Work-Conserving 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

Extended T-N Plane Abstraction (E-TNPA) proposed in this paper realizes work-conserving and efficient optimal real-time scheduling on multiprocessors relative to the original T-N Plane Abstraction (TNPA). Additionally a scheduling algorithm named NVNLF (No Virtual Nodal Laxity First) is presented for E-TNPA. E-TNPA and NVNLF relax the restrictions of TNPA and the traditional algorithm LNREF, respectively. Arbitrary tasks can be preferentially executed by both tie-breaking rules and time apportionment policies in accordance with various system requirements with several restrictions. Simulation results show that E-TNPA significantly reduces the number of task preemptions as compared to TNPA.

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 LNREF [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 energy efficiency, energy consumption is minimized when the workload is balanced among processors [2]. Energy efficiency is critically important for battery-based embedded systems. LNREF is an efficient algorithm on the balance as compared to the other optimal algorithms.

LNREF is based on the abstraction technique called T-N Plane Abstraction (TNPA). LNREF is an optimal real-time scheduling algorithm for multiprocessors; however LNREF is non-work-conserving due to the restriction of TNPA, whereas Extended T-N Plane Abstraction (E-TNPA) proposed in this paper realizes work-conserving algorithms. 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. Work-conserving algorithms have some advantages against non-work-conserving ones. In particular, average response time under work-conserving algorithms may be lower than that under non-work-conserving ones because no processor time is wasted. As a result, run-time costs under work-conserving algorithms may be lower than that under non-work-conserving ones because unnecessary task preemptions may be able to be avoided. Therefore one of the contributions of this paper is the practicality improvement of optimal real-time scheduling on actual environments.

The other contribution of this paper is that the restrictions of both TNPA and LNREF are relaxed by E-TNPA and NVNLF proposed in this paper to satisfy various system requirements with several restrictions. If a scheduling algorithm has tight restrictions, some types of systems do not prefer the algorithm. For example, the motor control of humanoid robots requires the minimized jitter to realize precise motions [7]. Aperiodic task scheduling with hard real-time periodic tasks such as aperiodic servers [8], which behave as periodic tasks, requires the minimized response time. In NVNLF based on E-TNPA, these tasks can be preferentially executed by both tie-breaking rules and time apportionment policies described in the later sections.

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. Each task can not be executed in parallel among 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

*This research is supported by CREST, JST.

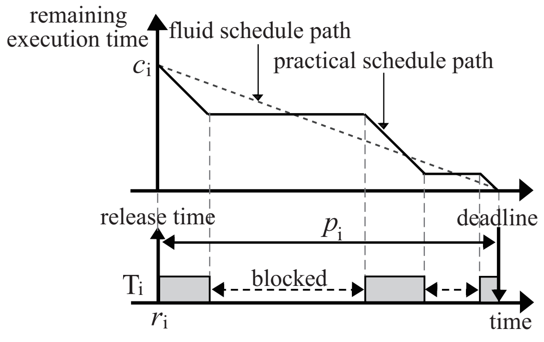

Figure 1. Fluid and practical schedules.

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 precedence for each other).

The remainder of this paper is organized as follows. The next section explains the traditional techniques TNPA and LNREF. In Section 3, E-TNPA and NVNLF are presented. Section 4 evaluates the effectiveness of E-TNPA. Finally we conclude with a summary and future work in Section 5.

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 [6]. 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 horizontal axis and task's remaining execution time on 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 simultaneously. 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 some 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

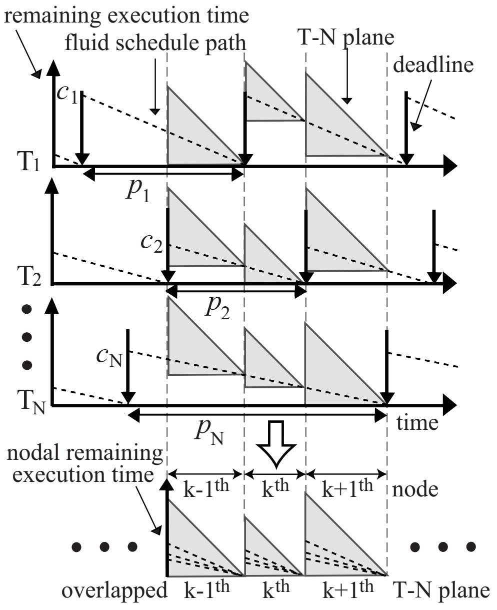

Figure 2. T-N Plane Abstraction.

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 horizontal axis and task's *nodal remaining execution time* on 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 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. The scheduler is invoked at time t_0, Event C, and Event B. We assume that j^{th} event occurs at

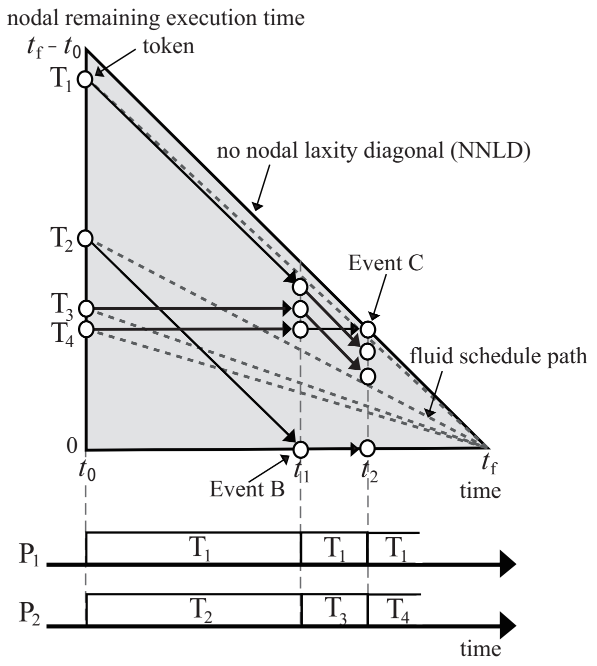

Figure 3. LNREF based on TNPA.

time t_j. For each task T_i, nodal remaining execution time at time t_j is defined as $l_{i,j}$. LNREF selects M tokens in Largest Nodal Remaining Execution time First. All tokens are nodally feasible if $U \leq M$. Besides scheduling policies other than LNREF may also provide the nodal feasibility. LNREF based on TNPA is optimal in the sense that any periodic taskset with utilization $U \leq M$ will be scheduled to meet all deadlines. The successful schedule to meet all deadlines is called *globally feasible* to accentuate the difference with *nodally feasible*. If $U > M$, no algorithm can realize the global feasibility. Thus we assume that $U \leq M$.

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 simultaneously. At time t_0, T_1 and T_2 are executed on P_1 and P_2 by LNREF. 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 oblique side (NNLD) of the T-N plane. As just described, the scheduler is invoked at every event.

3. Extended T-N Plane Abstraction

Extended T-N Plane Abstraction (E-TNPA) realizes work-conserving optimal real-time scheduling. E-TNPA differs from TNPA in the sense that the processor time unused by the taskset in TNPA is apportioned among tasks. Moreover the processor time apportionment is flexibly decided by arbitrary policies with several restrictions in E-TNPA, while TNPA allocates the stationary nodal remaining execution time $u_i(t_f - t_0)$ for each task T_i in each node.

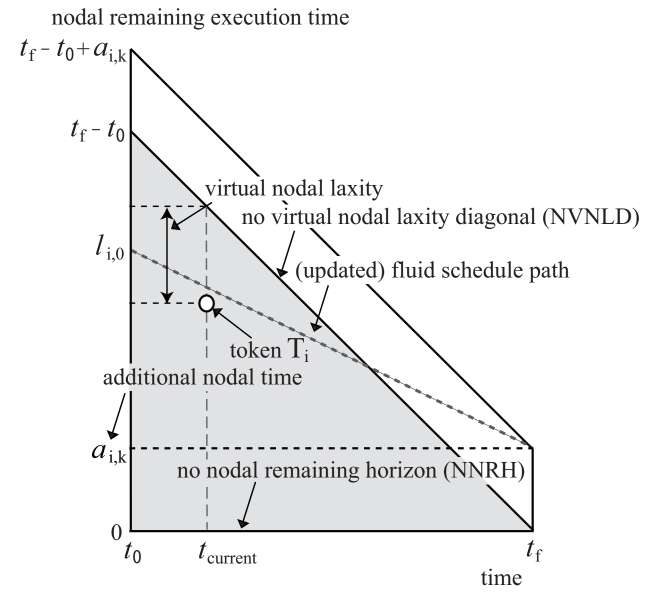

Figure 4. An extended T-N plane ($a_{i,k} \geq 0$).

Figures 4 and 5 show new T-N planes for E-TNPA. T-N planes are not triangles but trapeziums; furthermore they are no longer congruent even in the same node. In the k^{th} node, each task T_i has *additional nodal time* $a_{i,k}$ and initial nodal remaining execution time $l_{i,0} = u_i(t_f - t_0) + a_{i,k}$. $A = \sum_{T_i \in \mathbf{T}} a_i$ denotes total additional nodal time. $a_{i,k}$ never changes throughout the node; however it can be changed at time t_0 in each node. If $a_{i,k} \geq 0$, the fluid schedule path runs through the upper right vertex of the T-N plane; otherwise it runs through the lower right vertex. Thus the initial nodal remaining execution time $l_{i,0}$ is the intersection of the fluid schedule path and the left side of the T-N plane as shown in the figures. A line called *no virtual nodal laxity diagonal* (NVNLD) is drawn from $(t_0, t_f - t_0)$ to $(t_f, 0)$. The line which represents that tasks have no nodal remaining execution time is called *no nodal remaining horizon* (NNRH). Event C and Event B occur when tokens hit the NVNLD and the NNRH, respectively. The successful arrival to $(t_f, 0)$ is called *nodally feasible*. All tokens must be in the shaded triangle called *virtual T-N plane* for the nodal feasibility. Notice that the overlapped T-N plane of TNPA and the overlapped virtual T-N plane of E-TNPA have the same concepts since all the virtual T-N planes in the same node are congruent. Consequently all tasks are both nodally feasible and globally feasible if $a_{i,k}$ is assigned carefully.

The overview of E-TNPA is shown in Figure 6. The figure shows consecutive T-N planes of a task T_i. If a task T_i completes the execution in the k^{th} node even if $a_{i,k} = 0$, the task gets a zero or negative additional nodal time $a_{i,k} \leq 0$;

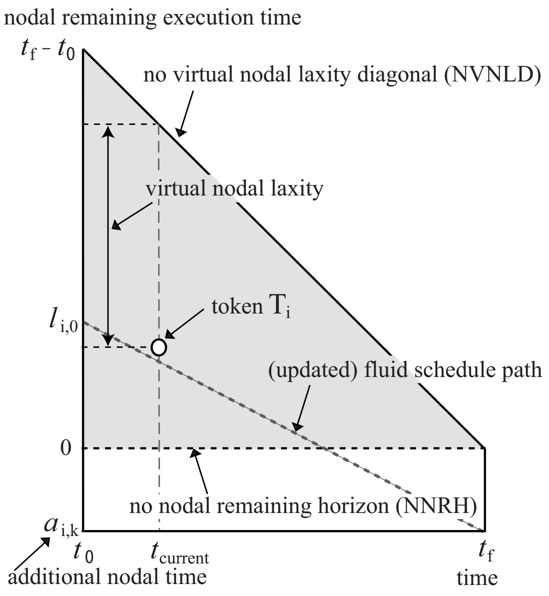
nodal remaining execution time
$t_f - t_0$
no virtual nodal laxity diagonal (NVNLD)
virtual nodal laxity
$l_{i,0}$
token T_i
(updated) fluid schedule path
0
no nodal remaining horizon (NNRH)
$a_{i,k}$
t_0
$t_{current}$
t_f
additional nodal time
time

Figure 5. An extended T-N plane ($a_{i,k} < 0$).

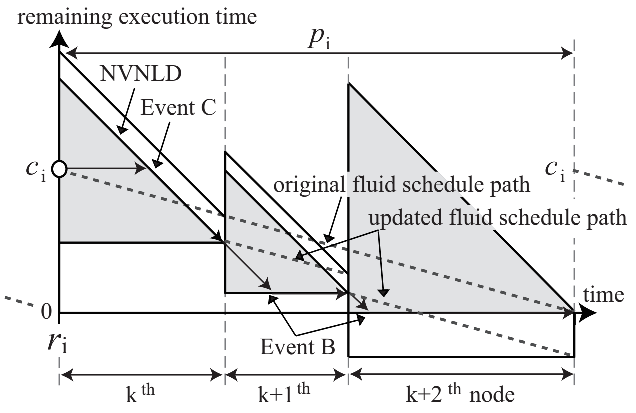

Figure 6. Consecutive extended T-N planes.

otherwise the task is assigned a zero or positive additional nodal time $a_{i,k} \geq 0$. If a task gets a positive additional nodal time $a_{i,k} > 0$ and the token arrives at the rightmost vertex of the current virtual T-N plane, the fluid schedule path is updated as shown in the figure; namely the updated fluid schedule path starts from the lower right vertex of the current T-N plane and runs parallel to the original fluid schedule path. Thus the position of the virtual T-N plane in the next node is always the same as or lower than the T-N plane of TNPA as shown in the figure. The $(k+2)^{\text{th}}$ T-N plane gets a negative additional nodal time $a_{i,k+2} < 0$ since the task completes the execution in the node. The time $(-a_{i,k+2})$ is apportioned among the other T-N planes in the same node. Therefore the negative additional nodal time is consumed by the other tasks.

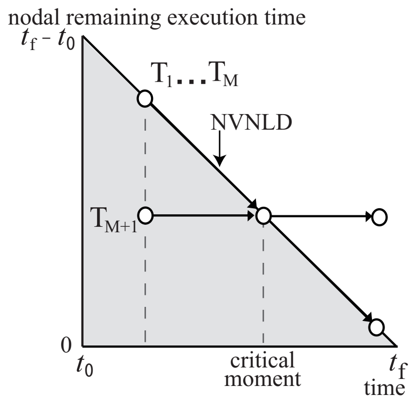

Figure 7. Critical moment.

3.1. Nodal and Global Feasibilities

E-TNPA must ensure both the nodal feasibility and the global feasibility. In TNPA, all tokens are globally feasible in obvious if they are nodally feasible. On the other hand, in E-TNPA, tokens may not be globally feasible even if they are nodally feasible. We focus on all the T-N planes in the k^{th} node. The problem is the way to assign $a_{i,k}$ (a_i for simplicity) for each task T_i. There are three restrictions to accomplish the nodal and global feasibilities in E-TNPA.

Cho et al. [4] show that *critical moment* is the sufficient and necessary condition where tokens are not nodally feasible in TNPA. It is partly available in E-TNPA. In E-TNPA, critical moment is the first time when more than M tokens simultaneously hit the NVNLD of the overlapped virtual T-N plane as shown in Figure 7. Theorem 1 shows that critical moment is undesirable for the nodal feasibility.

Theorem 1 (Critical Moment)**.** *If at least one critical moment occurs, tokens are not nodally feasible in E-TNPA.*

Proof. The sufficiency is the same as that of TNPA [4]. Assume that a critical moment occurs. At least one token protrudes the overlapped virtual T-N plane since at most M tokens can be selected on M processors. Non-selected tokens move out of the virtual T-N plane after the critical moment. The lunging tokens are no longer nodally feasible in the result since the slope of token paths is either 0 or -1. □

Theorem 1 implies that critical moment is the sufficient condition for the nodal feasibility in E-TNPA, while it is the sufficient and necessary condition in TNPA. The reason comes from the fact that tasks with initial nodal remaining execution time $l_{i,0} > t_f - t_0$ never arrive at the rightmost vertex of the virtual T-N plane (i.e., the tasks are outside the virtual T-N plane from the beginning). Consequently the condition $\exists i : l_{i,0} > t_f - t_0$ is not acceptable. Theorem 2 shows the first restriction (1) for the nodal feasibility.

Theorem 2 (Maximum Additional Nodal Time)**.** *The positions of all tokens are the same as or lower than that of the topmost vertex of the virtual T-N plane at time t_0 iff $a_i \leq (1-u_i)(t_f - t_0)$ for all i.*

Proof. The height of the virtual T-N plane is $t_f - t_0$. Thus the sufficient and necessary condition of the proposition is

$$\begin{aligned} & l_{i,0} \leq t_f - t_0 \\ \Leftrightarrow & u_i(t_f - t_0) + a_i \leq t_f - t_0 \\ \Leftrightarrow & a_i \leq (1-u_i)(t_f - t_0) \end{aligned} \tag{1}$$

for all i. □

The detailed condition at the time when a critical moment occurs can be derived from the idea of *total nodal utilization* introduced by Cho et al [4]. The *nodal utilization* of T_i at time t_j is defined as $r_{i,j} = l_{i,j}/(t_f - t_j)$. $S_j = \sum_{T_i \in \mathbf{T}} r_{i,j}$ denotes *total nodal utilization* at time t_j.

Theorem 3 (Total Nodal Utilization at Critical Moment)**.** *If a critical moment occurs at time t_j in E-TNPA, $S_j > M$.*

Proof. This proof is the same as that of TNPA [4] since the T-N plane of TNPA and the virtual T-N plane of E-TNPA have the same concepts. □

The contraposition of Theorem 3 implies that "no critical moment occurs if $S_j \leq M$ for all j." Therefore scheduling algorithms based on E-TNPA must ensure $S_j \leq M$ for all j. Theorem 4 presents the second restriction (2) for the nodal feasibility. The theorem only shows the initial condition at time t_0. The subsequent works (i.e., $^\forall j > 0 : S_j \leq M$) must be ensured by the scheduling algorithms.

Theorem 4 (Maximum Total Additional Nodal Time)**.** *$S_0 \leq M$ iff $A \leq (M-U)(t_f - t_0)$.*

Proof.

$$\begin{aligned} & S_0 \leq M \\ \Leftrightarrow & \sum_{T_i \in \mathbf{T}} r_{i,0} - \sum_{T_i \in \mathbf{T}} u_i \leq M - \sum_{T_i \in \mathbf{T}} u_i \\ \Leftrightarrow & \sum_{T_i \in \mathbf{T}} \left(\frac{l_{i,0}}{t_f - t_0} - u_i \right) \leq M - U \\ \Leftrightarrow & \sum_{T_i \in \mathbf{T}} (l_{i,0} - u_i(t_f - t_0)) \leq (M-U)(t_f - t_0) \\ \Leftrightarrow & \sum_{T_i \in \mathbf{T}} a_i \leq (M-U)(t_f - t_0). \end{aligned} \tag{2}$$

□

Finally the global feasibility is established. The rightmost vertex of the virtual T-N plane must be on or below the original fluid schedule path; otherwise the task may miss its deadline (time management between the current node and future nodes is not considered in this paper since the mechanism requires future information). Consequently remaining execution time must be retained to compare the rightmost vertex with the original fluid schedule path. $e_{i,j}$ denotes the remaining execution time of a task T_i at time t_j. The final restriction (3) for the global feasibility is as follows.

Theorem 5 (Minimum Additional Nodal Time)**.** *The rightmost vertexes of all the virtual T-N planes in the current node are on or below the original fluid schedule paths iff $a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i)$ for all i.*

Proof. The remaining execution time at the rightmost vertex of the virtual T-N plane is $e_{i,0} - u_i(t_f - t_0) - a_i$, and the remaining execution time of the original fluid schedule path at time t_f is $c_i - u_i(t_f - r_i)$ for each task T_i. Thus the right most vertexes of all the virtual T-N planes in the current node are on or below the original fluid schedule paths iff

$$\begin{aligned} & c_i - u_i(t_f - r_i) \geq e_{i,0} - u_i(t_f - t_0) - a_i \\ \Leftrightarrow & a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i) \end{aligned} \tag{3}$$

for all i. □

3.2. Time Apportionment

The purpose of time apportionment is that all processor time in each node is maximally consumed by active tasks. A simple time apportionment algorithm is shown in Figure 8. The algorithm first calculates the time unused by the taskset in TNPA at Line 2. Total processor time between time t_0 and t_f is $M(t_f - t_0)$, and the processor time used by TNPA in the interval is $U(t_f - t_0)$. Therefore the unoccupied time $(M-U)(t_f - t_0)$ can be apportioned among tasks. Then each nodal remaining execution time $l_{i,0}$ is initialized to $u_i(t_f - t_0)$ at Line 4 (i.e., it is the same as that in TNPA). The time kept by the tasks which complete the execution in the current node without any additional nodal time is retrieved at Lines 3-13. The unoccupied time L' is apportioned among the other tasks at Lines 14-25. All tasks run through Line 7, 18, or 20. The tasks which run through Line 7 certainly complete the execution in this node even if $a_i = 0$. The tasks which run through Line 18 complete the execution in this node if the task can receive enough a_i. The tasks which run through Line 20 can not complete the execution in this node. The algorithm ensures the nodal and global feasibilities described in the previous section as shown in the following three theorems.

```
Algorithm: ApportionTime
1: when time $t_0$ in each node
2:   $L' = (M - U)(t_f - t_0)$
3:   foreach $1..N$ as $i$
4:     $l_{i,0} = u_i(t_f - t_0)$
5:     if $e_{i,0} \le l_{i,0}$
6:       // retrieve additional nodal time ($a_i \le 0$)
7:       $a_i = e_{i,0} - l_{i,0}$
8:       $l_{i,0} = l_{i,0} + a_i$
9:       $L' = L' - a_i$
10:    else
11:      $a_i$ = NULL
12:    end if
13:  end foreach
14:  foreach $1..N$ as $i$
15:    if $a_i$ is NULL
16:      // apportion additional nodal time ($a_i \ge 0$)
17:      if $e_{i,0} \le t_f - t_0$
18:        $a_i = \min\{e_{i,0} - l_{i,0}, L'\}$
19:      else
20:        $a_i = \min\{(t_f - t_0) - l_{i,0}, L'\}$
21:      end if
22:      $l_{i,0} = l_{i,0} + a_i$
23:      $L' = L' - a_i$
24:    end if
25:  end foreach
26:  $L = L'$ // leftover time for proofs
27: end when
```

Figure 8. ApportionTime.

Theorem 6 (Maximum Additional Nodal Time by ApportionTime)**.** *Additional nodal time given by ApportionTime satisfies $a_i \le (1 - u_i)(t_f - t_0)$ for all i.*

Proof. The tasks which run through Line 7 satisfy the inequality since $a_i \le 0$ and $(1 - u_i)(t_f - t_0) \ge 0$. The tasks which run through Line 18 satisfy the inequality as follows:

$$\begin{aligned}
&(1 - u_i)(t_f - t_0) - a_i \\
=&(1 - u_i)(t_f - t_0) - (e_{i,0} - l_{i,0}) \\
&\Downarrow \text{since } e_{i,0} \le t_f - t_0 \text{ and } l_{i,0} = u_i(t_f - t_0) \\
\ge&(1 - u_i)(t_f - t_0) - (t_f - t_0) + u_i(t_f - t_0) = 0 \\
\Rightarrow&(1 - u_i)(t_f - t_0) \ge a_i.
\end{aligned}$$

The tasks which run through Line 20 also satisfy the inequality in the same manner as Line 18. □

Theorem 7 (Total Additional Nodal Time by ApportionTime)**.** *Total additional nodal time given by ApportionTime satisfies $A \le (M - U)(t_f - t_0)$.*

Proof. L' is initialized to $(M - U)(t_f - t_0)$ at Line 2. The tasks which run through Line 7 receive zero or negative additional nodal time, and the time is retrieved to L' at Line 9. Lines 18 and 20 limit the maximum receivable additional nodal time to L'. Thus total additional nodal time given by ApportionTime is at most $(M - U)(t_f - t_0)$. □

Theorem 8 (Minimum Additional Nodal Time by ApportionTime)**.** *Additional nodal time given by ApportionTime satisfies $a_i \ge e_{i,0} - c_i + u_i(t_0 - r_i)$ for all i.*

Proof. The tasks which run through either Line 7 or Line 18 satisfy the inequality for all i as follows:

$$\begin{aligned}
&a_i - (e_{i,0} - c_i + u_i(t_0 - r_i)) \\
=&(e_{i,0} - u_i(t_f - t_0)) - (e_{i,0} - c_i + u_i(t_0 - r_i)) \\
=&-u_i t_f + c_i + u_i r_i \\
=&c_i - (c_i/p_i)(t_f - r_i) \\
=&c_i\,(1 - (t_f - r_i)/p_i) \\
&\Downarrow \text{since } p_i \ge t_f - r_i \\
\ge&0 \\
\Rightarrow&a_i \ge e_{i,0} - c_i + u_i(t_0 - r_i).
\end{aligned}$$

The tasks which run through Line 20 also satisfy the inequality as follows. Since the tasks do not complete the execution in this node (i.e., $e_{i,0} > t_f - t_0$), the tasks are assigned zero or positive additional time $a_i \ge 0$ in the T-N planes between time r_i and t_f, where r_i is the release time of the current job. Therefore the T-N planes between time r_i and t_f have the shape which is shown in Figure 4. The rightmost vertex of the virtual T-N plane is on or below the original fluid schedule path as shown in Figure 4. Thus $a_i \ge e_{i,0} - c_i + u_i(t_0 - r_i)$ is satisfied by Theorem 5. □

Notice that ApportionTime assigns additional nodal time in increasing task index order which does not depend on any task parameters. Therefore arbitrary tasks can preferentially receive additional nodal time by task sorting (e.g., smallest remaining execution time first, aperiodic server first, and motor control first). Time apportionment algorithms can be designed to satisfy various types of systems.

The time apportionment can be realized by $O(N)$ or more complex algorithms. The number of the time apportionments can be bounded as follows. Fortunately the number of Event C and Event B does not affect the bound.

Theorem 9 (Upper-bound on the Number of the Time Apportionments)**.** *The number of the time apportionments during the time interval I is at most*

$$\sum_{T_i \in \boldsymbol{T}} \left\lceil \frac{I}{p_i} \right\rceil .$$

Proof. The number of task releases during the time interval I is $\sum_{T_i \in \mathbf{T}} \lceil I/p_i \rceil$. The number of the time apportionments is the same as that of task releases. □

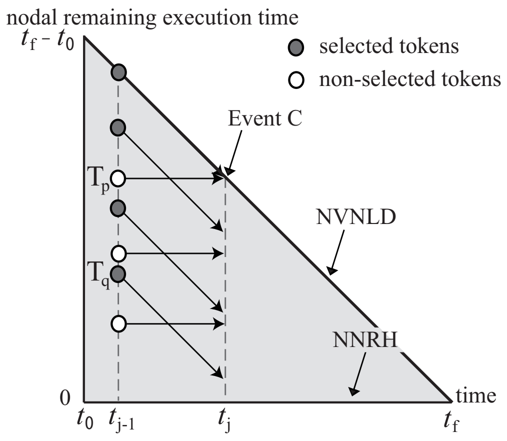

Figure 9. NVNLF ($M = 4$).

3.3. NVNLF Scheduling Algorithm

NVNLF assigns the highest priority to the tasks which have no virtual nodal laxity, and ties are broken arbitrarily. The tasks which are not on NVNLD can be flexibly executed by arbitrary tie-breaking rules as shown in Figure 9, while LNREF must select tokens in decreasing nodal remaining execution time. In NVNLF, either T_q or T_p incurs Event B or Event C at time t_j, respectively, where T_q has the smallest nodal remaining execution time in executed tasks, and T_p has the smallest virtual nodal laxity in non-executed tasks. First we show that S_j is monotonically decreasing.

Lemma 10 (Total Nodal Utilization)**.** $S_{j-1} \geq S_j$ *for all* j.

Proof. The induction hypothesis is

$$\begin{aligned} & S_{j-1} = S \\ \Leftrightarrow & \sum_{T_i \in \mathbf{T}} \frac{l_{i,j-1}}{t_f - t_{j-1}} = S \\ \Leftrightarrow & \sum_{T_i \in \mathbf{T}} l_{i,j-1} = S(t_f - t_{j-1}), \end{aligned} \tag{4}$$

where $S \leq M$. Assume that $N'(\leq M)$ tokens can be selected at time t_{j-1}. Since all tasks are in the virtual T-N plane at first from Theorems 2 and 6, $r_{i,j-1} \leq 1$ for all i. $S_{j-1} = S \leq N'$ is derived from $S \leq M$ and $r_{i,j-1} \leq 1$ for all i. The total remaining execution time decreases by $N'(t_j - t_{j-1})$ between time t_{j-1} and t_j. S_j is as follows:

$$\begin{aligned} S_j &= \frac{1}{t_f - t_j} \sum_{T_i \in \mathbf{T}} l_{i,j} \\ &= \frac{1}{t_f - t_j} \left(\left(\sum_{T_i \in \mathbf{T}} l_{i,j-1} \right) - N'(t_j - t_{j-1}) \right) \\ & \Downarrow \text{ since Equation (4)} \\ &= \frac{S(t_f - t_{j-1}) - N'(t_j - t_{j-1})}{t_f - t_j}. \end{aligned} \tag{5}$$

We have

$$\begin{aligned} & S_{j-1} - S_j \\ =& S - \frac{S(t_f - t_{j-1}) - N'(t_j - t_{j-1})}{t_f - t_j} \\ =& \frac{t_j - t_{j-1}}{t_f - t_j}(N' - S) \geq 0 \\ \Rightarrow& S_{j-1} \geq S_j. \end{aligned}$$

Thus $S_{j-1} \geq S_j$ for all j. □

Lemma 10 implies that the initial condition $S_0 \leq M$ is important for the nodal feasibility. NVNLF based on E-TNPA is optimal as shown in the following theorem.

Theorem 11 (Optimality of NVNLF)**.** *Any periodic taskset* $\boldsymbol{T}$ *with utilization* $U \leq M$ *will be scheduled to meet all deadlines on* M *processors by NVNLF.*

Proof. Theorem 7 and the proof of Theorem 4 show that $S_0 \leq M$ in E-TNPA. Lemma 10 shows that S_j is monotonically decreasing. Since $S_0 \leq M$ and S_j is monotonically decreasing, $S_j \leq M$ for all j. Therefore all tokens are nodally feasible since no critical moment occurs. The global feasibility is already established by E-TNPA. □

The number of NVNLF scheduler invocations can be bounded in the same manner as LNREF based on TNPA [4].

Theorem 12 (Upper-bound of NVNLF Scheduler Invocations over Time in E-TNPA)**.** *The number of NVNLF scheduler invocations during the time interval* I *in E-TNPA is at most*

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

Proof. This proof is the same as that of LNREF [4] since $a_i = 0$ for all i at the worst case. □

3.4. Work-Conserving

This section shows that NVNLF based on E-TNPA is work-conserving. A task T_i is *nodally active* at time t_j if and only if the task has non-zero nodal remaining execution time $l_{i,j} > 0$. Non-work-conserving algorithms based on TNPA can not select active and nodally inactive tasks.

The total processor time between time t_j and t_f is $M(t_f - t_j)$. When total nodal remaining execution time is $M(t_f - t_j)$, there is no margin for the nodal feasibility. In this case, total utilization becomes as follows:

$$S_j = \sum_{T_i \in \mathbf{T}} \frac{l_{i,j}}{t_f - t_j} = \frac{M(t_f - t_j)}{t_f - t_j} = M.$$

If $S_j = M$, no processor time can be unavailingly wasted between time t_j and t_f for the nodal feasibility. If $\exists j : S_j = M$, total utilization is always M throughout the current node as shown in the following lemma.

Lemma 13 (Total Nodal Utilization in Full-Loaded Condition)**.** $\forall j : S_j = M$ *if* $\exists j : S_j = M$.

Proof. Assume that there exists an integer x which satisfies $S_{x-1} = M$. (A) M is assigned to S in Equations (4) and (5). Then S_x becomes M. Thus $S_{x-1} = M \Rightarrow {}^{\forall}j > x-1 : S_j = M$ by the inductive method. (B) The proposition $S_{x-1} = M \Rightarrow {}^{\forall}j < x-1 : S_j = M$ is also correct as follows. Because S_j is monotonically decreasing from Lemma 10, ${}^{\forall}j < x-1 : S_j \geq M$. No critical moment occurs before time t_{j-1} since NVNLF is optimal from Theorem 11. Thus ${}^{\forall}j < x-1 : S_j \leq M$ from Theorem 3. For all $j < x-1$, $S_j = M$ is derived from $S_j \geq M$ and $S_j \leq M$. (A) and (B) show that $S_j = M$ for all j if there exists an integer j which satisfies $S_j = M$. □

The relation between S_0 and A is as follows.

Lemma 14 (Total Additional Nodal Time in Full-Loaded Condition)**.** $A = (M-U)(t_f - t_0)$ *iff* $S_0 = M$.

Proof. The inequality signs in the proof of Theorem 4 can be changed to the equal signs. Thus $A = (M-U)(t_f - t_0)$ iff $S_0 = M$. □

Maximum total additional nodal time is $(M-U)(t_f - t_0)$ based on Theorem 4. Consequently the time which can not be apportioned among tasks is calculated as $L = (M-U)(t_f - t_0) - A$ as shown at Line 26 in Figure 8. If the system is in a full-loaded condition $S_j = M$ at time t_j, L becomes zero as shown in the following theorem.

Lemma 15 (Leftover Time in Full-Loaded Condition)**.** $L = 0$ *iff* $\exists j : S_j = M$.

Proof.

$$\begin{aligned}
&\exists j : S_j = M \\
&\qquad \Downarrow \text{ since Lemmas 13 and 14} \\
&\qquad \Leftrightarrow A = (M-U)(t_f - t_0) \\
&\qquad \Downarrow \text{ since } L = (M-U)(t_f - t_0) - A \\
&\qquad \Leftrightarrow L = 0.
\end{aligned}$$

□

NVNLF based on E-TNPA is work-conserving as shown in the following theorem.

Theorem 16 (Work-Conserving)**.** *NVNLF based on E-TNPA is work-conserving.*

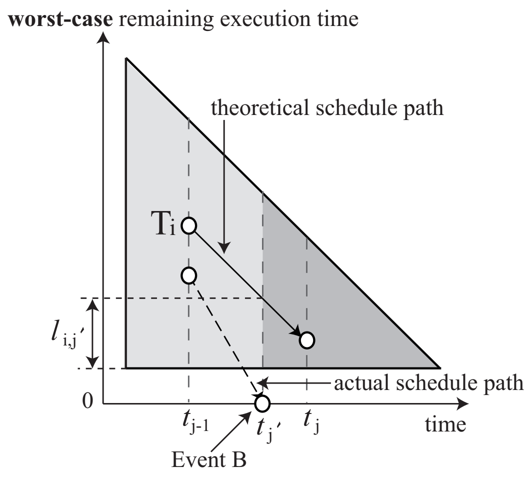

Figure 10. Actual and theoretical schedules.

Proof. Following three cases are possible. (A) If $\exists j : S_j = M$, there exist more than or equal to M nodally active tasks throughout the current node since $\forall j : S_j = M$ from Lemma 13 and $r_{i,j} \leq 1$ for all i, j. Thus M tokens can be always selected. (B-1) The case of $S_j < M$ and $L = 0$ is nonexistent since $L = 0 \Leftrightarrow S_j = M$ from Lemmas 13 and 15. (B-2) The case of $S_j < M$ and $L \neq 0$ is as follows. Only the tasks which run through Lines 18 and 20 incur non-work-conserving schedules (the tasks which run through Line 7 need not be considered since the tasks certainly complete the execution within the current node). Since $L \neq 0$, additional nodal time is not limited by L' at Lines 18 and 20 in Figure 8. Consequently the tasks which run through Lines 18 and 20 receive the maximum initial nodal remaining execution time $e_{i,0}$ and $t_f - t_0$ at Line 22 in Figure 8, respectively. The tasks which run through Line 18 complete the execution within the current node. The tasks which run through Line 20 are always executed in the current node. From (A), (B-1), and (B-2), NVNLF can always select all active tasks up to M in E-TNPA. □

3.5. T-N Plane Update

NVNLF based on E-TNPA may be non-work-conserving on 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 10 because c_i represents "worst-case" execution time. The figure shows a single virtual T-N plane for a task T_i. The scheduler can detect the early completion, and then Event B occurs at time $t_{j'}$. In this case, the time $l_{i,j'}$, which represents the theoretical value, can not be reused by the other tasks since the time is held by the inactive task T_i. Time reapportionment such as ReapportionTime shown in Figure 11 is performed at every task completion (i.e., the assumption "a_i

```
Algorithm: ReapportionTime
 1: when a task $T_c$ completes the execution at time $t_{j'}$
 2:   // retrieve the additional nodal time of the task $T_c$
 3:   $L' = l_{c,j'}$
 4:   foreach $1..N$ as $i$
 5:     if $e_{i,j'} > l_{i,j'}$
 6:       // apportion additional nodal time ($\Delta \geq 0$)
 7:       if $e_{i,j'} \leq t_f - t_{j'}$
 8:         $\Delta = \min\{e_{i,j'} - l_{i,j'}, L'\}$
 9:       else
10:         $\Delta = \min\{(t_f - t_{j'}) - l_{i,j'}, L'\}$
11:       end if
12:       $\alpha_i = \alpha_i + \Delta$
13:       $l_{i,j'} = l_{i,j'} + \Delta$
14:       $L' = L' - \Delta$
15:     end if
16:   end foreach
17: end when
```

Figure 11. ReapportionTime.

never changes throughout a node" shown in the previous section is overturned). ReapportionTime is based on ApportionTime shown in Figure 8. The difference between ApportionTime and ReapportionTime is that ReapportionTime retrieves unavailable time only from the completing task . All tokens remain nodally feasible and globally feasible, and NVNLF based on E-TNPA is work-conserving as shown in the following two theorems.

Theorem 17 (Feasibility on Practical Environments)**.** *All tokens remain nodally and globally feasible even if the T-N plane is updated by ReapportionTime.*

Proof. Theorem 11 shows that all tokens are nodally and globally feasible before the update. If all tokens are nodally feasible, no critical moment occurs from Theorem 1. Thus we obtain $S_{j'} \leq M$ before the update from Theorem 3. Consider $t_{j'}$ as t_0 (i.e., the virtual T-N plane becomes the deeply shaded triangle shown in Figure 10). $S_{j'} \leq M$ can be written to $S_0 \leq M$. Then the deeply shaded triangle is updated by ReapportionTime. In this case, Inequalities (1)(2)(3) are preserved since ReapportionTime is based on ApportionTime. Therefore all tokens remain nodally and globally feasible in the updated T-N plane. □

Theorem 18 (Work-Conserving on Practical Environments)**.** *NVNLF based on E-TNPA is work-conserving if the T-N plane is updated by ReapportionTime.*

Proof. Consider $t_{j'}$ as t_0. Then the deeply shaded triangle shown in Figure 10 is updated by ReapportionTime. Theorem 16 can be applied to the updated deeply shaded triangle since ReapportionTime is based on ApportionTime. Therefore NVNLF based on E-TNPA is work-conserving if the T-N plane is updated by ReapportionTime. □

The time reapportionment policies can be designed in the same way as the time apportionment. Therefore arbitrary tasks can preferentially receive additional nodal time.

Time reapportionment can be realized by $O(N)$ or more complex algorithms. The number of the time reapportionments can be bounded by the following theorem.

Theorem 19 (Upper-bound on the Number of Time Reapportionments)**.** *The number of the time reapportionments during the time interval I is at most*

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

Proof. The number of task releases during the time interval I is $\sum_{T_i \in \mathbf{T}} \lceil I/p_i \rceil$. Thus the number of task completions during the interval is at most $\sum_{T_i \in \mathbf{T}} (\lceil I/p_i \rceil + 1)$. □

4. Simulation

The advancement of E-TNPA is evaluated by comparing with TNPA in terms of the number of task preemptions. The number of task preemptions directly affects the scheduling overhead. In each node, tasks are scheduled by NVNLF, and ties are broken by LNREF. In E-TNPA, the tasks which have the smallest remaining execution time preferentially receive additional nodal time to reduce the number of active tasks as early as possible (i.e., tasks are indexed in increasing remaining execution time order in ApportionTime). EKG and EDZL [9] are also compared. Although EDZL is not optimal, it is known as an efficient algorithm. The schedulable total utilization bound of EDZL does not exceed $(1 - 1/e)M \simeq 0.6321M$, where e is the Euler's number (the exact bound has not been presented). In EDZL, the results of schedulable tasksets are presented (i.e., unschedulable tasksets are not considered). Pfair algorithms are not compared since the run-time overhead is clearly large due to their quantum-based scheduling approach.

Each simulation is modeled as sixteen processors and a taskset. The taskset is initially empty. A new task is appended to the taskset as long as $U \leq U_{\text{target}}$, where U_{target} is the target utilization for each simulation. For each task T_i, its utilization u_i is computed based on a uniform distribution within the range of $[0.01, 1.0]$. Only the utilization of the last task is adjusted so that U becomes equal to U_{target}. Each task T_i is generated with the period p_i in the integer rang of $[100, 3000]$ and the worst-case execution time $c_i = u_i p_i$. For each task T_i, actual execution time is equal to its worst-case execution time c_i since the worst-case number of task preemptions is important in real-time systems. The simulation interval is $[0, \min\{\text{lcm}\{p_i | T_i \in \mathbf{T}\}, 2^{32}\}]$. In order to measure the average number of task preemptions, hundred simulations are conducted for each total utilization.

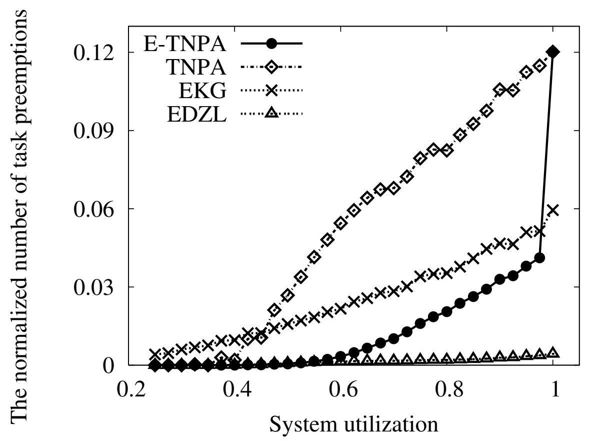

Figure 12. The number of task preemptions.

Figure 12 shows the results plotted at every 0.025 system utilization U/M for each algorithm. The figure shows system utilization U/M on the horizontal axis and the average number of task preemptions on the vertical axis normalized by M and the simulation interval. When the number of tasks is small, few preemptions occur in global scheduling (e.g., E-TNPA, TNPA, and EDZL) since the workload is distributed among processors. In EKG, on the other hand, the number of task preemptions in lower system utilization is larger than the other algorithms since EKG concentrates the workload on some processors. EDZL is the most efficient algorithm in the simulation; however some tasksets miss the deadlines in EDZL when U exceeds the schedulable bound. E-TNPA significantly reduces the number of task preemptions relative to TNPA since the number of active tasks decreases early. The last big jump of E-TNPA comes from the fact that (1) $M-U = 40\%$ time of a processor is still available for the time apportionment at the total utilization $U = 15.6$ ($U/M = 0.975$), and (2) E-TNPA creates the same schedule as TNPA at the worst case $U/M = 1$ because a_i becomes zero for all i. The difference between E-TNPA and EDZL is not significantly large in the utilization where EDZL can guarantee the schedulability.

5. Conclusions and Future Work

E-TNPA proposed in this paper leverages the ideas of *time apportionment* and *virtual nodal laxity*. NVNLF assigns the highest priority to the tasks which have no virtual nodal laxity, and ties are broken arbitrarily. E-TNPA and NVNLF have attractive advantages against TNPA and LNREF in the sense that (1) the schedule becomes work-conserving, and (2) arbitrary policies can be leveraged for both the time apportionment and the scheduling tie-break.

E-TNPA has only two disadvantages against TNPA. One is that E-TNPA must retain task's remaining execution time. However it causes little overhead because the remaining execution time is not actual one but theoretical one. The other is that E-TNPA must apportion time at every node to realize work-conserving schedule. It can be realized by $O(N)$ or more complex algorithms, where N is the number of tasks. The overhead of the time apportionment is worthy of little attention if the algorithm is reasonably simple. Additionally the number of the time apportionments over time can be bounded by the number of task releases.

Efficient time apportionment is a topic for the future work. The time apportionment algorithm proposed in this paper apportions time only among the T-N planes in the same node. The additional nodal time of the current node can be lent to or borrowed from the future nodes of the other tasks. In consequence, we might be hopefully able to find an optimal real-time scheduling algorithm for multiprocessors which can schedule tasks with much lower overhead.

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

[7] T. Matsui, H. Hirukawa, N. Yamasaki, H. Ishikawa, S. Kagami, F. Kanehiro, H. Saito, and T. Inamura. Distributed Real-Time Processing for Humanoid Robots. In *Proc. of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 205–210, Aug. 2005.

[8] A. Srinvasan, P. Holman, and J. Anderson. Integrating Aperiodic and Recurrent Tasks on Fair-Scheduled Multiprocessors. In *Proc. of the 14th Euromicro Conference on Real-Time Systems*, pages 19–28, June 2002.

[9] H.-W. Wei, Y.-H. Chao, S.-S. Lin, K.-J. Lin, and W.-K. Shih. Current Results on EDZL Scheduling for Multiprocessor Real-Time Systems. In *Proc. of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 21–24, Aug. 2007.