Regular Paper

Further Analysis with Linear Programming on Blocking Time Bounds for Partitioned Fixed Priority Multiprocessor Scheduling

ZHONGQI MA\(^1,a\) RYO KURACHI\(^1,b\) GANG ZENG\(^2,c\) HIROAKI TAKADA\(^1,d\)

Received: November 17, 2017, Accepted: May 10, 2018

Abstract: The recently developed FMLP\(^+\) provides significant advantages for partitioned fixed priority scheduling, since it ensures asymptotically optimal O(n) maximum priority-inversion blocking. The constraints under the FMLP\(^+\) can be exploited to determine bounds on the blocking time. However, these bounds may be pessimistic since shared resources local to a processor do not incur priority-inversion blocking in some cases. Consequently, a schedulable task set may be erroneously judged as unschedulable because of these pessimistic values. Based on our analysis, additional constraints were added to compute the maximum blocking time bound of each task with linear programming and the corresponding worst-case response time. The results of experiments show our proposed strategy is less pessimistic than existing strategies. Meanwhile, we also demonstrate that local resource sharing should be used instead of global resource sharing where possible.

Keywords: partitioned fixed priority, multiprocessor scheduling, FMLP\(^+\), blocking time bound, linear programming

1. Introduction

Since an increasing number of complex systems rely on computer control, real-time computing is becoming essential in a variety of fields. There are countless examples of applications requiring real-time embedded systems, including industrial automation, automotive networks, robotics, military systems, and even smart toys such as LEGO Mindstorms EV3.[1], [2], [3], [4].

Fixed priority preemptive partitioned scheduling algorithms for multiprocessor systems have been shown to be predictable [5]. The main advantage of using a partitioning approach in multiprocessor scheduling is that a wealth of real-time scheduling techniques and analyses for uniprocessor systems can be applied once an allocation of tasks to processors has been achieved [1]. The following optimality results for uniprocessor scheduling have a strong influence on research into partitioned multiprocessor scheduling. Rate monotonic (RM) priority assignment is the optimal policy for periodic task sets with implicit deadlines, if preemptive uniprocessor scheduling using fixed priority is considered [6]. On the other hand, since tasks are statically assigned to processors under partitioned scheduling algorithms, there are no job migrations and this serves to reduce overheads.

However, the task allocation problem is analogous to the bin packing problem. It is known to be NP-hard [7], which is the main disadvantage of applying a partitioning method to multiprocessor scheduling. Hence task allocation is a well-studied problem in the field of real-time systems. A synchronization-aware task allocation strategy has been proposed to reduce the scheduling penalties associated with remote task synchronization [8]. That is, tasks sharing a common mutex are bundled and then are colocated, transforming the shared mutex into a local mutex. This strategy is more efficient than the previous ones.

For the task synchronization, each task executes critical sections on its assigned processor in shared-memory systems. The FIFO Multiprocessor Locking Protocol (FMLP\(^+\)) [9], a refinement of the Flexible Multiprocessor Locking Protocol (FMLP) [10] for partitioned scheduling, is also a shared-memory locking protocol. Similarly, FIFO queues are employed to order requests for resources as well as blocked tasks. Although the FMLP cannot ensure asymptotically optimal priority-inversion blocking, the updated FMLP\(^+\) assigns priorities to tasks: the effective priority of a lock holder is the time at which it requested the lock. It is this improved rule that ensures asymptotically optimal maximum priority-inversion blocking [9].

Concerning the calculation of blocking time bound, a linear-programming-based analysis technique, which is significantly less pessimistic than previous approaches, was developed in Ref. [11]. The problem of obtaining bounds on the maximum blocking can be transformed into a linear programming (LP) by imposing a few constraints, whereas previous methods require the analyst to explicitly consider each critical section. Thus, LP solvers provide a straightforward approach for obtaining bounds, such as the GNU Linear Programming Kit (GLPK) [12] or the
Cplex Optimizer developed by IBM [13].

1.1 Contributions

Under the FMLP', the bounds on maximum blocking time may be less pessimistic if local critical sections are considered. As is described in Ref. [8], local synchronization eliminates the scheduling penalties associated with global synchronization. We analyzed the solutions derived from the LP solver and determined how the above technique can be enhanced with additional constraints. As a result, those task sets which were erroneously judged as unschedulable are judged as schedulable with our proposed approach. Otherwise expensive processors with higher performance might be considered to improve the schedulability, which leads to higher costs. The effectiveness and merits of our strategies were demonstrated in the experiments. We also evaluated how much the average blocking time bounds were improved.

1.2 Organization

We introduce the task model and related assumptions in Section 2. We then review three kinds of delays, the blocking fraction, and the objective function in Section 3. We also state how to formalize blocking time bounds as a linear optimization problem, and how to compute the worst-case response time (WCRT) of each task. In Section 4, we analyze the pessimism of existing constraints through a numerical example. Additional constraints are developed in Section 5, and evaluations of our approach are provided in Section 6. Finally, we present our conclusions and areas for future work in Section 7.

2. Definitions

2.1 Assumptions

Fixed priority scheduling, with tasks having conventional RM scheduling priorities, is considered in this paper. Each task’s worst-case execution time (WCET) and period are assumed to be known in advance. For simplicity, implicit-deadline task sets use worst-case execution times that any task accesses its maximum number of any critical section in a single access. That is, each task’s deadline is equal to its corresponding period.

All of the critical sections are assumed to be non-nested. Each job for a task requests and holds at most one resource at any time. Jobs release all resources before their completion. The FMLP', which ensures mutual exclusion, is utilized when two or more jobs access the same resource. If a job requires a locked resource, it must wait and incurs a delay until the requested resource is released. Semaphore protocols are used in this paper, under which jobs wait by suspending instead of spinning.

2.2 Task Model

Consider a real-time workload consisting of n sporadic tasks \( \tau = \{ T_1, T_2, \ldots, T_n \} \) scheduled on \( m \) identical processors \( \{P_1, P_2, \ldots, P_m\} \), whose cores have equal processing capabilities. Each task has a unique and fixed base priority under partitioned fixed priority scheduling. For brevity, tasks are ordered in strictly decreasing order of base priorities. That is, \( i < j \) implies that \( T_i \) has a higher priority than \( T_j \). Tasks are assigned to the \( m \) processors statically. The function \( P(T_i) \) returns \( T_i \)'s assigned processor.

Each task is considered to be an alternating sequence of normal execution segments and critical section execution segments [8]. \( T_i \) is described as follows:

\[
T_i : (E_{i,1}, C_{i,1}, E_{i,2}, C_{i,2}, \ldots, E_{i,nr(i)}, C_{i,nr(i)}),
\]

where \( E_{i,j} \) is the WCET of \( T_i \)'s \( j \)th normal execution, \( C_{i,k} \) is the WCET of \( T_i \)'s \( k \)th critical section, \( s(i) \) is the number of \( T_i \)’s normal execution segment, and thus \( T_i \)'s critical section execution segment must be \( s(i) - 1 \). Therefore, the WCET of \( T_i \) is denoted by \( e_i \) such that

\[
e_i = \sum_{j=1}^{s(i)} E_{i,j} + \sum_{k=1}^{s(i)-1} C_{i,k}.
\]

In this paper, we mainly consider the following situation: \( \forall E_{i,j} : E_{i,j} > 0 \).

The deadline (= period) of \( T_i \) is denoted as \( d_i \), and the utilization of \( T_i \) is defined as \( u_i = e_i/d_i \). Let \( J_i \) denote a job of \( T_i \), and \( J_i \)'s response time is the difference between its finishing time and arrival time. \( T_i \)'s WCRT \( r_i \) denotes the maximum value of any \( J_i \)'s response time. \( T_i \)'s bound on maximum blocking time is denoted by \( b_i \).

2.3 Resources

The tasks share \( n_r \) serially reusable resources \( l_1, l_2, \ldots, l_{n_r} \) besides the \( m \) processors. Here, \( N_{l,q} \) is the maximum number of times that any \( J_i \) access its \( l_q \), and \( L_{l,q} \) denotes \( T_i \)'s maximum critical section length, which means the maximum duration that any \( J_i \) uses its \( l_q \) in a single access. That is, \( L_{l,q} = 0 \) if \( N_{l,q} = 0 \).

A resource is called a local resource in this paper if all of the tasks accessing the resource are assigned to the same processor. Conversely, a resource which is accessed by tasks allocated to different processors is said to be a global resource [14]. We let \( P(l_q) \) denote the processor on which the local resource \( l_q \) is located.

Consider Table 1. There are 6 tasks sharing 3 resources. Tasks \( T_1, T_3, \) and \( T_5 \) are assigned to processor \( P_1 \) and \( T_2, T_4, \) and \( T_6 \) are assigned to \( P_2 \). Resource \( l_1 \) is accessed by \( T_1, T_3, \) and \( T_5; l_2 \) by \( T_1 \) and \( T_2; l_3 \) by \( T_2, T_4, \) and \( T_6. From the above definitions, \( l_1 \) and \( l_3 \) are local resources, while \( l_2 \) is a global resource.

Under the FMLP', priority of a task is raised to expedite request completion after it requests a resource.

3. Blocking Time Formulation

A linear-programming-based blocking analysis technique has been proposed which offers substantial improvements over prior blocking time bounds [11]. It can be adapted under various protocols besides the FMLP'.

3.1 Delay

There are three kinds of delays common to all shared-memory
blocking protocols [9].

(1) Direct Request Delay: This arises under any protocol whenever a job \( J_i \) requests an unavailable resource. \( J_i \) can potentially incur blocking while waiting for the lock holder to finish its critical section. Direct request delay occurs only via resources which \( J_i \) requests.

(2) Indirect Request Delay: This arises if \( J_i \) waits for another job \( J_a \) to release a resource but \( J_a \) has been preempted by a third job \( J_b \), thus increasing \( J_i \)'s total acquisition delay. Indirect request delay can arise due to shared resources which \( J_i \) never accesses.

(3) Preemption Delay: This arises when \( J_i \) is preempted by a priority-boosted, lower-priority job. Thus, preemption delay affects even tasks which do not access shared resources.

3.2 Blocking Fraction

The concept of the blocking fraction was proposed in Ref. [11] to express partial blocking. For each \( T_x \), \( \mathcal{R}^{q,p}_x \) denotes the \( \tau^{th} \) request for \( \mathcal{T} \) by jobs of \( T_x \) from \( J_i \)'s release until its completion. Here, \( b_{i}^{q,p} \) denotes the blocking incurred by \( J_i \) due to the execution of \( \mathcal{R}_i^{q,p} \). The corresponding blocking fraction is as follows:

\[
X_i^{q,p} = \frac{b_{i}^{q,p}}{\tau},
\]

where \( X_i^{q,p} \in [0, 1] \). \( X_i^{q,p} \) indicates the fraction of blocking time which was observed during \( \mathcal{T}_i \), out of the total blocking time that could arise from \( \mathcal{R}_i^{q,p} \). Here, \( X_i^{q,p} \), \( X_i^{q,p} \), and \( X_i^{q,p} \) are the fractions of blocking due to direct request delay, indirect request delay, and preemption delay, respectively.

We refer to blocking fractions by means of an illustration. Consider Fig. 1. \( J_1 \) suffers blocking twice. Preemption delay is incurred by \( J_1 \) during [3, 6) and direct request delay by \( J_2 \) during [7, 8]. We can see \( b_{1}^{3,1,1} = 3 \), \( b_{1}^{2,2,1} = 1 \) from Fig. 1 and \( L_{3,1} = 5 \), \( L_{2,2} = 3 \) from Table 1. Then,

\[
X_{1,1,1} = \frac{b_{1}^{3,1,1}}{L_{3,1}} = \frac{3}{5} \quad \text{and} \quad X_{1,2,1} = \frac{b_{1}^{2,2,1}}{L_{2,2}} = \frac{1}{3}.
\]

3.3 Objective Function

A task set is schedulable if each task’s WCRT, which depends on its maximum blocking time bound, is less than or equal to its deadline (i.e., \( r_i \leq d_i \) for each \( T_x \)). Therefore, we consider each task’s bound on maximum blocking time to be the objective function.

Blocking depends on the access of both the local and global resources. Let \( b_i^{l} \) and \( b_i^{r} \) denote bounds on the maximum local and remote blocking time, respectively. Similarly, \( \tau^{l} \) and \( \tau^{r} \) denote the sets of the local and remote tasks. \( N_q^{l} \) denotes the number of requests by \( T_x \) for \( \mathcal{T}_q \)'s release until its completion. \( N_q^{r} \) can be bounded for a sporadic task \( T_x \) [9]. The objective function is described as follows [11]:

\[
\begin{align*}
\text{maximize} & \quad b_i^{l}, b_i^{r} \\
\text{subject to} & \quad i = 1, 2, \ldots, n
\end{align*}
\]

where

\[
\begin{align*}
b_i^{l} &= b_i^{l} + b_i^{r}, \\
b_i^{l} &= \sum_{T_x \in \tau^{l}} \sum_{q \in \tau} X_{i,q}^{l} \cdot L_{x,q}^{l}, \\
b_i^{r} &= \sum_{T_x \in \tau^{r}} \sum_{q \in \tau} X_{i,q}^{r} \cdot L_{x,q}^{r}, \\
\tau^{l} &= (T_x | P(T_x) = P(T_i) \land x \neq i), \\
\tau^{r} &= (T_x | P(T_x) \neq P(T_i), \\
N_q^{l} &= \left( \frac{r_i + r_q}{d_i} \right) \cdot N_q^{l}.
\end{align*}
\]

The objective function is used to obtain each \( T_x \)’s theoretical maximum blocking time caused by other tasks \( T_1, T_2, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n \) in the same set task, i.e., \( T_x \)’s bound on maximum blocking time. In other words, each task has its own maximum blocking time bound. When we compute \( T_x \)’s maximum bound, we consider how \( T_i \) is delayed by other tasks and neglect other tasks’ blocking time. We have to compute all task bounds one by one so as to do the analysis of schedulability.

3.4 Linear Optimization

Figure 1 outlines the computation of the blocking fraction based on its theoretical definition. The worst-case scenarios in hard real-time issues are the main concern. However, Fig. 1 does not represent the worst-case scenario, and \( b_i^{3,1,1} = 3 \) or \( b_i^{2,2,1} = 1 \) is not the maximum blocking time bound, either. Only when a task’s maximum blocking time bound is obtained shall we be able to make the corresponding scenario.

For the FMLP*, the bound on maximum blocking time can be formalized as a linear optimization problem in terms of the blocking fractions (variables), Eq. (2) (objective function) mentioned above, and Constraint 1, 9–14 in Ref. [11].

3.5 Response Time Analysis

We use response time analysis to determine each \( T_x \)’s \( r_i \) [8], [15]. Under blocking conditions, the response time of a task with a fixed priority can be calculated by the following recurrent relation.

(1) Initial Condition: Iteration starts with Eqs. (5a) and (5b), which are the first points in time that \( T_i \) and \( T_x \) could possibly complete.

\[
\begin{align*}
t_i^{(0)} &= e_i, \\
t_x^{(0)} &= e_x, \\
N_q^{(0)} &= \left( \frac{t_i^{(0)} + t_x^{(0)}}{d_x} \right) \cdot N_q^{(0)}.
\end{align*}
\]
After each $N_{i,j}^t$ is substituted into the constraints, we can solve the LP and obtain $b_1^{(0)}$ and $b_2^{(0)}$, the initial bounds on the maximum local and remote blocking time respectively.

(2) Recurrence: If $r_i^{(0)} = r_i^{(2k-1)}$, then $r_i^{(2k)}$ is the actual WCRT for $T_i$; that is, $r_i = r_i^{(2k)}$. Otherwise, we have to compute each $r_i$ iteratively with Eq. (6) until $r_i$ converges for each task.

$$r_i^{(2k)} = e_i + P_i^{(2k-1)} + \sum_{T_j \in \mathcal{P}(T_i)} \left( \frac{\mathbf{d}_h^{(2k-1)}}{\mathbf{d}_h} \right) \cdot e_j.$$  \hspace{1cm} (6)

Here, $b_i^{(2k-1)} = b_i^{(2k-1)} + b_i^{(2k-1)}$. Once $r_i$ is calculated, the feasibility of $T_i$ is guaranteed if and only if $r_i \leq d_i$. Conversely, a task set is judged as unschedulable if $r_i > d_i$ for at least one $T_i$.

4. Analysis on Pessimism

A numerical example is presented to demonstrate the pessimistic results derived under the existing constraints. Recall Table 1. We made use of GLPK to solve the generated LPs of each task. Concerning $T_1$, the solution can be seen in the first row of Table 2. Other variables equal to 0, such as $X_{1,1}^{2,1}$, $X_{1,1}^{4,3}$, and $X_{1,1}^{6,3}$ are not listed in the table, because they do not alter the value of the objective function.

First, we consider the blocking incurred by $J_1$ due to direct request delays. Since $X_{1,1}^{3,1} = 1$ and $X_{1,1}^{5,1} = 1$, $J_1$ must incur blocking twice when requesting $l_1$. However, there is no way for $J_1$ to $J_3$ to execute if $J_1$ arrives or resumes earlier than (or at the same time as) $J_3$ or $J_5$ (see Fig. 2(a)). Conversely, $J_1$ cannot request $l_1$ if $J_3$ or $J_5$ is the lock holder of $l_1$, because of priority-boosted $J_3$ and $J_5$’s unfinished normal execution segment (see Fig. 2(b)). Thus, it is impossible that $R_{1,1}^{3,1}$ or $R_{1,1}^{5,1}$ causes $J_1$ to incur a direct request delay.

Similarly, $R_{1,1}^{2,1}$ and $R_{1,1}^{6,1}$ also cause $J_1$ to incur an indirect request delay twice in spite of $l_2$ which $J_1$ never accesses. However, only when $J_1$ waits for $J_2$ to release $l_2$ but $J_2$ has been preempted by $J_4$ or $J_6$ does $b_{1,1}^{2,1}$ or $b_{1,1}^{6,1}$ occur. Since $J_2$ in a critical section has the highest priority on $P_2$, $b_{1,1}^{2,1}$ or $b_{1,1}^{6,1}$ cannot occur.

It is possible for $b_{1,1}^{2,1}$ to occur because $T_1$ and $T_2$ are assigned to two different processors. $J_2$ can affect $J_1$ directly as long as $J_2$ has locked $l_2$ when $J_1$ issues a request for $l_2$.

On the other hand, there are other feasible solutions due to the symmetry of Constraint 1 in Ref. [11]. Now that $J_1$ is never directly delayed by $J_1$ or $J_2$, the priority-boosted $J_3$ and $J_5$ are likely to preempt $J_1$. $X_{1,1}^{2,1} = X_{1,1}^{3,1} = X_{1,1}^{4,1} = X_{1,1}^{6,1} = X_{1,1}^{6,3} = 1$ is another optimal solution. Here, $X_{1,1}^{3,1} = 1$ and $X_{1,1}^{6,1} = 1$ cannot change $b_1$ (or $r_i$) but they provide a better result than $X_{1,1}^{3,1} = 1$ and $X_{1,1}^{6,3} = 1$. Of course, the values of $X_{1,1}^{3,1}$ and $X_{1,1}^{6,3}$ are still pessimistic.

Overall, the actual execution time for $J_1$ is much lower than the theoretical upper bounds. It is impossible to make a corresponding figure for the worst-case scenario for $J_1$. We computed $b_1$ ($b_1 = 29$ in Table 2) by Eq. (3) and obtained $r_1 = 40$ and $r_2 = 51$, as listed in the second row of Table 3 after iteratively calculating $r_i$. Both $T_1$ and $T_2$ miss their deadlines, and so the task set is regarded as unschedulable. It is therefore necessary for us to derive more realistic and less pessimistic values.

### Table 2 Comparison of the blocking fractions and the maximum blocking time bounds.

<table>
<thead>
<tr>
<th>Method</th>
<th>$X_{1,1}^{2,1}$</th>
<th>$X_{1,1}^{3,1}$</th>
<th>$X_{1,1}^{4,1}$</th>
<th>$X_{1,1}^{6,1}$</th>
<th>$X_{1,1}^{5,1}$</th>
<th>$X_{1,1}^{6,3}$</th>
<th>$b_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing method</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>Proposed method</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

(a) $J_1$, $J_2$, and $J_3$ arrive at $t_0$. It is impossible for $J_2$ or $J_3$ to block $J_1$. Note that: if $C_{1,2}$ is delayed by another task on a remote processor, $J_1$ cannot be directly delayed by $J_2$ or $J_3$, because of $E_{1,2}$. In this case, $T_3$ is equivalent to another task: $(1, 2, 1)$.

(b) $J_1$, $J_2$, and $J_3$ arrive at $t_0$, $t_1$, and $t_2$ respectively. $J_3$ is blocked by $J_2$ because the effective priority of $J_3$ in a critical section is higher than the base priority of $J_3$, and similarly, $J_1$ is blocked by $J_3$. Note that: $R_{1,1}^{3,1}$ causes $J_1$, $R_{1,1}^{3,1}$ causes $J_2$ to incur preemption delay instead of direct request delay.

### Table 3 Comparison of the WCRT.

<table>
<thead>
<tr>
<th>$T_i$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
<th>$T_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_i$</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>$r_i$ (existing method)</td>
<td>40</td>
<td>51</td>
<td>26</td>
<td>26</td>
<td>28</td>
<td>38</td>
</tr>
<tr>
<td>$r_i$ (proposed method)</td>
<td>21</td>
<td>25</td>
<td>26</td>
<td>22</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Improvement (%)</td>
<td>47.50</td>
<td>50.98</td>
<td>23.08</td>
<td>0</td>
<td>21.43</td>
<td>26.32</td>
</tr>
</tbody>
</table>

5. Improvements

Since the above example showed us that the bounds on the maximum blocking time obtained by the original constraints were pessimistic, we tried to reduce pessimism through additional constraints.

### 5.1 Preliminaries

To simplify the necessary lemma and theorems, we define $\tilde{J}_i$ as a job starting with a normal execution segment (i.e., $E_{1,1} \neq 0$). The lemma is then expressed as follows.

**Lemma 1** Under the FMLP, if $\tilde{J}_i$ in a critical section is preempted by another job $J_a$, then $J_a$ must be in an earlier-initiated global critical section.

**Proof:** Suppose not. Then $J_a$ is in a local critical section, which was initiated earlier than $\tilde{J}_i$ but has not yet finished. Since $J_a$’s critical section is local, $J_a$ cannot be affected by jobs on other processors. Thus, there must exist a job $J_b$ which is local to $J_a$ and is executing its critical section. $J_a$ will continue executing its local critical section as soon as $J_a$ finishes its critical section. $\tilde{J}_i$ cannot execute its critical section because $J_a$’s effective priority is higher than $\tilde{J}_i$’s at that time. It is impossible that $J_a$ preempts $\tilde{J}_i$ in a critical section. Contradiction. □
5.2 Theorems

We were able to deduce the following theorem for the direct request delay after obtaining $X_{1,D} = 0$ and $X_{2,D} = 0$ in the example above.

**Theorem 1** $\bar{J}_a$ never incurs direct request delays due to other jobs in local critical sections under the FMLP$^\ast$.

**Proof:** Suppose not. Then there exists a lower-priority job $J_b$ local to $\bar{J}_a$, which delays $\bar{J}_a$ directly. According to the FMLP$^\ast$, lock holders are scheduled in order of increasing lock-request time [11]. $J_b$ must request a local lock earlier than $\bar{J}_a$. $\bar{J}_a$ has also requested the same lock before $J_b$ releases the lock. But the effective priority of a lock holder is higher than any other local jobs in normal execution segments under the FMLP$^\ast$. Thus, $\bar{J}_a$ cannot execute its normal execution segment before its critical section when $J_b$ is in the critical section. $\bar{J}_a$ cannot issue the request if $\bar{J}_a$ does not finish its normal execution segment first. Contradiction.

Similarly, the second theorem about indirect request delays can be derived from the fact that $X_{3,1,1} = 0$ and $X_{2,3,1} = 0$.

**Theorem 2** Under the FMLP$^\ast$, $\bar{J}_a$ never incurs indirect request delays caused by $J_b$’s local critical sections if $\bar{J}_a$ waits for $\bar{J}_b$ to release a resource.

**Proof:** Suppose not. From the definition of an indirect request delay, $\bar{J}_a$ must be preempted by a third job $J_b$. From the Lemma, it is impossible for $\bar{J}_a$ to be preempted by $J_b$ in a local critical section. Contradiction.

By analysis, we also surmised that the preemption delay is related to the global critical sections. Let $n(i)$ denote the number of global resources accessed by $T_i$.

**Theorem 3** The number of times that priority-boosted, lower-priority jobs $\bar{J}_a$ in critical sections can preempt $\bar{J}_a$ is at most $1 + \bar{n}(i)$ under the FMLP$^\ast$.

**Proof:** From the Lemma, $\bar{J}_a$ must be in a normal execution segment. Other lower-priority jobs local to $\bar{J}_a$ can delay $\bar{J}_a$ with at most one critical section whenever $\bar{J}_a$ resumes. According to Theorem 1, $\bar{J}_a$ cannot be suspended due to other jobs in local critical sections if $\bar{J}_a$ does not have global critical sections. That is, the number of times that $\bar{J}_a$ may be preempted by $\bar{J}_b$ depends on the number of global resources accessed by $\bar{J}_a$. In addition to the first normal execution segment of $\bar{J}_a$, $\bar{J}_a$ has at most $1 + \bar{n}(i)$ opportunities to affect $\bar{J}_a$.

5.3 Additional Constraints

We now consider an additional constraint arising from Theorem 1.

**Additional Constraint 1** In any schedule of $\tau$ under the FMLP$^\ast$:

$$\sum_{t, e, r} \sum_{i} X_{1,D}^t = 0,$$

where $t' = \tau \setminus \{T_i\}$ is the set of all tasks except $T_i$, $l_{fe} = \{l_i\} P(l_i) = P(T_i)$ denotes the local resources on the processor which $T_i$ is assigned to. Likewise, the following additional constraint is based on Theorem 2.

**Additional Constraint 2** In any schedule of $\tau$ under the FMLP$^\ast$:

$$\sum_{t, e, r} \sum_{i} X_{1,D}^t = 0,$$

where $t' = \tau \setminus \{T_i\}$ is the set of all tasks except $T_i$, $l_{fe} = \{l_i\} P(l_i) = P(T_i)$ denotes the local resources on the processor which $T_i$ is assigned to. Likewise, the following additional constraint is based on Theorem 2.

**Additional Constraint 3** In any schedule of $\tau$ under the FMLP$^\ast$:

$$\sum_{t, e, r} \sum_{i} X_{1,D}^t = 0,$$

where $t' = \tau \setminus \{T_i\}$ is the set of all tasks except $T_i$, $l_{fe} = \{l_i\} P(l_i) = P(T_i)$ denotes the local resources on the processor which $T_i$ is assigned to. Likewise, the following additional constraint is based on Theorem 2.

$\sum_{t, e, r} \sum_{i} X_{1,D}^t = 0,$

where $t' = \tau \setminus \{T_i\}$ is the set of all tasks except $T_i$, $l_{fe} = \{l_i\} P(l_i) = P(T_i)$ denotes the local resources on the processor which $T_i$ is assigned to. Likewise, the following additional constraint is based on Theorem 2.

5.4 Review of the Example

We now return to the numerical example in Table 1. We add the three additional constraints and compute each task’s bound on maximum blocking time. As the last column of Table 2 lists, $b_i$ has been greatly reduced from 29 to 15. The new value obtained by our proposed method is closer to the actual maximum blocking time than by the existing one.

The second row of Table 2 shows that $X_{1,i}^2$ might cause $J_1$ to incur direct request delay as before. Yet $X_{1,i}^3 = X_{1,i}^2 = 1$ is now $X_{1,i}^3 = X_{1,i}^3 = 1$. Although $J_1$ may still be affected by $J_3$
and J5’s critical sections, this is because Additional Constraint 1
works that the rate of occurrence of both kinds of delays is now
more reasonable. On the other hand, \( X_{4,1}^{3,1} = X_{4,1}^{6,3,1} = 1 \) becomes
\( X_{4,1}^{3,1} = X_{4,1}^{6,3,1} = 0 \), leading to less pessimism that J1 cannot be
indirectly delayed by J4 or J6 due to Additional Constraint 2.

We can now estimate \( b_1^{1,1,1} = 5 \), \( b_1^{2,2,1} = 3 \), and \( b_1^{3,1,1} = 7 \) from
Eq. (1). As Fig. 3 depicts, the schedule results in a worst-case
scenario for \( J_1 \). Next, we explain in detail how \( J_1 \) is delayed by
\( J_1 \), \( J_2 \), and then by \( J_5 \).

1. \( b_1^{1,1,1} = 5 \): \( J_1 \)'s critical section begins and its priority is
boosted at \( t_2 \). \( J_1 \)'s effective priority is higher than \( J_1 \)'s until
\( t_5 \), though \( J_1 \) has the highest base priority among all tasks.
Therefore, \( J_1 \) is blocked by \( J_5 \) as soon as it arrives.

2. \( b_1^{2,2,1} = 3 \): From the figure, both \( J_1 \) and \( J_2 \) request \( l_2 \) at \( t_5 \).
Then \( J_1 \) should have acquired the lock due to its higher base
priority. In theory, we have to consider the maximum blocking.
That is, \( J_1 \) requests \( l_2 \) after \( J_2 \) does. We suppose that
\( J_1 \) issues a request for \( l_2 \) at \( t_{8, e} \) instead of \( t_8 \), where \( e > 0 \). As a
result, \( J_2 \) acquires the lock and \( J_1 \) remains suspended until
\( t_{11} \). Thus,

\[
b_1^{2,2,1} = \lim_{\epsilon \to 0} [11 - (8 + \epsilon)] = 3.
\]

3. \( b_1^{3,1,1} = 7 \): In the same way as \( b_1^{3,1,1} \), \( J_1 \) is preempted by the
priority-boosted \( J_3 \) at \( t_{12} \). Also, \( J_3 \) is preempted by \( J_1 \) at \( t_{11} \)
because the lock holders are scheduled in order of increasing
lock-request time.

On the other hand, \( J_2 \) could not cause \( J_1 \) to incur a preemp-
tion delay at \( t_{12} \) if \( J_1 \) did not access the global resource \( l_2 \), or if \( l_2 \)
was a local resource instead of a global one. Without accessing
a global resource, \( J_1 \) could not be delayed by \( J_2 \) on the remote
processor. That is, \( J_2 \) could not be executed immediately as soon
as it released at \( t_{10} \). As is described in Additional Constraint 3,
preemption delay is related to the number of task’s global critical
sections. This example also illustrates that a global critical
section may incur additional penalties.

Based on the new blocking time bounds, we repeated the ex-
eriment in Section 3.5. The new WCRT of each task is listed
in the third row of Table 3. All of the tasks are completed be-
fore their deadlines. Unlike the result presented in Section 4,
the whole task set is now schedulable. By comparison, a task set
might be judged as unschedulable without additional constraints.

6. Experiments

The main purpose of our experiments was to verify whether
results could become less pessimistic if there are local resources
available after task allocation.

6.1 Task Generation

The following parameter settings were used in the experiments:

- \( n_s = 8 \)
- \( N_{\text{max}} \in [1, 3] \)
- \( N_{\text{iq}} \in [0, N_{\text{max}}] \)
- \( L_{\text{iq}}: 2 \) kinds of uniform distributions

- \( m = 8 \)
- \( n \in [m, 10 \cdot m] \)
- \( d_i \in [10 \text{ ms}, 100 \text{ ms}] \)
- \( u_i \in [0, 1, 0.2] \)

In order to generate critical sections, we considered three pa-
rameters: \( N_{\text{iq}} \), \( L_{\text{iq}} \), and the number of resources \( n_s \). \( N_{\text{iq}} \) was
related to the maximum request times \( N_{\text{max}} \) and randomly chosen
from \([0, 1, \ldots, N_{\text{max}}] \). \( T_i \) did not access \( l_i \) if \( N_{\text{iq}} = 0 \). Otherwise
the corresponding \( L_{\text{iq}} \) was uniformly distributed over the range
\([50 \mu s, 100 \mu s] \) (short) and \([100 \mu s, 500 \mu s] \) (long).

For simplicity, we considered only one multiprocessor plat-
form with 8 processors. The number of tasks \( n \) ranged from
\( n = m \) to \( n = 10 \cdot m \). \( d_i \) was chosen from a uniform distribution rang-
ing \([10 \text{ ms}, 100 \text{ ms}] \); \( u_i \) was also uniformly distributed on \([0, 1, 0.2] \).

6.2 Allocation

Our priority was to determine whether our additional con-
straints could eliminate pessimism. The more local resources
available after allocation, the more effective additional constraints
for local resources might be. Therefore, we considered the
synchronization-aware task allocation algorithm in Ref. [8]. First,
tasks which share the same resource were bundled together, and
then the bundles which could not be assigned as a single task to
a processor were split into bundles or tasks that would fit any ex-
isting processor.

6.3 Comparison

We used the same settings of parameters as Ref. [11]. To show
the merits of our proposed method more clearly, we added a new
Fig. 4 Average blocking time bounds under three combinations of parameters.

(b) long critical section, $N_{\text{max}} = 1$

Fig. 5 Maximum percentage decrease in blocking time bounds under three combinations of parameters. We draw purple curves to show the maximum decrease between two methods. Yet red or green ones only represent the blocking time bounds in the case of maximum difference under the improved or existing method, respectively. They are not discussed independently.

(c) long critical section, $N_{\text{max}} = 3$
evidently shows the maximum difference in blocking time bound for each $n$ between two methods. For $n < 13$: the maximum percentage decrease is up to about 90%. The percentage decrease in Fig. 4 (a) also illustrates that the bounds on maximum blocking time are effectively reduced with our strategy.

Next, we consider Fig. 4 (b) for the experiment with a long critical section and $N_{\text{max}} = 1$. Compared with the existing method, our strategy lowers the average blocking time bound by more than 27%. Most significantly, for $n = 9$, the blocking time bound decreases by as much as 65.7%. In Fig. 5 (b), the maximum decrease in blocking time bound can be seen obviously under the circumstance of long critical sections. Figure 6 (b) also shows a clearer advantage for our proposed method than Fig. 6 (a). The schedulability also rises because the bounds on maximum blocking time are less pessimistic. Especially at $n = 9$, the percentage of schedulable tasks improves 1%. Although the request times are unchanged, the additional constraints perform better if the critical sections are longer.

Finally, we consider the case when $N_{\text{max}} = 3$. More request times also increase the total length of the critical section. The result depicted in Fig. 4 (c) shows a larger advantage for the improved method than in Fig. 4 (b): about a 30% decrease in the average blocking time bound and 73% decrease in the maximum blocking time bound at $n = 10$. On the other hand, Fig. 5 (c) illustrates the greatest improvement in the maximum decrease among Fig. 5. Significantly at $n = 8, 9, 11, 12$ or 15, the percentage drops more than 90%. Figure 6 (c) shows a higher schedulability for the improved method when the number of supported tasks is between 11 and 45. For $n = 26$, schedulability increases by nearly 2 percentage points (maximal value). The results of experiments reveal that the additional constraints work better for more request times.

Overall, since the schedulability is relevant to the task’s WCRT and deadline, our proposed method sometimes brings a small improvement in the schedulability if task sets are randomly generated. On the contrary, shorter deadlines may lead to a bigger improvement. Nevertheless, the additional constraints make the blocking time bounds less pessimistic under the different combinations of parameters.

7. Conclusion

Using a numerical example, the existing approach is shown to result in pessimistic values. From a theoretical perspective, we succeed in solving the problem through the introduction of three additional constraints on the local critical section. As long as there exist local resources after task allocation, the additional constraints can function. The results of experiments indicate that it is more efficient to utilize both the original and additional constraints together, especially under the condition that the critical sections are relatively long. They also illustrate that the pessimism of maximum blocking time bound is reduced significantly. Although the schedulability rises slightly, our results still represent an improvement over the existing method.

In our future work, we intend to include further constraints under different protocols besides the distributed locking protocols. We shall also try to apply our strategy to the spin execution control policy as well as the global scheduling algorithm.

References


Zhongqi Ma was a Ph.D. student at the Graduate School of Information Science, Nagoya University. He graduated from East China University of Science and Technology with a bachelor’s degree, and further a master’s degree at Donghua University (China). He won the Japanese Government Scholarship and came to Japan to pursue a doctor’s degree after having worked as a software engineer in Shanghai for a few years. His research interests include real-time operating systems and multi-processor scheduling theory.

Ryo Kurachi is a Designated Associate Professor of Center for Embedded Computing Systems at Nagoya University. He graduated from Tokyo University of Science with undergraduate majors in applied electronics. After a few years working at AISIN AW CO., LTD, as a software engineer, he received his master’s degree in Management of Technology from Tokyo University of Science in 2007, followed by his Ph.D. in information science from the Nagoya University in 2012. His research interests includes embedded systems and real-time systems. Within that domain, he has investigated topics such as in-vehicle networks and real-time scheduling theory and embedded systems security.

Gang Zeng is an associate professor at the Graduate School of Engineering, Nagoya University. He received his Ph.D. degree in Information Science from Chiba University in 2006. From 2006 to 2010, he was a researcher, and then assistant professor at the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. His research interests mainly include power-aware computing, real-time embedded system design. He is a member of IEEE and IPSJ.

Hiroaki Takada is a professor at Institutes of Innovation for Future Society, Nagoya University. He is also a professor and the Executive Director of the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in Information Science from University of Tokyo in 1996. He was a Research Associate at University of Tokyo from 1989 to 1997, and was a Lecturer and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IPSJ, IEICE, JSSST, and JSAE.