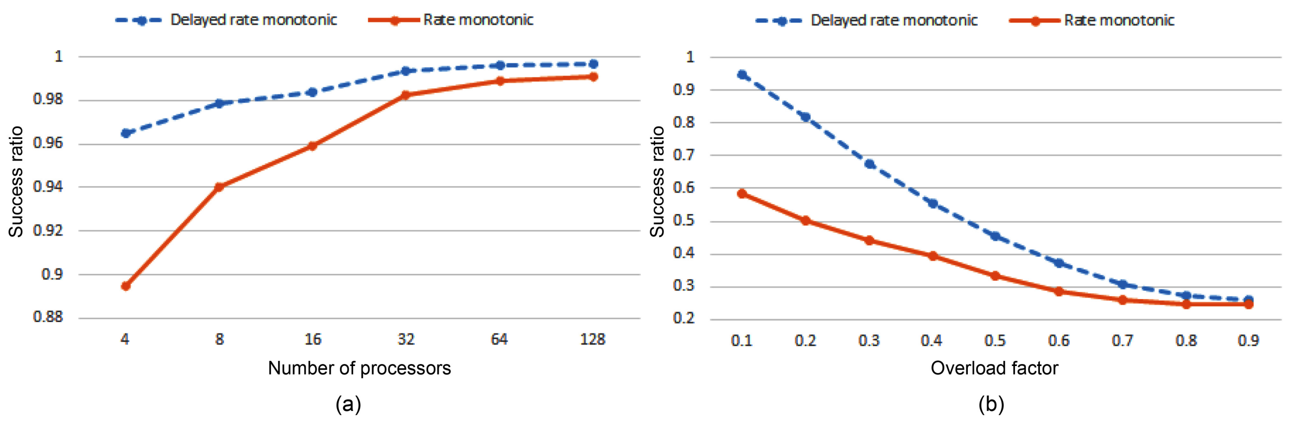


Fig. 5. Resistance of two algorithms against overload in which producing overload on (a) one task from the entire system (b) one task from each processor.

tests. At the end, the success ratio average of these tests is computed for the corresponding number of processors. Fig. 5(a) shows the result of this evaluation. As shown in Fig. 5(a), the success ratio of two algorithms when the number of processors is high becomes closer. This happened because the number of processors which are not overloaded is increased as well.

In the second evaluation, for each processor, a task is randomly selected, and the overload factor is added to its worst-case execution time. Fig. 5(b) shows the result of this evaluation for 32 processors. The success ratio of two algorithms converges together with increasing overload factor, because the number of processors with utilization larger than one is increased as well.

VII. CONSLUSION

In this paper, a new scheduling algorithm called SS-DRM was proposed by combining the delayed rate monotonic algorithm with a semi-partitioned technique. SS-DRM can safely schedule any task set which is feasible under the rate monotonic algorithm.

Our proposed approach can increase the processor utilization beyond the well-known L&L bound. SS-DRM can also be more resilient against possible overload compared with the rate monotonic algorithm. Further, it achieves the same utilization. Then, we expressed two improvements to achieve higher processor utilization in some special cases under the SS-DRM algorithm. In our future works, further improvements will be sought to change the allocation method of SS-DRM to improve the processor utilization.

REFERENCES

1. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: W. H. Freeman, 1979.
2. M. Naghibzadeh and K. H. K. Kim, “The yielding-first rate-monotonic scheduling approach and its efficiency assessment,” *Computer Systems Science & Engineering*, vol. 18, no 3, pp. 173-180, 2003.
3. M. Sabeghi, M. Naghibzadeh, and T. T. Razavizadeh, “A fuzzy algorithm for scheduling soft periodic tasks in preemptive real-time systems,” *New Mathematics and Natural Computation*, vol. 3, no. 3, pp. 371-384, 2007.
4. N. Guan, M. Stigge, W. Yi, and G. Yu, “Parametric utilization bounds for fixed-priority multiprocessor scheduling,” in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium*, Shanghai, China, 2012, pp. 261-272.
5. J. J. Anderson, V. Bud, and U. C. Devi, “An EDF-based scheduling algorithm for multiprocessor soft real-time systems,” in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, 2005, pp. 199-208.
6. A. Kandhalu, K. Lakshmanan, J. Kim, and R. Rajkumar, “pCOMPATS: period-compatible task allocation and splitting on multi-core processors,” in *Proceedings of the IEEE 18th Real Time and Embedded Technology and Applications Symposium*, Beijing, China, 2012, pp. 307-316.
7. N. Guan, M. Stigge, W. Yi, and G. Yu, “Fixed-priority multiprocessor scheduling with Liu and Layland’s utilization bound,” in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Stockholm, Sweden, 2010, pp. 165-174.
8. K. Lakshmanan, R. Rajkumar, and J. P. Lehoczky, “Partitioned fixed-priority preemptive scheduling for multi-core processors,” in *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, Dublin, Ireland, 2009, pp. 239-248.
9. M. Fan and G. Quan, “Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, Dresden, Germany, 2012, pp. 503-508.
10. M. Naghibzadeh, P. Neamatollahi, R. Ramezani, A. Rezae-

ian, and T. Dehghani, “Efficient semi-partitioning and rate-monotonic scheduling hard real-time tasks on multi-core systems,” in *Proceedings of the 8th IEEE International Symposium on Industrial Embedded Systems*, Porto, Portugal, 2013, pp. 85-88.
11. S. Lauzac, R. Melhem, and D. Mosse, “An improved rate-monotonic admission control and its applications,” *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 337-350, 2003.
12. M. K. Bhatti, C. Belleudy, and M. Auguin, “A semi-partitioned real-time scheduling approach for periodic task systems on multicore platforms,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, Riva, Trento, Italy, 2012, pp. 1594-1601.
13. L. George, P. Courbin, and Y. Sorel, “Job vs. portioned partitioning for the earliest deadline first semi-partitioned scheduling,” *Journal of Systems Architecture*, vol. 57, no. 5, pp. 518-535, 2011.
14. R. J. Bril, M. M. van den Heuvel, and J. J. Lukkien, “Improved feasibility of fixed-priority scheduling with deferred preemption using preemption thresholds for preemption points,” in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, Sophia Antipolis, France, 2013, pp. 255-264.
15. C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, 1973.
16. J. Lee, A. Easwaran, I. Shin, and I. Lee, “Zero-laxity based real-time multiprocessor scheduling,” *Journal of Systems and Software*, vol. 84, no. 12, pp. 2324-2333, 2011.
17. J. Lehoczky, L. Sha, and Y. Ding, “The rate monotonic scheduling algorithm: exact characterization and average case behavior,” in *Proceedings of the Real Time Systems Symposium,* Santa Monica, CA, 1989, pp. 166-171.

Saeed Senobary

Saeed Senobary received his B.S. degree as a first rank student in computer engineering from the Institute of Darolfonun Bojnourd, Iran, in 2011, and an M.S. degree in computer software engineering from Imam Reza International University, Mashhad, Iran, in 2013. His research interests include real-time systems, especially real-time scheduling algorithms, parallel programing, and recommendation systems.

Mahmoud Naghibzadeh

Mahmoud Naghibzadeh received his M.S. and Ph.D. degrees in computer science and computer engineering, respectively, both from the University of Southern California, USA. He is now a full professor at the Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran, where he teaches advanced database design and advanced operating systems to graduate students and supervises graduate students. In addition, he is the director of the Knowledge Engineering Research Group (KERG) laboratory. His research interests include the scheduling aspects of real-time systems, grid, cloud, multiprocessors, and multicores. A new subject of his interest is bioinformatics computer algorithms. He has published numerous papers in international journals and conference proceedings as well as eight books in the field of computer science and engineering. Prof. Naghibzadeh was the general chair of an international computer conference and technical chair of two others. He is the reviewer of many journals and a member of many computer societies as well as a senior member of IEEE. He is the recipient of many awards including M.S. and Ph.D. study awards and an outstanding professorship award.