Semi-Partitioning Technique for Multiprocessor Real-Time Scheduling *

Shinpei Kato and Nobuyuki Yamasaki
Department of Information and Computer Science
Keio University, Yokohama, Japan
{shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract

A semi-partitioning technique is presented for efficient scheduling of sporadic task systems on multiprocessors. The presented technique performs in the same manner as the traditional partitioning, as long as tasks are successfully partitioned, but a task is allowed to be shared among multiple processors for its execution, if a spare capacity of every individual processor is not enough to accept the full execution of the task, beyond partitioning. We also design an EDF-based algorithm in which the semi-partitioning technique is combined. According to the simulation results, the presented semi-partitioning approach improves schedulable multiprocessor utilization by 10 to 30%, over the traditional partitioning approach.

1 Introduction

Multiprocessor scheduling is generally classified into partitioning and global scheduling. In the partitioning scheme, each task is scheduled on a particular assigned processor and never migrates among processors. In the global scheduling scheme, on the other hand, all eligible tasks are stored in a unique priority queue; the global scheduler selects for execution the same number of the highest priority tasks as processors.

A main advantage of partitioning is that it reduces a problem of multiprocessor scheduling into a set of uniprocessor one, once tasks are partitioned, and thus the complexity of implementation and analysis is relaxed. In addition, the overhead of inter-processor communications and local cache misses is far smaller than global scheduling, since no migrations occur in partitioning. That is mainly why practical real-time systems prefer partitioning to global scheduling. However, the partitioning scheme has disadvantage in schedulability. According to Lopez *et al.* [12], partitioning may cause deadlines to be missed on m processors, if the total utilization of tasks exceeds $(\beta m+1)/(\beta+1)$, where $\beta = \lfloor 1/\alpha \rfloor$ and α is a maximum utilization of every individual task. Letting $\alpha = \beta = 1$ and $m \to \infty$, deadlines may be missed if multiprocessor utilization reaches slightly greater than 50%.

The global scheduling scheme is more attractive in schedulability. In fact, there exist optimal algorithms, such as Pfair [5, 4] and LLREF [6], which guarantee all tasks to be successfully scheduled even if multiprocessor utilization is 100%. There also exist other efficient algorithms, such as EDZL [7] and EDCL [9], which offer excellent real-time performance with lower run-time overhead. However, the overhead of inter-processor communications or local cache misses due to migrations is often criticized.

Recent work [1, 3, 2, 11, 10] have made a new class of multiprocessor scheduling, so-called semi-partitioning. In semi-partitioning, most of tasks are assigned to particular processors as partitioning, but the rest of tasks are allowed to migrate between assigned two processors to improve multiprocessor utilization. In other words, those tasks are split into two processors. As a result, it generally performs better than partitioning, while the number of migrations is much smaller than global scheduling.

In this paper, we propose a new approach of semi-partitioning, and design an EDF-based algorithm based on the proposed semi-partitioning technique. Like the previous approaches, the proposed approach splits tasks but differs in that the tasks are never split as long as they can be partitioned. Thus, it completely succeeds the property of the traditional partitioning technique.

2 Semi-Partitioning Technique

The semi-partitioning technique is a derivative of the traditional partitioning technique. Each task is assigned to a particular processor, as long as the total utilization of the processor does not exceed its schedulable bound. In the traditional partitioning, a task is not schedulable, if the spare capacity of every individual processor is not enough to accept full execution of the task. In the presented semi-partitioning, on the other hand, such an unpartitionable task

*This work was supported in part by Grant in Aid for the Global Center of Excellence Program for "Center for Education and Research of Symbiotic, Safe and Secure System Design" from Ministry of Education, Culture, Sport, and Technology in Japan. This work is also supported in part by the fund of Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists.

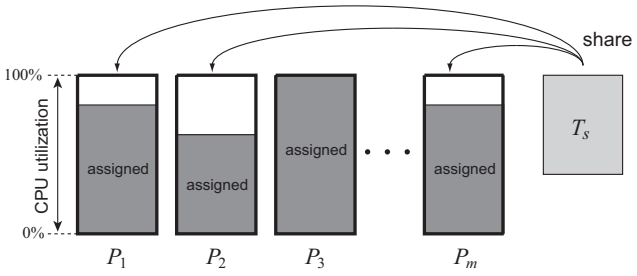

Figure 1. Concept of semi-partitioning

can be shared for its execution among multiple processors, beyond partitioning. In run-time scheduling, the shared tasks are allowed to migrate among the assigned multiple processors, while the partitioned tasks are executed on the particular processors without migrations.

Figure 1 shows the concept of semi-partitioning. No processors have spare capacity to accept full portion of a task T_s, and therefore it is for instance shared among three processors P_1, P_2, and P_m. In other words, T_s is "split" into the three processors: the entire portion of the task is not partitioned but the share is partitioned. The amount of each share is such a value that fills the assigning processor to capacity without timing violations.

In recent research, this kind of task splitting approach has been successfully used in multiprocessor scheduling [1, 3, 2, 8, 11, 10]. In the previous approaches, tasks are willingly split even though a task set can be successfully partitioned without splitting, and thus additional scheduler invocations and migrations are generated over the traditional partitioning. The presented semi-partitioning, on the other hand, performs in the completely same manner as the traditional partitioning, as long as a task set is successfully partitioned. Hence, the simplicity of the partitioning is succeeded as much as possible in the presented approach.

2.1 Scheduling Algorithm

In this section, we design a scheduling algorithm, called Earliest Deadline and Highest-priority Split (EDHS), based on the semi-partitioning technique. The scheduling policy of EDHS is straightforward.

- Migrating (shared) tasks are assigned the global and static highest-priority over partitioned tasks.
- Every job of a migrating task begins on the first processor to which it is assigned, and it is sequentially migrated onto the next processor when the assigned capacity is consumed on each processor.
- Partitioned tasks are then scheduled by EDF.

Figure 2 indicates a counterexample of scheduling a migrating task. The migrating task has share on P_1, P_2, and P_m. Then, it is executed by priority on each processor within the assigned capacity. Once the capacity is exhausted, it is migrated onto the next processor and is executed by priority again. For ease in guarantee of timing constraints, this paper considers such an algorithm that does not receive more than one shared task on one processor.

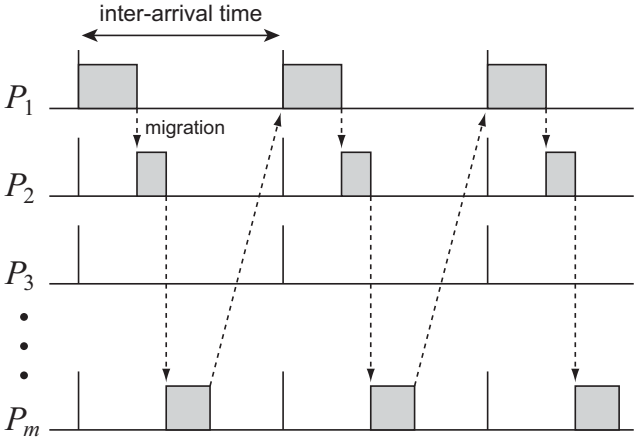

Figure 2. Scheduling of a migrating task

The simplified policy of EDHS offers such benefits that (i) the implementation complexity of scheduler is relaxed, and (ii) the number of migrations as well as preemptions is reduced, over the prior algorithms. For instance, EKG [3, 2] must generate two migrations and preemptions in every interval of successive job arrivals, to schedule shared tasks. EDDHP [8] EDDP [10], and RMDP [11] also produce unbounded preemptions and migrations, particularly when shared tasks have long inter-arrival time.

The issue of concern here is how to assign the capacity to each shared task on each processor. The capacity must be computed so as not to cause timing violations of already-partitioned tasks. We therefore consider scheduling analysis in the next section.

2.2 Scheduling Analysis

This section derives a schedulable condition for EDHS. All we need to do for this objective is to find the maximum execution amount c'_s that a job in a shared task T_s can consume on each processor without timing violations. Let d be the deadline of any job J in some non-shared (fixed) task T_i and t be the time instant at which J is released. Then, we derive a condition for c'_s to cause the deadline of J to be missed.

We first consider the maximum total amount $W_s(t, d)$ of processor time consumed by T_s on the processor in the interval $[t, d)$. For sporadic task systems, it is clear that the execution amount of T_s is maximized when it is periodically released at the minimum interval p_s. Thus, we reduce the problem of sporadic task systems to that of periodic ones. Given the case of periodic task systems, since T_s is assigned

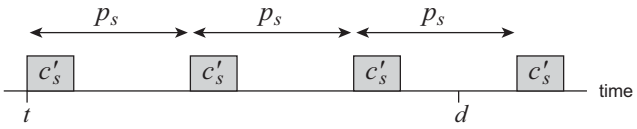

Figure 3. Case 1: $d - t \geq Fp_s + c'_s$

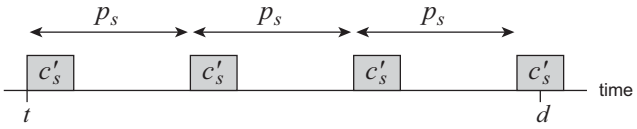

Figure 4. Case 2: $d - t \leq Fp_s + c'_s$

the highest priority, its execution is exactly repeated at its minimum inter-arrival time p_s on every processor where it is scheduled, as shown in Figure 2. This fact reduces the analysis to one processor. Therefore, we focus on the analysis for one processor henceforth.

In order for T_s to consume the most amount of time in $[t, d)$, t must be at the beginning of some job execution in T_s. Now let F be the maximum number of periods which T_s can have in $[t, d]$. F is given by the following expression.

$$F = \left\lfloor \frac{d-t}{p_s} \right\rfloor \tag{1}$$

If d is a time instant at which T_s is not executed, i.e. $d - t \geq Fp_s + c'_s$ as shown in Figure 3, $W_s(t, d)$ is described by Equation (2).

$$W_s(t, d) = c'_s + Fc'_s \tag{2}$$

In contrast, if d is a time instant at which T_s is in execution, i.e. $d - t \leq Fp_s + c'_s$ as shown in Figure 4, $W_s(t, d)$ is described by Equation (3)

$$W_s(t, d) = d - t - F(p_s - c'_s) \tag{3}$$

As for the total amount $W_k(t, d)$ of time used by each task T_k, except for T_s, in $[t, d)$, it is obtained more easily. Since all the tasks but T_s are scheduled according to EDF, it is obvious that $W_k(t, d)$ is bounded from above by $c_k(d-t)/p_k$. Thus, the condition for J to miss its deadline is given by the following expression, where Γ is a set of non-shared tasks on the processor for which we consider the analysis.

$$W_s(t, d) + \sum_{T_k \in \Gamma} W_k(t, d) > d - t \tag{4}$$

Now we can derive the condition for the value of c'_s to guarantee T_i to be schedulable. Note that $d_i = d - t$. If we assume $d_i \geq Fp_s + c'_s$, the condition is given as follows.

$$\begin{aligned} & c'_s + Fc'_s + \sum_{T_k \in \Gamma} \frac{c_k}{p_k} d_i \leq d_i \\ \Leftrightarrow\ & c'_s \leq \frac{d_i}{F+1} \left(1 - \sum_{T_k \in \Gamma} \frac{c_k}{p_k} \right) \end{aligned} \tag{5}$$

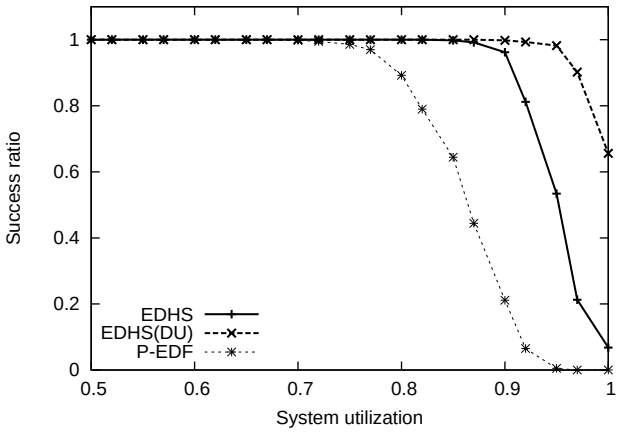

Figure 5. Success ratio (First-Fit).

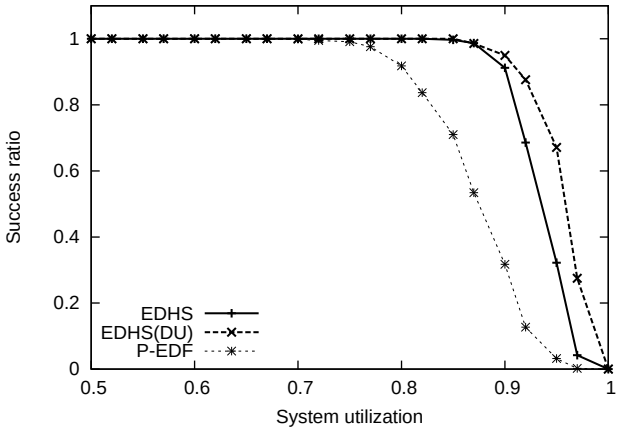

Figure 6. Success ratio (Best-Fit).

Otherwise, the condition is obtained as follows.

$$\begin{aligned} & d_i - F(p_s - c'_s) + \sum_{T_k \in \Gamma} \frac{c_k}{p_k} d_i \leq d_i \\ \Leftrightarrow\ & c'_s \leq p_s - \frac{d_i}{F} \sum_{T_k \in \Gamma} \frac{c_k}{p_k} \end{aligned} \tag{6}$$

Finally, c'_s is set such a maximum value that satisfies the condition (5) or (6) for every $T_i \in \Gamma$.

3 Simulation

In this section, we briefly show the results of simulations conducted with similar setups to [8, 11, 10]. 1,000,000 task sets are randomly generated for every multiprocessor utilization in the range of 0.5 to 1.0. The CPU utilization of each individual task is uniformly distributed in the range of 0.25 to 0.75. The minimum inter-arrival time of each individual task is randomly determined in the range of 100 to 10000 time units. The number of processors is 16.

Figure 5 depicts the ratio of successfully-scheduled task sets in the submitted 1,000,000 task sets in the case that the

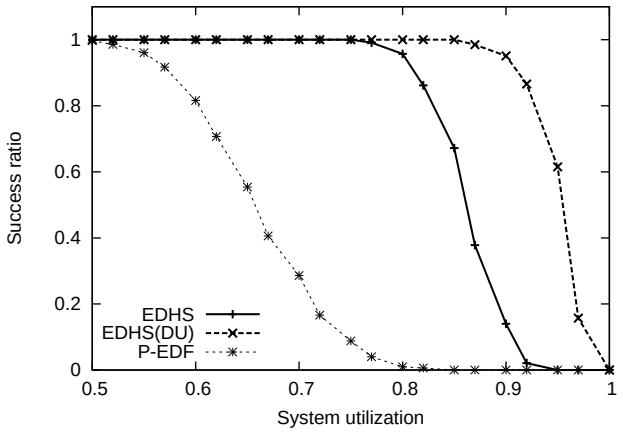

Figure 7. Success ratio (Worst-Fit).

tasks are partitioned based on the first-fit heuristic. EDHS indicates the presented algorithm. EDHS(DU) also indicates the presented algorithm, but the tasks are sorted in decreasing utilization before partitioning. P-EDF indicates the traditional partitioned EDF algorithm. Similarly, Figure 6 and Figure 7 depict the ratios in the cases that the tasks are partitioned based on the best-fit and worst-fit heuristics respectively.

Particularly in the case of worst-fit, EDHS outperforms P-EDF. That is mainly because the performance of worst-fit is potentially poor for partitioning, and thus the impact of task splitting is great. In contrast, P-EDF also performs well in the cases of first-fit and best-fit, since those heuristics offer better performance than worst-fit. Nonetheless, EDHS improves schedulable utilization by about 10% over P-EDF. The performance of EDHS is further improved when the tasks are sorted in decreasing utilization. A primary reason is that the shared tasks are more successfully assigned to processors, since heavy tasks are already partitioned and the utilizations of remaining shared tasks are likely to be small.

Throughout the simulation results, we observe that semi-partitioning is effective to improve the schedulability over traditional partitioning. Since the designed algorithm just places the highest priority to shared tasks, we believe that it is promising for practical use.

4 Future Work

This work is still ongoing. First, we will consider the scheduling of two shared tasks on one processor to improve multiprocessor utilization moreover. We are also interested in the worst-case bounds on the achievable multiprocessor utilization and the numbers of migrations as well as scheduler invocations to improve predictability. In addition, the analysis of tardiness bound is an interesting subject for soft real-time systems.

For the evaluation of the algorithm, we would like to attempt more variety of setups, such as the number of processors, the range and the distribution rules of individual utilizations, etc. Since this paper compared the presented algorithm with only the partitioned EDF algorithm, the other prior algorithms should be compared as well. We will moreover measure the numbers of migrations and context switches. The most significant future work is implementation of the algorithm in a practical operating system such as Linux, and its performance evaluation.

References

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

[2] B. Andersson and K. Bletsas. Sporadic Multiprocessor Scheduling with Few Preemptions. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 243–252, 2008.

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

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

[5] S. Baruah, J. Gehrke, and C.G. Plaxton. Fast Scheduling of Periodic Tasks on Multiple Resources. In *Proceedings of the International Parallel Processing Symposium*, pages 280–288, 1995.

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

[7] S. Cho, S.K. Lee, A. Han, and K.J. Lin. Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. *IEICE Transactions on Communications*, E85-B(12):2859–2867, 2002.

[8] S. Kato and N. Yamasaki. Real-Time Scheduling with Task Splitting on Multiprocessors. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 441–450, 2007.

[9] S. Kato and N. Yamasaki. Global EDF-based Scheduling with Efficient Priority Promotion. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 197–206, 2008.

[10] S. Kato and N. Yamasaki. Portioned EDF-based Scheduling on Multiprocessors. In *Proceedings of the ACM International Conference on Embedded Software*, 2008.

[11] S. Kato and N. Yamasaki. Portioned Static-Priority Scheduling on Multiprocessors. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2008.

[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.