Real-Time Scheduling with Task Splitting on Multiprocessors *

Shinpei Kato and Nobuyuki Yamasaki
School of Science for Open and Environmental Systems
Keio University, Yokohama, Japan
{shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract

This paper presents a real-time scheduling algorithm with high schedulability and few preemptions for multiprocessor systems. The algorithm is based on an unorthodox method called portioned scheduling that assigns each task to a particular processor like partitioned scheduling but can split a task into two processors if there is not enough capacity remaining on a processor. We describe an algorithm for assigning tasks to processors as well as an algorithm for scheduling the assigned tasks on per-processor. The schedulability analysis provides a formula to calculate the upper bound of the schedulable per-processor utilization for the algorithm. We then prove that the least upper bound of the whole system utilization is 50%. *In addition, we propose heuristic procedures to improve schedulability. The simulation results show that the algorithm can often successfully schedule a task set with system utilization much higher than* 50%, *though the least upper bound is* 50%. *We also show that the algorithm achieves higher schedulability with fewer preemptions compared to the existiting algorithms.*

1. Introduction

Recent advances of embedded real-time systems have raised the necessity for more powerful platforms. For instance, humanoid robots often require high-performance and multi-functional computing with low power consumption. Conventional uniprocessor systems are no longer capable of responding all those requirements. Therefore multiprocessor systems such as symmetric multiprocessors (SMP), simultaneous multithreaded (SMT) processors [18], chip multiprocessors (CMP) [14] and chip multithreaded (CMT) processors [16] have considerable attention nowadays. However it is known to be more complicated to schedule recurrent real-time tasks on multiprocessors compared to on uniprocessors. It has been proved that the optimal scheduling algorithms for uniprocessors such as Rate Monotonic (RM) [11] and Earliest Deadline First (EDF) [11] are no longer optimal for multiprocessors [9].

The Pfair scheduling method [6, 5], originally proposed by Baruah *et al.*, brought an optimality of scheduling recurrent real-time tasks on multiprocessors. In Pfair scheduling, tasks are divided into quantum-size pieces so-called *subtasks* and are scheduled based on the deadline of the subtasks. PF [5], PD [6] and PD^2 [2] are known to be optimal Pfair algorithms. LLREF [8] is another optimal scheduling algorithm based on a different technique that does not rely on the quantum-based approach but on the original notion so-called *T-L Plane*. Those sophisticated scheduling algorithms necessarily generate a number of preemptions and task migrations that incur run-time overhead due to their optimality, though they can always achieve the theoretical schedulable system utilization of 100%. Meanwhile simple algorithms such as EDF with a first fit bin-packing algorithm (EDF-FF) [12], EDF with a best-fit bin-packing algorithm (EDF-BF) [12] and EDF-US[1/2] (we omit '[1/2]' hereinafter) [10, 4] are inferior to the sophisticated ones in the schedulability point of view, though they can offer low overhead. The minimum value of the least upper bound for those algorithms is at most 50%. Most of algorithms that belong to so-called the global scheduling scheme or the partitioned schedulinging scheme are cataloged in [7].

Andersson *et al.* proposed the EKG algorithm that improves schedulability with few preemptions [3]. EKG assigns each task to a particular processor like conventional partitioned scheduling algorithms but can split a task into two portions if necessary, then assigns the first portion to the current processor on which the assignment is going and the second portion to the next processor on which the assignment will go. The two portions of a split task are scheduled exclusively. The least upper bound of the schedulable system utilization for EKG depends on the value of a parameter k which should be selected in the range of $1 \leq k \leq M$ where M is the number of processors in a system. A large k results in a higher bound but incurs more preemptions. The bound becomes 66% in the case of $k = 2$ and 100% in the case of $k = M$. Namely EKG is an optimal algorithm in the case of $k = M$, though more preemptions occur instead.

This paper presents a real-time scheduling algorithm with high schedulability and few preemptions for multiprocessors. The algorithm is based on an unorthodox method that assigns each task to a particular processor like partitioned scheduling but can split a task into two processors if there is not enough capacity remaining on a processor.

*This work is supported by the fund of Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists. This work is also supported in part by the fund of Core Research for Evolutional Science and Technology, Japan Science and Technology Agency.

In this viewpoint, the algorithm follows the approach of EKG, however we design the algorithm with different notions in assigning tasks to processors and in scheduling the assigned tasks on per-processor. Our goal is to achieve higher schedulability than simple algorithms such as EDF-FF, EDF-BF, EDF and EDF-US with fewer preemptions than EKG. To distinguish from the conventional partitioned scheduling scheme, we define a scheduling scheme that partitions the tasks but can split some of them into two portions if necessary to improve scheduling as *portioned scheduling*.

The rest of this paper is organized as follows. The next section defines the system model. In Section 3, we describe our algorithm and analyze its schedulability. Also we propose heuristic techniques to improve schedulability in Section 4. Section 5 evaluates the advancement of our algorithm compared to the existing algorithms by simulation. Finally we conclude our work in Section 6.

2. System Model

The system is composed of M processors: $P_1, P_2, ..., P_M$ and a periodic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ is given to the system. Each task τ_i is defined by tupple $(C_i,\ T_i)$ where C_i is a worst-case execution time and T_i is a period, then $U_i = C_i/T_i$ indicates a processor utilization of τ_i. The total utilization of the tasks in an arbitrary task set Λ is defined by $U(\Lambda) = \sum_{\tau_i \in \Lambda} U_i$, namely $U(\Gamma)$ denotes the load of the entire system. A task generates a sequence of jobs periodically. The kth job of τ_i, released at time $r_{i,k}$, is denoted by $\tau_{i,k}$. Its deadline is equal to the release time of the next job, i.e. $d_{i,k} = r_{i,k+1} = r_{i,k} + T_i$. The start time and the finish time of $\tau_{i,k}$ are denoted by $s_{i,k}$ and $f_{i,k}$ respectively.

The algorithm is designed under the following assumption. The system is a memory-shared multiprocessor and each processor shares the code and data. All the tasks are preemptive and independent. Any jobs of the task cannot be executed in parallel. Also no task joins and leaves the system at run-time. Since the subject of this paper is not to design a whole system but a scheduling algorithm, we do not take into account costs of preemptions and task migrations. In the scheduling algorithm point of view, it is more important to consider how often preemptions and task migrations occur. In addition, the cost of switching contexts between processors is almost equal to that of switching contexts within a processor, because we assume that the processors share the code and data. We never suffer from performance deterioration of each task due to a transient degradation of the cache hit ratio caused by task migrations, since the tasks are scheduled based on the worst-case execution time in this paper. Hence we take the number of preemptions as a performance metric for overhead.

3. Algorithm Ehd2-SIP

This section presents a new scheduling algorithm based on the portioned scheduling scheme. The algorithm is composed of the task assignment phase and the task execution phase. The reminder of this section describes an algorithm for each phase. In addition, we provide the schedulability analysis for the proposed algorithm in this section.

Assumption:
task set Γ is sorted so that $T_1 \le T_2 \le ... \le T_N$.
all the per-processor task sets are empty: $\forall m, \Lambda_m = \emptyset$.

1. $i = 1$, $m = 1$ and $U_{ub} = 1$;
2. **if** $U(\Lambda_m \cup \tau_i) \le U_{ub}$
3. $\quad \Lambda_m = \Lambda_m \cup \tau_i$;
4. **else if** $m < M$
5. $\quad U_{rem} = U_{ub} - U(\Lambda_m)$;
6. $\quad C'_i = U_{rem} T_i$ and $C''_i = C_i - C'_i$;
7. $\quad$ **if** $i < N$
8. $\quad\quad U_{ub} = calc\_ub(C'_i, C''_i, T_i, T_{i+1})$;
9. $\quad$ split τ_i into $\tau'_i(C'_i, T_i)$ and $\tau''_i(C''_i, T_i)$;
10. $\quad \Lambda_m = \Lambda_m \cup \tau'_i$ and $\Lambda_{m+1} = \{\tau''_i\}$;
11. $\quad m = m + 1$;
12. **else**
13. $\quad$ the algorithm fails;
14. **if** $i < N$
15. $\quad i = i + 1$ and go back to step 2.;
16. the algorithm successfully exits;

Figure 1. Algorithm SIP

3.1. Task Assignment Phase

We propose an algorithm called **S**equential assignment in **I**ncreasing **P**eriod (SIP) for assigning the tasks to the processors. The pseudo code of SIP is indicated in Figure 1. The algorithm assumes that the tasks are sorted so that $T_i \le T_{i+1}$. A task set composed of the tasks assigned to processor P_m is denoted by Λ_m initialized with $\emptyset$.

The algorithm first initializes the variables (line 1). The upper bound U_{ub} of the schedulable utilization on the first processor P_1 is always 100% (line 1). For the other processors, we need to calculate it based on the formula (line 8). We will explain the formula in Section 3.3 more in detail. Then the algorithm assigns τ_i to P_m if $U(\Lambda_m \cup \tau_i)$, the total utilization of P_m to which τ_i is assumed to be assigned, is less than or equal to U_{ub} (line 3). In the case of $U(\Lambda_m \cup \tau_i) > U_{ub}$, the algorithm checks whether there are remaning processors to assign the tasks (line 4). The algorithm fails if there are no remaining processors (line 13). Otherwise the algorithm splits the task (line 5-11). More specifically, it first calculates the remaining utilization U_{rem} of P_m (line 5). Then it calculates C'_i and C''_i that are reserved capacities for execution of τ_i in every T_i on P_m and P_{m+1} respectively (line 6). When C'_i, C''_i, T_i and T_{i+1} are all acquired, it can calculate the bound for the next processor based on those parameters (line 8). Note that if τ_i is the last task in Γ, we need not calculate the bound for P_{m+1}, because there will be only τ''_i on P_{m+1} and it will never miss the deadline. Finally it splits τ_i into τ'_i and τ''_i (line 9), then assigns τ'_i to P_m and τ''_i to P_{m+1} (line 10). Here 'split' does not mean that a task is really divided into two tasks. τ'_i and τ''_i are nothing but pseudo tasks to reserve a processor time on P_m and P_{m+1} for execution of τ_i. We call τ'_i *portion-1 task* of τ_i and τ''_i *portion-2 task* of τ_i. Then if there are still tasks to be assigned, it repeats the same procedure (line 15).

Assumption:
the following algorithm is dedicated to processor P_m.
$\tau''_i, \tau_{i+1}, \tau_{i+2}, ..., \tau'_j$ are assigned to P_m.

```
1.  when any task is released or is completed on P_m
2.    call schedule_on_P_m;
3.    if τ'_j is included in the released tasks
4.      call schedule_on_P_{m+1};

5.  subroutine schedule_on_P_m
6.    if τ''_i is ready and τ'_i is not being executed on P_{m-1}
7.      execute τ''_i;
8.    else
9.      execute a task with the earliest deadline but τ''_i;
10.     if the chosen task is τ'_j and τ''_j is being executed
        on P_{m+1}
11.       call schedule_on_P_{m+1};
```

Figure 2. Algorithm Ehd2

Otherwise the algorithm successfully exists (line 16).

The task assigment algorithm of EKG does not sort the tasks based on the periods before assigning the tasks. Meanwhile SIP assigns the tasks with holding a condition of $T_s \leq \min\{T_i \mid \tau_i \in \Lambda_m \setminus \tau''_s\}$ where τ''_s is a portion-2 task on processor P_m. This property will be essential for the schedulability analysis described in Section 3.3.

3.2. Task Execution Phase

We propose an algorithm called **E**arliest Deadline First with the **h**ighest-priority **d**eferrable portion-**2** task (Ehd2) for scheduling the tasks which are assigned to each processor P_m by SIP. A scheduling policy of Ehd2 is that the tasks are scheduled according to the EDF policy except that a portion-2 task always has the highest priority on a processor but it cannot be executed if its corresponding portion-1 task is being executed on a neighbor processor.

The algorithm is shown in Figure 2. Here we assume that task τ_i is split into τ'_i and τ''_i, then they are assigned to P_{m-1} and P_m respectively. We also assume that task τ_j is split into τ'_j and τ''_j, then they are assigned to P_m and P_{m+1} respectively. Namely all the tasks between τ_i and τ_j are assigned to P_m. When any task is released or is completed on P_m, the scheduler on P_m is invoked (line 2). The scheduler on P_m executes τ''_i when it is ready and τ'_i is not currently being executed on P_{m-1} (line 6-7). Otherwise it executes a task with the earliest deadline in the rest of the ready tasks but τ''_i (line 9-10). If the task that was executed right now is a portion-1 task and its corresponding portion-2 task is currently being executed on P_{m+1}, the scheduler on P_m calls the scheduler on P_{m+1} to preempt the portion-2 task to execute split portions exclusively (line 11-12). Since we assume a memory-shared multiprocessor, a cost for calling a scheduler on another processor is negligible.

Ehd2 only needs to execute a portion-2 task in priority to other tasks unless its portion-1 task is executed on a neighbor processor in addition to the behavior of EDF. Besides all the preemptions can be handled by the EDF scheduler. All we have to do is to track the executed times of a portion-1 task and a portion-2 task so as not to overrun their reserved capacity on each processor. This operation can be realized by a resource reservation technique [13, 15]. EKG, on the other hand, needs to calculate the additional times so-called *timea* and *timeb* beyond the schedule of EDF for each split task to suspend and resume its execution, every time any task is released on the processors where the split task resides. Therefore we consider that Ehd2 is more straightforward and reasonable than EKG in the computation and implementation points of view.

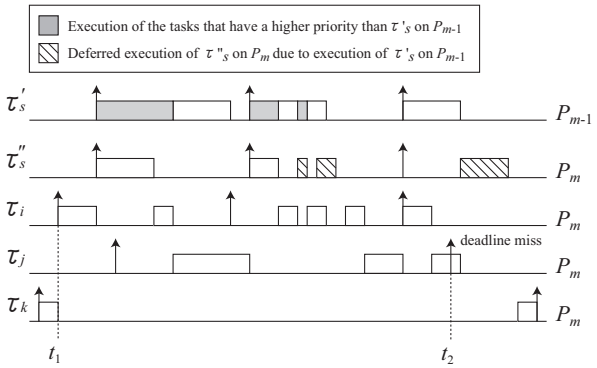


Figure 3. Unfeasible schedule of Ehd2

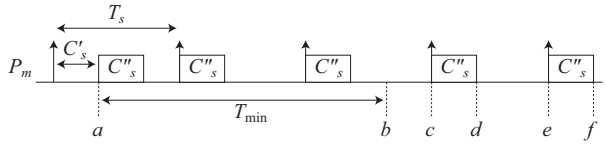
In Ehd2, a portion-2 task is always assigned the highest priority. In other words, it is statically prioritized over the rest of tasks on a processor. Hence Ehd2 is similar to the scheduling method of the EDF-fm algorithm [1] rather than EKG. However, unlike EDF-fm, a period of a portion-2 task is always shorter than that of any other tasks on a processor in Ehd2. This property will help to analyze the schedulable utilization in the next section.

3.3. Schedulability Analysis

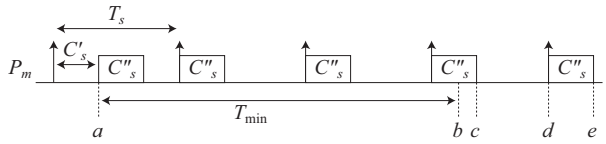
This section analyzes the schedulable utilization for Ehd2-SIP. Since all the tasks but a portion-2 task are scheduled according to the EDF policy on each processor in Ehd2, we need to understand how the portion-2 task behaves. Hereinafter we assume that task τ_s is split into τ'_s and τ''_s, then they are assigned to P_{m-1} and P_m respectively. Now we focus on only the schedulable utilization on P_m.

We show an example of the unfeasible schedule of Ehd2 on P_m in Figure 3. Note that the portion-2 task of τ'_s, i.e. τ''_s, is always assigned the highest priority on P_m unless its portion-1 task, i.e. τ'_s, is being executed on P_{m-1}. Now let us assume that any job $\tau_{j,c}$ missed its deadline as shown in the figure. Let t_1 be the last time at which the processor is idle or a job whose deadline is later than the deadline of $\tau_{j,c}$ is executed. Let t_2 be the time at which $\tau_{j,c}$ missed its deadline, that is, the deadline of $\tau_{j,c}$. In order to have $\tau_{j,c}$ miss its deadline, the following condition needs to be satisfied where $S(t_1, t_2)$ is the total amount of the time at which τ''_s is not executed within $[t_1, t_2]$, namely the total slack amount with respect to τ''_s within $[t_1, t_2]$.

$$S(t_1, t_2) < \sum_{\tau_i \in \Lambda_m \setminus \tau''_s} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$



(a) Case 1: $T_{min} \geq FT_s + C''_s - C'_s$



(b) Case 2: $T_{min} \leq FT_s + C''_s - C'_s$

Figure 4. Worst-case phasing for Ehd2

Since $\lfloor x \rfloor \leq x$ for any x, the following condition must be satisfied in order that $\tau_{j,c}$ may miss its deadline.

$$S(t_1, t_2) < \sum_{\tau_i \in \Lambda_m \setminus \tau''_s} (t_2 - t_1) U_i$$

In other words, any job $\tau_{j,c}$ is guaranteed to meet its deadline if the following condition is satisfied for any (t_1, t_2).

$$\sum_{\tau_i \in \Lambda_m \setminus \tau''_s} U_i \leq \frac{S(t_1, t_2)}{t_2 - t_1}$$

Now we seek to obtain the minimum value of $R(t_1, t_2) = \frac{S(t_1,t_2)}{t_2-t_1}$. Note that we have $t_2 - t_1 \geq \min\{T_i \mid \tau_i \in \Lambda_m \setminus \tau''_s\}$, since t_1 should be before or at the release time of $\tau_{j,c}$. It is obvious that τ''_s consumes the most processor time within $[t_1, t_2]$ when its first job within $[t_1, t_2]$ is deferred for the longest time and its last job within $[t_1, t_2]$ is executed immediately with no preemptions. This phasing results in minimizing $R(t_1, t_2)$. Taking this worst-case phasing into account, we need to consider two cases to obtain the minimum value of $R(t_1, t_2)$. The first one is the case of $T_{min} \geq FT_s + C''_s - C'_s$ and the second one is the case of $T_{min} \leq FT_s + C''_s - C'_s$. Hereinafter we define T_{min} and F as follows for simplicity of description.

$$T_{min} = \min\{T_i \mid \tau_i \in \Lambda_m \setminus \tau''_s\},\ F = \left\lfloor \frac{T_{min} + C'_s}{T_s} \right\rfloor$$

The two cases are shown in Figure 4. We obtain the minimum value of $R(t_1, t_2)$ for each case.

Lemma 1. *The minimum value of $R(t_1, t_2)$ in the case of $T_{min} \geq FT_s + C''_s - C'_s$ is described by Equation* (1) *where G represents $G = F + 1$ for limitation of space.*

$$\min\{R(t_1, t_2)\} = \min\left\{1 - \frac{GC''_s}{T_{min}},\ \frac{G(T_s - C''_s) - C'_s}{GT_s + C''_s - C'_s}\right\} \quad (1)$$

Proof. At first we assume $t_1 = a$ and $t_2 = b$ in Figure 4(a). Then $R(t_1, t_2)$ is described as follows.

$$R(t_1, t_2) = R(a, b) = \frac{S(a, b)}{b - a} = \frac{T_{min} - GC''_s}{T_{min}} = 1 - \frac{GC''_s}{T_{min}}$$

If t_2 is $b < t_2 \leq c$, $R(t_1, t_2)$ can be written as follows where $0 < \alpha \leq c - b$.

$$R(t_1, t_2) = R(a, b + \alpha) = \frac{S(a, b) + \alpha}{b - a + \alpha}$$

Since $\frac{x}{y} < \frac{x+z}{y+z}$ is always true for any $x > 0, y > 0$ and $z > 0$, we have $R(a, b) < R(a, b + \alpha)$. Next we assume $t_1 = a$ and $t_2 = d$. Then $R(t_1, t_2)$ is described as follows.

$$R(t_1, t_2) = R(a, d) = \frac{S(a, d)}{d - a} = \frac{G(T_s - C''_s) - C'_s}{GT_s + C''_s - C'_s}$$

If t_2 is $b \leq t_2 < d$, $R(t_1, t_2)$ can be written as follows where $0 < \beta \leq d - c$.

$$R(t_1, t_2) = R(a, d - \beta) = \frac{S(a, d)}{d - a - \beta}$$

Since $\frac{x}{y} < \frac{x}{y-z}$ is always true for any $x > 0, y > 0$ and $z > 0$, we have $R(a, d) < R(a, d - \beta)$. Also if t_2 is $d < t_2 \leq e$, $R(t_1, t_2)$ can be written as $R(a, d + \gamma)$ where $0 < \gamma \leq e - d$. Then we have $R(a, d) < R(a, d + \gamma)$ by the same reason of $R(a, b) < R(a, b + \alpha)$. At last if t_2 is $e < t_2 \leq f$, $R(t_1, t_2)$ is minimized when $t_2 = f$, since we have $R(a, f) < R(a, f - \delta)$ by the same reason of $R(a, d) < R(a, d - \gamma)$. For any $C'_s > 0$ and $C''_s > 0$, the following condition is obviously satisfied.

$$R(a, d) = \frac{S(a, d)}{d - a} < \frac{T_s - C''_s}{T_s}$$

Meanwhile $R(a, f)$ can be written as follows.

$$R(a, f) = \frac{S(a, d) + (e - d)}{(d - a) + (f - d)} = \frac{S(a, d) + (T_s - C''_s)}{(d - a) + T_s}$$

Hence we have $R(a, d) < R(a, f)$, which implies $R(a, d) < R(a, g)$ for any $d < g$. So the minimum value of $R(t_1, t_2)$ is either of $R(a, b)$ or $R(a, d)$. Let us assume $R(a, b) < R(a, d)$. Then the following condition must be satisfied.

$$\begin{aligned} \frac{T_{min} - GC''_s}{T_{min}} &< \frac{G(T_s - C''_s) - C'_s}{GT_s + C''_s - C'_s} \\ T_{min}(G + 1)C''_s &< GC''_s(GT_s + C''_s - C'_s) \\ T_{min} &< \frac{G(GT_s + C''_s - C'_s)}{G + 1} = \frac{G}{G + 1}(d - a) \end{aligned}$$

Here the above inequation is not always true. It is easy to prove it if we consider the case of $C'_s \simeq 0$ and $C''_s \simeq 0$. In this case, we can approximate $d - a \simeq d \simeq c$. Since $1 > \frac{G}{G+1} > \frac{F}{F+1}$ is always true, we have $\frac{F}{F+1}(d - a) \simeq \frac{F}{F+1}c \simeq b < \frac{G}{G+1}(d - a) < (d - a) \simeq c$. Because of $C'_s \simeq 0$ and $C''_s \simeq 0$, T_{min} can take any length within $[b, c]$. Hence it depends on the length of T_{min} whether the above inequation is true or false. Consequently the minimum value of $R(t_1, t_2)$ is described by Equation (1). □

Lemma 2. *The minimum value of $R(t_1, t_2)$ in the case of $T_{min} \leq FT_s + C''_s - C'_s$ is described by Equation* (2)*.*

$$\min\{R(t_1, t_2)\} = \frac{F(T_s - C''_s) - C'_s}{FT_s + C''_s - C'_s} \quad (2)$$

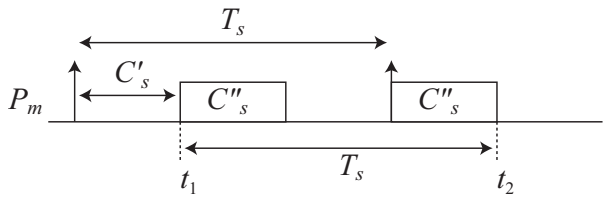


Figure 5. Absolute worst-case phasing

Proof. By the same token as Lemma 1, we assume $t_1 = a$ then find t_2 that minimizes $R(t_1, t_2)$. According to the discussion in Lemma 1, we can easily obtain $R(a, b) > R(a, c)$. We can also easily obtain $R(a, c) < R(a, e) < R(a, d)$. Therefore $R(t_1, t_2)$ is minimized when $t_1 = a$ and $t_2 = c$.

$$R(t_1, t_2) = R(a, c) = \frac{S(a, c)}{c - a} = \frac{F(T_s - C''_s) - C'_s}{FT_s + C''_s - C'_s}$$

Hence the minimum value of $R(t_1, t_2)$ is described by Equation (2). □

Theorem 1. *The upper bound of the schedulable utilization on processor P_m for* Ehd2 *is described by Equation* (3)*. We replace X and Y as follows for limitation of space.*

$$\begin{aligned} U_{ub} &= \frac{C''_s}{T_s} + \begin{cases} X & \textbf{\textit{if}}\ T_{min} \geq FT_s + C''_s - C'_s \\ Y & \textbf{\textit{otherwise}} \end{cases} \qquad (3) \\ X &= \min\left\{ \frac{T_{min} - GC''_s}{T_{min}},\ \frac{G(T_s - C''_s) - C'_s}{GT_s + C''_s - C'_s} \right\} \\ Y &= \frac{F(T_s - C''_s) - C'_s}{FT_s + C''_s - C'_s} \end{aligned}$$

Proof. It is trivial from Lemma 1 and Lemma 2. □

The *calc_ub* function in Figure 1 corresponds to Equation (3) that is a function of C'_s, C''_s, T_s and T_{min}. Since *calc_ub* is called when task τ_s is split, C'_s, C''_s and T_s are already known. Also the tasks are sorted so that $T_i \leq T_{i+1}$. Thereby we can consider T_{min} as T_{s+1}. If the tasks are not sorted in increasing order of the period, we need to find which task will have the shortest period in the tasks assigned to P_m. This procedure is not easy since we do not know how many tasks will be assigned to P_m before the upper bound is computed. That is why we assign the tasks in increasing order of the period.

Theorem 2. *The least upper bound of the schedulable system utilization for* Ehd2-SIP *is* 50%.

Proof. For the case of Lemma 1, it is obvious that $R(a, b)$ is monotonically increasing in T_{min} and F. Since we have the condition of $T_{min} \geq FT_s + C''_s - C'_s$ in this case, $R(a, b)$ is minimized when $T_{min} = FT_s + C''_s - C'_s$ and $F = 1$. Meanwhile $R(a, d)$ can be rewritten as follows.

$$\begin{aligned} R(a, d) &= \frac{G(T_s - C''_s) - C'_s}{GT_s + C''_s - C'_s} \\ &= 1 - \frac{(G + 1)C''_s}{GT_s + C''_s - C'_s} \\ &= 1 - \frac{2C''_s + FC''_s}{(C''_s - C'_s + T_s) + FT_s} \end{aligned}$$

Assumption:
the algorithm is applied only when $i < N$.
let τ''_k be a portion-2 task in Λ_m if there is.

1. $s = i$;
2. **for each** $\tau_j \in \Lambda_m \setminus \{\tau''_k, \tau_i\}$
3. $U_{rem2} = U_{rem} - U_s + U_j$;
4. **if** $U_{rem2} \geq 0$
5. $C'_j = U_{rem2} T_j$ and $C''_j = C_j - C'_j$;
6. $x = calc\_ub(C'_j, C''_j, T_j, T_{min})$;
7. **if** $U_{ub} < x$
8. $U_{ub} = x$, $U_{rem} = U_{rem2}$ and $s = j$;
9. $\Lambda_m = \Lambda_m \cup \tau_i \setminus \tau_s$;
10. $C'_s = U_{rem} T_s$ and $C''_s = C_s - C'_s$;
11. split τ_s into $\tau'_s(C'_s, T_s)$ and $\tau''_s(C''_s, T_s)$;
12. $\Lambda_m = \Lambda_m \cup \tau'_s$ and $\Lambda_{m+1} = \{\tau''_s\}$;

Figure 6. Procedure smb

Since $C''_s \leq T_s$ is always true, $R(a, d)$ is monotonically increasing in F and it is minimized when $F = 1$. Taking $T_{min} = FT_s + C''_s - C'_s \geq T_s$ into account, we can derive the following relations.

$$\begin{aligned} R(a, b) &\geq \frac{T_{min} - 2C''_s}{T_{min}} = 1 - \frac{2C''_s}{T_{min}} \geq 1 - \frac{2C''_s}{T_s} \\ R(a, d) &\geq 1 - \frac{3C''_s}{2T_s + C''_s - C'_s} = 1 - \frac{2C''_s + C''_s}{T_s + (T_s + C''_s - C'_s)} \end{aligned}$$

$C''_s \leq T_s + C''_s - C'_s$ is always true from $C'_s \leq T_s$. Also $\frac{x}{y} > \frac{x+z}{y+w}$ is always true for any $w > z > 0$, so $R(a, b) < R(a, d)$ is derived in the case of minimization. Namely the minimization of U_{ub} occurs for Figure 5. That is, the minimum value of U_{ub} can be described as following U_{lub}.

$$U_{lub} = \frac{C''_s}{T_s} + 1 - \frac{2C''_s}{T_s} = 1 - \frac{C''_s}{T_s}$$

Because of $C'_s = C''_s$ and $C'_s + C''_s = C_s \leq T_s$, C''_s is in the range of $0 \leq C''_s \leq T_s/2$. Hence the absolute minimum value of U_{lub} is 50%. As for the case of Lemma 2, Equation (2) is monotonically increasing in F by the same reason as the above $R(a, d)$, so it is minimized when $F = 1$. This is completely the same phasing as the one in Figure 5. Namely the minimum value of U_{ub}, i.e. the least upper bound, is also 50% in this case. Since the least upper bound of the schedulable per-processor utilization is 50%, that of the schedulable whole system utilization is also 50%. □

4. Technique for Improving Schedulability

According to the previous section, the least upper bound for Ehd2-SIP is only 50%. However, as we mentioned, this bound is obtained only in the special case and Equation (3) is often higher than 50%. Also it can be moreover improved if we efficiently choose a task to split in the task assignment phase because the bound is a function of the period and the execution time of a split task. The necessity condition for the schedulability analysis for processor P_m

Assumption:	
the algorithm is applied only when $i < N$.	
let τ''_k be a portion-2 task in Λ_m if there is.	
1.	**if** $U_{ub} + U_{rem} > 1.0$
2.	$C'_i = U_{rem} T_s$ and $C''_i = C_i - C'_i$;
3.	split τ_i into $\tau'_i(C'_i, T_i)$ and $\tau''_i(C''_i, T_i)$;
4.	$\Lambda_m = \Lambda_m \cup \tau'_i$ and $\Lambda_{m+1} = \{\tau''_i\}$;
5.	**else**
6.	$\Lambda_{m+1} = \{\tau_s\}$;
7.	$U_{ub} = 1.0$;

Figure 7. Procedure sbi

is $T_s \leq \min\{T_i \mid \tau_i \in \Lambda_m \setminus \tau''_s\}$. Since SIP always holds $\max\{T_j \mid \tau_j \in \Lambda_{m-1}\} \leq \min\{T_i \mid \tau_i \in \Lambda_m\}$ that results in satisfying the above inequation, we can choose any task in P_{m-1} to split into P_{m-1} and P_m.

Based on this foundation, we propose a technique called **s**plitting a task that **m**aximizes the **b**ound (smb) to increase the bound of the schedulable utilization. The smb procedure can be combined with SIP by replacing line 9-10 in Figure 1 with the procedure in Figure 6. This procedure seeks a task that can increase the bound if it is split instead of τ_i (line 1-8). Let s be an index of a task that will be split and it is initialized with i (line 1). Then it tests each task τ_j if it meets $U_{rem2} = U_{rem} - U_s + U_j \geq 0$ where U_{rem2} is the remaining utilization of P_m on which the entire portion of τ_s is presumed to be assigned instead of τ_j (line 3-4). If this condition is not satisfied, there is no room for any portion of τ_j to be assigned on P_m. For only the tasks that passed this test, it finds the task that results the highest bound for the next processor (line 5-8). Here we need to remove τ_s from Λ_m and, instead, add τ_i to Λ_m (line 9). This replacement is valid only when $i \neq s$. Then it updates the values of U_{ub}, U_{rem} and s. Finally it splits τ_s (line 11).

There is another approach to improve the schedulability of Ehd2-SIP. As we already discussed, the bound of the schedulable processor utilization for Ehd2 could be 50% in the worst case, whereas that of EDF is always 100%. Therefore the splitting method of Ehd2-SIP may degrade schedulability compared to the nonsplitting method such as EDF-FF and EDF-BF. For instance, consider task set $\Gamma = \{\tau_1(2,5), \tau_2(2,5), \tau_3(6,10), \tau_4(4,11)\}$ with two processors P_1 and P_2. Being as $U_1 + U_2 = 0.8 < 1$ and $U_1 + U_2 + U_3 = 1.4 > 1$, SIP splits τ_3 into P_1 and P_2. Then we have $C'_s = 2$, $C''_s = 4$, $T_s = 10$, $T_{min} = 11$ and $F = 1$ in Equation (3). Because of $T_{min} > FT_s + C''_s - C'_s$, we acquire $U_{ub} \simeq 0.733$. Here SIP cannot assign all the tasks successfully as $U''_3 + U_4 \simeq 0.763 > U_{ub}$. Meanwhile we can successfully assign them if we do not split τ_3 and instead assign it to P_2 as $U_3 + U_4 \simeq 0.963 < 1$, since there will be no portion-2 task on P_2 and the bound will be 100%.

Taking this situation into account, we propose a technique called **s**plitting a task only if the **b**ound is **i**ncreased (sbi). The procedure is shown in Figure 7. It can be combined with SIP by replacing line 9-10 in Figure 1 with the procedure or it can be also combined with smb by replacing line 10-12 in Figure 6 with the procedure. Note that, in the latter case, index i in Figure 7 should be considered to be s. The procedure is straightforward. It splits a task only if $U_{ub} + U_{rem} > 1.0$, that is, only if the schedulability is improved compared to the nonsplitting approach (line 1-4). Otherwise it assigns the task that was supposed to be split by SIP to the next processor (line 5-7).

5. Simulation

Beyond the theoretical schedulability analysis, this section evaluates the actual schedulability of Ehd2-SIP compared to the existing algorithms such as EDF-FF, EDF-BF, EDF, EDF-US and EKG with randomly-generated task sets by simulation. We measure the success ratio[1] for each algorithm as a performance metric of the schedulability. Also we measure the number of preemptions for each algorithm as a performance metric of the scheduling overhead.

5.1. Experimental Setup

The simulations are characterized by the parameters of U_{min}, U_{max}, U_{total} and M. U_{min} and U_{max} are the minimum and maximum values of the per-task utilization in the given task set. U_{total} is the total utilization of the tasks. M is the number of processors. The system utilization is defined by $U_{sys} = U_{total}/M$. For each combination of (U_{min}, U_{max}, M), the simulation goes through from $U_{sys} = 30\%$ to $U_{sys} = 100\%$ with 1000 task sets. Although various combinations of the parameters can be considered, we only attempt the following settings due to the limitation of space. Taking the scale of the current processor for embedded systems into account, we evaluate the cases of $M = 2$, $M = 4$ and $M = 8$. For the range of the per-task utilization, we refer to the implementation of the humanoid robot that we have been developing [17]. The simple activities are realized with only light tasks. Meanwhile, when the quality of the activities needs to be enhanced, it requires heavy tasks. So we prepare two sets of (U_{min}, U_{max}), $(0.01, 0.1)$ and $(0.01, 1.0)$, to simulate the task sets with only light tasks and the ones with both light and heavy tasks. Each task set Γ is generated as follows. A new task is appended to Γ as long as $U(\Gamma) \leq U_{total}$. For each task τ_i, its utilization U_i is computed based on a uniform distribution in the range of $[U_{min}, U_{max}]$. Only the utilization of the task generated at the last is adjusted so that $U(\Gamma)$ gets equal to U_{total}. T_i is determined in the range of [100, 3000] randomly. Then its execution time $C_i = U_i T_i$ is calculated. The range of the periods is set as above, since the feedback interval of the tasks in the humanoid is usually about $1ms \sim 30ms$. Each simulation runs during time interval $[0, \max\{\mathrm{lcm}(\{T_i \mid \tau_i \in \Gamma\}), 2^{32}\})$.

The definition of the successfully scheduled task set depends on each scheduling algorithm. The three algorithms, Ehd2-SIP, EDF-FF and EDF-BF, are designed so that no tasks will miss the deadline once they are successfully assigned to the processors. Hence we define that a task set

[1] The success ratio is defined by $\frac{\text{\# of successfully scheduled task sets}}{\text{\# of scheduled task sets}}$

is said to be successfully scheduled if all the tasks can be assigned to the processors. The upper bound of the per-processor utilization for Ehd2-SIP is calculated by (3). That for EDF-FF and EDF-BF is always 100%. Meanwhile task set Γ cannot be guaranteed to be schedulable (i) by EDF unless $U(\Gamma) \leq M(1 - U_{max}) + U_{max}$ [10] and (ii) by EDF-US unless $U(\Gamma) \leq (M + 1)/2$ [4]. Especially in the presence of heavy tasks, the schedulability of EDF degrades dramatically, which is called Dhall's effect [9]. However it is known that the task set can be often scheduled without missing any deadlines by those algorithms even if $U(\Gamma)$ exceeds the above bounds. Therefore, in those algorithms, we define that a task set is said to be successfully scheduled if all the tasks are scheduled without missing any deadline.

We also evaluate the effectiveness of the procedures proposed in Section 4. The version of Ehd2-SIP with smb and sbi is described as Ehd2-SIP-ss in this experiment. As for EKG, we implemented it with $k = 2$, $k = 4$ and $k = 8$ as the number of the processors increases in the simulations. Note that EKG is an optimal algorithm when $k = M$. The ties are broken arbitrarily for all the algorithms.

5.2. Success Ratio

Figure 8 shows the success ratio for each algorithm. We first discuss the cases with $(U_{min}, U_{max}) = (0.01, 0.1)$. In the cases with $M = 2$ and $M = 4$, there was little difference in performance among the algorithms. Ehd2-SIP-ss and EKG slightly outperformed the other algorithms. The success ratio of EKG(k=M) dropped below 100% before $U_{sys} = 100\%$ in spite of its optimality. This problem happened because the execution time of a task cannot be split less than the minimum time unit 1 in the simulations. Hence each processor inevitably remains a little room unless the remaining processor utilization before splitting task τ_i is an integer multiple of $1/T_i$. As a result, a few tasks may fail to be assigned if U_{sys} is very close to 100%. All the following simulations had the same phenomenon for EKG(k=M).

In the case with $M = 8$, EKG(k=4) outperformed the other algorithms but EKG(k=8). Its maximum schedulable system utilization was 87%, whereas the success ratio of the other algorithms dropped below 100% at $U_{sys} = 85\%$. After the success ratio dropped below 100%, EKG(k=4) still performed well compared to the other algorithms. Nonetheless the difference in performance among the algorithms was still a little. Apart from EKG(k=4) and EKG(k=8), Ehd2-SIP-ss and EKG(k=2) relatively performed well.

We next discuss the cases with $(U_{min}, U_{max}) = (0.01, 1.0)$. The results of these cases were quite different from the previous cases. In contrast to the cases with only light tasks, the cases with both light and heavy tasks clearly showed the difference in performance among the algorithms. In the case with $M = 2$, Ehd2-SIP, Ehd2-SIP-ss and EDF-BF were apparently superior to EDF-FF, EDF, and EDF-US. While the maximum schedulable system utilization of Ehd2-SIP, Ehd2-SIP-ss and EDF-BF algorithms was around 77% $\sim$ 80%, that of EDF-FF, EDF, and EDF-US was around only 53% $\sim$ 57%. An interesting result of this case is that EDF-BF was very competitive to Ehd2-SIP and Ehd2-SIP-ss. It rather outperformed Ehd2-SIP at the system utilization higher than 90%. This fact implies that the proposed task splitting approach does not always improve schedulability. Meanwhile Ehd2-SIP-ss, the improved version of Ehd2-SIP with the procedures of smb and sbi, consistently outperformed EDF-BF. This result proved the effectiveness of smb and sbi.

The case with $M = 4$ made a difference in performance between Ehd2-SIP and EDF-BF. While Ehd2-SIP kept the success ratio 100% to $U_{sys} = 80\%$, EDF-BF dropped the success ratio below 100% at $U_{sys} = 73\%$. It is remarkable that Ehd2-SIP performed competitively to EKG(k=2) with fewer preemptions (we will show the number of preemptions afterwards in the next section). Combining the procedures of smb and sbi, Ehd2-SIP-ss consistently outperformed EKG(k=2). Also EDF and EDF-US were apparently inferior to EDF-FF unlike the case with $M = 2$.

Finally in the case with $M = 8$, we can observe the difference in performance among the algorithms clearly. As in the case with $M = 4$, Ehd2-SIP, Ehd2-SIP-ss and EKG(k=2) were competitive, though Ehd2-SIP-ss was slightly better than the other two algorithms. The maximum schedulable system utilization of Ehd2-SIP-ss was 80%, whereas that of Ehd2-SIP and EKG(k=2) was 77%. Also Ehd2-SIP-ss outperformed the others at any system utilization after the success ratio dropped below 100%. EKG(k=4) kept the success ratio 100% to $U_{sys} = 90\%$, however its number of preemptions was much larger than the other algorithms as we will show in the next section. The performance of EDF-BF and EDF-FF was almost same as the case with $M = 4$. In contrast, the performance of EDF and EDF-US further degraded. Their maximum schedulable system utilization was around only 45% $\sim$ 50%.

We observe that we cannot acquire the advancement of the proposed algorithm very much for task sets with only light tasks. Meanwhile we can observe the effectiveness of the proposed algorithm in the presence of heavy tasks. Ehd2-SIP and Ehd2-SIP-ss consistently achieved high schedulability regardless of the number of processors, whereas the other algorithms deteriorated schedulability as the number of processors increased. EKG also performed well but it may suffer from run-time overhead caused by a number of preemptions as we will show in the next section.

5.3. Number of Preemptions

Figure 9 shows the relative number of preemptions for each algorithm to the number of preemptions for Ehd2-SIP. We measured and calculated the average numbers of preemptions for each algorithm only if the success ratios of both the measured algorithm and Ehd2-SIP were 100%. We first discuss the case with $(U_{min}, U_{max}) = (0.01, 0.1)$. While the system utilization was less than 50%, which means that only the first processor was used in those algorithms, they had the same number of preemptions. Once the system utilization exceeded 50%, they scheduled the tasks differently and, as a result, made a difference in the numbers of

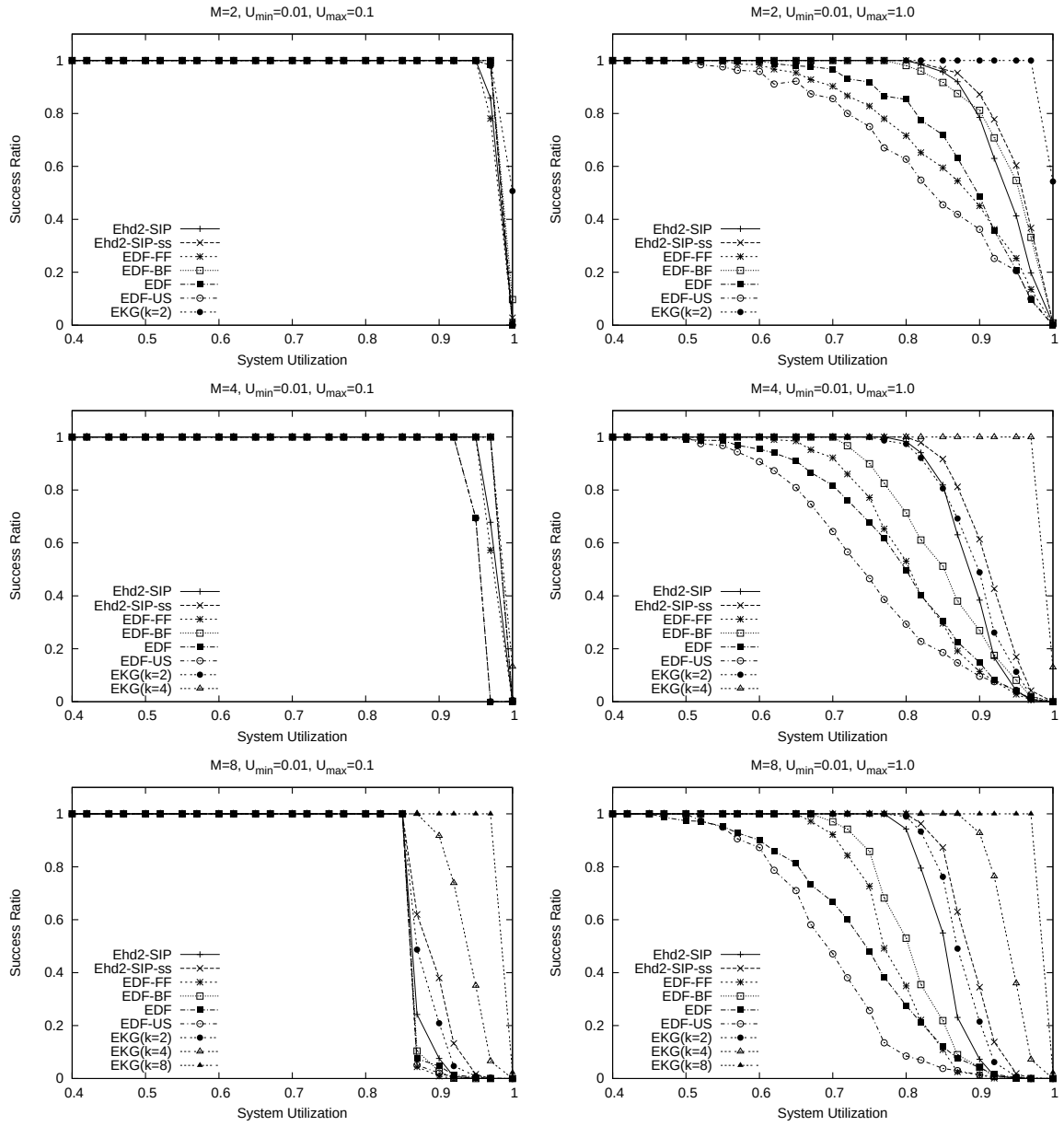


Figure 8. Success ratio as function of system utilization

preemptions. EKG(k=2) caused 1.2 ∼ 1.4 times as many preemptions as Ehd2-SIP at $U_{sys} > 50\%$, though it could optimally schedule the tasks in this case. Meanwhile EDF-FF and EDF-FF had fewer preemptions than Ehd2-SIP at $U_{sys} < 90\%$. They had about 0.8 times as many preemptions as Ehd2-SIP at $U_{sys} = 65\%$. However they caused more preemptions than Ehd2-SIP at $U_{sys} > 90\%$. EDF and EDF-US had fewer preemptions than the other algorithms while U_{sys} was small. However the number of preemptions for EDF increased as U_{sys} increased. It finally caused about 50% more preemptions than Ehd2-SIP at $U_{sys} = 95\%$. Ehd2-SIP-ss also had more preemptions than Ehd2-SIP. This fact implies that smb incurred more preemptions, because sbi never increases preemptions.

In the case with $M = 4$, Ehd2-SIP, Ehd2-SIP-ss, EDF-FF and EDF-BF were competitive. By the same reason that Ehd2-SIP achieved fewer preemptions than EDF-FF and EDF-BF at $U_{sys} > 90\%$ in the case with $M = 2$, it achieved slightly fewer preemptions than those algorithms at $U_{sys} > 45\%$ in this case. Also it slightly outperformed Ehd2-SIP-ss. EKG(k=2) caused 1.4 ∼ 1.5 times as many preemptions as Ehd2-SIP. EKG(k=4) also caused 1.5 ∼ 1.9 times as many preemptions as Ehd2-SIP. We realized that the number of preemptions for EKG transiently increased when any processor is just full of the tasks. For example, at $U_{sys} = 50\%$ (2/4) and $U_{sys} = 75\%$ (3/4), namely when the second processor and the third processor are full of the tasks respectively, both EKG(k=2) and EKG(k=4) transiently caused more preemptions. Like the case with $M = 2$, EDF and EDF-US caused more preemptions as the system utilization increased. It finally caused about three times as many preemptions as Ehd2-SIP. We can observe

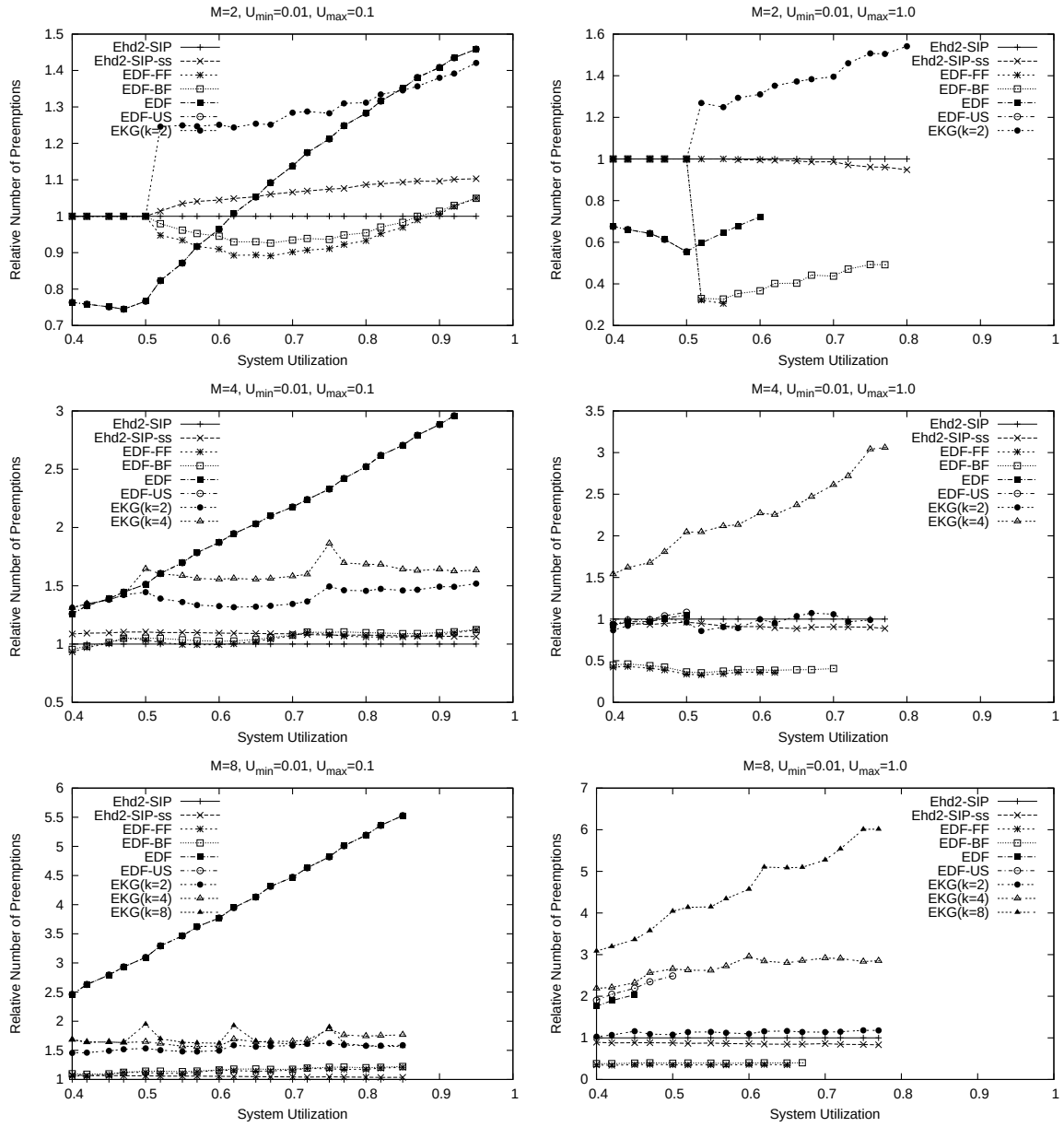


Figure 9. Number of preemptions as function of system utilization

almost the same results in the case with $M = 8$ as the case with $M = 4$. In this case, however, EDF finally incurred about five times and half as many preemptions as Ehd2-SIP. EKG(k=8) transiently caused twice as many preemptions as Ehd2-SIP in the worst case, at $U_{sys} = 50\%$ (4/8), $U_{sys} = 62.5\%$ (5/8) and $U_{sys} = 75\%$ (6/8). Ehd2-SIP, Ehd2-SIP-ss, EDF-FF and EDF-BF, were still competitive.

Next we discuss the cases with $(U_{min}, U_{max}) = (0.01, 1.0)$. The results were quite different from the cases with $(U_{min}, U_{max}) = (0.01, 0.1)$. It is remarkable that EDF-FF and EDF-BF highly outperformed the other algorithms. Especially in the case with $M = 2$, EDF-FF and EDF-BF occured only 0.3 times as many preemptions as Ehd2-SIP at the system utilization slightly higher than 50%. This fact implies that the proposed task splitting approach incurred more preemptions in the presence of heavy tasks. If there are only light tasks, the execution time of each task is relatively short, thereby the schedule times of a portion-1 task and portion-2 task tend not to be overlapped. If there are heavy tasks, on the other hand, the execution time of each task can be long, thereby the schedule times of the split portions tend to be overlapped as the second invocation in Figure 3. This overlapping obviously incurs more preemptions. That is why Ehd2-SIP caused much more preemptions than EDF-FF and EDF-BF compared to the cases with $(U_{min}, U_{max}) = (0.01, 0.1)$. It is also remarkable that Ehd2-SIP-ss had slightly fewer preemptions than Ehd2-SIP despite of its higher schedulability. This superiority is given by sbi that can reduce unnecessary task splits in Ehd2-SIP. Namely sbi has an effectiveness of improving schedulability as well as reducing preemptions.

In the cases with $M = 4$ and $M = 8$, EDF-FF and EDF-

BF still consistently incurred only half as many preemptions as Ehd2-SIP. Ehd2-SIP, Ehd2-SIP-ss and EKG(k=2) were competitive. An important result is that Ehd2-SIP-ss achieved slightly fewer preemptions than EKG(k=2) all through the simulations, though it achieved higher schedulability as we showed in the previous section. As for EKG with large K, EKG(k=4) incurred about three times as many preemptions as Ehd2-SIP and EKG(k=8) incurred about six times as many preemptions as Ehd2-SIP respectively in the worst case. EDF and EDF-US were competitive to each other and they caused at most about twice as many preemptions as Ehd2-SIP in the case with $M = 8$.

6. Conclusion

We proposed the Ehd2-SIP algorithm for scheduling recurrent real-time tasks on multiprocessors with high schedulability and few preemptions. The schedulability analysis provided the formula to calculate the upper bound of the schedulable per-processor utilization, and then proved that the least upper bound of the schedulable whole system utilization is 50%. We also proposed the smb and sbi procedures to improve schedulability.

The simulation results showed that Ehd2-SIP with smb and sbi, denoted by Ehd2-SIP-ss, consistently achieves higher schedulability than the existing algorithms but EKG with large k. In addition to the superiority in schedulability, Ehd2-SIP-ss relatively achieves fewer preemptions especially in the presence of only light tasks. Throughout the simulations, we found the trade-off between schedulability and preemptions. EKG with large k can achieve higher schedulability than Ehd2-SIP-ss, though it causes more preemptions. EDF-FF and EDF-BF can reduce preemptions compared to Ehd2-SIP-ss, though they are inferior to Ehd2-SIP-ss in schedulability. In consequence, we consider that Ehd2-SIP-ss can be a good choice for system designers, since it performs well in both aspects.

We give an insight of our future work here. In order to prove the total superiority of the proposed algorithm in real environments, we are implementing the algorithm on our original operating system. Then we will evaluate the schedulability of our algorithm with whole run-time overhead. Also, in a theoretical aspect, we are considering to improve the algorithm so as to reduce unnecessary preemptions including migrations between processors.

References

[1] J. H. Anderson, V. Bud, and U. C. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proc. of the Euromicro Conf. on Real-Time Systems*, pages 199–208, 2005.

[2] J. H. Anderson and A. Srinivasan. Early-Release Fair Scheduling. In *Proc. of the Euromicro Conf. on Real-Time Systems*, pages 35–43, 2000.

[3] B. Andersson and E. Tovar. Multiprocessor Scheduling with Few Preemptions. In *Proc. of the IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

[4] T. P. Baker. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE TRANS. ON PARALLEL AND DISTRIBUTED SYSTEMS*, 16:760–768, 2005.

[5] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate Progress: A Notion of Fairness in Resource Allocation. *Algorithmica*, 15:600–625, 1996.

[6] S. Baruah, J. Gehrke, and C. G. Plaxton. Fast Scheduling of Periodic Tasks on Multiple Resources. In *Proc. of the Int'l Parallel Processing Symp.*, pages 280–288, 1995.

[7] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. H. Anderson, and S. Baruah. A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms. In *Handbook of SCHEDULING Algorithms, Models and Performance Analysis*, pages 30.1–30.19. CHAPMAN & HALL/CRC, 2004.

[8] H. Cho, B. Ravindran, and E. D. Jensen. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *Proc. of the IEEE Real-Time Systems Symp.*, pages 101–110, 2006.

[9] S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26:127–140, 1978.

[10] J. Goosens, S. Funk, and S. Baruah. Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems*, 25:187–205, 2003.

[11] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20:46–61, 1973.

[12] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utlization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. *Real-Time Systems*, 28:39–68, 2004.

[13] C.W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Application. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, 1994.

[14] K. Olukotun, B. A. Nayfe, L. Hammond, K. Wilson, and K. Chang. The Case for a Single-Chip Multiprocessor. In *Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 2–11, 1996.

[15] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 298–307, 1997.

[16] L. Spracklen and S. G. Abraham. Chip Multithreading: Opportunities and Challenges. In *Proc. of the IEEE Int'l Symp. on High-Performance Computer Architecture*, pages 248–252, 2005.

[17] T. Taira, N. Kamata, and N. Yamasaki. Design and Implementation of Reconfigurable Modular Robot Architecture. In *Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 3566–3571, 2005.

[18] D. M. Tullsen, S. J. Eggers, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In *Proc. of the Annual Int'l Symp. on Computer Architecture*, pages 191–202, 1996.