Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems *

Rhan Ha and Jane W. S. Liu
{rhanha, janeliu}@cs.uiuc.edu
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Abstract

In multiprocessor and distributed real-time systems, scheduling jobs dynamically on processors is likely to achieve better performance. However, analytical and efficient validation methods to determine whether all the timing constraints are met do not exist for systems using modern dynamic scheduling strategies, and exhaustive simulation and testing are unreliable and expensive. This paper describes several worst-case bounds and efficient algorithms for validating systems in which jobs have arbitrary timing constraints and variable execution times and are scheduled on processors dynamically in a priority-driven manner. The special cases of the validation problem considered here are concerned with independent jobs that are (1) *preemptable and migratable, or* (2) *preemptable and non-migratable, or* (3) *nonpreemptable.*

1 Introduction

In recent years, many real-time load balancing and scheduling algorithms that dynamically dispatch and schedule jobs on processors have been developed. Examples include algorithms described in [1-3]. These algorithms are likely to make better use of the processors and achieve a higher responsiveness than the traditional approach to scheduling real-time 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 [4-9]. Here, the term *job* refers to a basic unit of work to be scheduled. A job may be the formatting of a text file, the processing of a query, the transmission of a message, etc. To execute, it requires a CPU, a database server, or a data link, respectively; they are all modelled abstractly as processors. (In queuing theory literature, a processor is called a server.) A real-time job J_i has a *release time* r_i and a *deadline* d_i. The release time is the point in time after which the job can begin execution whenever its constraints due to data and control dependencies are met. The deadline is the point in time by which the job is required to complete. We say that a job meets its timing constraint if it executes only at or after its release time and completes by its deadline.

The validation of a system containing real-time jobs involves validating not only that the results produced by all jobs are functionally correct but also that all jobs meet their timing constraints. By statically binding jobs to processors, the traditional approach allows timing constraints to be validated using analytical methods or by efficient algorithms [4, 10]. In contrast, such validation methods do not exist for systems using modern dynamic scheduling strategies, and exhaustive simulation and testing are unreliable and expensive. Until reliable and efficient validation methods become available, the modern approaches to scheduling can not be used in real-life, safety-critical systems.

To illustrate the difficulties we are likely to encounter, we consider a replicated database system. Time-critical queries arriving at the communication server are queued and dispatched to identical database servers in a greedy manner in an attempt to produce all responses on time. Specifically, in the simple system shown in Figure 1, there are two database servers; they are modelled abstractly as processors P_1 and P_2. There are 6 queries; each is a job. These jobs are *independent*, that is, they can execute in any order. Jobs are assigned priorities, and those ready for execution are placed in a common queue ordered by their priorities. Scheduling decisions are made when jobs are released and completed. At each scheduling decision time, the priority of the job at the head of the queue is compared with the priorities of the jobs executing on the processors. If the priority of the job is higher

*This work has been partially supported by ONR Contract Nos. N00014-89-J-1181 and N000-92-J-1815 and NASA Grant No. NAG 1-1613.

job	r_i	d_i	$[e_i^-, e_i^+]$
J_1	0	10	[5,5]
J_2	0	10	[2,6]
J_3	4	15	[8,8]
J_4	0	20	[10,10]
J_5	5	200	[100,100]
J_6	7	25	[2,2]

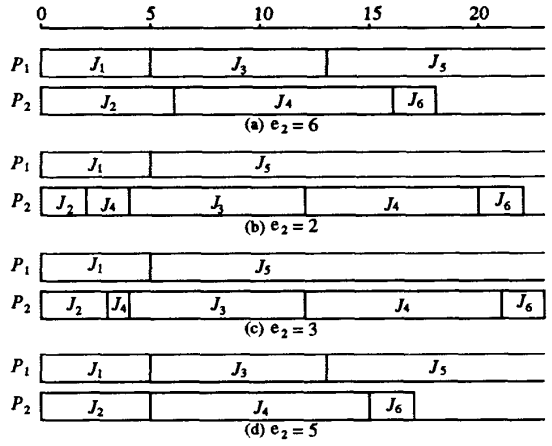

Figure 1: An example illustrating scheduling anomalies

than the lowest priority of the executing jobs, or if a processor is idle, the job is dispatched to the processor executing the lowest-priority job or to the idle processor. Because it is costly to migrate jobs among processors, once a job is dispatched, it is never migrated. At any time, each processor executes the job with the highest-priority among all the jobs dispatched to execute on it.

The table in Figure 1 lists the release times and deadlines of the jobs in our example. The amount of time a job J_i requires to execute to completion on a processor by itself is called its *execution time* and is denoted by e_i. J_2's execution time can be any value in the range [2,6]. The execution times of all the other jobs are fixed. The priority order is J_1, J_2, J_3, J_4, J_5 and J_6 with J_1 having the highest priority and J_6 having the lowest priority. The schedules in Figure 1 (a) and (b) arise when the execution time of J_2 has the maximum value 6 and the minimum value 2, respectively. Looking at these two schedules, we would conclude that all jobs always complete by their deadlines and variations in their completion times are small. Unfortunately, this conclusion is false as indicated by the schedule in Figure 1 (c). We see that in the worst case, J_4 completes at time 21 and misses its deadline; this occurs when J_2's execution time is 3. In the best case, as shown in Figure 1 (d), J_4 completes at 15 when J_2's execution time is 5.

The phenomenon illustrated by Figure 1 is known as a scheduling anomaly: an unexpected behavior exhibited by a large class of scheduling algorithms known as *priority-driven algorithms*. A distinguishing feature of these algorithms is that they do not intentionally leave any processor idle. Almost all event-driven scheduling algorithms, such as FIFO, LIFO, shortest-processing-time-first, earliest-deadline first, and rate-monotonic algorithms, are priority-driven. Scheduling anomalies can occur when execution times, resource requirements and release times vary, and variations in the job parameters are unavoidable. Because of these anomalies, we must try an exponential number of combinations of job parameters in order to ensure full coverage in simulation and testing.

In this paper, we focus on analytical methods and efficient algorithms for validating multiprocessor and distributed systems in which ready jobs are dispatched and scheduled on available processors in a priority-driven manner. The problem addressed here, often known as the *schedulability analysis problem*, can be stated in general as follows: given a set of jobs, a set of processors available to execute the jobs, and a scheduling algorithm to allocate processors to jobs, determine whether all the jobs always meet their deadlines. A variant of this problem that has been studied extensively is concerned with periodic tasks. (A periodic task is an infinite sequence of jobs whose release times occur periodically.) There are now worst-case bounds and efficient algorithms which allow us to determine whether all jobs can meet their deadlines when the periodic tasks are assigned and bound to processors statically [4-10]. These methods can be easily generalized to deal with jobs with arbitrary release times. Many worst-case performance bounds for priority-driven multiprocessor scheduling algorithms can be found in literature on classical scheduling theory [11]. These bounds are typically too pessimistic to be of practical use. Worse, they cannot be extended to the case where jobs have arbitrary release times and deadlines.

The rest of the paper is organized as follows. Section 2 gives the formal definition of the schedulability analysis problem. In this paper, we focus on the case when jobs are independent and the system is homogeneous, that is, the processors are identical. Sections 3, 4 and 5 present several efficient algorithms for bounding the response times of jobs. Section 6 is a summary. It discusses future work as well as how the results here

can be applied when the system contains heterogeneous processors. Due to space limitation only proofs that offer some insights into the conditions that lead to predictable or unpredictable executions are included here; other proofs can be found in [12].

2 Definitions and Notation

We characterize the workload to be scheduled and analyzed as a set $J = \{J_1, J_2, \ldots, J_n\}$ of jobs. The jobs in J are independent and can execute in any order. Each job J_i is defined by its release time r_i, its deadline d_i and its execution time e_i. (These parameters are rational numbers; they were defined earlier in Section 1.) We say that J_i has a jittered release time when r_i can have any value in the range $[r_i^-, r_i^+]$ where r_i^- and r_i^+ are the *earliest release time* and the *latest release time* of J_i, respectively. Without loss of generality, we assume that no job is released before $t = 0$. We say that the jobs have fixed release times, or there is no jitter, when $r_i^- = r_i = r_i^+$ and that they have identical, or zero, release times when $r_i^- = r_i^+ = 0$ for all i. Similarly, the execution time of J_i is in the range $[e_i^-, e_i^+]$ and therefore can be as small as its *minimum execution time* e_i^- and as large as its *maximum execution time* e_i^+. We assume that the maximum and minimum values of these parameters of all jobs are known before the execution of any job begins, but their actual release times and execution times are not known *a priori*.

The underlying system contains m identical processors. The scheduling algorithm is priority-driven. Jobs are assigned fixed priorities. We assume that the system is sufficiently tightly-coupled so that it is possible for the scheduler(s) of all processors to maintain a common priority list; all jobs ready for execution are ordered according to this list. If there is more than one scheduler, each scheduler knows the current state of the global priority queue of ready jobs. The schedulers cooperatively behave like a centralized scheduler.

We confine our attention to the class of algorithms that do not make use of the information on the actual execution times when assigning priorities to jobs. 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 use the list $(J_1, J_2, \ldots, J_n)$ in decreasing priority order throughout this paper. In other words, we always index the jobs so that J_i has a higher priority than J_j according to the given scheduling algorithm if $i < j$. Let $\boldsymbol{J}_i = \{J_1, J_2, \ldots, J_i\}$ denote the subset of jobs with priorities equal to or higher than the priority of J_i.

We use $\boldsymbol{J}_i^+$ to denote the set $\{J_1^+, J_2^+, \ldots, J_i^+\}$ of jobs in which every job has its maximum execution time. Similarly, $\boldsymbol{J}_i^-$ denotes the set $\{J_1^-, J_2^-, \ldots, J_i^-\}$ in which every job has its minimum execution time. We refer to the schedule of $\boldsymbol{J}_i$ produced by the given algorithm as the *actual schedule* $\boldsymbol{A}_i$ and the schedule of $\boldsymbol{J}_i^+$ (or $\boldsymbol{J}_i^-$) produced by the same algorithm as the *maximal* (or the *minimal*) *schedule* $\boldsymbol{A}_i^+$ (or $\boldsymbol{A}_i^-$) of $\boldsymbol{J}_i$.

Let $S(J_i)$ be the instant of time at which the execution of J_i begins according to the actual schedule $\boldsymbol{A}_n$. $S(J_i)$ is the (actual) *start time* of J_i. Let $S^+(J_i)$ and $S^-(J_i)$ be the *observable start times* of J_i in the schedules $\boldsymbol{A}_n^+$ and $\boldsymbol{A}_n^-$, respectively. ($S^+(J_i)$ and $S^-(J_i)$ are observable because they can easily be found by constructing the maximal and minimal schedules and observing when J_i starts according to these schedules.) 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 instant at which J_i completes execution according to the actual schedule $\boldsymbol{A}_n$. $F(J_i)$ is the *completion time* of J_i. Let $F^+(J_i)$ and $F^-(J_i)$ be the *observable 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 minimum release time r_i^- of J_i gives J_i's worst-case response time. J_i meets its deadline if $F^+(J_i) \leq d_i$.

Again, the objective of schedulability analysis is to determine, analytically or by using an efficient algorithm, whether every job can meet its deadline. We will consider the following cases of the schedulability analysis problem in Sections 3, 4 and 5:

(1) preemptable and migratable: In this case, a 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.

(2) preemptable and nonmigratable: As in case (1), each job can begin its execution on any processor and is preemptable. However, it is constrained to execute to completion on the same processor. Figure 1 gives an example of this case.

(3) nonpreemptable: Each job can be scheduled on any processor. Some or all of the jobs are nonpreemptable.

Preemptive scheduling algorithms that may migrate jobs among processors are not practical for distributed

systems; case (1) is included here for the sake of completeness.

We will refer to these cases by three capital letters separated by /. The first letter denotes preemptability. It can be either P, for preemptable, or N, for nonpreemptable. The second letter defines the migratability of jobs. It can be either M, for migratable, or N, for nonmigratable. The third letter describes the release time characteristics. It can be either Z, for zero release times, or F, for fixed arbitrary release times, or J, for jittered release times. For example, by P/M/F jobs, we mean jobs that are preemptable and migratable and have fixed, but arbitrary, release times. N/N/Z jobs are nonpreemptable (and therefore not migratable) and have zero, or identical, release times.

3 Preemptable and Migratable Case

It is easy to find the worst-case and best-case response times of independent jobs when they are scheduled preemptively and may be migrated among processors. When jobs have fixed release times, we can find the worst-case (or best-case) response time of a job J_i in a set $\boldsymbol{J}_n$ of independent P/M/F jobs by applying the given scheduling algorithm on the set $\boldsymbol{J}_n^+$ (or $\boldsymbol{J}_n^-$) where all jobs have their maximum (or minimum) execution times. The response times of J_i according to the resultant schedule $\boldsymbol{A}_n^+$ (or $\boldsymbol{A}_n^-$) is its largest (or smallest) possible response time. We are sure that J_i always meets its deadline if it meets its deadline in the maximal schedule $\boldsymbol{A}_n^+$. The following theorem and corollary ensure us that this strategy is correct.

Theorem 3.1 *The start time of every job in a set of independent P/M/F jobs is predictable, that is,* $S^-(J_i) \leq S(J_i) \leq S^+(J_i)$.

Proof : Clearly, $S(J_1) \leq S^+(J_1)$ is true for the highest-priority job J_1. Assuming that $S(J_k) \leq S^+(J_k)$ for $k = 1, 2, \cdots, i-1$, we now prove $S(J_i) \leq S^+(J_i)$ by contradiction.

Suppose that $S(J_i) > S^+(J_i)$. Because the scheduling algorithm is priority driven, every job whose release time is at or earlier than $S^+(J_i)$ and whose priority is higher than J_i has started by $S^+(J_i)$ according to the maximal schedule $\boldsymbol{A}_n^+$. From the induction hypothesis, we can conclude that every such job has started by $S^+(J_i)$ in the actual schedule $\boldsymbol{A}_n$. Because $e_k \leq e_k^+$ for all k, in $(0,\ S^+(J_i))$, the total time demand of all jobs with priorities higher than J_i in the maximal schedule $\boldsymbol{A}_n^+$ is larger than the total time demand of these jobs in the actual schedule $\boldsymbol{A}_n$. In $\boldsymbol{A}_n^+$, a processor is available at $S^+(J_i)$ for J_i to start; a processor must also be available in $\boldsymbol{A}_n$ at or before $S^+(J_i)$ on which J_i or a lower-priority job can be scheduled. Therefore, the start time $S(J_i)$ of J_i in the actual schedule $\boldsymbol{A}_n$ being later than the observable start time $S^+(J_i)$ implies that in $\boldsymbol{A}_n$ either some job(s) whose priority is lower than J_i is scheduled in $(S^+(J_i),\ S(J_i))$ or at least one processor is left idle in this interval. This contradicts the fact that $\boldsymbol{A}_n$ is a priority-driven schedule.

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

Corollary 3.1 *The completion time of every job in a set of independent P/M/F jobs is predictable, that is,* $F^-(J_i) \leq F(J_i) \leq F^+(J_i)$.

When there is jitter in release times, whether J_i is schedulable depends not only on its own actual release time but also on the actual release times of all the higher-priority jobs. As an illustrative example, we consider two P/M/J jobs, J_1 and J_2, that are to be scheduled on a processor. Suppose that J_1's release time is in [0,5] and its execution time is 5. J_2's release time, execution time, and deadline are 3, 5, and 12, respectively. If J_1 is released early, for example, in [0,2], J_2 can meet its deadline. However, if J_1 is released after 2, then J_2 cannot complete by its deadline. On the other hand, if J_1's release time is in [5,10], J_2 can meet its deadline when J_1 is released late, in [8,10] for example. However, if J_1 is released before 8, then J_2 cannot complete by its deadline. This example illustrates that in trying to find the worst-case completion time of a job, we cannot simply choose the earliest or the latest release times of higher-priority jobs.

Algorithm $\mathcal{IPMJ}$ is based on this observation. It finds bounds for start times and completion times of independent P/M/J jobs by considering one job at a time, from the job with the highest priority to the job with the lowest priority. In order to find the worst-case completion time of J_i, it first transforms J_i and the jobs that have priorities higher than J_i into jobs with fixed release times as follows. Let K_k denote the job transformed from J_k and $\boldsymbol{K}_i$ denote the set of transformed jobs $\{K_1, K_2, \ldots, K_i\}$. The algorithm has three steps. Step 1 computes the parameters of the transformed job K_i from the parameters of J_i. Specifically, K_i's execution time is equal to J_i's maximum execution time, e_i^+, plus the length $(r_i^+ - r_i^-)$ of its jitter interval. K_i's release time is J_i's earliest release time, r_i^-, and its deadline is d_i. Step 2 computes the parameters of K_k for each $k = 1, \cdots, i-1$. K_k's execution time is e_k^+, that is, the maximum execution time of J_k. Its deadline is d_k. K_k's release time is chosen according to the following rule:

(1) if $r_k^- < r_i^- < r_k^+$, K_k's release time is r_i^-,

(2) if $r_k^+ \leq r_i^-$, K_k's release time is r_k^+ (*i.e.*, as late as possible), and

(3) if $r_k^- \geq r_i^-$, K_k's release time is r_k^- (*i.e.*, as early as possible).

In other words, K_k's release time is chosen among r_i^-, r_k^+ and r_k^- so that the interval between the release time and deadline of K_i overlaps with the interval between the release time and deadline of K_k as much as possible. Step 3 schedules $\boldsymbol{K}_i$ according to the given preemptable, migratable, priority-driven algorithm. An upper bound of the completion time $F(J_i)$ of J_i is equal to the completion time of K_i in the resultant schedule $\boldsymbol{A}_i'$.

Because $\boldsymbol{K}_{i-1}$ is not a subset of $\boldsymbol{K}_i$, $\boldsymbol{K}_i$ needs to be constructed from scratch for every job J_i. Consequently, the complexity of Algorithm $\mathcal{IPMJ}$ is $O(n^2)$. The following theorem allows us to conclude that if K_i completes by the deadline d_i of J_i in the schedule $\boldsymbol{A}_i'$ generated by the $\mathcal{IPMJ}$ algorithm, then J_i always meets its deadline for all possible combinations of release times and execution times.

Theorem 3.2 *The completion time $F(J_i)$ of J_i is no later than the completion time of the transformed job K_i in the observable schedule $\boldsymbol{A}_i'$ of $\boldsymbol{K}_i$ generated by Algorithm $\mathcal{IPMJ}$.*

4 Preemptable, Nonmigratable, Fixed Release Time Case

It is often too costly to migrate jobs among processors in a distributed system. Consequently, jobs are not migrated.

Conditions for Predictable Execution

In the special case when independent jobs have zero, or identical, release times and jobs have fixed priorities, preemption and migration can never occur. Therefore, it does not matter whether preemption and migration are allowed or not. The following theorem follows straightforwardly from this observation.

Theorem 4.1 *The execution of the independent N/N/Z (P/N/Z) jobs is predictable.*

As illustrated by the example in Figure 1, when jobs have arbitrary release times and are not migratable, their execution behavior is no longer predictable. In Figure 1, the completion time of job J_4 is not predictable. The start time of J_6 is 16, 20, 21, and 15 when the execution time of J_2 is 6, 2, 3, and 5, respectively, and therefore is also unpredictable.

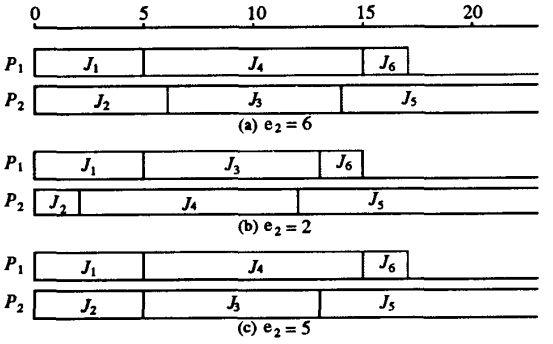

Figure 2: Predictable execution of P/N/F jobs scheduled on the FIFO basis

While the execution of independent P/N/F jobs is not predictable for arbitrary priority assignments, it is predictable when the jobs are scheduled in the order in which they are released. Figure 2 illustrates this fact which is stated formally in the following theorem. The job set in this figure is the same as the one in Figure 1 except that the priority list is $(J_1, J_2, J_4, J_3, J_5, J_6)$. J_2's execution time in parts (a), (b) and (c) is 6, 2, and 5, respectively.

Theorem 4.2 *When the priorities of independent P/N/F jobs are assigned on the FIFO basis (that is, the earlier the release time, the higher the priority), the execution of the jobs is predictable.*

A General Upper Bound

When independent P/N/F jobs are not scheduled on the FIFO basis and their execution is no longer predictable, we can bound their start times and completion times according to the following theorem. We note that when we want to determine whether J_i is schedulable, there is no need to consider $J_{i+1}, J_{i+2}, \ldots, J_n$, since jobs are preemptable and independent. Therefore, we can confine our attention to $\boldsymbol{J}_i$. The theorem is stated in terms of the set $\boldsymbol{D}_i$; $\boldsymbol{D}_i$ is a subset of $\boldsymbol{J}_i$ in which each job J_k is released after some job in $\boldsymbol{J}_i$ with a priority lower than itself (that is, J_k) and is not scheduled on the same processor as J_i to start and complete before J_i in the maximal schedule. In the example in Figure 1, $\boldsymbol{D}_1$, $\boldsymbol{D}_2$, $\boldsymbol{D}_3$ and $\boldsymbol{D}_5$ are null. $\boldsymbol{D}_4$ and $\boldsymbol{D}_6$ are $\{J_3\}$.

Theorem 4.3 $S(J_i) \leq S^+(J_i) + \sum_{J_k \in D_i} e_k^+$, *and*

$$F(J_i) \leq F^+(J_i) + \left(\sum_{J_k \in D_i} e_k^+ \right) - (e_i^+ - e_i).$$

The worst-case bound of completion time given by this theorem is sometimes too pessimistic to be useful

in practice. For example, this theorem tells us that the completion times of the jobs in Figure 1 are no greater than 5, 6, 13, 24, 113, and 26, respectively. The bounds for $F(J_4)$ and $F(J_6)$ are pessimistic. As another example, we consider a simple system containing m independent jobs and m identical processors. The release times of the jobs $J_1, J_2, \ldots, J_m$ are such that $r_m < r_{m-1} < \cdots < r_1$. The priority order is $J_1, J_2, \ldots, J_m$ with J_1 having the highest priority. Obviously, every job J_i can be scheduled immediately after its release time and can always complete at or before its observable worst-case completion time $F^+(J_i)$. The bound of the worst-case completion time of J_m computed from Theorem 4.3 can be as large as m times the actual completion time of J_m.

The remainder of this section discusses whether other information provided by the two observable schedules can be used to derive a more accurate prediction of job completion times. Specifically, we consider schedulability tests that begin by examining the sequences in which the jobs start execution according to the minimal and maximal schedules and whether there is preemption in these observable schedules.

A Tight Upper Bound

Let $\wp_i^+(t)$ (or $\wp_i^-(t)$) be the sequence of jobs whose observable start times are at or before t according to the maximal schedule $\boldsymbol{A}_i^+$ (or minimal schedule $\boldsymbol{A}_i^-$) of $\boldsymbol{J}_i$; the jobs in the sequence appear in order of increasing start times. For example, in Figure 1, $\wp_4^+(S^+(J_4))$ is (J_1, J_2, J_3, J_4) and $\wp_4^-(S^-(J_4))$ is (J_1, J_2, J_4). Similarly, let $\wp_i(t)$ be the corresponding sequence of jobs in increasing order of their actual start times according to the actual schedule $\boldsymbol{A}_i$, including all jobs whose actual start times are at or before t. We call $\wp_i(t)$ the actual starting sequence, and $\wp_i^+(t)$ and $\wp_i^-(t)$ the maximal and minimal observable starting sequences according to $\boldsymbol{A}_i^+$ and $\boldsymbol{A}_i^-$, respectively. We say that a sequence X is a *subsequence* of a sequence Y if Y contains X and the elements in both X and Y appear in X and Y in the same order. For example, in Figure 1, $\wp_4^-(S^-(J_4)) = (J_1, J_2, J_4)$ is a subsequence of $\wp_4^+(S^+(J_4)) = (J_1, J_2, J_3, J_4)$. Similarly, $\wp_4^+(S^+(J_3)) = (J_1, J_2, J_3)$ is a subsequence of $\wp_4^-(S^-(J_3)) = (J_1, J_2, J_4, J_3)$.

When we want to determine whether J_i is schedulable, we first examine whether there is preemption in the maximal schedule. In the simpler case, no job in $\boldsymbol{J}_i$ is preempted in the maximal schedule $\boldsymbol{A}_i^+$. Then, we examine whether the two observable starting sequences $\wp_i^+(S^+(J_i))$ and $\wp_i^-(S^-(J_i))$ are identical. If the two sequences are identical, we can conclude that no job in $\boldsymbol{J}_i$ is preempted and the orders in which jobs start execution are the same, according to all the schedules of $\boldsymbol{J}_i$ for all combinations of execution times of jobs in $\boldsymbol{J}_i$. Therefore, the latest completion time of J_i is $F^+(J_i)$. This fact is stated in Theorem 4.4 whose proof requires the following seven lemmas and their corollaries. Because of the limitation in space and the lengthiness of the proofs, we present here only the proof of Lemma 4.4 to provide some insight into why job execution is predictable when the condition stated in the lemma is true. Again, the other proofs can be found in [12].

Lemma 4.1 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$ and $\boldsymbol{A}_i^-$ in the time interval $[0,t)$, and in $\boldsymbol{A}_i^+$, at most $(m-1)$ processors are busy at time t, then in $\boldsymbol{A}_i^-$, at most $(m-1)$ processors are busy at time t.*

Corollary 4.1 *If no job in $\boldsymbol{J}_i$ is preempted in the interval $[0,t)$ according to $\boldsymbol{A}_i^+$ and $\boldsymbol{A}_i$, and in $\boldsymbol{A}_i^+$, at most $(m-1)$ processors are busy at t, then in $\boldsymbol{A}_i$, at most $(m-1)$ processors are busy at t.*

Lemma 4.2 *If no job in $\boldsymbol{J}_{i-1}$ is preempted according to $\boldsymbol{A}_{i-1}^+$ and $\boldsymbol{A}_{i-1}^-$, then $S^-(J_i) \leq S^+(J_i)$.*

Corollary 4.2 *If no job in $\boldsymbol{J}_{i-1}$ is preempted according to $\boldsymbol{A}_{i-1}^+$ and $\boldsymbol{A}_{i-1}$, then $S(J_i) \leq S^+(J_i)$.*

Corollary 4.3 *If no job in $\boldsymbol{J}_{i-1}$ is preempted according to $\boldsymbol{A}_{i-1}$ and $\boldsymbol{A}_{i-1}^-$, then $S^-(J_i) \leq S(J_i)$.*

Lemma 4.3 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$ and $\wp_i^+(S^+(J_i))$ is a subsequence of $\wp_i^-(S^-(J_i))$, then no job in $\boldsymbol{J}_i$ is preempted according to the minimal schedule $\boldsymbol{A}_i^-$.*

Lemma 4.4 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$, and $\wp_i^+(S^+(J_i))$ is a subsequence of $\wp_i^-(S^-(J_i))$, then no job in $\boldsymbol{J}_i$ is preempted according to the actual schedule $\boldsymbol{A}_i$.*

Proof: The lemma is trivially true for the highest-priority job J_1. Assuming that no job in $\boldsymbol{J}_{i-1}$ is preempted according to $\boldsymbol{A}_{i-1}$, we now prove that J_i is not preempted according to $\boldsymbol{A}_i$.

Suppose that J_i is preempted in the actual schedule $\boldsymbol{A}_i$ and J_l is the first job to preempt J_i. Clearly, J_l is not in $\wp_i(S(J_i))$. Moreover, because J_l has a priority higher than J_i and J_l starts after J_i, $S(J_l)$ must be the release time r_l of J_l. J_l cannot start earlier than $S(J_l)$ in any schedule. We need to consider the following two cases:

(1) J_l is in $\wp_i^+(S^+(J_i))$, that is, J_l starts no later than J_i in $\boldsymbol{A}_i^+$. From Lemma 4.3 and the induction hypothesis, no job in $\boldsymbol{J}_{i-1}$ is preempted according to both $\boldsymbol{A}_{i-1}^-$ and $\boldsymbol{A}_{i-1}$. Corollary 4.3 tells us

that $S^-(J_i) \leq S(J_i)$. In other words, J_l is not in $\wp_i^-(S^-(J_i))$. This contradicts the assumption that $\wp_i^+(S^+(J_i))$ is a subsequence of $\wp_i^-(S^-(J_i))$.

(2) J_l is not in $\wp_i^+(S^+(J_i))$, that is, J_i starts before J_l in $\boldsymbol{A}_n^+$. From the induction hypothesis and assumption, no job in $\boldsymbol{J}_{i-1}$ is preempted according to both $\boldsymbol{A}_{i-1}$ and $\boldsymbol{A}_{i-1}^+$. Corollary 4.2 tells us that $S(J_i) \leq S^+(J_i)$.

The facts that J_i is preempted at r_l and that $S(J_i) \leq S^+(J_i)$ tell us that J_i is not completed at r_l in the maximal schedule A_i^+. Nevertheless, J_i is not preempted at r_l in $\boldsymbol{A}_i^+$. We therefore can conclude that among all the jobs that are in $\boldsymbol{J}_i - \{J_l\}$ and have release times at or earlier than r_l, at most $(m-1)$ are not yet completed. In other words, at r_l, at most $(m-1)$ processors are busy executing these jobs according to $\boldsymbol{A}_i^+$ and there is an idle processor on which J_l can be scheduled without preempting J_i. Corollary 4.1 allows us to conclude that an idle processor is also available at r_l according to $\boldsymbol{A}_i$; J_l is scheduled on this processor, not the one executing J_i at r_l, and J_l does not preempt J_i in $\boldsymbol{A}_i$. This conclusion contradicts our supposition that J_l preempts J_i. □

In fact, we can restate Lemmas 4.3 and 4.4 more precisely. Instead of $\wp_i^+(S^+(J_i))$ being a subsequence of $\wp_i^-(S^-(J_i))$, we say " if $\wp_i^+(S^+(J_i))$ and $\wp_i^-(S^-(J_i))$ are identical." The following lemma says that this seemingly more restrictive condition is in fact not more restrictive.

Lemma 4.5 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$ and $\wp_i^+(S^+(J_i))$ is a subsequence of $\wp_i^-(S^-(J_i))$, then $\wp_i^+(S^+(J_i))$ is identical to $\wp_i^-(S^-(J_i))$.*

Lemma 4.6 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$ and the two observable starting sequences $\wp_i^+(S^+(J_i))$ and $\wp_i^-(S^-(J_i))$ are identical, then the actual starting sequence $\wp_i(S(J_i))$ is identical to $\wp_i^+(S^+(J_i))$ (or $\wp_i^-(S^-(J_i))$).*

Lemma 4.7 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$ and the two observable starting sequences $\wp_i^+(S^+(J_i))$ and $\wp_i^-(S^-(J_i))$ are identical, then $S^-(J_i) \leq S(J_i) \leq S^+(J_i)$.*

Theorem 4.4 *If no job in $\boldsymbol{J}_i$ is preempted according to $\boldsymbol{A}_i^+$ and the two observable starting sequences $\wp_i^+(S^+(J_i))$ and $\wp_i^-(S^-(J_i))$ are identical, then $F^-(J_i) \leq F(J_i) \leq F^+(J_i)$.*

Obviously, the upper bound of $F(J_i)$ given by this theorem is tight. For example, for the system in Figure 1, no job in $\boldsymbol{J}_3$ is preempted according to the maximal schedule $\boldsymbol{A}_3^+$, and $\wp_3^+(S^+(J_3))$ is identical to $\wp_3^-(S^-(J_3))$. Consequently, we can conclude that the completion time of J_3 is never later than $F^+(J_3)$, which is 13 according to Figure 1 (a). Similarly, for the system consisting of m jobs that is mentioned earlier, no job in $\boldsymbol{J}_m$ is preempted according to the maximal schedule $\boldsymbol{A}_m^+$, and $\wp_m^+(S^+(J_m))$ is identical to $\wp_m^-(S^-(J_m))$. Hence the completion time of each job J_i is at most equal to $F^+(J_i)$.

Conditions for Unpredictable Executions

A natural question to ask at this point is whether a tight bound can be derived in a similar manner for the case when there is no preemption in the maximal schedule and $\wp_i^-(S^-(J_i))$ is a subsequence of $\wp_i^+(S^+(J_i))$. The example in Figure 3 shows that this is not possible. Parts (a) and (b) show the maximal and minimal schedules, respectively. Part (c) shows a possible actual schedule. J_1, J_2, J_3 and J_4 have the same parameters as the ones in Figure 1 except that the execution time of J_2 is in the range [1,6] and the release times of J_4 is 2. The parameters of J_5, J_6 and J_7 are listed in the table in Figure 3. As shown in Figure 3, $\wp_7^-(S^-(J_7)) = (J_1, J_2, J_7)$ is a subsequence of $\wp_7^+(S^+(J_7)) = (J_1, J_2, J_3, J_4, J_6, J_7)$, but the actual starting sequence $\wp_7(S(J_7)) = (J_1, J_2, J_4, J_3, J_6, J_5, J_7)$, according to the actual schedule in part (c), is not. Furthermore, the set of jobs in the actual starting sequence $\wp_7(S(J_7))$ is not a subset of the set of jobs in the maximal starting sequence $\wp_7^+(S^+(J_7))$. (J_5 is not in $\wp_7^+(S^+(J_i))$ but is in $\wp_7(S(J_7))$.) The actual start time of J_7 is larger than the maximal start time $S^+(J_7)$, illustrating that the start time is not predictable in this case. Moreover, according to both $\boldsymbol{A}_7^+$ and $\boldsymbol{A}_7^-$, no job is preempted before the start time of J_7, but J_4 and J_6 are preempted before $S(J_7)$ according to the actual schedule.

Similarly, when some job(s) in $\boldsymbol{J}_i$ is preempted before the completion time of J_i in the maximal schedule, the start time and completion time of J_i may be unpredictable, even though all the starting sequences are same. Figure 4 shows an illustrative example. The release times of J_1, J_2 and J_4 are 0. The release times of J_3 and J_5 are indicated by the arrows. The schedules in Figure 4 (a)-(c) are the maximal, the minimal and the actual schedules, respectively. All the starting sequences are identical, that is, $\wp_5^+(S^+(J_5)) = \wp_5^-(S^-(J_5)) = \wp_5(S(J_5)) = (J_1, J_2, J_4, J_3, J_5)$. However, the start time and completion time of J_5 in the actual schedule are later than the ones in the maximal schedule. In particular, in the maximal schedule,

job	r_i	d_i	$[e_i^-, e_i^+]$
J_5	18	25	[3,3]
J_6	5	200	[100,100]
J_7	0	20	[2,2]

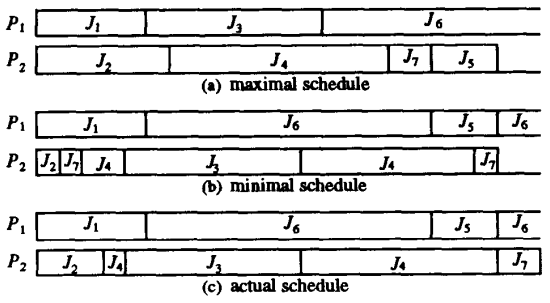

Figure 3: An example illustrating unpredictable start times and completion times

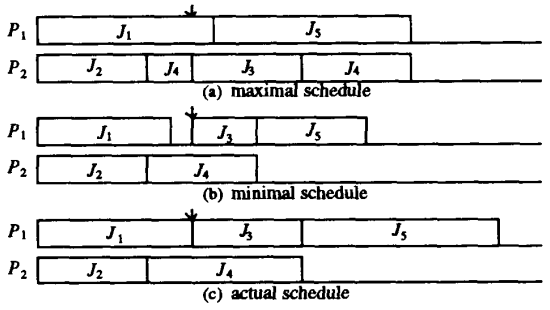

Figure 4: An unpredictable job with two identical observable starting sequences

J_3 preempts J_4, and J_5 starts during the time when J_4 is preempted. In the actual schedule, J_3 does not preempt J_4, J_3 is scheduled on the same processor as J_5, and the start time of J_5 is postponed by J_3.

We note that the completion times of the jobs in Figure 4 are in fact accurately predicted by the upper bounds given by Theorem 4.3. This example, as well as similar examples in [12], causes us to ask whether the bounds in Theorem 4.3 are tight in some sense when there is preemption in the maximal schedule. This question remains to be addressed in the future.

5 Some Jobs are Nonpreemptable

It is well known that when some of the jobs are nonpreemptable their execution is not predictable [13]. We consider here three cases: when all jobs are nonpreemptable, when preemptable jobs are migratable, and when preemptable jobs are not migratable. For all three cases, the release times of all jobs are assumed to be fixed, and jobs are independent.

Totally Nonpreemptable Case

A lower-priority nonpreemptable job whose release time is earlier than J_i may be executed to completion after J_i in the observable schedules but before J_i in the actual schedule. Consequently, we cannot ignore lower-priority jobs when trying to find the start time and the completion time of J_i. Let $\boldsymbol{N}$ denote the set of nonpreemptable jobs and $\boldsymbol{N_i}$ denote the subset of nonpreemptable jobs that have release times earlier than J_i and priorities lower than J_i. Let $\boldsymbol{P_n^-}$ be a schedule of $\boldsymbol{J_n^-}$ constructed by assuming all the jobs are preemptable and migratable. Let $\boldsymbol{B_i}$ denote the set of jobs in $\boldsymbol{N_i}$ which start before J_i in $\boldsymbol{P_n^-}$. The following lemmas give us the basis to find upper bounds of the start times and completion times of independent N/N/F jobs. The proofs of these lemmas can be found in [12].

Lemma 5.1 *Every job J_i is blocked for at most the duration of one lower-priority job with release time earlier than r_i.*

Lemma 5.2 *A job J_l in $\boldsymbol{N_i}$ but not in $\boldsymbol{B_i}$ cannot start before J_i in the actual schedule $\boldsymbol{A_n}$.*

Lemma 5.2 allows us to identify among all the jobs in $\boldsymbol{N_i}$ the lower-priority jobs that have a chance to be scheduled before J_i in the actual schedule. We only need to be concerned with the subset $\boldsymbol{B_i}$ when trying to bound the completion time of J_i.

We note that Lemma 5.1 does not mean that we can simply bound the start time of J_i by $S^+(J_i)$ plus the largest of the maximum execution times of jobs in $\boldsymbol{B_i}$. During the time when J_i is blocked, higher-priority jobs with release times later than r_i may become ready. These higher-priority jobs may be scheduled before J_i, further delaying its start time. The postponement of J_i's start and execution may in turn postpone the start times of jobs with lower priorities than J_i.

Algorithm $\mathcal{INNF}$ takes these considerations into account when trying to bound the worst case completion times. It considers one job at a time, from the job with the highest priority to the job with the lowest priority. In order to find the worst-case completion time of J_i, Algorithm $\mathcal{INNF}$ transforms J_i and jobs in $\boldsymbol{J_{i-1}}$ as follows. In this transformation, J_i is transformed into two jobs, and each of the other jobs in $\boldsymbol{J_{i-1}}$ is transformed to one job. Let G_i and H_i denote the two jobs transformed from J_i. In Step 1 the parameters of the transformed jobs G_i and H_i are computed

from those of J_i. G_i's execution time is equal to the largest of the maximum execution times of jobs in $\boldsymbol{B}_i$, and G_i's release time is release time r_i of J_i. H_i's execution time is equal to J_i's maximum execution time, e_i^+, and H_i's release time h_i is r_i plus G_i's execution time. Let 0 be a priority that is higher than the priority of J_1. The priority of G_i is 0, and the priority of H_i is equal to that of J_i. G_i simulates the job that blocks J_i, and H_i simulates J_i blocked by G_i. Let H_k denote the job transformed from J_k, for $k = 1, 2, \cdots, i-1$. Step 1 also computes the parameters of H_k for each J_k for $k = 1, 2, \cdots, i-1$. H_k's release time is equal to r_k, its execution time is equal to e_k^+ plus the largest of the maximum execution times of jobs in $\boldsymbol{B}_k$, and its priority is equal to that of J_k. Let $\boldsymbol{H}_i$ denote the set of jobs $\{H_1, \ldots, H_i\}$. In Step 2, G_i and $\boldsymbol{H}_i$ are scheduled according to the given nonpreemptable, priority-driven algorithm. An upper bound on the completion time $F(J_i)$ of J_i is equal to the completion time of H_i in the resultant schedule.

$\boldsymbol{H}_{i-1}$ is not a subset of $\boldsymbol{H}_i$; $\boldsymbol{H}_i$ needs to be constructed for every job J_i. Consequently, the complexity of Algorithm $\mathcal{INNF}$ is $O(n^2)$. The proof of a theorem stating that if H_i can complete by the deadline d_i of J_i in the schedule generated by $\mathcal{INNF}$ algorithm, J_i is schedulable can be found in [12].

Because no job is ever blocked by a preemptable lower-priority job, we can use Algorithm $\mathcal{INNF}$ with very little change to find the completion times of all jobs when some jobs are preemptable and migratable. Specifically, Step 2 of Algorithm $\mathcal{INNF}$ treats (1) G_i as nonpreemptable and (2) H_k in $\boldsymbol{H}_i$ as preemptable (nonpreemptable) if J_k is preemptable (nonpreemptable).

Nonpreemptable and Nonmigratable Case

In the case where some jobs are nonpreemptable and some jobs are preemptable but not migratable, the actual start time and completion time of J_i may be postponed beyond $S^+(J_i)$ and $F^+(J_i)$ by two kinds of jobs. First, a nonpreemptable lower-priority job may block some jobs in the actual schedule but not in the maximal schedule. Second, some higher-priority jobs may preempt some jobs in the actual schedule but not in the maximal schedule. When preemptable jobs are not migratable, Lemma 5.2 no longer holds. Similarly, the schedule of $\boldsymbol{J}_n^-$ constructed by assuming all the jobs are preemptable and nonmigratable gives us no information on which lower-priority jobs can actually start before J_i. Counterexamples can be found in [12].

Algorithm $\mathcal{INNF} - \mathcal{N}$ can be used to bound the completion times in this case. It consists of two steps. Step 1 considers the delays in the start time of each job J_i by nonpreemptable lower-priority jobs. It uses Algorithm $\mathcal{INNF}$ to construct a schedule for each J_i, using the set $\boldsymbol{N}_i$ instead of $\boldsymbol{B}_i$ in Step 1 of Algorithm $\mathcal{INNF}$. Let $F_*^+(J_i)$ denote the completion time of H_i in this schedule. In Step 2 it computes the delays in the completion time of J_i due to higher-priority jobs which may preempt J_i, or some job starting before J_i, in the actual schedule. Step 2 makes use of the following theorem. This theorem is stated in terms of the set $\boldsymbol{E}_i$; $\boldsymbol{E}_i$ is a subset of $\boldsymbol{J}_i$ in which each job J_k is released after some preemptable job in $\boldsymbol{J}_i$ with a priority lower than itself (that is, J_k), and its transformed job H_k (the job created in Step 1 of Algorithm $\mathcal{INNF}$) is not scheduled on the same processor as the transformed job H_i to complete before H_i in the schedule constructed by Algorithm $\mathcal{INNF}$. The complexity of Algorithm $\mathcal{INNF} - \mathcal{N}$ is $O(n^2)$.

Theorem 5.1 $F(J_i) \leq F_*^+(J_i) + \left(\sum_{J_k, H_k \in E_i} e_k^+ \right) - (e_i^+ - e_i)$.

6 Summary and Future Work

In this paper, we have presented several worst-case upper bounds and efficient algorithms. They can be used to reliably compute the worst-case completion times of independent jobs in homogeneous distributed systems in which jobs are dynamically dispatched and scheduled on available processors in an event-driven manner. One of the algorithms allows us to take into account release times jitters. The others assume fixed arbitrary release times but take into account the effects of nonpreemptability and nonmigratability.

These upper bounds and algorithms can be easily modified and applied to find upper bounds on the completion times of jobs in some heterogeneous systems. Processors in a heterogeneous system are divided into c different types. Let m_j be the number of type-j processors and $m = \sum_{j=1}^{c} m_j$. There are two cases: when a job can execute on only one type of processors and when a job can execute on several types of processors. In the former case, a job J_i is said to be of type j if it can execute only on a type-j processor. The job set $\boldsymbol{J}$ is divided into disjoint subsets according to the processor types of jobs. We can find the worst-case completion times of type-j jobs by applying the algorithms described in the previous sections to the subset of type-j jobs on m_j identical processors. An example of such systems is a distributed database system. A user query arrives at one of many communication processors. After the user is authenticated,

the query is sent via one of many data links to one of several replicated database servers. After the query is processed, the response is sent back over the data link, back to the user through the communication processor. In our model, this system has three types of processors: the communication processors, data links, and database servers. User authentication jobs can only be executed on the communication processors, query/response transmission can only be over the data links, and queries can only be processed by database servers.

We are developing an algorithm to handle the case where each job can execute on several types of processors. Our model of heterogeneous processors is known as the uniform-processor model [11]. Jobs are preemptable and migratable and have fixed release times. The reliability and performance of this algorithm (in terms of the tightness of the upper bounds produced by it) remain to be ascertained.

The results presented here constitute a small part of the theoretical basis needed for a comprehensive validation strategy that is capable of dealing with dynamic distributed real-time systems. Much of the work on schedulability analysis remains to be done. For example, we must be able to deal with dependencies between jobs. Ways to reliably compute the worst-case completion times of jobs that have precedence constraints and/or share resources are yet not available. This is a part of our future work.

Acknowledgement

The authors wish to thank Dr. C. L. Liu, Dr. W. K. Shih, and D. Hull for their comments and suggestions.

References

[1] J. A. Stankovic, K. Ramamritham, and S. Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Transactions on Computers*, 34(12):1130–1143, December 1985.

[2] K.G. Shin and Y.C. Chang. Load sharing in distributed real-time systems with state-change broadcasts. *IEEE Transactions on Software Engineering*, 38(8):1124–1142, August 1989.

[3] K. Schwan and H. Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736–748, August 1992.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[5] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, February 1978.

[6] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[7] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of IEEE 9th Real-Time Systems Symposium*, pages 259–269, December 1988.

[8] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[9] T. P. Baker. A stack-based allocation policy for real-time processes. In *Proceedings of IEEE 11th Real-Time Systems Symposium*, pages 191–200, December 1990.

[10] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotone scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of IEEE 10th Real-Time Systems Symposium*, pages 166–171, December 1989.

[11] J. Blazewicz. Selected topics in scheduling theory. *Annals of Discrete Mathematics*, 31:1–60, 1987.

[12] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. Technical Report UIUCDCS-R-93-1833, University of Illinois at Urbana-Champaign, 1993.

[13] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.