Validating Timing Constraints of Dependent Jobs with Variable Execution Times in Distributed Real-Time Systems

Hojung Cha[1] and Rhan Ha[2]

[1] Dept. of Computer Science, Yonsei University, Seoul 120-749, Korea
hjcha@cs.yonsei.ac.kr
[2] Dept. of Computer Engineering, Hongik University, Seoul 121-791, Korea
rhanha@cs.hongik.ac.kr

Abstract. In multiprocessor and distributed real-time systems, scheduling jobs dynamically on processors can be used to achieve better performance. However, analytical and efficient validation methods for determining whether all the timing constraints are met do not yet exist for systems using modern dynamic scheduling strategies, and exhaustive methods are often infeasible or unreliable since the execution time and release time of each job may vary. In this paper, we present several upper bounds and efficient algorithms for computing the worst-case completion times of dependent jobs in dynamic systems where jobs are dispatched and scheduled on available processors in a priority-driven manner. The bounds and algorithms consider arbitrary release times and variable execution times. We present conditions under which dependent jobs execute in a predictable manner.

1 Introduction

Many safety-critical real-time applications, (*e.g.*, air-traffic control and factory automation) have timing constraints that must be met for them to be correct. Their timing constraints are often specified in terms of the release times and deadlines of the jobs that make up the system. A *job* is a basic unit of work to be scheduled and executed. It can begin execution when its data and control dependencies are met only after its *release time*, and it must complete by its *deadline*. The failure of a job to complete by its deadline can lead to a performance degradation or complete failure.

We assume here that the scheduler never schedules any job before its release time and focus on the problem of how to validate that every job indeed completes by its deadline when executed according to a given priority-driven discipline used by the scheduler. A scheduling algorithm is *priority-driven* if it never leaves any processor idle intentionally. Such an algorithm can be implemented by assigning priorities to jobs, placing jobs ready for execution in one or more queues and scheduling the jobs with the highest priorities among all jobs in the queue(s) on the available processors. Specifically, we consider here only *dynamic systems*: in a dynamic system, ready jobs are placed in a common queue and are dispatched to available processors in a priority-driven manner.

Recently, many real-time load balancing and scheduling algorithms for dynamic systems have been developed. These algorithms are likely to make better use of the processors and achieve a higher responsiveness than the traditional approach to scheduling jobs in multiprocessor and distributed environments. According to the traditional approach, jobs are first statically assigned and bound to processors, and then a uniprocessor scheduling algorithm is used to schedule the jobs on each processor.

A system that uses a priority-driven scheduling algorithm can have unexpected timing behavior [1]. A job may complete later where it and other jobs execute for shorter amounts of time and when jobs are released sooner. Consequently, it is impractical and unreliable to validate that all jobs meet their deadlines using exhaustive testing and simulation when their execution times and release times may vary. Recently, several efficient and analytical methods for validating timing constraints of static multiprocessor and distributed systems have been developed, *e.g.*, [2,3,4] (In a static system, jobs are assigned and bound to processors.). These methods are based on worst-case bounds and schedulability conditions for uniprocessor systems[5,6,7,8]. They allow us to bound the completion times of jobs that are scheduled in a priority-driven manner even when their release times and execution times may vary. Several efficient algorithms for computing the worst-case completion times of independent jobs in dynamic systems now exist[9]. This paper is concerned with the case where jobs have dependencies, the processors are identical, and the system is dynamic. The works in [10,11] are also related to the problem solved in this paper. In [10,11], they have studied the validation problem to bound the completion times of jobs on one processor. Our work provides the bounds of completion times of jobs in multiprocessor or distributed systems.

The rest of the paper is organized as follows. Section 2 presents the formal definition of the validation problem addressed by the paper. Sections 3 and 4 give conditions for predictable execution and present an efficient algorithm for bounding the completion times of dependent jobs. Conclusions are given in Section 5.

2 Problem

The general validation problem addressed here and in [9] can be stated as follows: given n jobs, m identical processors, and the priority-driven scheduling algorithm that dynamically schedules the jobs on the processors, determine whether all the jobs meet their deadlines analytically or by using an efficient algorithm. We let $\boldsymbol{J} = \{J_1, J_2, \ldots, J_n\}$ denote the set of jobs. As in [3,4,5,6,7,8] each job J_i has the following parameters: release time r_i, deadline d_i and execution time e_i. These parameters are rational numbers. The actual execution time e_i of J_i is in the range $[e_i^-, e_i^+]$. We call e_i^- its *minimum execution time* and e_i^+ its *maximum execution time*. We assume that the scheduler knows the parameters r_i, d_i and

$[e_i^-, e_i^+]$ of every job J_i before any job begins execution, but the actual execution time e_i is unknown.

The jobs in $\boldsymbol{J}$ are dependent; data and control dependencies between them impose *precedence constraints* in the order of their execution. A job J_i is a *predecessor* of another job J_j (and J_j is a *successor* of J_i) if J_j cannot begin to execute until J_i completes. We denote this precedence relation by $J_i \prec J_j$. J_i is an *immediate predecessor* of J_j (and J_j is an *immediate successor* of J_i) if $J_i \prec J_j$ and there is no other job J_k such that $J_i \prec J_k \prec J_j$. Two jobs J_i and J_j are *independent* if neither $J_i \prec J_j$ nor $J_j \prec J_i$. Independent jobs can be executed in any order. We use a *precedence graph* $G = (\boldsymbol{J}, \boldsymbol{R})$ to represent the precedence relations between jobs. There is a node J_i in this graph for each job J_i in $\boldsymbol{J}$, and there is an edge from J_i to J_j in $\boldsymbol{R}$ whenever J_i is an immediate predecessor of J_j. We say that a job becomes *ready* when the time is at or after its release time and either it has no predecessor or all of its predecessors are completed, and it remains ready until it completes.

We confine our attention to scheduling algorithms that assign fixed priorities to jobs and do not choose priorities based on the actual execution times of the jobs. Therefore, the given scheduling algorithm is completely defined by the list of priorities it assigns to the jobs. Without loss of generality, we assume that the priorities of jobs are distinct. We will always index the job in decreasing priority order (*i.e.*, the priority list is $(J_1, J_2, \ldots, J_n)$) except where it is stated to be otherwise. $\boldsymbol{J_i} = \{J_1, J_2, \ldots, J_i\}$ denotes the subset of jobs with priorities equal to or higher than the priority of J_i.

The scheduler maintains a common priority queue and places all ready jobs in the queue. In this paper, we consider the case when jobs are preemptable and migratable. In this case, a ready job can be scheduled on any processor. It may be preempted when a higher priority job becomes ready. Its execution may resume on any processor. We refer the jobs as P/M/Z or P/M/F jobs depending on whether the jobs have identical (zero) release times or fixed arbitrary release times.

We will use $\boldsymbol{J_n^+} = \{J_1^+, J_2^+, \ldots, J_n^+\}$ as a shorthand notation to mean that every job has its maximum execution time. Similarly, $\boldsymbol{J_n^-} = \{J_1^-, J_2^-, \ldots, J_n^-\}$ means that every job has its minimum execution time. We refer to the schedule of $\boldsymbol{J_n}$ produced by the given algorithm as the *actual schedule* $\boldsymbol{A_n}$. To determine whether any job completes in time, we sometimes generate simulated schedules of $\boldsymbol{J_n}$ using the given scheduling algorithm and assuming every job has its maximum execution time or every job has its minimum execution time. In particular, we call the schedule of $\boldsymbol{J_n^+}$ (or $\boldsymbol{J_n^-}$) produced by the same algorithm the *maximal* (or the *minimal*) *schedule* $\boldsymbol{A_n^+}$ (or $\boldsymbol{A_n^-}$) of $\boldsymbol{J_n}$.

Let $S(J_i)$ be the (actual) *start time* of J_i, the instant of time at which the execution of J_i begins according to the actual schedule $\boldsymbol{A_n}$. Let $S^+(J_i)$ and $S^-(J_i)$ be the *start times* of J_i in the schedules $\boldsymbol{A_n^+}$ and $\boldsymbol{A_n^-}$, respectively. We say that the start time of J_i is *predictable* if $S^-(J_i) \leq S(J_i) \leq S^+(J_i)$. Similarly, let $F(J_i)$ be the (actual) *completion time* of J_i, the instant at which J_i completes execution according to the actual schedule $\boldsymbol{A_n}$. The *response time* of a job is the length of time between its release time and its completion time. Let $F^+(J_i)$ and $F^-(J_i)$ be the *completion times* of J_i according to the schedules $\boldsymbol{A_n^+}$ and $\boldsymbol{A_n^-}$, respectively. The completion times of J_i is said to be *predictable* if $F^-(J_i) \leq F(J_i) \leq F^+(J_i)$.

We say that *the execution of* J_i *is predictable* if both its start time and completion time are predictable. In this case, the completion time $F^+(J_i)$ in the schedule $\boldsymbol{A_n^+}$ minus the release time r_i of J_i gives J_i's worst-case response time. J_i meets its deadline if $F^+(J_i) \leq d_i$.

To find the worst-case completion time of J_i when all the jobs are preemptable, we only need to consider the jobs which are the predecessors of J_i, jobs which have higher priorities than J_i and the predecessors of J_i, and jobs which are predecessors of these higher-priority jobs. To be more precise, we define a new precedence relation $\boldsymbol{R_i^*}$ as follows: Let $\boldsymbol{Y_n^i}$ be the subset of $\boldsymbol{J_n}$ which contains all the jobs that are not successors of J_i. Let $\boldsymbol{R_i^*}$ be the union of (1) the given precedence relation $\boldsymbol{R}$ restricted to the subset $\boldsymbol{Y_n^i}$ and (2) the relation over $\boldsymbol{Y_n^i}$ defined by a graph in which there is a node for each job in $\boldsymbol{Y_n^i}$ and there is an edge from J_i to J_j whenever J_i has a higher priority than J_j. Let $\boldsymbol{X_n^i}$ be the subset of $\boldsymbol{Y_n^i}$ which contains J_i and all the predecessors of J_i under the new relation $\boldsymbol{R_i^*}$. When we try to find the worst-case completion time of J_i, we must consider the jobs in the subgraph of precedence graph G induced by $\boldsymbol{X_n^i}$. We call the subset $\boldsymbol{X_n^i}$ the *interfering subset* of J_i.

3 Identical Release Time Case

Unlike independent P/M/Z jobs [9], the execution of P/M/Z jobs with precedence constraints is not predictable in general. This fact is illustrated by the example in Figure 1 where the precedence graph is an out-tree. According to the maximal and minimal schedules shown in parts (b) and (c), J_8's completion times are 8 and 6, respectively. (We omit the values of the execution times here because they are not relevant. It suffices that the execution time of every job in (b) is larger than the execution time of the corresponding job in (c).) Part (d) shows a possible actual schedule obtained when the execution time of every job is between the extreme values used in (b) and (c). According to this actual schedule, J_8 completes at 18. Hence the execution of J_8 is not predictable.

In the next section, we will present an algorithm that can be used to bound the completion times of P/M/Z jobs with arbitrary precedence graphs. Here, we examine a special case when all the jobs have identical release times and the precedence graph is a forest of in-trees. The execution of P/M/Z jobs with this type of precedence graph is predictable. Theorem 1 states this fact. To prove it, we need the following lemmas.

Lemma 1. *No job in a set of P/M/Z jobs is preempted in the actual schedule when every job has at most one immediate successor.*

Proof. A job preempts another job if when it becomes ready, all the processors are busy. At the time when the preempting job becomes ready, the number of

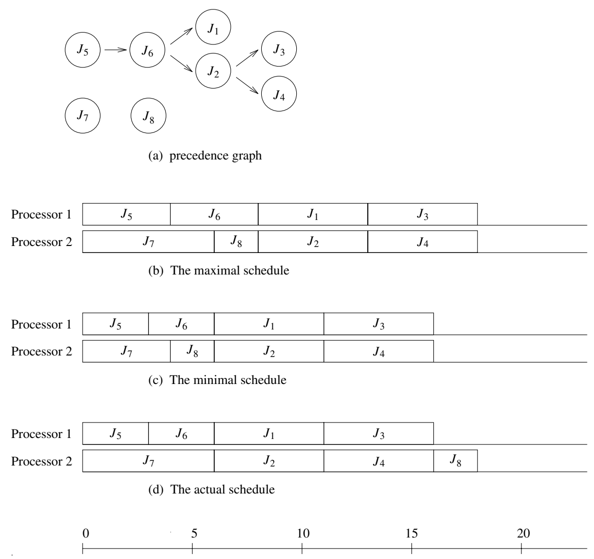

Fig. 1. Unpredictable execution of P/M/Z jobs in out-trees

jobs that are ready increases by at least one. However, when the precedence graph is a forest of in-trees, every job has at most one immediate successor. Moreover, all jobs have the same release time. As each job J_i completes, at most one immediate successor of J_i becomes ready. Hence, as time increases, the number of jobs that are ready to execute cannot increase. The lemma follows. □

Lemma 2. *Let $\boldsymbol{A_n}(i, \epsilon)$ be the schedule of the set of n jobs whose parameters and precedence constraints are the same as those of jobs in $\boldsymbol{J_n}$ except that the execution time of J_i is $e_i - \epsilon$ for some $\epsilon > 0$. When every job is a P/M/Z job and the precedence graph is an in-tree forest, the start time of every job J_j according to $\boldsymbol{A_n}(i, \epsilon)$ is no later than its actual start time $S(J_j)$ according to the actual schedule $\boldsymbol{A_n}$ of $\boldsymbol{J_n}$.*

Proof. We need to consider only the jobs that start at or later than $F(J_i) - \epsilon$ in $\boldsymbol{A_n}$. Let J_l be such a job. According to $\boldsymbol{A_n}$, a processor is available to execute J_l at $S(J_l)$. From Lemma 1, no job is preempted in $\boldsymbol{A_n}$. Hence at most $(m-1)$ higher-priority jobs that become ready at or before $S(J_l)$ are not yet complete at $S(J_l)$ according to the schedule $\boldsymbol{A_n}$. In the set of jobs scheduled according to $\boldsymbol{A_n}(i, \epsilon)$, the execution time of every job is no larger than the execution time of the corresponding job in $\boldsymbol{J_n}$, and as time increases, the number of ready jobs cannot increase. Moreover, from Lemma 1, no job is preempted according to $\boldsymbol{A_n}(i, \epsilon)$. Consequently, we can conclude that according to $\boldsymbol{A_n}(i, \epsilon)$, at most $(m-1)$ higher-priority jobs that become ready at or before $S(J_l)$ are not complete at $S(J_l)$. There is a processor available to execute J_l at or before $S(J_l)$. The lemma follows. $\square$

Corollary 1. *Let $\boldsymbol{A}_{\boldsymbol{n}}^{+}(i,\epsilon)$ be the schedule of the set of n jobs whose parameters and precedence constraints are the same as those of the jobs in $\boldsymbol{J}_{\boldsymbol{n}}^{+}$ except that the execution time of J_i is $e_i^{+} - \epsilon$ for some $\epsilon > 0$. When every job is a P/M/Z job and the precedence graph is an in-tree forest, the start time of every job J_j according to $\boldsymbol{A}_{\boldsymbol{n}}^{+}(i,\epsilon)$ is no later than its maximal start time $S^{+}(J_j)$ according to the maximal schedule $\boldsymbol{A}_{\boldsymbol{n}}^{+}$ of $\boldsymbol{J}_{\boldsymbol{n}}^{+}$.*

Let $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k]$, for $1 \leq k \leq n$, be the schedule of the set of n jobs which is such that (1) the precedence constraints of all jobs are the same as those of the jobs in $\boldsymbol{J}_{\boldsymbol{n}}$, (2) the execution time of J_j is e_j for $j = 1, 2, \cdots, k$, and (3) the execution time of J_j is e_j^{+} for $j = k+1, k+2, \cdots, n$. Let $\boldsymbol{A}_{\boldsymbol{n}}^{+}[0]$ be another notation for $\boldsymbol{A}_{\boldsymbol{n}}^{+}$.

Lemma 3. *When the precedence graph of a set of n P/M/Z jobs is an in-tree forest, the start time of every job according to $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k]$ is no later than the start time of the corresponding job according to the schedule $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k-1]$ for $1 \leq k \leq n$.*

Proof. We note that $\boldsymbol{A}_{\boldsymbol{n}}^{+}[1]$ is just $\boldsymbol{A}_{\boldsymbol{n}}^{+}(1, e_1^{+} - e_1)$ defined in Corollary 1. From this corollary, the start time of every job in $\boldsymbol{A}_{\boldsymbol{n}}^{+}[1]$ is no later than the maximal start time of the corresponding job in $\boldsymbol{A}_{\boldsymbol{n}}^{+}$.

To show that the lemma is true for $k > 1$, let J_l be a job which starts at or after $(e_k^{+} - e_k)$ time units before the completion time of J_k^{+} according to $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k-1]$. Because no job is preempted, at the time t when J_l starts in $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k-1]$, at most $(m-1)$ higher-priority jobs that become ready at or before t are not complete. The number of higher-priority jobs that become ready at or before t but are not complete at t cannot increase when the execution time of J_k^{+} is reduced from e_k^{+} to e_k because every job has at most one immediate successor and the release times of all jobs are the same. Consequently, according to $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k]$, there are at most $(m-1)$ ready jobs with priorities higher than J_l at the time t. This means that J_l can start no later than t in $\boldsymbol{A}_{\boldsymbol{n}}^{+}[k]$. $\square$

Let $\boldsymbol{A}_{\boldsymbol{n}}[k]$, for $1 \leq k \leq n$, be the schedule of the set of n jobs whose precedence constraints are the same as those of jobs in $\boldsymbol{J}_{\boldsymbol{n}}$, the execution time of J_j is e_j^{-} for $j = 1, 2, \cdots, k$ and the execution time of J_j is e_j for $j = k+1, k+2, \cdots, n$. $\boldsymbol{A}_{\boldsymbol{n}}[0]$ is another notation of $\boldsymbol{A}_{\boldsymbol{n}}$. The proof of the following corollary is similar to that of Lemma 3.

Corollary 2. *When the precedence graph of a set of n P/M/Z jobs is an in-tree forest, the start time of every job according to $\boldsymbol{A}_{\boldsymbol{n}}[k]$ is no later than the start time of the corresponding job according to the schedule $\boldsymbol{A}_{\boldsymbol{n}}[k-1]$ for $1 \leq k \leq n$.*

Theorem 1. *When the precedence graph of a set of n P/M/Z jobs is an in-tree forest, the start time of every job is predictable, that is, $S^-(J_i) \leq S(J_i) \leq S^+(J_i)$.*

Proof. We note that $\boldsymbol{A}_{\boldsymbol{n}}^+[n]$ is just the actual schedule $\boldsymbol{A}_{\boldsymbol{n}}$. That the start time $S(J_i)$ of every job J_i in $\boldsymbol{A}_{\boldsymbol{n}}$ is no later than $S^+(J_i)$ in $\boldsymbol{A}_{\boldsymbol{n}}^+$ follows straightforwardly from Lemma 3. Similarly, the fact that $S^-(J_i) \leq S(J_i)$ follows from Corollary 2. □

Corollary 3. *When the precedence graph of a set of n P/M/Z jobs is an in-tree forest, the completion time of every job is predictable, that is, $F^-(J_i) \leq F(J_i) \leq F^+(J_i)$.*

4 Arbitrary Release Time Case

In contrast to P/M/Z jobs, the execution of jobs with arbitrary release times is not predictable even when the precedence graph of jobs is a set of chains. This fact is illustrated by the example in Figure 2. The simple system in this figure contains eight jobs and two identical processors. The release time of each job is indicated by a number on top of each node in the precedence graph. The execution times of all the jobs are known except for J_3. Its execution time can be any value in the range [2,8]. Parts (b), (c) and (d) of this figure show the maximal, minimal and actual schedules, respectively. The completion time of J_6 in the actual schedule is 13; it is later than its completion time of 11 in the maximal schedule. Similarly, the execution of P/M/F jobs is not predictable when the precedence graph is a forest of in-trees. This is illustrated in Figure 3. In this example, all jobs are released at 0 except J_2 whose release time is marked by an arrow in the schedules. We note that the execution of J_4 and J_5 is not predictable. However, in the following special cases, we have predictable execution.

4.1 Predictable Execution Case

We consider here a set of P/M/F jobs whose precedence graph is a forest of in-trees. The execution of every job is predictable when the assumption of the following theorem holds. The assumption is stated in terms of $S_z^-(J_i)$, which is the start time of J_i in the schedule of $\boldsymbol{J}_{\boldsymbol{n}}^-$ constructed by assuming that all jobs are released at time 0.

Theorem 2. *When the precedence graph of a set $\boldsymbol{J}_{\boldsymbol{n}}$ of P/M/F jobs is a forest of in-trees, the execution of every job is predictable if every job J_k is released at or before $S_z^-(J_k)$.*

Proof. When all the jobs are released at time 0, their execution is predictable according to Theorem 1 and Corollary 3. Moreover, no job J_k can start execution before $S_z^-(J_k)$ in any schedule of the set $\boldsymbol{J}_{\boldsymbol{n}}$. $S_z^-(J_k)$ gives the earliest time

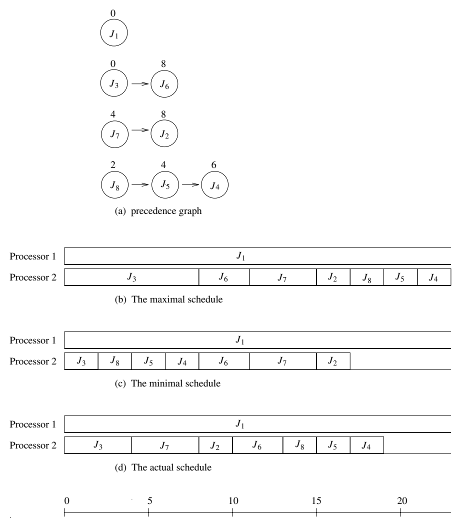

Fig. 2. Unpredictable execution of P/M/F jobs in chains

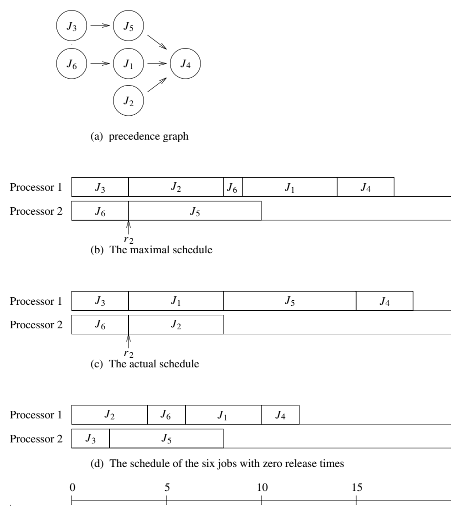

Fig. 3. Unpredictable execution of P/M/F jobs in in-trees

instant at which all the predecessors of J_k and all but at most $(m-1)$ jobs with higher priorities than J_k can complete. Hence, if every job J_k is released at or before $S_z^-(J_k)$, the schedule of the given P/M/F jobs is the same as their schedule constructed by assuming that all jobs are released at time 0. From Theorem 1 and Corollary 3, the theorem follows. □

Theorem 2 gives us a way to check whether the execution of a set of jobs whose release times are arbitrary and precedence graph is an in-tree is predictable. For example, the assumption of Theorem 2 does not hold for the jobs in Figure 3. The schedule in Figure 3 (d) is constructed based on the assumptions of zero release times and minimum execution times. According to this schedule, $S_z^-(J_2)$ is 0, but J_2 is released after 0. In contrast, the assumption holds for the system in Figure 4; hence, the jobs in this system have predictable execution. This system has the same set of jobs, the same precedence graph and the same number of processors as the system in Figure 3, except for the release times of jobs. The release time of each job J_k in Figure 4 is equal to its start time $S_z^-(J_k)$ in the schedule in Figure 3 (d) and is indicated by the number on top of each node.

Theorem 3 stated below gives us another condition for predictable execution: the precedence graph consists of chains and every job has a higher priority than its successor. To prove it, we need the following lemma.

Lemma 4. *Let J_i be a job in a set of P/M/F jobs which is such that* (1) *its precedence graph is a set of chains and* (2) *every job in the set has a priority higher than its immediate successor. The completion time of J_i is predictable if its start time is predictable and the start times of all the jobs which start before its completion time $F^+(J_i)$ according to the maximal schedule $\boldsymbol{A_n^+}$ are predictable.*

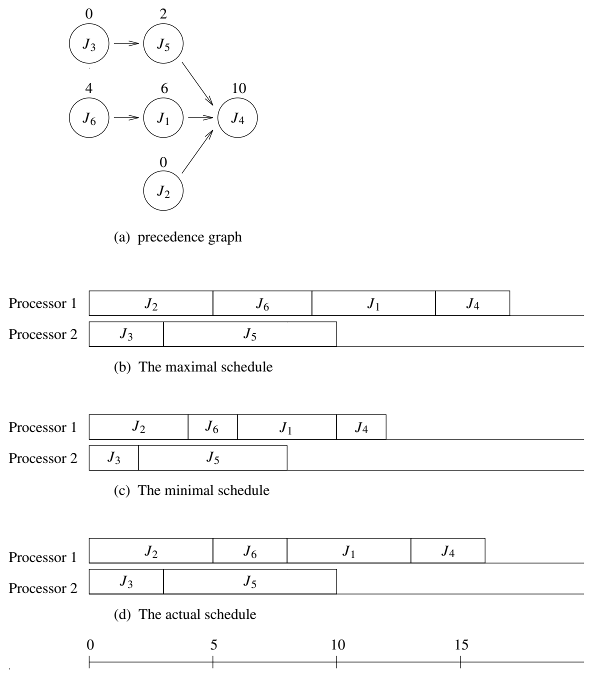

Fig. 4. Predictable execution of P/M/F jobs in in-trees

Proof. We need to consider two cases, when J_i has no predecessors and when J_i has predecessors.

Suppose that J_i has no predecessors. To show that its completion time is predictable when the assumption of the lemma is true, we suppose for the moment that $F(J_i) > F^+(J_i)$. For this condition to be true, the total amount of processor time demanded in the interval $[S^+(J_i), F^+(J_i)]$ by all the jobs with higher priorities than J_i according to $\boldsymbol{A_n}$ must be larger than their total demand of time in this interval according to $\boldsymbol{A_n^+}$; otherwise, since $S(J_i) < S^+(J_i)$, $e_k^+ \geq e_k$ for all k, and the scheduling algorithm is preemptable and migratable, J_i would be able to complete by $F^+(J_i)$. Therefore, there must be at least one job J_h with a higher priority than J_i which is ready in $[S^+(J_i), F^+(J_i)]$ according to $\boldsymbol{A_n}$ but is not ready in this interval according to $\boldsymbol{A_n^+}$. (J_h is not in the subset of all chains each of which contains higher-priority jobs that are ready in $[S^+(J_i), F^+(J_i)]$ according to $\boldsymbol{A_n^+}$.) This job J_h must have a predecessor that has a priority lower than J_i and is not completed before $F^+(J_i)$ in $\boldsymbol{A_n^+}$ but is completed before $F^+(J_i)$ in $\boldsymbol{A_n}$. However, J_h cannot have a priority higher than J_i because every job has a higher priority than its successors. We therefore have a contradiction. The supposition $F(J_i) > F^+(J_i)$ cannot be true. Similarly, we can prove that $F^-(J_i) \leq F(J_i)$.

We prove that the completion time of a job J_i that has predecessors is predictable by induction. The case above gives us the basis. Suppose that the completion times of all of its predecessors are predictable. The proof of that the completion time of J_i is also predictable follows straightforwardly from the assumption of the lemma, the induction hypothesis and the proof for the case where the job has no predecessor. □

Theorem 3. *When the precedence graph of a set of P/M/F jobs is a set of chains and every job has a priority higher than its immediate successor, the start time of every job is predictable, that is,* $S^-(J_i) \leq S(J_i) \leq S^+(J_i)$.

Proof. Let J_k be the job that starts earliest among all the jobs in $\boldsymbol{A_n^+}$. Clearly, J_k starts at its release time; it has the earliest release time among all jobs that have no predecessors. $S^-(J_k) \leq S(J_k) \leq S^+(J_k)$. We now prove that $S(J_i) \leq S^+(J_i)$ for any J_i by contradiction, assuming that all the jobs whose start times are before $S^+(J_i)$ in $\boldsymbol{A_n^+}$ have predictable start times.

Suppose that $S(J_i) > S^+(J_i)$. According to $\boldsymbol{A_n^+}$, at $S^+(J_i)$, a processor is available to execute J_i. In other words, at $S^+(J_i)$, at most $(m-1)$ of all the higher-priority jobs that become ready at or earlier than $S^+(J_i)$ are not yet completed and hence are ready. However, according to $\boldsymbol{A_n}$, at $S^+(J_i)$, at least m higher-priority jobs are ready. Therefore, there must be a job J_h with a priority higher than J_i which is ready at $S^+(J_i)$ according to $\boldsymbol{A_n}$ but is not ready at $S^+(J_i)$ according to $\boldsymbol{A_n^+}$. We need to consider two cases: J_h is not a predecessor of J_i and J_h is a predecessor of J_i.

Because the scheduling algorithm is preemptable and migratable, if J_h is not a predecessor of J_i, it must have a predecessor J_p that has a priority lower than J_i and is not completed before $S^+(J_i)$ in $\boldsymbol{A_n^+}$ but is completed at or before $S^+(J_i)$ in $\boldsymbol{A_n}$. Because every job has a priority higher than its successors, J_h, which is a successor of J_p, cannot have a priority higher than J_i. This fact contradicts that J_h has a priority higher than J_i.

It is possible that J_h is a predecessor of J_i. At $S^+(J_i)$, it is completed according to $\boldsymbol{A_n^+}$ but is not completed according to $\boldsymbol{A_n}$. However, this is not possible because of the induction hypothesis and Lemma 4.

Similarly, $S^-(J_i) \leq S(J_i)$ can be proved. □

Corollary 4. *When the precedence graph of a set of P/M/F jobs is a set of chains and every job has a priority higher than its immediate successor, the completion time of every job is predictable, that is,* $F^-(J_i) \leq F(J_i) \leq F^+(J_i)$.

Proof. The proof of this corollary follows straightforwardly from Theorem 3 and the proof of Lemma 4. □

Similarly, in the case where the precedence graph of jobs is a forest of in-trees, when every job has a priority higher than its immediate successor, we have the following corollaries. Their proofs follow from those of Theorem 3 and Corollary 4 and hence are omitted.

Corollary 5. *When the precedence graph of a set of P/M/F jobs is an in-tree forest and every job has a priority higher than its immediate successor, the start time of every job is predictable, that is,* $S^-(J_i) \leq S(J_i) \leq S^+(J_i)$.

Corollary 6. *When the precedence graph of a set of P/M/F jobs is an in-tree forest and every job has a priority higher than its immediate successor, the completion time of every job is predictable, that is,* $F^-(J_i) \leq F(J_i) \leq F^+(J_i)$.

4.2 Arbitrary Precedence Case

As stated in Section 2, in general, when we want to bound the completion time of a job J_i, we need to consider all the jobs in its interfering subset $\boldsymbol{X_n^i}$. Again, $\boldsymbol{X_n^i}$ is a subset of $\boldsymbol{J_n}$ containing jobs that are predecessors of J_i, jobs that have higher priorities than J_i and predecessors of J_i, and jobs that are predecessors of these higher-priority jobs. However, in the given interfering set $\boldsymbol{X_n^i}$ and for the given scheduling algorithm, there may be some lower-priority jobs that can never complete before the completion of J_i. We want to identify these jobs and their successors, because we do not need to consider them when computing an upper bound of $F(J_i)$. The following lemma allows us to do so. It is stated in terms of $bs(J_i)$, the start time of J_i according to the schedule of $\boldsymbol{J_n^-}$ on an infinite number of processors. $bs(J_i)$ is equal to $max(r_i, max\{bs(J_p) + e_p^-\})$ where J_p is an immediate predecessor of J_i. $bs(J_i)$ is the *best possible start time* of J_i, and no job J_i can start before $bs(J_i)$.

Lemma 5. *When J_i has no predecessor, J_i completes no later than any lower-priority job J_k according to the actual schedule $\boldsymbol{A_n}$ if $r_i \leq bs(J_k)$ and $e_i^+ \leq e_k^-$.*

Proof. If $r_i \leq bs(J_k)$, J_i starts no later than J_k in the actual schedule since the scheduling algorithm is priority-driven. Even when J_k can start before J_i completes in the actual schedule, J_i completes no later than J_k since $e_i^+ \leq e_k^-$ and J_k has a priority lower than J_i. □

The algorithm, called Algorithm $\mathcal{DPMF}$, described by the pseudo code in Figure 5, is based on this lemma. In this description, we use the notation $\boldsymbol{C_i^{(1)}}$, which is the subset of the interfering set $\boldsymbol{X_n^i}$ containing every job J_k in $\boldsymbol{X_n^i}$ that has a lower priority than J_i and parameters $bs(J_k)$ and e_k^- larger than or equal to r_i and e_i^+, respectively. Because of Lemma 5, when J_i has no predecessor, we do not need to consider the jobs in $\boldsymbol{C_i^{(1)}}$ and their successors for computing an upper bound of the completion time of J_i.

Algorithm $\mathcal{DPMF}$ computes upper bounds of the completion times of preemptable and migratable jobs by performing three steps. Its time complexity is $O(n^4)$. Its first step finds the jobs in $\boldsymbol{X_n^i}$ that cannot postpone the completion of J_i in the actual schedule. Specifically, when J_i has no predecessor, Step 1 finds the subset $\boldsymbol{C_i^{(1)}}$ of $\boldsymbol{X_n^i}$ and eliminates them from further consideration.

```
Algorithm $\mathcal{DPMF}$:

Input: Parameters of the given job set $\boldsymbol{J}$ and the precedence graph $G$
Output: Upper bounds of completion times of $J_i$ for $i = 1, 2, \cdots, n$

Begin
   Step 1. for $i = 1$ to $n$ do
               $\boldsymbol{C_i} = \boldsymbol{X_n^i}$.
               if $J_i$ has no predecessor in $G$,
                 { find the subset $\boldsymbol{C_i^{(1)}}$ of $\boldsymbol{X_n^i}$ in which every job $J_k$ has a priority
                 lower than $J_i$ and parameters $bs(J_k)$ and $e_k^-$ larger than or equal
                 to $r_i$ and $e_i^+$, respectively.
                    $\boldsymbol{C_i} = \boldsymbol{C_i} - \boldsymbol{C_i^{(1)}} - \{$successors of jobs in $\boldsymbol{C_i^{(1)}}\}$. }
           endfor
   Step 2. Construct a new dependence graph $G'_i$ for each $J_i$ for $i = 1, 2, \cdots, n$:
           for $i = 1$ to $n$ do
               $G'_i$ = subgraph of $G$ induced by $\boldsymbol{X_n^i}$
               for $l = 1$ to $n$ do
                   if $J_l$ has no predecessor in $G'_i$, flag[$l$] = 1
                      else flag[$l$] = 0
               endfor
               for each job $J_p$ in $G'_i$ such that flag[$p$] is 0 and flag[$l$] is 1 for every
                   predecessor $J_l$ of $J_p$ do
                   Insert new dependencies into $G'_i$ from each job $J_k$ in $\boldsymbol{C_p}$ to $J_p$
                     if $J_k$ has a higher priority than $J_p$ and there is no dependency
                     from $J_k$ to $J_p$ or any predecessor of $J_p$ in $G'_i$.
                   flag[$p$] = 1
               endfor
               Prune the jobs not in $\boldsymbol{C_i} \cup \{J_i\}$ from $G'_i$.
           endfor
   Step 3. for $i = 1$ to $n$ do
               Schedule the remaining jobs in $G'_i$ with their maximum execution
               times. An upper bound of $F(J_i)$ = the completion time of $J_i$
               in the generated schedule.
           endfor
End
```

Fig. 5. Pseudo code of Algorithm $\mathcal{DPMF}$

Let $\boldsymbol{C_i}$ be the set $\boldsymbol{X_n^i} - \boldsymbol{C_i^{(1)}} - \{$successors of jobs in $\boldsymbol{C_i^{(1)}}\}$, if J_i has no predecessor, or $\boldsymbol{X_n^i}$ if J_i has a predecessor. When trying to bound the completion time of J_i, Step 2 of Algorithm $\mathcal{DPMF}$ inserts pseudo dependencies to the jobs in $\boldsymbol{C_i}$ and J_i as follows: we use G'_i to denote a new precedence graph after inserting pseudo dependencies. Initially, G'_i is the subgraph of G induced by $\boldsymbol{X_n^i}$. In order to construct a new precedence graph G'_i, for every job J_p in G'_i, a new edge from each job J_k in $\boldsymbol{C_p}$ to J_p is inserted if J_k has a higher priority than J_p and G'_i does not yet contain an edge from J_k to J_p or to any predecessor of J_p. Then Step 2 prunes the jobs not in $\boldsymbol{C_i} \cup \{J_i\}$ from G'_i. In Step 3, the completion time of the job J_i is bounded by J_i's completion time according to the schedule of the jobs that are in G'_i assuming every job has its maximum execution time, and their precedence relations are given by G'_i.

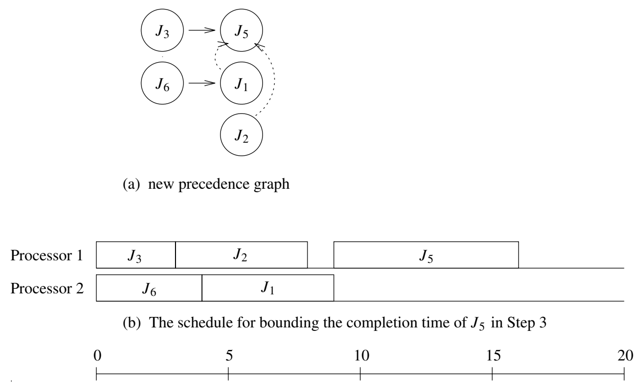

Fig. 6. An example illustrating Algorithm $\mathcal{DPMF}$ for P/M/F jobs

Theorem 4 allows us to conclude that if J_i can complete by the deadline d_i in the schedule of the jobs in G'_i generated by Algorithm $\mathcal{DPMF}$, then J_i always completes by d_i. Figure 6 illustrates how Algorithm $\mathcal{DPMF}$ bounds the completion time of J_5 in Figure 3. Figure 6 (a) is the new precedence graph G'_5 where the dashed edges are the edges added in Step 2. Figure 6 (b) is the schedule of the five jobs with their maximum execution times and precedence graph G'_5. According to this schedule, J_5 completes at time 16. Thus, 16 is the upper bound of the completion time of J_5. In this example, this bound is only one unit of time higher than the actual completion time of J_5, 15, given by Figure 3. In general, the bounds obtained by this algorithm are loose, and can be much looser than indicated by this example.

Theorem 4. *The completion time $F(J_i)$ of J_i is no later than the completion time of J_i according to the schedule generated by Algorithm $\mathcal{DPMF}$.*

Proof. From Lemma 5, when J_i has no predecessor, the lower-priority jobs in $\boldsymbol{C_i^{(1)}}$ do not have a chance to complete before $F(J_i)$ in the actual schedule. Therefore, if J_i has no predecessor, we do not need to consider the jobs in $\boldsymbol{C_i^{(1)}}$ and their successors when computing an upper bound of the completion time of J_i. This justifies Step 1. After pruning, $\boldsymbol{C_i}$ contains all the jobs that can possibly start before the completion of J_i in the actual schedule. Algorithm $\mathcal{DPMF}$ inserts new precedence relations into the precedence graph G'_i so that every job in $\boldsymbol{C_i}$ completes before J_i starts. Therefore the theorem follows. $\square$

5 Conclusion

This paper is concerned with validating timing constraints of dependent jobs that have arbitrary timing constraints and variable execution times and are scheduled on processors dynamically in a priority-driven manner. We present conditions under which dependent jobs execute in a predictable manner, *i.e.*, the completion times of jobs are no later when the execution times of some jobs decrease. We also present algorithms and bounds with which the latest completion times of all jobs can be determined.

In the literature, scheduling jobs with precedence constraints has been well studied. In most systems of practical interests, the precedence graphs are restricted to structures no more complicated than trees and forests. Often, all jobs have identical release times. Execution of jobs in many such systems is predictable. Consequently, it is possible to find tight bounds to their completion times with our results. For general precedence structures, however, our results are less encouraging. Even in the case where jobs have identical release times and are preemptable and migratable, the bounds on a job set with a general precedence graph are much looser than the bounds on a job set with a forest of in-tree precedence graph.

The results presented here, as well as the results on independent jobs in homogeneous systems and heterogeneous systems in [9,12], make up the theoretical basis for comprehensive validation methods in dynamic distributed real-time systems. Methods for bounding the worst-case completion times of jobs that share resources and have precedence constraints are not yet available. Furthermore, release time jitter is often unavoidable in real-life systems. The release time jitter problem needs to be further investigated.

Acknowledgement. This work is supported by the University Research Program of the Ministry of Information & Communication in republic of Korea (2001-076-3) and BK(Grant 991031001).

References

1. Graham, R.: Bounds on multiprocessing timing anomalies. SIAM Journal of Applied Mathematics. **17(2)** (1969) 416–429
2. Sha, L., Sathaye, S.: A systematic approach to designing distributed real-time systems. IEEE Computer. **26(9)** (1993) 68–78
3. Rajkumar, L. Sha, L, Lehoczky, J.: Real-time synchronization protocols for multiprocessors. Proc. of IEEE 9th Real-Time Systems Symposium. (1988) 259–269
4. Sun, J., Bettati, R., Liu, J.: An end-to-end approach to schedule tasks with shared resources in multiprocessor systems. Proc. of IEEE 11th Workshop on Real-Time Operating Systems and Software. (1994) 18–22
5. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery. **20(1)** (1973) 46–61
6. Lehoczky, J., Sha, L., Ding, Y.: The rate monotone scheduling algorithm: Exact characterization and average case behavior. Proc. of IEEE 10th Real-Time Systems Symposium. (1989) 166–171
7. Han, C., Tyan, H.: A better polynomial-time schedulability test for teal-time fixed-priority scheduling algorithms. Proc. of IEEE 18th Real-Time Systems Symposium. (1997) 36–45

8. Palencia, J., Harbour, M.: Schedulability analysis for tasks with static and dynamic offsets. Proc. of IEEE 19th Real-Time Systems Symposium. (1998) 26–37
9. Liu, J., Ha, R.: Methods for validating real-time constraints. Journal of Systems and Software. **30** (1995) 85–98
10. Sun, J., Liu, J.: Bounding completion times of jobs with arbitrary release times and variable execution times. Proc. of IEEE 17th Real-Time Systems Symposium. (1996) 2–12
11. Meyer, M., Wong-Toi, H.: Schedulability analysis of acyclic processes. Proc. of IEEE 19th Real-Time Systems Symposium. (1998) 274–283
12. Ha, R., Cha, H., Liu, J.: Validating real-time constraints in heterogeneous multiprocessor and distributed systems. Journal of Systems Integration. **9(3)** (1999) 207–222