# Predictability of Least Laxity First Scheduling Algorithm on Multiprocessor Real-Time Systems*

Sangchul Han and Minkyu Park**

School of Computer Science and Engineering, Seoul National University, Seoul, Korea
{schan, mkpark}@ssrnet.snu.ac.kr

**Abstract.** A priority-driven scheduling algorithm is said to be *start time (finish time) predictable* if the start time (finish time) of jobs in the schedule where each job executes for its actual execution time is bounded by the start times (finish times) of jobs in the schedules where each job executes for its maximum/minimum execution time. In this paper, we study the predictability of a job-level dynamic priority algorithm, LLF (Least Laxity First), on multiprocessor real-time systems. We present a necessary and sufficient condition for a priority-driven algorithm to be start time (finish time) predictable. Then, in LLF scheduling, we show that both the start time and the finish time are predictable if the actual execution times cannot be known. However, solely the finish time is predictable if the actual execution times can be known.

## 1 Introduction

As the workload of embedded systems becomes heavier, multiprocessor architecture becomes more common to many embedded systems. For example, Multiprocessor cores (MPcore) and Multiprocessor System-On-Chips (MPSoCs) have been adopted in various embedded systems such as cellular phones, digital video entertainment systems, mobile multimedia applications [1,2]. Recently as embedded systems handle complex and dynamic applications, many researchers study priority-driven real-time scheduling on multiprocessor embedded systems [3,4].

Priority-driven algorithms assign priorities to jobs and execute the highest-priority jobs on the available processors. Most well-known scheduling algorithms such as FIFO (First-In-First-Out), LIFO (Last-In-Last-Out), SJF (Shortest-Job-First) are priority-driven [5]. Priority-driven algorithms are classified into three categories: static priority, job-level fixed priority (task-level dynamic priority) and job-level dynamic priority algorithms. Static priority algorithms assign a unique priority to each task, and each job of a task has the same priority associated with that task. Examples of such algorithms are RM (Rate Monotonic) and

* This work was supported by the Korea Research Foundation Grant funded by the Korean Government(MOEHRD)(KRF-2005-041-D00636).

** Corresponding author.

X. Zhou et al. (Eds.): EUC Workshops 2006, LNCS 4097, pp. 755–764, 2006.

DM (Deadline Monotonic) [6,7]. In job-level fixed priority algorithms, each job is given a unique priority and the priority remains fixed until the job completes its execution. EDF (Earliest Deadline First), EDF-US$[m/(2m-1)]$, and fpEDF are in this class [7,8,9]. In job-level dynamic priority algorithms, the priority of a job may change at any time. A well known job-level dynamic priority algorithm is LLF (Least Laxity First) [10,11].

The execution requirement of a hard real-time job is usually specified as the maximum execution time or an upper bound on the actual execution time. It is assumed that the maximum execution time is known *a priori*. But the actual execution time is known only when the job arrives or is not known until the job finishes. In this context, Ha and Liu [12] defined predictability as follows. The execution of a job is *start time predictable* if the start time of the job in the actual schedule can be bounded by the start times of the job in the maximum/minimum schedule (a schedule where each job executes for its maximum/minimum execution time). Similarly, the execution of a job is *finish time predictable* if the finish time of the job in the actual schedule can be bounded by the finish times in the maximum/minimum schedule. The execution of a job is *predictable* if it is both start time predictable and finish time predictable.

In most real systems only predictable scheduling algorithms are useful. In general, the feasibility of a given set of jobs is determined on the basis of jobs' maximum execution time during the system design stage. At runtime the actual execution time may be less than the maximum execution time, and thus the actual schedule may be different from the maximum schedule. If a scheduling algorithm is not finish time predictable, the finish time in the actual schedule might be later than the finish time in the maximum schedule. It might fail to schedule actual jobs to meet their deadlines even though every job meets its deadline in the maximum schedule.

Fortunately, Ha and Liu [12] showed that any job-level fixed priority algorithm is predictable in preemptible and migratable systems, where every job can be dispatched to execute on any processor, can be preempted at any time, and can be resumed on any processor. With job-level fixed priority algorithms, validation methods can confine their attention to the maximum execution time and ignore the variations in the actual execution time [5].

Until now, however, little research has addressed this problem for job-level dynamic priority algorithms. Job-level dynamic priority algorithms such as LLF and EDZL (Earliest Deadline Zero Laxity) [13,14] may or may not be predictable.[1] In order to use a job-level dynamic priority algorithm in real systems, validation methods should include a test for the predictability of the algorithm.

In this paper, we study the predictability of a job-level dynamic priority algorithm, LLF, for multiprocessor real-time systems. We present a necessary and sufficient condition for a priority-driven algorithm to be predictable. We show that LLF satisfies this condition if the actual execution times cannot be known until jobs finish. However, if the actual execution times can be known,

[1] Recently Xuefeng Piao et al. [15] proved the predictability of EDZL algorithm.

LLF is finish time predictable but is not start time predictable, which may be counterintuitive.

This paper is organized as follows. Section 2 briefly explains the system model and notations. Section 3 discusses the predictability of priority-driven algorithms and Section 4 presents the predictability of LLF algorithm. Then, we conclude in Section 5.

## 2 System Model and Notations

We consider preemptive and migratable scheduling of independent real-time jobs on $m$ identical multiprocessors. A job set $J = \{j_1, j_2, \ldots, j_n\}$ is a set of jobs $j_i = (r_i, e_i, d_i)$. $j_i$ is released at time $r_i$, requires execution of $e_i$ time units, and must complete its execution by deadline $d_i$. The start time of $j_i$, denoted by $s(j_i)$, is a time instance at which the execution of $j_i$ begins. The finish time of $j_i$ is denoted by $f(j_i)$, and $j_i$ is said to meet its deadline if $f(j_i) \leq d_i$. The remaining execution time of $j_i$ at time $t$ is denoted by $rc(j_i, t)$. The laxity of $j_i$ at time $t$ is defined as $l(j_i, t) = d_i - t - rc(j_i, t)$.

A job is said to be *active* if it is released and has not completed its execution, i.e., $j_i$ is active at time $t$ if $t \geq r_i$ and $rc(j_i, t) > 0$. The schedule of $J$ produced by an algorithm $Q$ is $Q(J) = \{(t, \sigma_Q(J,t))|\ t = 0, 1, 2, \cdots\}$ where $\sigma_Q(J,t)$ is the set of indexes of at most $m$ active jobs such that their priorities are higher than other jobs' at time $t$. In the rest of this paper, we mean $\sigma_{LLF}(J,t)$ by $\sigma(J,t)$.

**Observation 1.** *For any priority-driven algorithm $Q$, $l(j_i, t+1) = l(j_i, t)$ for $i \in \sigma_Q(J,t)$ and $l(j_i, t+1) = l(j_i, t) - 1$ for $i \notin \sigma_Q(J,t)$.*

A job $j_i$ is chosen to execute during $[t, t+1)$ if it is active and the number of jobs with higher priorities than $j_i$'s is smaller than $m$ at time $t$. During that interval the laxity of $j_i$ does not change. In case that $j_i$ is not active or the number of jobs with higher priorities than $j_i$'s is larger than or equal to $m$, $j_i$ is not chosen to execute and its laxity decreases by one. Note that the laxity of a job decreases by definition even when the job is not released yet or has finished.

The actual execution time $e_i$ can be any value in the range $[e_i^-, e_i^+]$. $e_i^-$ is the minimum execution time and $e_i^+$ is the maximum execution time of actual job $j_i$. In general $e_i^-$ and $e_i^+$ are known *a priori*. But $e_i$ may not be known in advance because the actual execution time is affected by input data, branches, cache misses, etc. Hence, it is usually assumed that either (A1) the actual execution time can be known when the job arrives, or (A2) the actual execution time cannot be known until the job finishes.

We define $j_i^+ = (r_i, e_i^+, d_i)$ as the maximum job of $j_i$ and $j_i^- = (r_i, e_i^-, d_i)$ as the minimum job. $J^+$ is the set of the maximum jobs $\{j_1^+, j_2^+, \cdots, j_n^+\}$ and $J^-$ is the set of the minimum jobs $\{j_1^-, j_2^-, \cdots, j_n^-\}$. Similarly, the maximum schedule of $J$ produced by algorithm $Q$ is $Q(J^+)$ and the minimum schedule is $Q(J^-)$.

## 3 Predictability of Priority-Driven Algorithms

Ha and Liu [12] defined the predictability as follows. An actual job $j_i$ is *start time predictable* if $s(j_i^-) \leq s(j_i) \leq s(j_i^+)$. Similarly, $j_i$ is *finish time predictable* if $f(j_i^-) \leq f(j_i) \leq f(j_i^+)$. And $j_i$ is *predictable* if it is both start time predictable and finish time predictable. A set of actual jobs $J$ is predictable if every job in $J$ is predictable. A scheduling algorithm $Q$ is said to be predictable if any job set $J$ is predictable as long as it is scheduled by $Q$.

Now consider two job sets $G = \{g_1, g_2, \cdots, g_n\}$ where $g_i = (gr_i, ge_i, gd_i)$, and $G' = \{g'_1, g'_2, \ldots, g'_n\}$ where $g'_i = g_i$ for $i \neq k$ and $g'_k = (gr_k, ge'_k = ge_k - 1, gd_k)$, i.e., $g'_k$ has the same release time and deadline as $g_k$ and execution requirement smaller than $g_k$ by 1 time unit. A job whose execution requirement is 0 time units is called *null job*. The start time and the finish time of a null job are defined as its release time, that is, $s(g_i) = f(g_i) = gr_i$ if $ge_i = 0$. In case $ge_k = 1$, $ge'_k = 0$ and $g'_k$ is a null job.

This section shows that the start/finish time predictability of $Q$ can be determined by comparing the start/finish time in $Q(G)$ and the start/finish time in $Q(G')$ for each job. Lemma 1 and Lemma 2 show that the start time and finish time in the actual schedule are respectively no later than the start time and the finish time in the maximum schedule if and only if the start time and finish time in $Q(G')$ are respectively no later than the start time and finish time in $Q(G)$. Then, Theorem 1 and Theorem 2 provide a necessary and sufficient condition for the start time predictability and the finish time predictability, respectively.

**Lemma 1.** *$s(j_i) \leq s(j_i^+)$ for $1 \leq i \leq n$ if and only if $s(g'_i) \leq s(g_i)$ for $1 \leq i \leq n$.*

*Proof.* Trivially $(\forall i, s(j_i) \leq s(j_i^+)) \Rightarrow (\forall i, s(g'_i) \leq s(g_i))$. We have to show $(\forall i, s(g'_i) \leq s(g_i)) \Rightarrow (\forall i, s(j_i) \leq s(j_i^+))$

Let $J'' = \{j''_1, j''_2, \ldots, j''_n\}$ denote a job set such that $j''_i = j_i^+$ for $i \neq k$ and $j''_k = (r_k, e_k, d_k)$. In other words, the execution requirement of $j''_k$ is the same as the actual execution time of $j_k$ and the execution requirements of other jobs are the same as the maximum execution time. We prove this lemma in two steps. Step 1 shows $(\forall i, s(g'_i) \leq s(g_i)) \Rightarrow (\forall i, s(j''_i) \leq s(j_i^+))$. Then, Step 2 shows $(\forall i, s(j''_i) \leq s(j_i^+)) \Rightarrow (\forall i, s(j_i) \leq s(j_i^+))$.

**Step 1:** Consider a job set $J^{(p)} = \{j_1^{(p)}, j_2^{(p)}, \ldots, j_n^{(p)}\}$ where $p = 0, 1, \cdots, e_k^+ - e_k$. $j_i^{(p)} = j_i^+ (i \neq k)$ and $j_k^{(p)} = (r_k, e_k^{(p)}, d_k)$ where $e_k^{(p)} = e_k^+ - p$. Then, $J^{(0)} = J^+$ and $J^{(e_k^+ - e_k)} = J''$. $s(j_i^{(0)}) = s(j_i^+)$ and $s(j_i^{(e_k^+ - e_k)}) = s(j''_i)$. Since $(\forall i, s(g'_i) \leq s(g_i))$ implies $(\forall i, s(j_i^{(p+1)}) \leq s(j_i^{(p)}))$, $\forall i, s(j_i^{(e_k^+ - e_k)}) \leq \cdots \leq s(j_i^{(0)})$. Therefore, $\forall i, s(j''_i) \leq s(j_i^+)$.

**Step 2:** Consider a set of jobs $J^{<q>} = \{j_1^{<q>}, j_2^{<q>}, \ldots, j_n^{<q>}\}$ where $q = 1, 2, \cdots, n$. $j_i^{<q>} = j_i$ for $1 \leq i \leq q$ and $j_i^{<q>} = j_i^+$ for $q < i \leq n$. Define $J^{<0>} = \{j_i^+, j_2^+, \cdots, j_n^+\}$. Then, $J^{<0>} = J^+$ and $J^{<n>} = J$. $s(j_i^{<0>}) = s(j_i^+)$ and $s(j_i^{<n>}) = s(j_i)$. Now consider $J^{<q>}$ and $J^{<q+1>}$. Since $J^{<q>} = \{j_1, j_2, \cdots, j_q, j_{q+1}^+, j_{q+2}^+, \cdots, j_n^+\}$ and $J^{<q+1>} = \{j_1, j_2, \cdots, j_q, j_{q+1}, j_{q+2}^+, \cdots, j_n^+\}$, $\forall i, s(j_i^{<q+1>}) \leq s(j_i^{<q>})$ by the result of Step 1. Hence, $\forall i, s(j_i^{<n>}) \leq \cdots \leq s(j_i^{<0>})$. Therefore, $\forall i, s(j_i) \leq s(j_i^+)$. □

**Lemma 2.** *$f(j_i) \leq f(j_i^+)$ for $1 \leq i \leq n$ if and only if $f(g'_i) \leq f(g_i)$ for $1 \leq i \leq n$.*

*Proof.* Similar to the proof of Lemma 1. □

**Theorem 1.** (start time predictability) *Suppose $G$ and $G'$ are scheduled by a priority-driven algorithm $Q$ respectively. Then, $Q$ is start time predictable on the domain of integers if and only if $(\forall i, s(g'_i) \leq s(g_i))$.*

*Proof.* $(\forall i, s(j_i^-) \leq s(j_i))$ straightforwardly follows from Lemma 1 and the fact that $(\forall i, e_i^- \leq e_i)$. Hence $s(j_i^-) \leq s(j_i) \leq s(j_i^+)$. □

**Theorem 2.** (finish time predictability) *Suppose $G$ and $G'$ are scheduled by a priority-driven algorithm $Q$ respectively. Then, $Q$ is finish time predictable on the domain of integers if and only if $(\forall i, f(g'_i) \leq f(g_i))$.*

*Proof.* $(\forall i, f(j_i^-) \leq f(j_i))$ follows from Lemma 2 and the fact that $(\forall i, e_i^- \leq e_i)$. Hence $f(j_i^-) \leq f(j_i) \leq f(j_i^+)$. □

## 4 Predictability of Least Laxity First Algorithm

Job-level fixed priority algorithms, such as EDF, do not refer to the remaining execution time in assigning priorities. They refer to only fixed values and thus assign job-level fixed priorities. Ha and Liu [12] showed that any job-level fixed priority algorithm is predictable. However, little research has addressed the predictability of job-level dynamic priority algorithms.

In this section, we discuss the predictability of LLF [10,11]. LLF is a well-known job-level dynamic priority algorithm. In LLF scheduling, an active job with less laxity is given higher priority. Jobs with the same laxity are executed in a static order determined by tie-breaking rules. Without loss of generality, we assume that $j_i$ is preferred to $j_{i+1}$ if $l(j_i, t) = l(j_{i+1}, t)$.

LLF is optimal on uniprocessor, and it outperforms many job-level fixed priority algorithms on multiprocessors [16]. However, LLF may cause a huge number of context switches when laxities of two or more jobs tie. To reduce the number of context switches, Oh and Yang [17] proposed a modified LLF algorithm that allows laxity inversions. Hilderbrandt et al. [18] developed a coprocessor dedicated for LLF scheduling. Livani and Kaiser [19] presented a CAN bus scheduling scheme that implemented LLF algorithm.

### 4.1 Start Time Predictability

In this section we deal with the start time predictability depending on the knowledge of the actual execution time. If the actual execution times can be known when jobs arrive, LLF can refer to the actual execution time in computing the laxity in the actual schedule. Otherwise, LLF has to use the maximum execution time instead of the actual execution time.[2]

[2] This version of LLF can be called Least Estimated Laxity First.

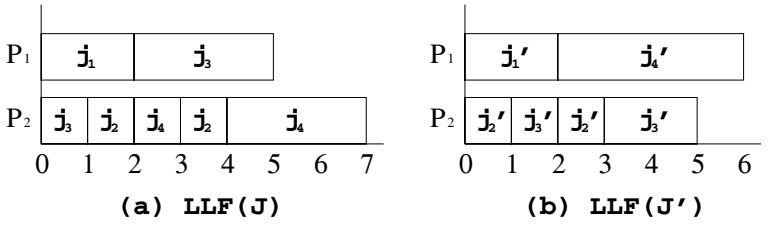


**Fig. 1.** LLF is not start time predictable if the actual execution time is known

Assume that the actual execution time is known. LLF is not start time predictable. For example, let $J = \{j_1 = (0, 2, 3),\ j_2 = (0, 2, 5),\ j_3 = (0, 4, 6),\ j_4 = (0, 4, 7)\}$ be the set of the maximum jobs and let $J' = \{j_1' = (0, 2, 3),\ j_2' = (0, 2, 5),\ j_3' = (0, 3, 6),\ j_4' = (0, 4, 7)\}$ be the set of the actual jobs. Suppose they are scheduled by LLF algorithm on two processors. The resulting schedules are shown in Fig. 1. Consider the start times of $j_3$ and $j_3'$, that is, $s(j_3)$ and $s(j_3')$, respectively. In $LLF(J)$, $l(j_1, 0) = 1$, $l(j_2, 0) = 3$, $l(j_3, 0) = 2$, $l(j_4, 0) = 3$. $j_1$ and $j_3$ is chosen to execute during $[0, 1)$. In $LLF(J')$, however, $l(j_1', 0) = 1$, $l(j_2', 0) = 3$, $l(j_3', 0) = 3$, $l(j_4', 0) = 3$, and $j_2'$ is given a higher priority than $j_3'$ according to the tie-breaking rule. $j_3'$ cannot execute during $[0, 1)$. Hence, $s(j_3) = 0 < s(j_3') = 1$.

Above example shows that LLF is not start time predictable if the actual execution times are known, that is, it is not guaranteed that the actual start time of each job is no later than the start time in the maximum schedule.

Counterintuitively, however, LLF is start time predictable if the actual execution times are not known. Consider two job sets $G$ and $G'$. $G = \{g_1, g_2, \ldots, g_n\}$ where $g_i = (r_i, e_i, d_i)$, and $G' = \{g_1', g_2', \ldots, g_n'\}$ where $g_i' = g_i (i \neq k)$ and $g_k' = (r_k, e_k - 1, d_k)$. Without loss of generality, we can assume $G$ is the maximum job set and $G'$ is the actual one. LLF has no choice but to use the maximum execution time $e_k$ in computing the laxity of $g_k'$ in the actual schedule $LLF(G')$. Each job in $LLF(G')$ is given the same priority as it is in $LLF(G)$ until $g_k'$ completes its execution. At that time, we come to know that the remaining execution time of $g_k'$ is zero.

To begin with, Lemma 3 shows that the laxity of each actual job in $LLF(G')$ is bounded from below by the laxity of the corresponding maximum job in $LLF(G)$ if the actual execution times are known. Based on this, Theorem 3 shows that LLF is start time predictable if the actual execution time cannot be known.

**Lemma 3.** *Suppose that $G$ and $G'$ are scheduled by LLF, respectively. At each time $t$ there exists an integer $k_t (1 \leq k_t \leq n)$ such that $l(g_{k_t}', t) = l(g_{k_t}, t) + 1$ and $l(g_i', t) = l(g_i, t)$ for $i \neq k_t$.*

*Proof.* By induction on time $t$. When $t = 0$, $l(g_i', 0) = l(g_i, 0)$ for $i \neq k$ and $l(g_k', 0) = l(g_k, 0) + 1$. Hence $k_0 = k$. Assume that at time $t$ there exists an integer $k_t$ such that $l(g_i', t) = l(g_i, t)$ for $i \neq k_t$ and $l(g_{k_t}', t) = l(g_{k_t}, t) + 1$. Then, we will show that at time $t + 1$ there exists an integer $k_{t+1}$ such that $l(g_{k_{t+1}}', t + 1) = l(g_{k_{t+1}}, t + 1) + 1$ and $l(g_i', t + 1) = l(g_i, t + 1)$ for $i \neq k_{t+1}$.

**Case 1:** $k_t \in \sigma(G,t)$ and $k_t \in \sigma(G',t)$. In this case, $\sigma(G,t) = \sigma(G',t)$. For $i \notin \sigma(G,t)$, $l(g_i, t+1) = l(g_i,t) - 1 = l(g'_i,t) - 1 = l(g'_i, t+1)$. For $i \in \sigma(G,t)$ and $i \neq k_t$, $l(g_i,t+1) = l(g_i,t) = l(g'_i,t) = l(g'_i,t+1)$. For $i = k_t$, $l(g_i,t+1) = l(g_i,t) = l(g'_i,t) - 1 = l(g'_i,t+1) - 1$. Hence, $k_{t+1} = k_t$. See Fig. 2 (a).

**Case 2 :** $k_t \notin \sigma(G,t)$ and $k_t \notin \sigma(G',t)$. Similarly, $\sigma(G,t) = \sigma(G',t)$ and $k_{t+1} = k_t$. See Fig. 2 (b).

**Case 3 :** $k_t \in \sigma(G,t)$ and $k_t \notin \sigma(G',t)$. There must be a job $g_s (s \neq k_t)$ such that $s \notin \sigma(G,t)$ but $s \in \sigma(G',t)$. In this case, $\sigma(G,t) - \{k_t\} = \sigma(G',t) - \{s\}$. For $i \in \sigma(G,t)$ and $i \neq k_t$, $l(g_i,t+1) = l(g_i,t) = l(g'_i,t) = l(g'_i,t+1)$. For $i \notin \sigma(G,t)$ and $i \neq s$, $l(g_i,t+1) = l(g_i,t) - 1 = l(g'_i,t) - 1 = l(g'_i,t+1)$. For $i = k_t$, $l(g_i,t+1) = l(g_i,t) = l(g'_i,t) - 1 = l(g'_i,t+1)$. For $i = s$, $l(g_i,t+1) = l(g_i,t) - 1 = l(g'_i,t) - 1 = l(g'_i,t+1) - 1$. Hence, $k_{t+1} = s$. See Fig. 2 (c).

**Case 4 :** $k_t \notin \sigma(G,t)$ and $k_t \in \sigma(G',t)$. Since $l(g_i,t) = l(g'_i,t)$ for $i \neq k_t$ and $l(g_{k_t},t) < l(g'_{k_t},t)$, at time $t$ the number of jobs with less laxity than $g_{k_t}$ in $LLF(G)$ is not greater than the number of jobs with less laxity than $g'_{k_t}$ in $LLF(G')$. This contradicts the assumption that $k_t \notin \sigma(G,t)$ and $k_t \in \sigma(G',t)$. Hence, this case cannot happen. See Fig. 2 (d). □

**Theorem 3.** *LLF is start time predictable if the actual execution time is not known.*

*Proof.* Suppose that $G$ and $G'$ are scheduled by $LLF$ respectively. Since the actual execution times cannot be known, $LLF$ assigns priorities to $g'_k$ based on $e_k$ (not on $e'_k$) in $LLF(G')$. The actual schedule is the same as the maximum schedule until $g'_k$ finishes, i.e., $\sigma(G,t) = \sigma(G',t)$ for $0 \leq t < f(g'_k)$. Hence, for each job $g'_i$ such that $s(g'_i) < f(g'_k)$, $s(g'_i) = s(g_i)$.

Now we will shown that $s(g'_i) \leq s(g_i)$ for each job $g'_i$ such that $s(g'_i) \geq f(g'_k)$. At time $f(g'_k)$, $g'_k$ finishes in $LLF(G')$ but $g_k$ does not in $LLF(G)$, i.e., $rc(g_k, f(g'_k)) = 1$ and $rc(g'_k, f(g'_k)) = 0$. At time $f(g'_k)$, $G$ can be thought as a job set $H = \{h_1, h_2, \cdots, h_n\}$ where $h_i = g_i$ for $i$ such that $r_i \geq f(g'_k)$ and $h_i = (f(g'_k), rc(g_i, f(g'_k)), d_i)$ for $i$ such that $r_i < f(g'_k)$. Similarly, $G'$ can be thought as a job set $H' = \{h'_1, h'_2, \cdots, h'_n\}$ where $h'_i = g'_i$ for $i$ such that $r_i \geq f(g'_k)$ and $h'_i = (f(g'_k), rc(g'_i, f(g'_k)), d_i)$ for $i$ such that $r_i < f(g'_k)$. Then $h_k = (f(g'_k), 1, d_k)$, $h'_k = (f(g'_k), 0, d_k)$, and $h_i = h'_i (i \neq k)$.

It is true that $s(g'_i) \leq s(g_i)$ for $i$ such that $s(g'_i) \geq f(g'_k)$, provided $s(h'_i) \leq s(h_i)$ for all $i$. Thus, we will show that $h'_i$ has already started by $t$ in $LLF(H')$ if $h_i$ starts execution at $t \geq f(g'_k)$ in $LLF(H)$.

Suppose $h_i$ starts execution at $t$. By Lemma 3, there exists an integer $k_t$ such that $l(h'_j,t) = l(h_j,t)$ for $j \neq k_t$ and $l(g'_j,t) = l(g_j,t) + 1$ for $j = k_t$.

**Case** $i = k_t$**:** $l(h_i,t) \neq l(h'_i,t)$. This implies that $h'_i$ already started before $t$ or $i = k$. In both cases, $h'_i$ already started before $t$ since $h'_k$ already finished ($h'_k$ is a null job).

**Case** $i \neq k_t$**:** $l(h_i,t) = l(h'_i,t)$. This implies $h'_i$ does not start execution yet. Assume that $h'_i$ does not start execution at $t$, that is, $i \notin \sigma(H',t)$. There must exist a job $h_s$ such that $s \notin \sigma(H,t)$ and $s \in \sigma(H',t)$. Then, $l(h_s,t) = l(h'_s,t)$ because $s \neq k_t$ by Case 4 in Lemma 3. This contradicts the fact that $l(h_i,t) = l(h'_i,t)$ because $h_s$ is given a lower priority than $h_i$ at $t$ in $LLF(H)$ but $h'_s$ is

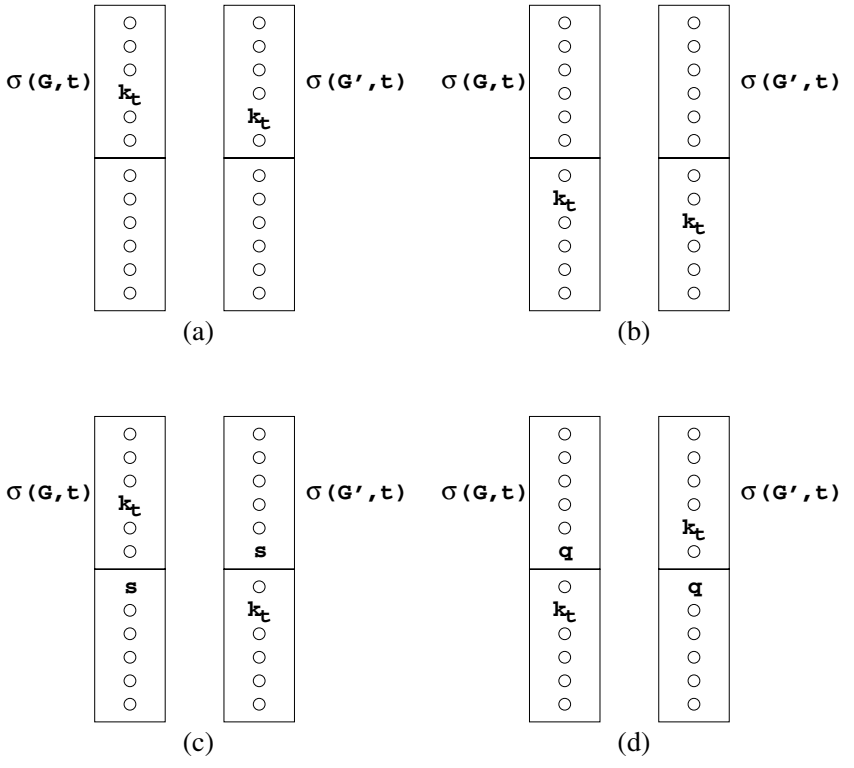


**Fig. 2.** Four possible cases depending on the execution of $g_k$ and $g'_k$ at time $t$. The number of processors is 6, and the number of active jobs is 12.

given a higher priority than $h'_i$ at $t$ in $LLF(H)$. Therefore, $i \in \sigma(H', t)$, that is to say, $h'_i$ starts execution at $t$. □

### 4.2 Finish Time Predictability

The finish time predictability is more important than the start time predictability. The purpose of validation is to determine whether all jobs in a system indeed meet their timing constraints, one of which is deadline. In hard real-time systems, especially, the usefulness of a job is measured by whether it completes by its deadline or not.

In the first place, Lemma 4 shows that for any scheduling algorithm $Q$ the finish time of each job in $Q(G')$ is no later than the finish time of the corresponding job in $Q(G)$ if the laxity of each job in $Q(G')$ is always no less than that of the corresponding job in $Q(G)$. Then, Theorem 4 and Theorem 5 show that LLF is finish time predictable regardless of the knowledge of the actual execution times.

**Lemma 4.** *Suppose $G$ and $G'$ are scheduled by a priority-driven algorithm $Q$ respectively. Then, for any job $g_i$, if $l(g'_i, t) \geq l(g_i, t)$ for all $t$, then $f(g'_i) \leq f(g_i)$.*

*Proof.* By contradiction. Suppose that for some job $g_j$ $l(g'_j, t) \geq l(g_j, t)$ for all $t$ but $f(g'_j) > f(g_j)$. Then, $rc(g'_j, f(g_j)) > 0$ and $rc(g_j, f(g_j)) = 0$. $l(g'_j, f(g_j)) = d_j - f(g_j) - rc(g'_j, t)$ and $l(g_j, f_j)) = d_j - f(g_j)$. Thus, $l(g'_j, f(g_j)) < l(g_j, f(g_j))$, which is a contradiction. □

**Theorem 4.** *LLF is finish time predictable if the actual execution time is known.*

*Proof.* Suppose that $G$ and $G'$ are scheduled by $LLF$ respectively. By Lemma 3, we have $(\forall i, l(g'_i, t) \geq l(g_i, t))$ at each time $t$. Then, by Lemma 4, $\forall i, f(g'_i) \leq f(g_i)$. □

**Theorem 5.** *LLF is finish time predictable even if the actual execution time is not known.*

*Proof.* Suppose that $G$ and $G'$ are scheduled by $LLF$ respectively. Since the actual execution times cannot be known, $LLF$ assigns priorities to $g'_k$ based on $e_k$ (not on $e'_k$) in $LLF(G')$. The actual schedule is the same as the maximum schedule until $g'_k$ finishes, i.e., $\sigma(G, t) = \sigma(G', t)$ for $0 \leq t < f(g'_k)$. Hence, for each job $g'_i$ such that $f(g'_i) < f(g'_k)$, $f(g'_i) = f(g_i)$.

Now we will shown that $f(g'_i) \leq f(g_i)$ for each job $g'_i$ such that $f(g'_i) \geq f(g'_k)$. Similar to the proof of Theorem 3, $G$ and $G'$ can be thought as $H$ and $H'$, respectively. At $f(g'_k)$ the execution requirement of every job $h_i$ is known. By Theorem 4, $(\forall i, f(h'_i) \leq f(h_i))$. Hence, for each job $g'_i$ such that $f(g'_i) > f(g'_k)$, $f(g'_i) \leq f(g_i)$. □

## 5 Conclusion

Validation methods for dynamic systems can concentrate on the feasibility of jobs based on the maximum execution time, ignoring the variation in the actual execution time only if the scheduling algorithm is predictable. It was shown by previous works that any job-level fixed priority algorithm is predictable. However, the predictability of job-level dynamic priority algorithms has not been focused on.

In this paper, we presented a necessary and sufficient condition for priority-driven algorithms to be start time (or finish time) predictable on the domain of integers. Then, we showed that LLF is start time predictable if the actual execution times cannot be known. This may be counterintuitive because LLF is not start time predictable if the actual execution times are known. Besides, we showed that LLF is finish time predictable regardless of the knowledge of the actual execution time. Since a major concern in validating hard real-time jobs is whether all jobs meet their deadlines, the finish time predictability of LLF is a useful property.

## References


1. Wolf, W.: The Future of Multiprocessor Systems-on-Chips. In: Proceedings of the 41st annual conference on Design automation. (2004) 681–685
2. ARM: ARM11 MPCore. http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html (2006)
3. Richter, K., Racu, R., Ernst, R.: Scheduling Analysis Integration for Heterogeneous Multiprocessor Soc. In: Proceedings of the 24th International Real-Time Systems Symposium. (2003) 236–245
4. Richter, K., Jersak, M., Ernst, R.: A Formal Approach to Mpsoc Performance Verification. IEEE Computer **36**(4) (2003) 60–67
5. Liu, J.W. In: Real-Time Systems. Prentice Hall (2000) 70
6. Audsley, N., Burns, A., Richardson, M., Wellings, A.: Hard Real-Time Scheduling: The Deadline Monotonic Approach. In: Proceedings of IEEE Workshop on Real-Time Operating Systems and Software. (1991) 133–137
7. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM **20**(1) (1973) 46–61
8. Srinivasan, A., Baruah, S.: Deadline-based Scheduling of Periodic Task Systems on Multiprocessors. Information Processing Letters **84**(2) (2002) 93–98
9. Baruah, S.K.: Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors. IEEE Trans. on Computers **53**(6) (2004) 781–784
10. Leung, J.: A New Algorithm for Scheduling Periodic Real-Time Tasks. Algorithmica **4** (1989) 209–219
11. Dertouzos, M.L., Mok, A.K.: Multiprocessor On-Line Scheduling of Hard Real-Time Tasks. IEEE Trans. on Software Engineering **15**(12) (1989) 1497–1506
12. Ha, R., Liu, J.: Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems. In: Proceedings of 14th IEEE International Conference on Distributed Computing systems. (1994) 162–171
13. Cho, S., Lee, S., Ahn, S., Lin, K.: Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. IEICE Trans. Commun. **E85-B**(12) (2002) 2859–2867
14. Park, M., Han, S., Kim, H., Cho, S., Cho, Y.: Comparison of Deadline-based Scheduling Algorithms for Periodic Real-Time Tasks on Multiprocessor. IEICE Trans. Inf. and Syst. **88-D**(3) (2005) 658–661
15. Piao, X., Han, S., Kim, H., Park, M., Cho, Y., Cho, S.: Predictability of Earliest Deadline Zero Laxity Algorithm for Multiprocessor Real-Time Systems. In: Proceedings of 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06). (2006) 359–364
16. Park, M., Han, S., Kim, H., Cho, S., Cho, Y.: Dominance and Performance of Real-Time Scheduling Algorithms on Multiprocessors. Journal of KISS **32**(7) (2005) 368–376
17. Oh, S., Yang, S.: A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks. In: Proceedings of Fifth International Conference on Real-Time Computing Systems and Applications (RTCSA'98). (1998) 31–36
18. Hildebrandt, J., Golatowski, F., Timmermann, D.: Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems. In: Proceedings of 11th Euromicro Conference on Real-Time Systems (ECRTS99). (1999) 208–215
19. Livani, M., Kaiser, J.: Evaluation of a Hybrid Real-time Bus Scheduling Mechanism for CAN. Lecture Notes in Computer Science **1586** (1999) 425–429