New Abstraction for Optimal Real-Time Scheduling on Multiprocessors*

Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki
Graduate School of Science and Technology
Keio University, Yokohama, Japan
{funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract

T-R Plane Abstraction (TRPA) proposed in this paper is an abstraction technique of real-time scheduling on multiprocessors. This paper presents that NNLF (No Nodal Laxity First) based on TRPA is work-conserving and optimally solves the problem of scheduling periodic tasks on a multiprocessor system. TRPA can accommodate to dynamic environments due to its dynamic time reservation, while T-N Plane Abstraction (TNPA) and Extended TNPA (E-TNPA) reserve processor time statically at every task release.

1. Introduction

Optimal real-time scheduling algorithms realize efficient systems theoretically. They achieve the schedulable utilization bound which is equal to the system capacity. Three optimal real-time scheduling approaches for multiprocessors are hitherto presented (i.e., Pfair [3], EKG [1], and TNPA [4, 5]). Pfair algorithms incur significant run-time overhead due to their quantum-based scheduling approach. Furthermore all task parameters must be multiples of the quantum size in Pfair algorithms. EKG concentrates the workload on some processors due to the approach similar to partitioned scheduling. This characteristic causes some problems on practical environments. For example, from the viewpoint of system overhead, EKG can not generate work-conserving schedules. A scheduling algorithm is *work-conserving* if and only if it never idles processors when there exists at least one active task awaiting the execution in the system. Run-time costs under work-conserving algorithms may be lower than that under non-work-conserving ones with same scheduling frameworks because unnecessary task preemptions may be able to be avoided. From the viewpoint of energy efficiency, energy consumption is minimized when the workload is balanced among processors [2]. Energy efficiency is critically important for battery-based systems. Theoretically optimal real-time static voltage and frequency scaling techniques based on TNPA are presented [6]. TNPA is an efficient approach on the balance as compared to the other optimal approaches.

TNPA is an optimal real-time scheduling approach for multiprocessors; however TNPA generates non-work-conserving schedules. E-TNPA [8] relaxes the restrictions of TNPA and generates work-conserving schedules. E-TNPA leverages the ideas of time apportionment and virtual nodal laxity to reduce the problem to the same concept as TNPA. E-TNPA has two theoretical weaknesses against TNPA in the sense that (1) virtual nodal laxity is tighter than nodal laxity, and (2) E-TNPA must apportion and reapportion additional nodal time at every task release and at every task completion. However E-TNPA can significantly reduce the number of task preemptions as compared to TNPA.

TRPA proposed in this paper realizes work-conserving optimal real-time scheduling without static time reservations, while E-TNPA apportions additional nodal time statically at every task release. It is desirable for dynamic environments such as aperiodic task scheduling [7] since aperiodic task arrivals are unknown priori. If processor time is reserved in every two consecutive task releases, aperiodic tasks can not be executed until the next task release.

The problem of scheduling a set of periodic tasks on a multiprocessor system is presented. The system is modeled as M processors and a taskset $\mathbf{T} = \{T_1, \ldots, T_N\}$, which is a set of N periodic tasks. Each processor can execute at most one task at the same time. Each task can not be executed in parallel among multiple processors. Each task T_i is characterized by two parameters, worst-case execution time c_i and period p_i. A task T_i requires c_i processor time at every p_i interval (i.e., a task generates a sequence of jobs periodically). The relative deadline d_i is equal to its period p_i. All tasks must complete the execution by the deadlines. The ratio c_i/p_i, denoted u_i $(0 < u_i \leq 1)$, is called task utilization. $U = \sum_{T_i \in \mathbf{T}} u_i$ denotes total utilization. We assume that all tasks may be preempted and migrated among processors at any time, and are independent (i.e., they do not share resources and do not have any precedences).

*This research is supported by CREST, JST.

1533-2306/08 $25.00 © 2008 IEEE
DOI 10.1109/RTCSA.2008.15

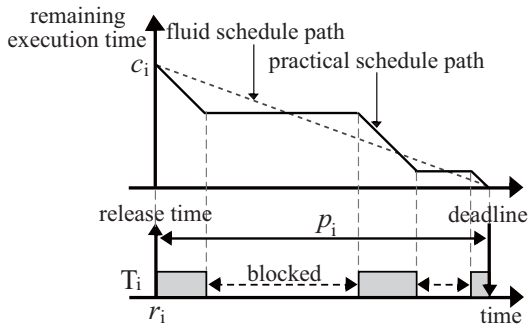

Figure 1. Fluid and practical schedules.

The remainder of this paper is organized as follows. The next section explains the traditional technique TNPA. Section 3 shows NNLF based on TRPA is optimal and work-conserving. We conclude with a summary in Section 4.

2. T-N Plane Abstraction

T-N Plane Abstraction (TNPA) [4, 5] is an abstraction technique of real-time scheduling. TNPA is based on the fluid scheduling model [9]. In the fluid scheduling model, each task is executed at a constant rate at all times. Figure 1 illustrates the difference between the fluid schedule and a practical schedule. The upper area of the figure represents time on the horizontal axis and task's remaining execution time on the vertical axis. In practical scheduling, the task will be blocked by the other tasks as shown in the lower area of the figure since a processor can execute only one task at the same time. On the other hand, in the fluid scheduling model, each task T_i is always executed along its fluid schedule path, the dotted line from (r_i, c_i) to $(r_i + p_i, 0)$, where r_i represents the release time of the current job. The fluid scheduling can not realize optimal schedule on practical environments since a processor must execute multiple tasks simultaneously. Notice that tasks need not constantly track their fluid schedule paths. Namely deadlines are the only time at which tasks must track the fluid schedule paths.

Figure 2 shows the way TNPA abstracts real-time scheduling. Time is divided by the deadlines of all tasks as the vertical dotted lines in the figure. The intervals between every two consecutive deadlines are called *nodes*. The right isosceles triangles called T-N planes (Time and Nodal remaining execution time domain planes) are placed inside the nodes of all tasks. The rightmost vertex of each T-N plane coincides with the intersection of the fluid schedule path and the right side of each node. Since all the T-N planes in the same node are congruent, we have only to keep in mind an overlapped T-N plane shown in the lower area of the figure at a time. The overlapped T-N plane represents time on the horizontal axis and task's *nodal remaining execution time* on the vertical axis. If the nodal remaining execution time becomes zero at the rightmost vertex of each T-N plane, the task execution follows the fluid schedule path at every deadline. Since T-N planes are repeated over time, good scheduling algorithms for a single T-N plane can help all tasks to meet their deadlines. Therefore the problem is the way to conduct all tasks to the rightmost vertex of the T-N plane. Note that all the algorithms based on TNPA are non-work-conserving. The tasks, the nodal remaining execution time of which is zero, are not executed within the current node even if the remaining execution time is not zero. In fact, these tasks can be executed in unoccupied time; however it incurs unnecessary task preemptions.

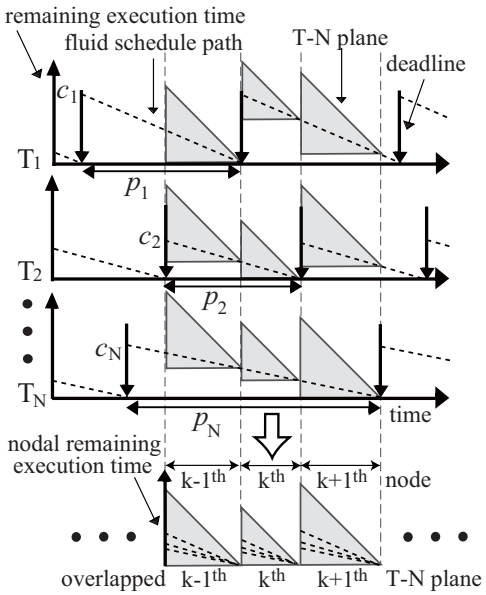

Figure 2. T-N Plane Abstraction.

Figure 3 shows an overlapped T-N plane, where *tokens* representing tasks move from time t_0 to t_f. All tokens are on their fluid schedule paths at time t_0. A token moves diagonally down if the task is executed; otherwise it moves horizontally. If all tokens arrive at the rightmost vertex, all tasks meet their deadlines. The successful arrival to the rightmost vertex is called *nodally feasible*. For the nodal feasibility, Events C and B occur when tokens hit the no nodal laxity diagonal (NNLD) and the bottom side of the T-N plane, respectively. Schedulers are invoked at every time t_0, Event C, and Event B. NNLF [7] selects at most M tokens in No Nodal Laxity First order, and ties are broken arbitrarily. All tokens are nodally feasible if $U \leq M$. NNLF based on TNPA is optimal in the sense that any periodic taskset with utilization $U \leq M$ will be scheduled to meet all deadlines.

For example, there are four tasks (T_1, T_2, T_3, T_4) and two processors (P_1, P_2) as shown in Figure 3. Since there are two processors, two tasks can be executed simultane-

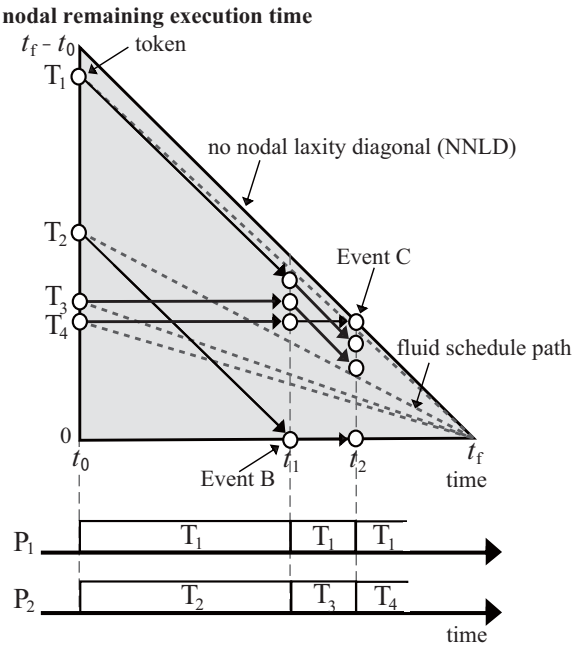

Figure 3. NNLF based on TNPA.

ously. At time t_0, T_1 and T_2 are executed on P_1 and P_2 in NNLF (ties are broken in Largest Nodal Remaining Execution time First [4, 5] in this example). Event B occurs at time t_1 since T_2 hits the bottom side of the T-N plane. Then two tasks T_1 and T_3 are selected again in the same manner. Event C occurs at time t_2 since T_4 hits the NNLD. The scheduler is invoked at every event.

Extended T-N Plane Abstraction (E-TNPA) [8] moves trapezial T-N planes up and down at the beginning of each node by using *time apportionment* as shown in Figure 4. The shaded triangle in each T-N plane is called virtual T-N plane. No virtual nodal laxity diagonal (NVNLD) is leveraged by NVNLF (No Virtual Nodal Laxity First), while NNLD is leveraged by NNLF in TNPA. Tasks are scheduled in virtual T-N planes in the same manner as TNPA. A disadvantage of E-TNPA is that the positions of virtual T-N planes are given at time t_0. Therefore aperiodic tasks can not be executed immediately (i.e., until the next task release) while there exist at least M active tasks in the system.

3. T-R Plane Abstraction

T-R Plane Abstraction (TRPA) realizes work-conserving optimal real-time scheduling on multiprocessors. TRPA leverages T-R planes (Time and Remaining execution time domain planes) as shown in Figure 5. TRPA differs from TNPA in the sense that (1) a T-R plane is a triangle or a trapezium, (2) the vertical axes of T-R planes represent remaining execution time, and (3) T-R planes are not congru-

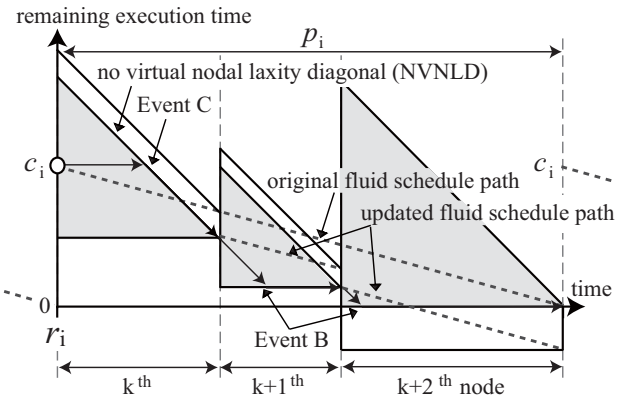

Figure 4. Extended T-N Plane Abstraction.

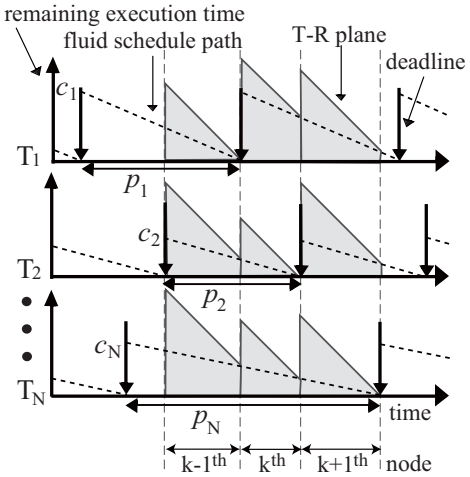

Figure 5. T-R Plane Abstraction.

ent even in the same node. The successful arrival to the right side of the T-R plane is called *nodally feasible*. If all tokens are nodally feasible, all tasks meet their deadlines as in the case of TNPA. In the later sections, only one node is focused on since the successful schedule in a node helps all tasks to meet their deadlines as well as in TNPA.

3.1. Abstraction Model

Figure 6 illustrates a T-R plane for a task T_i. The concepts of T-R planes are mostly the same as those of T-N planes. Tokens representing tasks move from time t_0 to t_f. The fluid schedule path runs through the upper right vertex of the T-R plane. The upper right diagonal is called no nodal laxity diagonal (NNLD) as well as in TNPA. The bottom side is called no remaining execution time horizon (NRETH). A T-R plane is divided into two areas: "safe area" and "unsafe area" as shown in Figure 6. A task T_i is *safe* if and only if the token is in the safe area. Otherwise

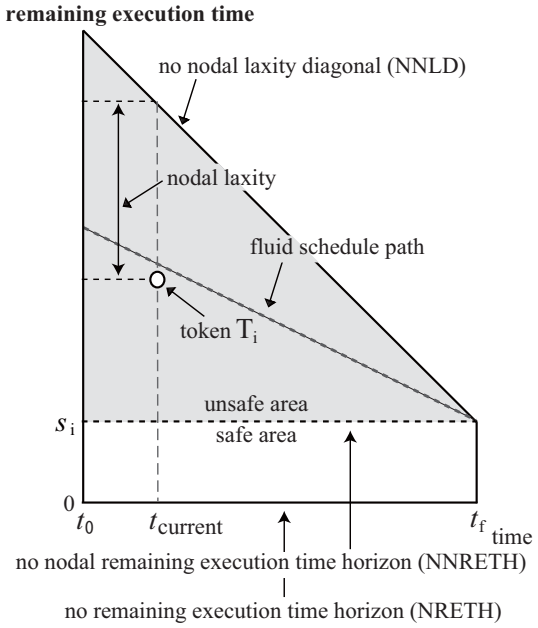

Figure 6. A T-R plane for a task T_i.

the task is *unsafe*. Unsafe areas are orthogonal isosceles triangles and are congruent in the same node. The borderline between the safe area and the unsafe area called no nodal remaining execution time horizon (NNRETH) is defined as $s_i = c_i - u_i(t_f - r_i)$, where r_i represents the release time of the current job of T_i. The borderline s_i changes at every node in accordance with the changes of t_f and r_i. We assume that j^{th} event occurs at time t_j. $e_{i,j}$ denotes the remaining execution time of T_i at time t_j. The nodal remaining execution time of T_i at time t_j is defined as follows:

$$l_{i,j} = \max\{e_{i,j} - s_i, 0\}. \quad (1)$$

The nodal utilization of T_i at time t_j is defined as $r_{i,j} = l_{i,j}/(t_f - t_j)$. If a task T_i is safe, both the nodal remaining execution time and the nodal utilization of the task T_i are zero. $S_j = \sum_{T_i \in \mathbf{T}} r_{i,j}$ denotes the total nodal utilization at time t_j. Flag M represents the two-phase condition: *on* and *off*. Flag M is *on* at time t_j if and only if $S_j \geq M$.

3.2. NNLF Scheduling Algorithm

NNLF selects at most M tokens in No Nodal Laxity First order at every event, and ties are broken arbitrarily as shown in Figure 7. NNLF based on TRPA differs from that based on TNPA in the sense that (1) Event M occurs at the time when Flag M becomes on from off, and (2) execution candidates and the condition where Event B occurs change in accordance with the condition of Flag M as shown in Table 1. When Flag M is off, Event B occurs at the time when

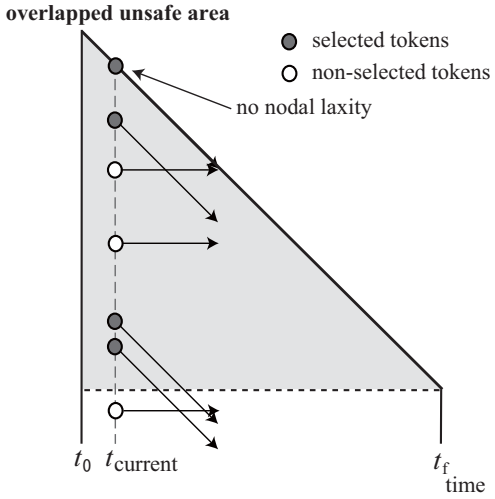

Figure 7. NNLF based on TRPA ($M = 4$).

Table 1. NNLF based on TRPA.

Flag M	on	off
candidates	unsafe tasks	all tasks
Event B	NNRETH	NRETH
Event C	NNLD	NNLD
Event M	does not occur	Flag M becomes on

tokens hit not NNRETHs but NRETHs as opposed to in TNPA. NNLF based on TRPA creates the same schedule as that on TNPA at the time when Flag M is on since (1) only unsafe tasks are executed, (2) Event B occurs when tokens hit NNRETHs, and (3) Event M does not occur.

3.3. Optimality

This section shows that NNLF based on TRPA is optimal in the sense that any periodic taskset with utilization $U \leq M$ will be successfully scheduled to meet all deadlines.

Cho et al. [4] show that *critical moment* is the sufficient and necessary condition where tokens are not nodally feasible in TNPA. Critical moment is the first time when more than M tokens simultaneously hit the NNLD as shown in Figure 8. It is the same as in TRPA as shown in Lemma 1.

Lemma 1. *All tokens are nodally feasible in TRPA if and only if no critical moment occurs in the current node.*

Proof. This proof is the same as that of TNPA [4]. □

The detailed condition at the time when a critical moment occurs at time t_j is shown in Lemma 2.

Lemma 2. *When a critical moment occurs at time t_j in TRPA, $S_j > M$ holds.*

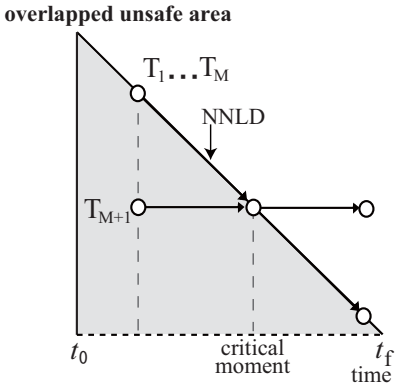

Figure 8. Critical moment.

Proof. This proof is the same as that of TNPA [4]. □

Theorem 3 is summarily derived from Lemmas 1 and 2.

Theorem 3. *All tokens are nodally feasible in TRPA if and only if $S_j \leq M$ for all j.* □

The fact that no critical moment occurs in TRPA, if $U \leq M$, is shown by the inductive method. The induction is based on the event time t_j to establish the total nodal utilization $S_j \leq M$. Lemma 4 shows the induction base, and Theorem 5 shows the nodal feasibility based on Lemma 4.

Lemma 4 (Initial Total Nodal Utilization)**.** *Assume that all tokens are nodally feasible in the previous node. In the assumption, $S_0 \leq M$ holds if $U \leq M$.*

Proof. Since all tokens are nodally feasible in the previous node, the position of each token is on or below its fluid schedule path at time t_0. In the result, the nodal utilization $r_{i,0}$ is less than or equal to the utilization u_i for all i. Therefore $S_0 \leq M$ is derived from $S_0 \leq U$ and $U \leq M$. □

Theorem 5 (Nodal Feasibility)**.** *Assume that all tokens are nodally feasible in the previous node. In the assumption, all tokens are nodally feasible in TRPA if $U \leq M$ holds.*

Proof. Lemma 4 shows that $S_0 \leq M$ holds, and the proof of Lemma 4 shows that all tokens are on their T-R planes at time t_0. When Flag M is off, no critical moment occurs from Lemma 2. Thus we have only to keep in mind whether all tokens are nodally feasible when Flag M is on. Since total nodal utilization is a continuous function, Event M occurs when total nodal utilization becomes M. Assume that total nodal utilization becomes M at time t_M. The induction base is $S_M = M$. The induction hypothesis is:

$$S_{j-1} = M$$
$$\Leftrightarrow \sum_{T_i \in \mathbf{T}} \frac{l_{i,j-1}}{t_f - t_{j-1}} = M$$
$$\Leftrightarrow \sum_{T_i \in \mathbf{T}} l_{i,j-1} = M(t_f - t_{j-1}). \quad (2)$$

If the number of unsafe tasks is less than M at time t_{j-1}, all tokens are nodally feasible in obvious. Therefore assume that M unsafe tokens can be selected at time t_{j-1}. The total remaining execution time decreases by $M(t_j - t_{j-1})$ between time t_{j-1} and t_j. S_j is calculated as follows:

$$\begin{aligned} S_j &= \frac{1}{t_f - t_j} \sum_{T_i \in \mathbf{T}} l_{i,j} \\ &= \frac{1}{t_f - t_j} \left(\left(\sum_{T_i \in \mathbf{T}} l_{i,j-1} \right) - M(t_j - t_{j-1}) \right). \\ &\Downarrow \text{From Equation (2)} \\ &= \frac{M(t_f - t_{j-1}) - M(t_j - t_{j-1})}{t_f - t_j} \\ &= M. \end{aligned}$$

Thus $S_j = M$ for all $t_j \geq t_M$. From Theorem 3, all tokens are nodally feasible since $S_j \leq M$ holds for all j and all the tokens with no nodal laxity are always selected. □

NNLF based on TRPA is optimal as follows.

Theorem 6 (Optimality)**.** *Any periodic taskset **T** with utilization $U \leq M$ will be scheduled to meet all deadlines on M processors by NNLF based on TRPA.*

Proof. In the first node, all tokens are on their fluid schedule paths at time t_0. Therefore the nodal utilization $r_{i,0}$ is equal to the utilization u_i for all i. In the condition, all tokens are nodally feasible based on the same way as the proof of Theorem 5. In the following nodes, all tokens are nodally feasible from Theorem 5. If all tokens are nodally feasible for all nodes, all tokens meet their deadlines. □

3.4. Work-Conserving

NNLF based on TRPA is work-conserving as follows. A task is *ready* if and only if the task has non-zero remaining execution time. The two cases of Flag M = on and Flag M = off are considered. (1) When Flag M is off, all the ready tasks up to M can be selected since NNRETHs do not block task executions as opposed to TNPA. (2) When Flag M is on, NNRETHs will block task executions; however NNLF based on TRPA is work-conserving even in the condition as follows. Since all tokens are nodally feasible as shown in the previous section, all tokens are in their T-R planes. Therefore the nodal utilization of each task is equal to or less than one. When Flag M is on, the total nodal utilization is equal to or larger than M. Therefore there exist at least M ready tasks at any time. From (1) and (2), NNLF based on TRPA can select all the ready tasks up to M.

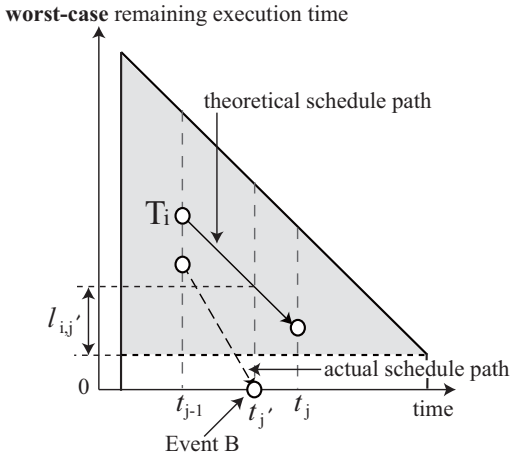

Figure 9. Actual and theoretical schedules.

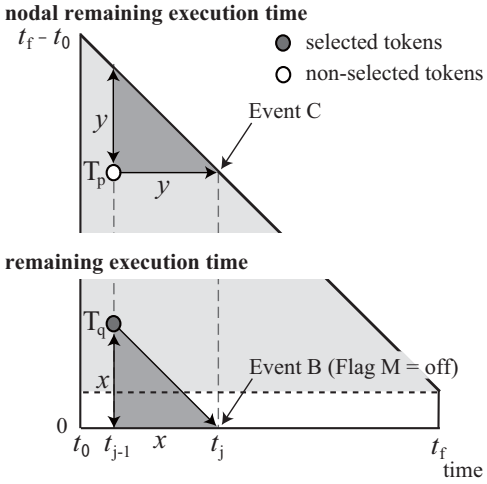

Figure 10. Events B and C.

3.5. Algorithm Overhead

Overhead is one of the main concerns against real-time scheduling algorithms since real-time scheduling algorithms incur frequent task preemptions. Cho et al. [4] discuss that the upper-bound on the number of scheduler invocations is a good metric for overhead measurement.

In NNLF based on TRPA, the scheduler is invoked at every event. The number of events except Event M in a node is at most $N + 1$ as well as in TNPA [4]. There exist $1 + \sum_{T_i \in \mathbf{T}} \lceil I/p_i \rceil$ nodes in Interval I. Therefore the upper-bound on the number of NNLF scheduler invocations in Interval I except for Event M is as follows:

$$(N+1)\left(1 + \sum_{T_i \in \mathbf{T}} \left\lceil \frac{I}{p_i} \right\rceil\right). \tag{3}$$

Event M theoretically occurs at most one time in a node as follows. Once Event M occurs at time t_M, the total nodal utilization S_j theoretically becomes M for all $t_j \geq t_M$ as shown in the proof of Theorem 5. However Event M may occur multiple times in a node in practical environments as follows. Each task T_i almost always completes the execution at time $t_{j'}$ earlier than c_i is completely consumed as shown in Figure 9 because c_i represents "worst-case" execution time. The scheduler can detect the early completion, and Event B occurs at time $t_{j'}$. In this case, the time $l_{i,j'}$ shown in Figure 9 becomes zero. Therefore the total nodal utilization $S_{j'}$ becomes less than M. In this case, Flag M becomes off, and Event M may occur again. The early completions occur at most the number of task completions $(1 + \sum_{T_i \in \mathbf{T}} \lceil I/p_i \rceil)$ in Interval I. Therefore the upper-bound on the number of Event M in Interval I is:

$$2\left(1 + \sum_{T_i \in \mathbf{T}} \left\lceil \frac{I}{p_i} \right\rceil\right). \tag{4}$$

From the results, the upper-bound on the number of scheduler invocations in Interval I is (3) + (4) as follows:

$$(N+3)\left(1 + \sum_{T_i \in \mathbf{T}} \left\lceil \frac{I}{p_i} \right\rceil\right).$$

The upper-bound is larger than that of TNPA [4] since Event M does not occur in TNPA. This estimation is rather undue since most tasks do not incur Event B when Flag M is off. More exact bound will be complex because it highly depends on the taskset. However TRPA is more efficient than TNPA since NNLF based on TRPA is work-conserving.

3.6. Next Event Time

Next event time is an interesting topic for the implementation. Once an event occurs at time t_{j-1}, scheduler invocations can be omitted until the next event time t_j. Assume that current time is t_{j-1}. The next event time t_j can be calculated from the current time t_{j-1}. The times at Events B, C, and M in the case where the other events do not occur are denoted by t_B, t_C, and t_M, respectively. The next event time t_j is the earliest event time calculated as follows:

$$t_j = \min\{t_B, t_C, t_M\}.$$

Event B occurs at the time t_B when the selected token T_q or T_r, the remaining execution time or the nodal remaining execution time of which is the least in selected tokens,

hits the NRETH or the NNRETH in accordance with Flag M, respectively. The length x shown in Figure 10 is equal to $e_{q,j-1}$ since the deeply shaded triangle is isosceles. Therefore Event B occurs at time $t_{j-1} + e_{q,j-1}$ when Flag M is off at time t_{j-1}. On the other hand, Event B occurs at time $t_{j-1} + l_{r,j-1}$ in the same way when Flag M is on at time t_{j-1}. Thus Event B occurs at time t_B as follows:

$$t_B = \begin{cases} t_{j-1} + e_{q,j-1} & \text{(Flag M is off)} \\ t_{j-1} + l_{r,j-1} & \text{(Flag M is on)} \end{cases}.$$

Event C occurs at the time t_C when the non-selected token T_p, the nodal laxity of which is the least in non-selected tokens, hits the NNLD as shown in Figure 10. The length y is equal to $t_f - t_{j-1} - l_{p,j-1}$ since the deeply shaded triangle is isosceles. Thus Event C occurs at time t_C as follows:

$$\begin{aligned} t_C &= t_{j-1} + (t_f - t_{j-1} - l_{p,j-1}) \\ &= t_f - l_{p,j-1}. \end{aligned}$$

Event M occurs at the time t_M when the total nodal utilization becomes M. Event M occurs only if Flag M is off at time t_{j-1}. The total nodal utilization at time t_M is:

$$S_M = M.$$

From the definition of total nodal utilization,

$$\frac{1}{t_f - t_M} \sum_{T_i \in \mathbf{T}} l_{i,M} = M. \tag{5}$$

The problem to rewrite Equation (5) is how much total nodal remaining execution time decreases between time t_{j-1} and t_M. Since nodal remaining execution time is conditionally branched by $\max$ as shown in Equation (1), the total nodal remaining execution time at time t_M based on at time t_{j-1} can not be simplified as opposed to Events B and C. Therefore whether Event M occurs is calculated at every interval shown by the dual-directional arrows in Figure 11 (non-selected tokens are not presented). Selected unsafe tasks are numbered in increasing nodal remaining execution time. Interval m denotes the time interval between time $t_{j-1} + l_{m-1,j-1}$ and $t_{j-1} + l_{m,j-1}$, where $l_{0,j-1}$ is zero.

In the following, whether Event M occurs in Interval m is checked. N' denotes the number of selected unsafe tasks in the interval. The decrease of the total nodal remaining execution time of selected safe tasks from time t_{j-1} is defined as $X = \sum_{i=1}^{m-1} l_{i,j-1}$. The total nodal remaining execution time of selected unsafe tasks decreases by $N'(t_M - t_{j-1})$ from time t_{j-1}. Consequently the total nodal remaining execution time of all tasks decreases by $N'(t_M - t_{j-1}) + X$ from time t_{j-1}. The total nodal remaining execution time at time t_M can be rewritten from Equation (5) as follows:

$$\frac{1}{t_f - t_M} \left(\sum_{T_i \in \mathbf{T}} l_{i,j-1} - (N'(t_M - t_{j-1}) + X) \right) = M.$$

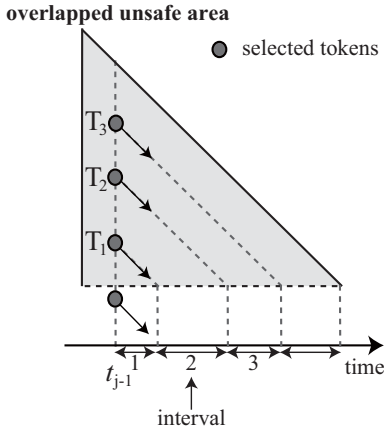

Figure 11. Intervals for Calculate t_M.

By a simple deformation,

$$\frac{t_f - t_{j-1}}{t_f - t_M} \frac{\sum_{T_i \in \mathbf{T}} l_{i,j-1}}{t_f - t_{j-1}} - \frac{N'(t_M - t_{j-1}) + X}{t_f - t_M} = M.$$

From the definition of total nodal utilization,

$$\frac{(t_f - t_{j-1}) S_{j-1} - N'(t_M - t_{j-1}) - X}{t_f - t_M} = M.$$

Finally we obtain

$$t_M = \frac{(M - S_{j-1}) t_f + (S_{j-1} - N') t_{j-1} + X}{M - N'}.$$

If t_M is in Interval m (i.e., $t_{j-1} + l_{m-1,j-1} < t_M \leq t_{j-1} + l_{m,j-1}$), Event M occurs at time t_M. Otherwise Interval $m+1$ is checked whether Event M occurs. If Event M does not occur in all intervals, the rightmost interval is checked.

Figure 12 shows Algorithm Calculate t_M based on the discussions. The algorithm must be performed at every event to check whether Event M occurs in the case where Flag M is off, while the time apportionment of E-TNPA is performed only at every task release and task completion.

3.7. TRPA versus E-TNPA

This section distinguishes the differences between TRPA and E-TNPA. Both of them are work-conserving and optimally solve the problem of scheduling periodic tasks on a multiprocessor system. First difference is that the complexity per invocation of Calculate t_M in TRPA is $O(M)$, whereas the complexity per invocation of ApportionTime of E-TNPA is $O(N)$. Second difference is that the averaged number of invocations per node is less than $N + 3$ in TRPA as shown in Section 3.5 and equal to 2 in E-TNPA.

Final difference is the most important. Some types of systems are required to accommodate to dynamic environments such as taskset changes and aperiodic services [7].

Algorithm: Calculate t_M

if Flag M is on
  return Event M does not occur
end if
$N' =$ the number of executed unsafe tasks at time t_{j-1}
$X = 0$
foreach executed unsafe tasks as T_m in increasing $l_{i,j-1}$
  $t_M = \dfrac{(M - S_{j-1})t_f + (S_{j-1} - N')t_{j-1} + X}{M - N'}$
  if $t_{j-1} + l_{m-1,j-1} < t_M \leq t_{j-1} + l_{m,j-1}$
    return Event M occurs at time t_M
  end if
  $N' = N' - 1$
  $X = X + l_{m,j-1}$
end foreach
if $t_{j-1} + l_{m,j-1} < t_f$
  // check the rightmost interval ($N' = 0$)
  $t_M = \dfrac{(M - S_{j-1})t_f + S_{j-1}t_{j-1} + X}{M}$
  if $t_{j-1} + l_{m,j-1} < t_M < t_f$
    return Event M occurs at time t_M
  end if
end if
return Event M does not occur

Figure 12. Algorithm: Calculate t_M.

E-TNPA reserves processor time at every task release and task completion statically. Therefore if an aperiodic task arrives at time $t_0 + \Delta$, where Δ is a very small number, the task can not be executed until time t_f. TRPA overcomes the disadvantage of E-TNPA. Figure 13 shows the difference of total nodal utilization flow between E-TNPA and TRPA. For work-conserving schedules, E-TNPA forces total nodal utilization to keep M, while TRPA forces total nodal utilization not to exceed M. In E-TNPA, when a task T_i finishes its execution earlier than c_i is completely consumed as shown in Figure 9, E-TNPA reapportions processor time to keep $S_j = M$ again. In TRPA, on the other hand, when a task T_i finishes its execution earlier, TRPA does nothing. As shown in Figure 13, there is no space for aperiodic servers [7] in E-TNPA, while TRPA has leeway for dynamic system changes such as aperiodic services.

4. Conclusions

This paper presents another work-conserving optimal real-time scheduling algorithm for multiprocessors. TRPA proposed in this paper overcomes the weaknesses of E-TNPA presented in the previous research with certain overhead. TRPA reserves processor time automatically by tie-breaking rules, while E-TNPA reserves processor time statically at every task release and task completion. However E-TNPA is still a considerable technique since the model of E-TNPA accepts the case where tokens move to the upper area

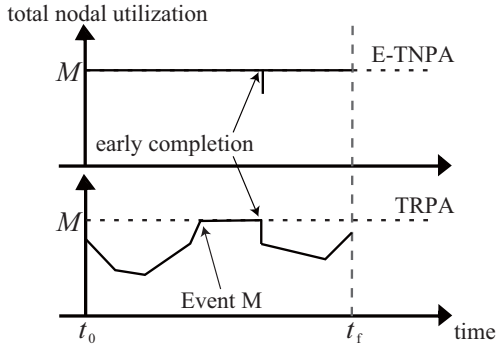

Figure 13. Total nodal utilization flow.

of NNLD. The combination of TRPA and E-TNPA may become a good abstraction for optimal real-time scheduling.

References

[1] B. Andersson and E. Tovar. Multiprocessor Scheduling with Few Preemptions. In *Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, Aug. 2006.

[2] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *Proc. of the 17th IEEE International Parallel and Distributed Processing Symposium*, pages 22–26, Sept. 2003.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate Progress: A Notion of Fairess in Resource Allocation. *Algorithmica*, 15(6):600–625, June 1996.

[4] H. Cho, B. Ravindran, and E. D. Jensen. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *Proc. of the 27th IEEE Real-Time Systems Symposium*, pages 101–110, Dec. 2006.

[5] H. Cho, B. Ravindran, and E. D. Jensen. Synchronization for an Optimal Real-Time Scheduling Algorithm on Multiprocessors. In *Proc. of the 2nd IEEE International Symposium on Industrial Embedded Systems*, pages 9–16, July 2007.

[6] K. Funaoka, S. Kato, and N. Yamasaki. Energy-Efficient Optimal Real-Time Scheduling on Multiprocessors. In *Proc. of the 11th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 23–30, May 2008.

[7] K. Funaoka, S. Kato, and N. Yamasaki. Optimal Real-Time Scheduling for Efficient Aperiodic Services on Multiprocessors. In *Proc. of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 245–251, Feb. 2008.

[8] K. Funaoka, S. Kato, and N. Yamasaki. Work-Conserving Optimal Real-Time Scheduling on Multiprocessors. In *Proc. of the 20th Euromicro Conference on Real-Time Systems*, July 2008.

[9] P. Holman and J. H. Anderson. Adapting Pfair Scheduling for Symmetric Multiprocessors. *Journal of Embedded Computing*, 1(4):543–564, May 2005.