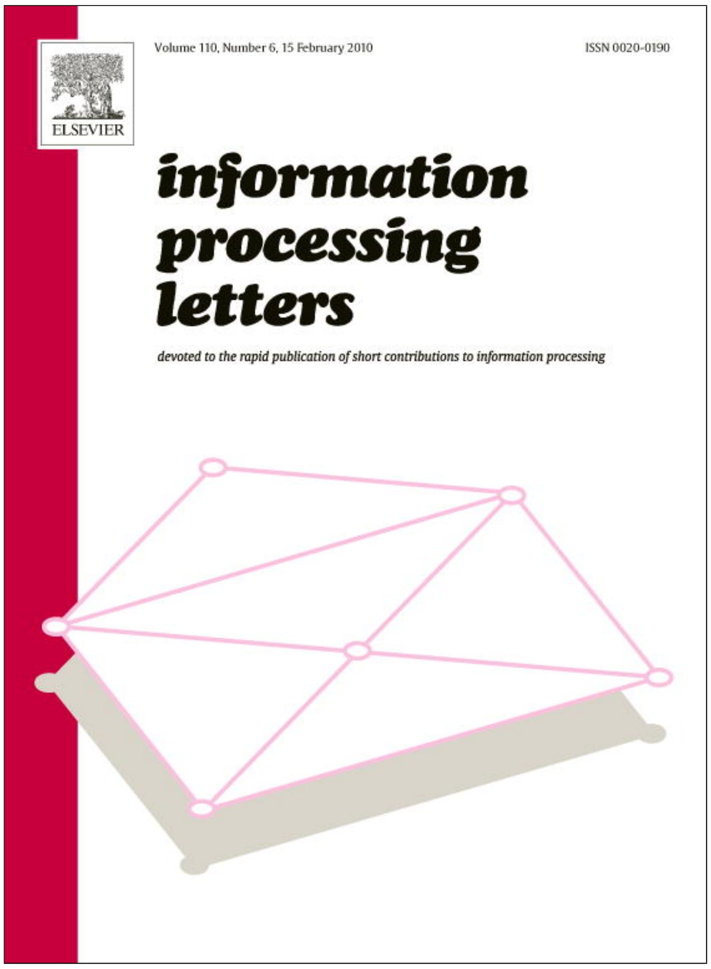
Volume 110, Number 6, 15 February 2010
ISSN 0020-0190
ELSEVIER
information processing letters
devoted to the rapid publication of short contributions to information processing

Information Processing Letters 110 (2010) 226–231

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Rate monotonic scheduling re-analysed

Qiwen Xu [a,*], Naijun Zhan [b]

[a] Faculty of Science and Technology, University of Macau, Macau, PR China
[b] Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences & State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing, PR China

ARTICLE INFO

Article history:
Received 10 March 2009
Received in revised form 22 December 2009
Accepted 30 December 2009
Available online 4 January 2010
Communicated by A.A. Bertossi

Keywords:
Real time systems
Scheduling
Schedulability conditions

ABSTRACT

In this paper, we re-analyse the rate monotonic scheduler. Traditionally, the schedulability condition was obtained from the greatest lower bound of utilisation factors over all the task sets that (are schedulable and) fully utilise the processor. We argue that full utilisation is not very appropriate for this purpose. We re-establish Liu and Layland's classic schedulability theorem by finding the greatest lower bound of utilisation factors over all the unschedulable task sets instead. The merits of our approach include: Firstly, the fact that the bound is both sound and tight for schedulability follows directly from definition; Secondly, our proof is simpler technically.

1. Introduction

In their seminal paper [6], Liu and Layland studied two scheduling algorithms, one of them is the rate monotonic scheduler (RMS). For the RMS, they established a schedulability condition based on the utilisation factors represented by the following classic theorem

Theorem 1. *(See Liu and Layland [6].) Given a set of m tasks, if its utilisation factor is less than or equal to $m(2^{\frac{1}{m}} - 1)$, then the task set can be scheduled by the RMS.*

To prove this, they introduced the notion of full utilisation. A set of tasks is said to fully utilise the processor if the task set is schedulable and any increase of the required execution time for any task will cause the task set to be unschedulable.

> **Liu and Layland's Claim**: A set of m tasks is schedulable if its utilisation factor is less than or equal to the minimum of the utilisation factors over all sets of m tasks that fully utilise the processor.

Since then, full utilisation has been used as a standard method by various researchers (see e.g. the recent book [1] and paper [2]) for proving the theorem. We shall argue that it is not very appropriate for this purpose.

First, Liu and Layland just stated the claim without proof or any further discussion. However, its soundness is not trivial. As far as we know, it was only proved recently by Devillers and Goossens [2] and by us in a slightly different setting [3]. Both proofs made use of the fact that the bound expression $m(2^{\frac{1}{m}} - 1)$ monotonically decreases as m increases (m is the number of tasks to be scheduled). Moreover, Liu and Layland did not discuss whether the bound is tight, that is, whether the bound can be lower, although this is implied by the constructive proof in their paper.

A set of m tasks is schedulable if its utilisation factor is less than the utilisation factors of any unschedulable set of m tasks (otherwise, the utilisation factor of the task set has to be less than itself). Since there are infinitely many sets of unschedulable tasks, there may not be a minimum value of the utilisation factors of them. Therefore, we shall use the term greatest lower bound instead of minimum. It

[*] Corresponding author at: Av. Padre Tomás Pereira, Taipa, Macau, PR China.
E-mail addresses: qwxu@umac.mo (Q. Xu), znj@ios.ac.cn (N. Zhan).

0020-0190/$ – see front matter © 2009 Published by Elsevier B.V.
doi:10.1016/j.ipl.2009.12.010

simply follows from the definition that a set of m tasks is schedulable if its utilisation factor is less than the greatest lower bound of utilisation factors over all the unschedulable sets of m tasks. Moreover, for any value greater than that greatest lower bound, it is also trivial to prove that there is an unschedulable set of m tasks whose utilisation factor is less than that value.

Due to these observations, the greatest lower bound of utilisation factors over all the unschedulable task sets immediately provides a sound and tight bound for schedulability test. Moreover, the calculation of the bound for the RMS in this way is simpler than using full utilisation, and this is perhaps expected since the notion of full utilisation is more complicated.

This paper is organized as follows. In Section 2, we show that the greatest lower bound of utilisation factors of all the unschedulable sets of tasks provides the bound for schedulability test. A formal characterisation for the static scheduler to be schedulable (hence also unschedulable) is developed in Section 3. In Section 4, Liu and Layland's theorem for the RMS is proven. The paper ends with a discussion in Section 5.

2. Utilisation factor and schedulability condition

We first define the greatest lower bound and establish a property that is needed later.

Definition 1. Given a set S of elements, its greatest lower bound $\mathsf{glb}(S)$ is defined to be the element satisfying the following two properties

- $\mathsf{glb}(S) \leqslant s$ for any $s \in S$.
- If $r \leqslant s$ for any $s \in S$, then $r \leqslant \mathsf{glb}(S)$.

Note $\mathsf{glb}(S)$ does not necessarily belong to S.

Lemma 1. *Given two sets R and S, if for any $s \in S$, there exists $r \in R$ such that $r \leqslant s$, then $\mathsf{glb}(R) \leqslant \mathsf{glb}(S)$.*

Proof. For any $s \in S$, there exists $r \in R$ such that $r \leqslant s$. It follows that $\mathsf{glb}(R) \leqslant r \leqslant s$ and therefore $\mathsf{glb}(R) \leqslant \mathsf{glb}(S)$. □

In this paper, we consider the same basic scheduling problem as in Liu and Layland's paper [6]. Consider a set of m tasks $\Gamma = \{\tau_1, \ldots, \tau_m\}$, each independent of one another. Task τ_i has a period T_i and requires C_i execution time in every period. We assume T_i and C_i are rational numbers, $T_i > 0$ and $C_i \geqslant 0$ (for notational convenience we allow $C_i = 0$ although this case is clearly not useful in practice). The deadline is equal to the period. All tasks begin their first requests at time 0. Such a task set Γ will be denoted by $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$. There is one processor on which the tasks are executed and tasks can be preempted. The time needed for switching tasks is omitted. As an example, let us consider two tasks τ_1 and τ_2. Task τ_1 needs one unit time and τ_2 needs two unit time. If starting from time 0, τ_1 runs first and τ_2 second, it is understood that execution will complete by time point 3. At time point 1, it is immaterial whether τ_1 is executing or τ_2 is executing. Therefore, when we refer to the execution of a task from time point a to time point b, we shall use open interval (a, b).

For a set of tasks $\Gamma = \{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$, the utilisation factor is defined to be $u(\Gamma) = \sum_{i=1}^{m} \frac{C_i}{T_i}$. Given a set of task sets S, we shall use $u(S)$ to denote the set of all the utilisation factors of the task sets in S, namely,

$$u(S) = \{u(\Gamma) \mid \Gamma \in S\}.$$

A set of tasks is schedulable under a scheduling policy if every instance of every task has enough execution time before the corresponding deadline, when the tasks are scheduled according to the policy. In other words, a set of tasks is unschedulable if there is one instance of one task that misses the deadline. In the following, we shall assume our discussion is with respect to a certain given scheduling policy, that is, if we say a set of tasks is schedulable or unschedulable, it is understood that this is with respect to the given particular policy.

Let

$$S_1 = \{\Gamma \mid \Gamma \text{ is a set of } m \text{ tasks that is unschedulable}\}.$$

Definition 2. $v(m) = \mathsf{glb}(u(S_1))$.

The following theorem is a variation of Liu and Layland's claim.

Theorem 2. *A set of m tasks Γ is schedulable if $u(\Gamma) < v(m)$.*

Proof. Given a set of m tasks Γ, if it is not schedulable, namely $\Gamma \in S_1$, then by the definition of $v(m)$, $v(m) \leqslant u(\Gamma)$. This is the contraposition of the theorem. □

Theorem 3. *For any $x > v(m)$, there exists an unschedulable set of m tasks Γ such that $u(\Gamma) < x$.*

Proof. If for any set of m unschedulable tasks Γ, $x \leqslant u(\Gamma)$, then by the definition of $v(m)$, $x \leqslant v(m)$. This is the contraposition of the theorem. □

The above two theorems do not cover the case that the utilisation factor happens to be equal to $v(m)$. For the RMS, which is our major concern, given our assumption that C_i and T_i are rational numbers, there is obviously no utilisation factor that can be equal to $m(2^{\frac{1}{m}} - 1)$, except when $m = 1$ (one task) and in this case, the task (set) is trivially schedulable. For a scheduling algorithm in general, we expect that $v(m)$ is not an element of $u(S_1)$ (in other words, $u(S_1)$ has no minimum element), which means that a set of tasks whose utilisation factor is equal to $v(m)$ is schedulable. Suppose this is not true and assume an unschedulable task set Γ whose utilisation factor is $v(m)$. We can reduce the execution time of one task of Γ by a very small amount, and the new task set is still unschedulable. The utilisation factor of the new task set is less than $v(m)$, hence a contradiction. Note we do not consider this argument as a proof, because it lacks the rigor that is usually required in mathematics.

3. Static scheduling

For static scheduling, the priorities of tasks are fixed. Without loss of generality, we hereafter assume priorities are in decreasing order, namely, for a set of tasks $\tau_1, \tau_2, \ldots, \tau_m$, the priority order is $\tau_1 > \tau_2 > \cdots > \tau_m$. In this section, we discuss static scheduling and shall not state this all the time.

A set of tasks is schedulable if the timing requirement of each instance of every task is satisfied before the corresponding deadline. Let us take one task, say τ_k, to consider. We first study the condition for its first request to be satisfied. Suppose the timing requirement is satisfied at time t ($t \leqslant T_k$). Over the interval $(0, t)$, the processor is occupied by $\tau_1, \tau_2, \ldots, \tau_k$, i.e., one of them is running and no tasks $\tau_{k+1}, \tau_{k+2}, \ldots, \tau_m$ can be executed, since τ_k's request is outstanding until t. It follows that $\sum_{i=1}^{k} \lceil \frac{t}{T_i} \rceil C_i \leqslant t$ (where $\lceil \frac{t}{T_i} \rceil$ is the smallest integer greater than or equal to $\frac{t}{T_i}$), since tasks with higher priorities need $\sum_{i=1}^{k-1} \lceil \frac{t}{T_i} \rceil C_i$ executing time. On the other hand, if there exists $0 < t \leqslant T_k$, such that $\sum_{i=1}^{k} \lceil \frac{t}{T_i} \rceil C_i \leqslant t$, then obviously there is enough time for task τ_k, along with the tasks with higher priorities, to complete the execution time. We next investigate a condition for an arbitrary request of a task.

Lemma 2. *The j-th request of task τ_k is satisfied, if for any $0 \leqslant b \leqslant (j-1)T_k$, there exists t such that $(j-1)T_k < t \leqslant jT_k$, $\sum_{i=1}^{k} \lceil \frac{t-b}{T_i} \rceil C_i \leqslant (t-b)$.*

Proof. We search backwards from $(j-1)T_k$ for the first time point b such that the processor is not used by a task from τ_1 to τ_k (either no task is running or a task from τ_{k+1} to τ_m is running) over interval $(b-\epsilon, b)$ where ϵ is arbitrarily small. If such a point cannot be found, then let $b = 0$. It follows that the processor is totally occupied by $\tau_1, \tau_2, \ldots, \tau_k$ over $(b, (j-1)T_k)$ and there are no outstanding requests of $\tau_1, \tau_2, \ldots, \tau_k$ that start before b. The total amount of requested execution time from $\tau_1, \tau_2, \ldots, \tau_k$ during (b, t) is at most $\sum_{i=1}^{k} \lceil \frac{t-b}{T_i} \rceil C_i$. The processor will continue to be occupied by $\tau_1, \tau_2, \ldots, \tau_k$ from $(j-1)T_k$ until their requests are satisfied. Since $(j-1)T_k < t \leqslant jT_k$ and $\sum_{i=1}^{k} \lceil \frac{t-b}{T_i} \rceil C_i \leqslant (t-b)$, there is enough time for τ_k to complete execution by t. $\square$

Liu and Layland [6] discovered a property, which became well known afterwards, about the static scheduling: the critical instant (the arriving time of a task that needs the maximal time to complete) of a task occurs when the task is requested simultaneously with requests of all higher priority tasks. Apart from its application in the schedulability theorem, it also provides the basis for the response time method studied by many researchers, for example, Joseph and Pandya [5]. However, the proof was informal in the form of illustrative diagrams and found to have flaws by Goossens [4].

In [3], we formally proved a similar result: under the static scheduler, if the timing requirement of a task's first instance is satisfied, then all instances of this task will be satisfied. The proof in [3] was given in Duration Calculus [7], an interval temporal logic. For easy reference, we reformulate the proof using traditional mathematics and now the proof follows immediately from Lemma 2.

Theorem 4. *For any task, the timing requirements of all its instances are satisfied under the static scheduler if and only if its first instance is satisfied.*

Proof. The only if part is by the definition. We next show the if part. Consider any task τ_k. Its first instance is satisfied, and therefore there exists time $0 < t \leqslant T_k$ such that

$$\sum_{i=1}^{k} \left\lceil \frac{t}{T_i} \right\rceil C_i \leqslant t. \tag{1}$$

For any instance $j > 0$, it follows from $0 < t \leqslant T_k$ that for any $b \leqslant (j-1)T_k$ there exists $n > 0$ such that $(j-1)T_k < b + nt \leqslant jT_k$. Thus,

$$\begin{aligned}
\sum_{i=1}^{k} \left\lceil \frac{(b+nt)-b}{T_i} \right\rceil C_i &\leqslant \sum_{i=1}^{k} n \left\lceil \frac{t}{T_i} \right\rceil C_i \\
&\quad \left(\text{by} \left\lceil \frac{nt}{T_i} \right\rceil \leqslant n \left\lceil \frac{t}{T_i} \right\rceil \right) \\
&= n \sum_{i=1}^{k} \left\lceil \frac{t}{T_i} \right\rceil C_i \\
&\leqslant nt.
\end{aligned}$$

Therefore, by Lemma 2, the j-th instance of τ_k is satisfied. $\square$

This indicates that a set of tasks $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$ is schedulable if and only if for any $1 \leqslant k \leqslant m$ there exists $0 < t \leqslant T_k$ such that (1) holds. In other words, a set of tasks is unschedulable if and only if there exists $1 \leqslant k \leqslant m$ such that for any $0 < t \leqslant T_k$, $\sum_{i=1}^{k} \lceil \frac{t}{T_i} \rceil C_i > t$.

4. Rate monotonic scheduler

The RMS is a static scheduler where tasks with shorter periods have higher priorities. Recall we have the convention that priorities are in decreasing order, so for a set of tasks $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$ this implies $T_1 \leqslant T_2 \leqslant \cdots \leqslant T_m$. In this section, we discuss the RMS and shall not mention this any more.

We now come to establish the greatest lower bound of utilisation factors of all unschedulable task sets under the RMS.

Theorem 5. *For the RMS, $v(m) = m(2^{1/m} - 1)$.*

Although our starting point, and subsequently some proofs, are different from that of Liu and Layland [6], we use many ideas from their proofs. In particular, Liu and Layland suggested first consider the case that one period is at least half of any other period. We can express this formally by a predicate $H(T_1, \ldots, T_m) = \forall 1 \leqslant i, j \leqslant m.\ 0 <$

$T_i/T_j < 2$. In the definitions below, assume $\Gamma = \{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$. Let

$$S_2 = \bigl\{\Gamma \bigm| \mathit{Unschedulable}(\Gamma) \wedge H(T_1, \dots, T_m)\bigr\}$$

and we shall prove that the bound over it has the same value as over the following set of tasks

$$S_3 = \left\{\Gamma \,\middle|\, \left(\bigwedge_{i=1}^{m-1} C_i = T_{i+1} - T_i\right) \wedge \right. \\ \left. C_m = 2T_1 - T_m \wedge H(T_1, \dots, T_m)\right\}.$$

To fit in our framework, we introduce another set of tasks

$$S_4 = \left\{\Gamma \,\middle|\, \left(\bigwedge_{i=1}^{m-1} C_i = T_{i+1} - T_i\right) \wedge \right. \\ \left. C_m > 2T_1 - T_m \wedge H(T_1, \dots, T_m)\right\}$$

and we shall prove

$$\operatorname{glb}\bigl(u(S_1)\bigr) = \operatorname{glb}\bigl(u(S_2)\bigr) = \operatorname{glb}\bigl(u(S_4)\bigr) \\ = \operatorname{glb}\bigl(u(S_3)\bigr) = m\bigl(2^{1/m} - 1\bigr).$$

Lemma 3. $\operatorname{glb}(u(S_2)) \leqslant \operatorname{glb}(u(S_4))$.

Proof. We prove this by showing $S_4 \subseteq S_2$, namely, any task set $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\} \in S_4$ is unschedulable. We shall prove that the m-th task will miss its deadline. Consider any $0 < t \leqslant T_m$. For notational convenience, let $T_0 = 0$, then there exists $0 \leqslant k < m$, $T_k < t \leqslant T_{k+1}$. It follows that

$$\begin{aligned}\sum_{i=1}^{m} \left\lceil \frac{t}{T_i} \right\rceil C_i &= 2C_1 + \dots + 2C_k + C_{k+1} + \dots + C_{m-1} + C_m \\ &= 2(T_2 - T_1) + \dots + 2(T_{k+1} - T_k) \\ &\quad + T_{k+2} - T_{k+1} + \dots + T_m - T_{m-1} + C_m \\ &= T_{k+1} + T_m - 2T_1 + C_m \\ &> T_{k+1} \\ &\geqslant t.\end{aligned}$$

This completes the proof of the lemma. □

Lemma 4. $\operatorname{glb}(u(S_2)) \geqslant \operatorname{glb}(u(S_4))$.

Proof. For any task set $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\} \in S_2$, we shall find another set of task $\{(C'_i, T'_i) \mid 1 \leqslant i \leqslant m\} \in S_4$, $\sum_{i=1}^{m} \frac{C'_i}{T'_i} \leqslant \sum_{i=1}^{m} \frac{C_i}{T_i}$. Suppose the k-th task misses its deadline. For $t = T_1, \dots, T_k$, we have $\sum_{i=1}^{k} \lceil t/T_i \rceil C_i > t$. Since $T_k > T_i$ and $T_k < 2T_i$ for all $1 \leqslant i < k$, it follows immediately

$$\begin{aligned}&C_1 + C_2 + \dots + C_{k-1} + C_k > T_1 \\ &2C_1 + C_2 + \dots + C_{k-1} + C_k > T_2 \\ &\quad\vdots \\ &2C_1 + 2C_2 + \dots + 2C_{k-1} + C_k > T_k\end{aligned}$$

if $T_1, \dots, T_k$ are mutually distinct. If the periods are not mutually distinct, these still hold. For example, if $T_i = T_{i+1}$ are the first two equal periods, then we have

$$2C_1 + \dots + 2C_{i-1} + C_i + \dots + C_k > T_i$$

therefore

$$2C_1 + \dots + 2C_i + C_{i+1} \dots + C_k > T_i = T_{i+1}.$$

It follows that there exist $\alpha_i > 1, i = 1 \dots k$ such that

$$\begin{aligned}&C_1 + C_2 + \dots + C_{k-1} + C_k = \alpha_1 T_1 \\ &2C_1 + C_2 + \dots + C_{k-1} + C_k = \alpha_2 T_2 \\ &\quad\vdots \\ &2C_1 + 2C_2 + \dots + 2C_{k-1} + C_k = \alpha_k T_k.\end{aligned}$$

Thus,

$$\begin{aligned}&C_1 = \alpha_2 T_2 - \alpha_1 T_1 \\ &\quad\vdots \\ &C_{k-1} = \alpha_k T_k - \alpha_{k-1} T_{k-1} \\ &C_k = 2\alpha_1 T_1 - \alpha_k T_k\end{aligned}$$

Hence,

$$\begin{aligned}&\sum_{i=1}^{k} \frac{C_i}{T_i} - \left(\left(\sum_{i=1}^{k-1} \frac{T_{i+1} - T_i}{T_i}\right) + \frac{2T_1 - T_k}{T_k}\right) \\ &\quad = \sum_{i=1}^{k-1} \left[\frac{(\alpha_{i+1} - 1)T_{i+1}}{T_i} - (\alpha_i - 1)\right] \\ &\qquad + \frac{(\alpha_1 - 1)2T_1}{T_k} - (\alpha_k - 1) \\ &\quad = \sum_{i=1}^{k-1} \left[\frac{(\alpha_{i+1} - 1)T_{i+1}}{T_i} - (\alpha_{i+1} - 1)\right] \\ &\qquad + \frac{(\alpha_1 - 1)2T_1}{T_k} - (\alpha_1 - 1) \\ &\quad = \delta \quad \text{(denote by } \delta \text{ for reference in the later)} \\ &\quad > 0 \quad (T_i \leqslant T_{i+1}, 2T_1 > T_k).\end{aligned}$$

We can choose the following task set

$$\begin{aligned}&C'_1 = 0 \\ &T'_1 = T_1 \\ &\quad\vdots \\ &C'_{m-k} = 0\end{aligned}$$

$$T'_{m-k} = T_1$$
$$C'_{m-k+1} = T_2 - T_1$$
$$T'_{m-k+1} = T_1$$
$$C'_{m-k+2} = T_3 - T_2$$
$$T'_{m-k+2} = T_2$$
$$\vdots$$
$$C'_{m-1} = T_k - T_{k-1}$$
$$T'_{m-1} = T_{k-1}$$
$$C'_m = 2T_1 - T_k + \varepsilon$$
$$T'_m = T_k,$$

where $0 < \varepsilon/T_k < \delta$.

Obviously, the task set is an element of S_4. Moreover, $\sum_{i=1}^m \frac{C'_i}{T'_i} \leqslant \sum_{i=1}^k \frac{C_i}{T_i} \leqslant \sum_{i=1}^m \frac{C_i}{T_i}$. □

Lemma 5. $\mathrm{glb}(u(S_1)) = \mathrm{glb}(u(S_2))$.

Proof. Since $S_1 \supseteq S_2$, $\mathrm{glb}(u(S_1)) \leqslant \mathrm{glb}(u(S_2))$ follows immediately.

Therefore, we only need to prove $\mathrm{glb}(u(S_1)) \geqslant \mathrm{glb}(u(S_2))$. For any task set $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\} \in S_1$, we shall find another set of tasks $\{(C'_i, T'_i) \mid 1 \leqslant i \leqslant m\} \in S_2$, $\sum_{i=1}^m \frac{C'_i}{T'_i} \leqslant \sum_{i=1}^m \frac{C_i}{T_i}$. Assume the k-th task misses it deadline. Therefore

$$\forall 0 < t \leqslant T_k. \sum_{i=1}^k \left\lceil \frac{t}{T_i} \right\rceil C_i > t. \tag{2}$$

Let $Q_i = \lfloor \frac{T_k}{T_i} \rfloor$ and $T'_i = Q_i T_i$ for $1 \leqslant i \leqslant k$. It is easy to prove the following:

I. $Q_i \geqslant 1$ for any $1 \leqslant i \leqslant k$;
II. $T'_i \leqslant T_k$ for any $1 \leqslant i < k$;
III. $T'_k = T_k$;
IV. $\lceil \frac{T_k}{T_i} \rceil \leqslant Q_i + 1$ for any $1 \leqslant i \leqslant k$;
V. $0 < \frac{T'_i}{T'_j} < 2$ for any $1 \leqslant i, j \leqslant k$.

Let $C'_i = C_i$ for $i = 1, \ldots, k-1$ and $C'_k = C_k + \sum_{i=1}^{k-1}(Q_i - 1)C_i$. Let us sort $T'_1, T'_2, \ldots, T'_k$ in increasing order, or in other words, decreasing order of priorities. Suppose $U_1, U_2, \ldots, U_{k-1}$ is a permutation of $\{1, 2, \ldots, k-1\}$ and $T'_{U_1} \leqslant T'_{U_2} \leqslant \cdots \leqslant T'_{U_{k-1}}$. Since $T_k = T'_k$, we can let $U_k = k$. For convenience, let $T'_{U_0} = 0$. We next show execution time requirement of task (C'_{U_k}, T'_{U_k}) cannot be satisfied. For any $0 < t \leqslant T'_{U_k}$, there exists $0 \leqslant j < k$ such that $T'_{U_j} < t \leqslant T'_{U_{j+1}}$. It follows

$$Q_{U_j} T_{U_j} < t \leqslant Q_{U_{j+1}} T_{U_{j+1}} \leqslant Q_{U_i} T_{U_i}, \quad \text{for } i = j+1, \ldots, k-1. \tag{3}$$

Moreover, we have

$$\begin{aligned}
\sum_{i=1}^k \left\lceil \frac{t}{T'_{U_i}} \right\rceil C'_{U_i} &= \sum_{i=1}^k \left\lceil \frac{T'_{U_{j+1}}}{T'_{U_i}} \right\rceil C'_{U_i} \quad \text{(by V)} \\
&= \sum_{i=1}^j 2C'_{U_i} + \sum_{i=j+1}^k C'_{U_i} \quad \text{(by V and } T'_{U_1} \leqslant \cdots \leqslant T'_{U_j} < T'_{U_{j+1}} \leqslant \cdots \leqslant T'_{U_k}) \\
&= \sum_{i=1}^j (Q_{U_i} + 1) C_{U_i} + \sum_{i=j+1}^{k-1} Q_{U_i} C_{U_i} + C_k \quad \text{(by the definition)} \\
&\geqslant \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_{U_i}} \right\rceil C_{U_i} + C_k \quad \text{(by IV and (3))} \\
&= \sum_{i=1}^k \left\lceil \frac{t}{T_i} \right\rceil C_i \\
&> t \quad \text{(by (2))}.
\end{aligned}$$

Therefore $\{(C'_i, T'_i) \mid 1 \leqslant i \leqslant k\}$ is unschedulable. We can add $m-k$ tasks with $C'_{k+1} = C'_{k+2} = \cdots = C'_m = 0$ and $T'_{k+1} = T'_{k+2} = \cdots = T'_m = T'_k$. The task set $\{(C'_i, T'_i) \mid 1 \leqslant i \leqslant m\}$ is still unschedulable and together with $0 < T'_i/T'_j < 2$ for any $1 \leqslant i, j \leqslant k$, we now know $\{(C'_i, T'_i) \mid 1 \leqslant i \leqslant m\} \in S_2$. On the other hand, it is easy to prove

$$\begin{aligned}
\sum_{i=1}^m \frac{C'_i}{T'_i} &= \sum_{i=1}^k \frac{C'_i}{T'_i} \\
&= \sum_{i=1}^k \frac{C_i}{T_i} + \sum_{i=1}^{k-1} C_i (Q_i - 1)\left(\frac{1}{T_k} - \frac{1}{T'_i}\right) \\
&\leqslant \sum_{i=1}^k \frac{C_i}{T_i} \\
&\leqslant \sum_{i=1}^m \frac{C_i}{T_i}. \quad \square
\end{aligned}$$

Lemma 6. $\mathrm{glb}(u(S_3)) = \mathrm{glb}(u(S_4))$

Proof. It is obvious that $\mathrm{glb}(u(S_3)) \leqslant \mathrm{glb}(u(S_4))$. We next prove $\mathrm{glb}(u(S_3)) \geqslant \mathrm{glb}(u(S_4))$ by contradiction. Suppose $\mathrm{glb}(u(S_3)) < \mathrm{glb}(u(S_4))$, then there exists a task set $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\} \in S_3$ such that $\sum_{i=1}^m \frac{C_i}{T_i} < \mathrm{glb}(u(S_4))$. We can increase C_m by a small amount $\epsilon > 0$, such that the utilisation factor of the new task set is still less than $\mathrm{glb}(u(S_4))$. However, the new task set belongs to S_4, hence a contradiction. □

Now we are ready to calculate the greatest lower bound. The calculation in Liu and Layland's paper [6] is complicated and missing steps. In the following, a straightforward proof using only elementary mathematics is given.

Lemma 7. $\mathrm{glb}(u(S_3)) = m(2^{1/m} - 1)$.

Proof. For $i = 1 \dots m-1$, $\frac{C_i}{T_i} = \frac{T_{i+1}}{T_i} - 1$, and $\frac{C_m}{T_m} = \frac{2T_1}{T_m} - 1$. It follows that

$$\sum_{i=1}^{m} \frac{C_i}{T_i} = \left(\sum_{i=1}^{m-1} \frac{T_{i+1}}{T_i}\right) + \frac{2T_1}{T_m} - m.$$

The following is a simple property for positive reals, that is the arithmetic mean of n positive reals is greater than or equal to their geometric mean,

$$\frac{x_1 + x_2 + \cdots + x_m}{m} \geqslant (x_1 x_2 \cdots x_m)^{\frac{1}{m}}$$

and the equality holds when $x_1 = x_2 = \cdots = x_m$. Therefore, the greatest lower bound of

$$\frac{(\sum_{i=1}^{m-1} \frac{T_{i+1}}{T_i}) + \frac{2T_1}{T_m}}{m}$$

is equal to

$$\left(\left(\frac{T_2}{T_1}\right)\left(\frac{T_3}{T_2}\right)\cdots\left(\frac{T_m}{T_{m-1}}\right)\left(\frac{2T_1}{T_m}\right)\right)^{\frac{1}{m}} = 2^{\frac{1}{m}}.$$

Hence, the greatest lower bound of $\sum_{i=1}^{m} \frac{C_i}{T_i}$ is $m(2^{\frac{1}{m}} - 1)$, when C_is and T_is are non-negative reals. It is easy to prove that the greatest lower bound of $\sum_{i=1}^{m} \frac{C_i}{T_i}$ is still $m(2^{\frac{1}{m}} - 1)$ when C_is and T_is are non-negative rationals. □

Theorem 5 follows from Lemmas 2 to 7.

5. Discussions

It is actually not by chance that the bound calculated from the sets of unschedulable tasks is the same as that from using full utilisation. Given the formal characterisation developed early for a set of tasks to be unschedulable under the static scheduler, it is easy to see that a set of tasks $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$ that is at or above the border line (that is, any increase of the execution time for any task will cause the new task set to be unschedulable and these also include task sets that are already unschedulable) is characterised by the following condition:

$$\text{for any } 0 < t \leqslant T_m, \quad \sum_{i=1}^{m} \left\lceil \frac{t}{T_i} \right\rceil C_i \geqslant t. \tag{4}$$

Therefore formally, a set of tasks $\{(C_i, T_i) \mid 1 \leqslant i \leqslant m\}$ fully utilises the processor if

1) the task set is schedulable, characterised by condition (1) and
2) condition (4).

Let

$$S_0 = \{\Gamma \mid \Gamma \text{ is a set of } m \text{ tasks that fully utilises the processor}\}.$$
$$S_5 = \{\Gamma \mid \Gamma \text{ is a set of } m \text{ tasks that satisfies condition (4)}\}.$$
$$S_6 = \{\Gamma \mid \Gamma \text{ is a set of } m \text{ tasks that is schedulable}\}.$$

Obviously, $S_0 = S_5 \cap S_6$. Let $l(m) = \mathsf{glb}(u(S_0))$ and $w(m) = \mathsf{glb}(u(S_5))$. We can prove

$$l(m) = w(m) = v(m).$$

This shows Liu and Layland's results are indeed correct. However, as we observed, conceptually the bound over the set of unschedulable tasks is more appropriate and also technically simpler.

Another contribution of this paper is that our proofs are rigorous by common standard in mathematics. In particular, we have developed a formal characterisation for a set of tasks to be schedulable, hence also unschedulable, under the static scheduler, and along the way, a formal characterisation for full utilisation. When we reason whether a set of tasks is schedulable or unschedulable, we check against the formal characterisation.

Acknowledgements

We thank Dong Shuzhen for her work in some proofs, Sun Haiwei for suggesting the proof for Lemma 7 using elementary mathematics, and the anonymous referee for comments that have led to improvement.

The second author is supported in part by the projects NSFC-60721061, NSFC-60573007, NSFC-90718041, NSFC-60736017, NSFC-60970031, and RCS2008K001.

References

[1] G.B. Buttazzo, Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers, 1997.
[2] R. Devillers, J. Goossens, Liu and Layland's schedulability test revisited, Information Processing Letters 73 (2000) 157–161.
[3] S. Dong, Q. Xu, N. Zhan, A formal proof for the rate monotonic scheduler, in: Proc. the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99), IEEE Computer Society Press, Hong Kong, 1999, pp. 500–507.
[4] J. Goossens, Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints, PhD thesis, Université Libre de Bruxelles, 1999.
[5] M. Joseph, P. Pandya, Finding response times in a real-time system, The Computer Journal 29 (5) (1986) 390–395.
[6] C.L. Liu, J.W. Layland, Scheduling algorithm for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 46–61.
[7] C.C. Zhou, C.A.R. Hoare, A.P. Ravn, A calculus of durations, Information Processing Letters 40 (5) (1991) 269–276.