IEEE Transactions on Software Engineering, Vol. 23, No. 10, October 1997, pp. 603-615.

Bounding Completion Times of Jobs with Arbitrary Release Times, Variable Execution Times, and Resource Sharing

Jun Sun, Mark K. Gardner, and Jane W.S. Liu

Abstract— **The workload of many real-time systems can be characterized as a set of preemptable jobs with linear precedence constraints. Typically their execution times are only known to lie within a range of values. In addition jobs share resources and access to the resources must be synchronized to ensure the integrity of the system. This paper is concerned with the schedulability of such jobs when scheduled on a priority-driven basis. It describes three algorithms for computing upper bounds on the completion times of jobs that have arbitrary release times and priorities. The first two are simple but do not yield sufficiently tight bounds, while the last one yields the tightest bounds but has the greatest complexity.**

Keywords— **Real-time systems, schedulability analysis, precedence constraints, resource sharing**

I. Introduction

In a real-time system, an external event may cause a set of jobs to be released and executed. The execution of some jobs may depend upon the completion of others. For example, pushing a button may trigger an interrupt handling job which produces data to be analyzed by an event processing job. Because the system state is not consistent until the interrupt handling job completes, the event processing job cannot start before the interrupt handling job completes. In addition to such precedence constraints, the release time of a job may depend on the release time of another job. For example, a data sampling job in a data acquisition system may not be allowed to start until a specified amount of time after the power-on job is released in order to ensure that the system is stable. In this paper we focus our attention on uniprocessor systems where the workload consists of chains of jobs, or *job chains*. Jobs in the same job chain are dependent, i.e., a job cannot start until all the preceding jobs in the same chain complete. In other words, no job can start executing until its release time or until all its predecessors have completed, whichever is later. Each job has an arbitrary but fixed release time which specifies the earliest possible time that the job can start.

Each job has a variable execution time which is known to lie within a given range. We assume that jobs are scheduled preemptively in a priority-driven manner. Each job has a fixed priority. Jobs may have critical sections which synchronize access to shared resources. We confine our attention to systems where all resource accesses are made according to the Non-preemptable Critical Section (NPS) protocol; each job is nonpreemptable when it is in a critical section [1]. Therefore uncontrolled priority inversion [2] cannot occur.

We do not consider the problem of how to assign priorities to jobs in this paper. Rather, we form ways to find upper bounds on the completion times of jobs under a given priority assignment, i.e., the problem addressed here is that of schedulability analysis of the system rather than the scheduling problem. Specifically, we describe here a progression of three algorithms that compute upper bounds on the completion times of jobs when they are scheduled according to a priority-driven algorithm. The first algorithm computes an upper bound on the *effective response times (ERT)* of individual jobs in each chain and is therefore called Algorithm ERT. The second algorithm is based on an analysis of the *critical job*, a concept which we will define later; it is called Algorithm CJA. The third algorithm iteratively applies Algorithm CJA to yield tighter bounds and is called Algorithm ITR. Algorithms ERT, CJA and ITR yield increasingly tighter upper bounds, but at increasingly higher complexity. While Algorithm ERT and Algorithm CJA can be used for on-line admission control (i.e., to determine whether the system can accept a new chain of jobs and still ensure on-time completion of all jobs), Algorithm ITR is more suitable for off-line schedulability analysis.

A great deal of work has been done on timing analysis for periodic tasks [3–7]. A periodic task is an infinite stream of identical jobs that are released periodically. The objective of timing analysis is to bound

J. Sun is with Geoworks, Inc., 2001 Center St., Berkeley, CA 94704 USA, jsun@geoworks.com.

M. K. Gardner and J. W.S. Liu are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, {mkgardne,janeliu}@cs.uiuc.edu.

the response times of all jobs in each task. Lehoczky [5] developed a time-demand analysis method for this purpose. Harbour et.al. [7] developed a method to bound the response times of jobs where each periodic task is a chain of subtasks. These existing methods either cannot be applied to bound the completion times of dependent jobs that have arbitrary release times (e.g., the methods based on schedulable utilization bounds [3, 8]) or yield unsatisfactorily loose bounds (e.g., the time-demand analysis method [5]). A reason for the poor performance of existing methods is that they ignore the actual release times of jobs but work with the worst-case combination of release times. While this treatment makes sense in timing analysis for periodic tasks which may have release time jitter, it causes the algorithms to be very pessimistic when applied to dependent jobs that have known and fixed release times. As a matter of fact, Algorithm ERT described in this paper also ignores the release times of jobs. As we will see in Section VI, the performance of this algorithm is poor compared with the other two algorithms which make use of information on release times.

The problem solved by our algorithms is also related to the *validation problem*. Both problems deal with a set of jobs with variable execution times. Ha [9] has studied the validation problem in multiprocessor or distributed systems. In her work, a system is predictable if the completion time of a job can be bounded by the completion times of the job in the maximum schedule and minimum schedule, where the maximum (minimum) schedule is obtained by applying the priority-driven algorithm to the set of jobs assuming that all jobs have their maximum (minimum) execution times. As shown by an example in the next section, the execution of a set of dependent, preemptable jobs (with or without critical sections) on a single processor is not predictable. Bounding the completion times of jobs is a reasonable approach to validating the timing constraints for these kinds of systems. Our algorithms provide tighter bounds and, thus, more accurate conclusions on the satisfiability of timing constraints than the general bounds provided by algorithms in [9].

The rest of the paper is organized as follows. Section II formally defines the problem addressed and introduces the notations used in the paper. Sections III, IV, and V present Algorithms ERT, CJA and ITR, respectively. Section VI presents the performance of the three algorithms obtained by simulation while Section VII discusses modifications to the algorithms when jobs have jittered release times. Section VIII concludes the paper.

II. Problem Formulation

Again, the problem addressed here is how to determine whether every job in n independent job chains, denoted $\mathbf{J}_1, \mathbf{J}_2, \ldots, \mathbf{J}_n$, can complete in time when the jobs are scheduled on a processor according to a priority-driven algorithm. We let $J_{i,j}$ denote the jth job of job chain $\mathbf{J}_i$. By independent chains, we mean that $J_{i,1}$ has no predecessor for every $i = 1, 2, \ldots, n$, and there are no precedence constraints between any pair of jobs in two different chains. We assume that each job $J_{i,j}$ has a fixed priority $\phi_{i,j}$ and is preemptable, except where stated otherwise. The execution time of job $J_{i,j}$ is in the range $[e^-_{i,j}, e^+_{i,j}]$. Both the maximum execution time $e^+_{i,j}$ and the minimum execution time $e^-_{i,j}$, as well as the release time $r_{i,j}$ of $J_{i,j}$ are known, but the actual execution time $e_{i,j}$ is not known.

The release time $r_{i,j}$ of job $J_{i,j}$ is arbitrary but fixed. $J_{i,1}$ is ready for execution at its release time $r_{i,1}$; for each $j > 1$, $J_{i,j}$ cannot execute until its immediate predecessor $J_{i,j-1}$ completes. We assume that the release time $r_{i,j}$ of every job $J_{i,j}$ is consistent with its precedence constraints. Specifically, the release time $r_{i,j}$ of a job $J_{i,j}$ is no sooner than $r_{i,j-1} + e^-_{i,j-1}$, which is the earliest time at which its immediate predecessor can complete. When the given release times do not satisfy this assumption, we replace them with *effective release times* that do. The effective release time of $J_{i,1}$ is equal to its given release time. The effective release time of $J_{i,j}$ is equal to its given release time or the sum of the effective release time of $J_{i,j-1}$ and $e^-_{i,j-1}$, whichever is larger. We lose no generality by working with the effective release times of jobs.

Each job may also require exclusive use of some resources. We assume that the overlapping critical sections of every job are properly nested. Hereafter, by a critical section we mean an outermost critical section. The maximum execution time of every critical section is known a priori. We let $d^k_{i,j}$ denote the maximum duration of the kth critical section $x^k_{i,j}$ of the job $J_{i,j}$: $d^k_{i,j}$ is the maximum execution time of $x^k_{i,j}$ when the job executes alone. Finally, we let $D_{i,j}$ denote the maximum duration of all the critical sections of the job.

As stated earlier, the NPS protocol is used to control accesses to resources by jobs. Such a protocol guarantees that each job can be blocked by at most one lower priority job for the duration of one critical section. This fact is stated formally by the following lemma, the proof of which can be found in [1]. The term *blocking time* used in the lemma refers to the length of the delay suffered by a job due to priority

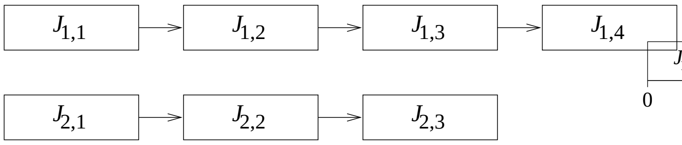

Fig. 1. Simple Job Set in Example 1

TABLE I
Parameters of Jobs in Example 1

$J_{i,j}$	$\phi_{i,j}$	$r_{i,j}$	$e^-_{i,j}$	$e^+_{i,j}$	$D_{i,j}$
$J_{1,1}$	2	0	10	40	
$J_{1,2}$	4	20	5	10	
$J_{1,3}$	2	75	20	30	10
$J_{1,4}$	4	130	15	50	
$J_{2,1}$	3	30	10	10	
$J_{2,2}$	3	60	5	40	20
$J_{2,3}$	1	120	20	70	60

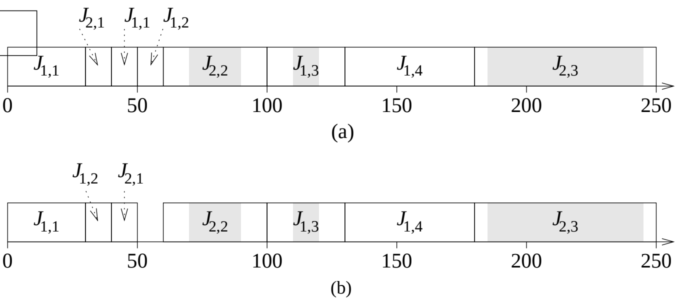

Fig. 2. Schedules of Jobs in Example 1

inversion: a lower priority job executes while the job waits. (The lower priority job is said to block the job.)

Lemma 1: The maximum blocking time of a job is the duration of the longest critical section of all lower priority jobs that can block it.

Clearly, a job $J_{i,j}$ can never be blocked by a lower priority job in the same job chain $\mathbf{J}_i$. The set of jobs that can block a job $J_{i,j}$ in $\mathbf{J}_i$ includes all the jobs that are in chains other than $\mathbf{J}_i$, have lower priorities than $J_{i,j}$ and have one or more critical sections.

An example of such a system is shown in Figure 1. In this example, referred to as Example 1 later, there are two job chains, $\mathbf{J}_1$ and $\mathbf{J}_2$. $\mathbf{J}_1$ has four jobs and $\mathbf{J}_2$ has three. There are three critical sections, one in job $J_{1,3}$, one in job $J_{2,2}$, and the other in job $J_{2,3}$. The parameters of each job are given in Table I. We use integers to represent priorities; the greater the integer, the higher the priority.

A job is *ready* at the instant when it is released or when its immediate predecessor completes, whichever is later. Let $y_{i,j}$ denote the *ready time* of job $J_{i,j}$ and $c_{i,j}$ denote its completion time. Since the first job in a job chain J_i has no predecessors, it is ready when it is released, i.e., $y_{i,1} = r_{i,1}$. For a later job $J_{i,j}$ ($j > 1$), we have $y_{i,j} = \max\{r_{i,j}, c_{i,j-1}\}$. The *response time* of $J_{i,j}$ is equal to its completion time less its release time (i.e., the duration of the interval $(r_{i,j}, c_{i,j}]$), and its *effective response time* is equal to its completion time less its ready time (i.e., the duration of the interval $(y_{i,j}, c_{i,j}]$).

Because the execution times of jobs may vary and the scheduling algorithm is priority driven, there may be many different schedules for a given set of jobs. According to some of these schedules, a job may have its worst-case (i.e., the latest) completion time while other jobs may not. In particular, when all jobs have their maximum execution times, we may not observe the worst-case completion times of all the jobs. For example, Figure 2 shows two schedules of the job chains in Example 1. In both cases, jobs are scheduled according to their priorities. We have the schedule in Figure 2(a) when all the jobs have their maximum execution times. According to this schedule, job $J_{2,1}$ completes at time 40. However when the execution time of $J_{1,1}$ is reduced from 40 to 30, we obtain the schedule in Figure 2(b) where the completion time of $J_{2,1}$ is 50. As it turns out, this is the worst-case completion time of $J_{2,1}$. This example shows that systems considered in this paper are not predictable in general [9]. To bound the completion times of jobs, we can exhaustively simulate the execution of the system and search for the worst-case completion times of all jobs. The complexity of a brute-force search is $O(E^N)$, where E is the length of the range $[e^-_{i,j}, e^+_{i,j}]$ for all i and j and N is the total number of jobs in the system. Clearly the exhaustive approach is impractical for most real-life systems. We focus here on analytical methods which give us upper bounds on the completion times of jobs rather than finding the exact worst case completion time.

III. Algorithm ERT

Algorithm ERT first bounds the effective response times of jobs and then derives the bounds on completion times from the effective response times. To motivate this algorithm, we focus on a job $J_{i,j}$ in job chain $\mathbf{J}_i$. Obviously, a job other than $J_{i,j}$ can execute during the interval $(y_{i,j}, c_{i,j}]$ only if it is in a different job chain than $\mathbf{J}_i$. Furthermore, it must have a priority no lower than the priority $\phi_{i,j}$ of $J_{i,j}$ or it must be in a critical section at $y_{i,j}$.

Figure 3 illustrates a job chain $\mathbf{J}_k$ ($k \neq i$). The shaded boxes represent the jobs in $\mathbf{J}_k$ whose priorities are lower than $\phi_{i,j}$, and white boxes represent

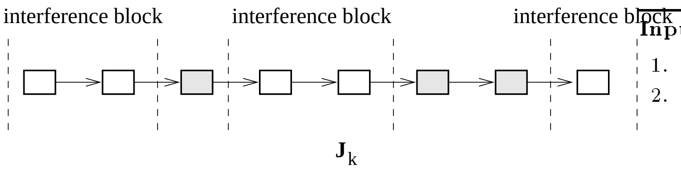

Fig. 3. Interference Blocks

jobs whose priorities are equal to or higher than $\phi_{i,j}$. The lower priority jobs divide the chain $\mathbf{J}_k$ into subchains, each of which contains only jobs with priorities higher than or equal to $\phi_{i,j}$. In this example, there are three such equal or higher priority subchains. We call such a subchain an *interference block* of $J_{i,j}$. In general, an interference block of $J_{i,j}$ is a subchain $\{J_{k,l}, J_{k,l+1}, \ldots, J_{k,l+u}\}$ of $\mathbf{J}_k$, for some $k \neq i$ that has the following properties. Priorities $\phi_{k,l}, \phi_{k,l+1}, \ldots, \phi_{k,l+u}$ are higher than or equal to $\phi_{i,j}$; either $J_{k,l}$ has no predecessor or $\phi_{k,l-1}$ is lower than $\phi_{i,j}$; and either $J_{k,l+u}$ has no successor or $\phi_{k,l+u+1}$ is lower than $\phi_{i,j}$. Except for a critical section of a lower priority job that has already begun at $y_{i,j}$, only jobs from the interference blocks of $J_{i,j}$ can execute during the interval $(y_{i,j}, c_{i,j}]$. Furthermore, since the interference blocks are separated by one or more jobs whose priorities are lower than $J_{i,j}$, it is impossible for jobs in more than one interference block of the same chain to execute in $(y_{i,j}, c_{i,j}]$. Consequently, when we want to bound the interference of other jobs on $J_{i,j}$ (i.e., the amount of time $J_{i,j}$ can be delayed by other jobs of equal or higher priority), we only need to consider one interference block from each job chain other than $\mathbf{J}_i$ This allows us to more tightly bound the possible interference of jobs on $J_{i,j}$ once it becomes ready at $y_{i,j}$.

We focus now on finding the maximal interference suffered by $J_{i,j}$. Hereafter, we call the job whose completion time we are trying to bound the *target job*. By an interference block, we mean specifically an interference block of the target job. Suppose that a job chain $\mathbf{J}_k$ has m_k interference blocks, and $M_{k,l}$ is equal to the sum of the maximum execution time of jobs in the lth interference block in $\mathbf{J}_k$. As we have discussed in the previous paragraph, the *maximum interference* by equal or higher priority jobs in $\mathbf{J}_k$ on the target job $J_{i,j}$ is never more than the maximum of $M_{k,l}$ for all $l = 1, 2, \ldots, m_k$ (i.e., $\max_{1 \leq l \leq m_k}\{M_{k,l}\}$). This is the basis of Algorithm Interference. For a target job $J_{i,j}$, the algorithm computes an upper bound $totalInter(J_{i,j}, \mathbf{J})$ of the total maximum interference by equal or higher priority jobs in all job chains other than $\mathbf{J}_i$. In other words, $totalInter(J_{i,j}, \mathbf{J}) = \sum_{k \neq i} \max_{1 \leq l \leq m_k}\{M_{k,l}\}$. The algorithm also calculates $\min_{k \neq i} \max_{1 \leq l \leq m_k}\{M_{k,l}\}$; we denote this minimum of maximum interferences by jobs in other jobs chains by $minInter(J_{i,j}, \mathbf{J})$. Figure 4 describes this algorithm.

Input:

1. The target job $J_{i,j}$.
2. A set $\mathbf{J}$ of jobs where each job $J_{k,l}$ has the release time $r_{k,l}$, the priority $\phi_{k,l}$, and the range $[e^-_{k,l}, e^+_{k,l}]$ of execution times.

Output:

1. $totalInter(J_{i,j}, \mathbf{J})$, the total maximum interference by jobs in job chains other than $\mathbf{J}_i$.
2. $minInter(J_{i,j}, \mathbf{J})$, the minimum among the maximum interference by jobs in job chains other than $\mathbf{J}_i$.

Algorithm:

1. $totalInter = 0$, $minInter = \sum_{k \neq i} \sum_{l=1}^{m_k} e^+_{k,l}$
2. For every job chain $\mathbf{J}_k$ other than $\mathbf{J}_i$,
   (a) find the interference blocks in $\mathbf{J}_k$;
   (b) if there are no interference blocks in $\mathbf{J}_k$, $M_k = 0$; otherwise, for every l from 1 to m_k, find $M_{k,l}$ and $M_k = \max_{1 \leq l \leq m_k}\{M_{k,l}\}$;
   (c) $totalInter = totalInter + M_k$;
   (d) $minInter = \min(minInter,\ M_k)$.
3. Return $totalInter$ and $minInter$.

Fig. 4. Algorithm Interference

Because jobs contend for resources, we must also consider lower priority jobs that can block the target job $J_{i,j}$. We let $block(J_{i,j}, \mathbf{J})$ denote the duration of the longest critical section of all the jobs that are in job chains other than $\mathbf{J}_i$ and have lower priorities than $J_{i,j}$.

An upper bound of the total delay the target job may suffer (i.e., the total maximum execution times of all the jobs in other job chains that may execute in $(y_{i,j}, c_{i,j}]$) is

$$delay(J_{i,j}, \mathbf{J}) = totalInter(J_{i,j}, \mathbf{J}) + block(J_{i,j}, \mathbf{J}) \tag{1}$$

However, we observe that it is not possible for the target job to be delayed by an interference block of equal or higher priority jobs in a job chain $\mathbf{J}_k$ and at the same time be blocked by a lower priority job in $\mathbf{J}_k$. (The reason is obvious: if a job in $\mathbf{J}_k$ blocks the target job, it will be preempted by the target job when it exits the critical section and, therefore, a subsequent interference block in $\mathbf{J}_k$ cannot become ready for execution until the target job completes and the blocking job resumes.) We can safely improve the upper bound by subtracting $\min\{block(J_{i,j}, \mathbf{J}), minInter(J_{i,j}, \mathbf{J})\}$ from the sum $totalInter(J_{i,j}, \mathbf{J}) + block(J_{i,j}, \mathbf{J})$. Therefore, an upper bound on the duration of interval $(y_{i,j}, c_{i,j}]$, which is the effective response time of $J_{i,j}$, satisfies the inequality $c_{i,j} - y_{i,j} \leq e^+_{i,j} + \mathit{delay}(J_{i,j}, \mathbf{J})$ where

$$\begin{aligned} \mathit{delay}(J_{i,j}, \mathbf{J}) &= \mathit{totalInter}(J_{i,j}, \mathbf{J}) + \mathit{block}(J_{i,j}, \mathbf{J}) \\ &\quad - \min\{\mathit{minInter}(J_{i,j}, \mathbf{J}), \mathit{block}(J_{i,j}, \mathbf{J})\} \end{aligned} \quad (2)$$

A simple transformation gives

$$c_{i,j} \leq y_{i,j} + e^+_{i,j} + \mathit{delay}(J_{i,j}, \mathbf{J}) \quad (3)$$

For the first job $J_{i,1}$ in the job chain $\mathbf{J}_i$, $r_{i,1} = y_{i,1}$. An upper bound $\hat{c}_{i,1}$ of $c_{i,1}$ is given by

$$\hat{c}_{i,1} = r_{i,1} + e^+_{i,1} + \mathit{delay}(J_{i,1}, \mathbf{J}) \quad (4)$$

For a job $J_{i,j}$ $(j > 1)$ that is not the first job in the chain, its ready time $y_{i,j}$ is equal to $\max\{c_{i,j-1}, r_{i,j}\}$. Therefore an upper bound $\hat{c}_{i,j}$ of $c_{i,j}$ is

$$\hat{c}_{i,j} = \max\{\hat{c}_{i,j-1}, r_{i,j}\} + e^+_{i,j} + \mathit{delay}(J_{i,j}, \mathbf{J}) \quad (5)$$

By applying Eqs. (4) and (5) to jobs in their execution precedence order, we can obtain an bound on the completion time of every job.

In summary, for each target job $J_{i,j}$, Algorithm ERT first calculates $\mathit{totalInter}(J_{i,j}, \mathbf{J})$ and $\mathit{minInter}(J_{i,j}, \mathbf{J})$ according to Algorithm Interference and finds the blocking time $\mathit{block}(J_{i,j}, \mathbf{J})$. It then computes $\mathit{delay}(J_{i,j}, \mathbf{J})$ and $\hat{c}_{i,j}$ according to Eqs. (3), (4) and (5). The complexity of Algorithm ERT is $O(N^2)$, where N is the total number of jobs in the system.

As an example, we compute the delay for every job in Example 1. From the perspective of $J_{1,1}$, jobs $J_{2,1}$ and $J_{2,2}$ have equal or higher priorities and form one interference block. The maximum execution time of this block is the sum of the maximum execution times of $J_{2,1}$ and $J_{2,2}$, which is 50 time units. Since $J_{2,3}$ contains a critical section and has a lower priority than $J_{1,1}$, the possible blocking delay suffered by $J_{1,1}$ is 60 time units. Thus $\mathit{delay}(J_{1,1}, \mathbf{J})$, the amount of delay suffered by $J_{1,1}$, is $50 + 60 - \mathit{min}\{50, 60\} = 60$. From the perspective of job $J_{2,1}$, jobs $J_{1,1}$ and $J_{1,3}$ have lower priorities, and jobs $J_{1,2}$ and $J_{1,4}$ have higher priorities. Since $J_{1,2}$ and $J_{1,4}$ are separated by $J_{1,3}$, they form two different interference blocks. The maximum execution times of these two interference blocks are 10 time units and 50 time units, respectively. The delay due to the critical section in $J_{1,3}$ is 10 units. Consequently $\mathit{delay}(J_{2,1}, \mathbf{J})$, the amount of delay suffered by $J_{2,1}$, is 50 time units. In a similar manner, the delay suffered by each of the other jobs in Example 1 is computed and listed in Table II. Based on these bounds on the maximal delays each job can suffer according to Algorithm ERT,

TABLE II
$\mathit{delay}(J_{i,j}, \mathbf{J})$ FOR JOBS IN EXAMPLE 1

$J_{1,1}$	$J_{1,2}$	$J_{1,3}$	$J_{1,4}$	$J_{2,1}$	$J_{2,2}$	$J_{2,3}$
60	60	60	60	50	50	130

we apply Eqns. (4) and (5) to obtain bounds on the completion times of all the jobs in Example 1.

$$\begin{aligned} \hat{c}_{1,1} &= r_{1,1} + e^+_{1,1} + \mathit{delay}(J_{1,1}, \mathbf{J}) = 100 \\ \hat{c}_{1,2} &= \max\{\hat{c}_{1,1}, r_{1,2}\} + e^+_{1,2} + \mathit{delay}(J_{1,2}, \mathbf{J}) = 170 \\ \hat{c}_{1,3} &= \max\{\hat{c}_{1,2}, r_{1,3}\} + e^+_{1,3} + \mathit{delay}(J_{1,3}, \mathbf{J}) = 260 \\ \hat{c}_{1,4} &= \max\{\hat{c}_{1,3}, r_{1,4}\} + e^+_{1,4} + \mathit{delay}(J_{1,4}, \mathbf{J}) = 370 \\ \hat{c}_{2,1} &= r_{2,1} + e^+_{2,1} + \mathit{delay}(J_{2,1}, \mathbf{J}) = 90 \\ \hat{c}_{2,2} &= \max\{\hat{c}_{2,1}, r_{2,2}\} + e^+_{2,2} + \mathit{delay}(J_{2,2}, \mathbf{J}) = 180 \\ \hat{c}_{2,3} &= \max\{\hat{c}_{2,2}, r_{2,3}\} + e^+_{2,3} + \mathit{delay}(J_{2,3}, \mathbf{J}) = 380 \end{aligned}$$

IV. Algorithm CJA

Algorithm ERT is simple to understand, easy to implement, and has relatively low complexity. However, it often does not give satisfactory bounds. To illustrate, we examine job $J_{2,2}$ in Example 1. The bound on its completion time is 180 time units. Half of the bound is contributed by the completion of its immediate predecessor $J_{2,1}$ at time 90, while 50 units of possible delay are contributed by job $J_{1,4}$. We observe that its predecessor $J_{2,1}$ completes at time 90 only if $J_{2,1}$ has been delayed by $J_{1,4}$ for 50 time units. Yet the same 50 time units of delay from $J_{1,4}$ is counted again in computing $\hat{c}_{2,2}$. Similarly, the delay from $J_{2,3}$ is counted twice in the upper bound of the completion time $\hat{c}_{1,3}$ of $J_{1,3}$. Algorithm CJA overcomes this problem by considering the subchain that contains the target job as a whole rather than dealing with the target job in isolation.

To motivate Algorithm CJA, suppose that a job chain $\mathbf{J}_i$ has five jobs and we want to bound the completion time of the target job $J_{i,5}$. Figure 5 shows a possible *worst-case schedule* for $J_{i,5}$, i.e., a schedule in which $J_{i,5}$ has its maximum completion time. Since all the release times are known, the completion time $c_{i,5}$ is equal to $r_{i,k}$ plus the duration of the interval $(r_{i,k}, c_{i,5}]$ for $k = 1, 2, \ldots, 5$. If we can find a tighter bound on the duration of these intervals, we can find a tighter bound on the completion time of $J_{i,5}$. We examine the *critical job* $J_{i,c(j)}$ of each target job $J_{i,j}$ for this purpose. In a schedule, $J_{i,c(j)}$ is the last job in $\mathbf{J}_i$ before and including $J_{i,j}$ whose ready time is equal to its release time. We call the interval $(r_{i,c(j)}, c_{i,j}]$ the *critical interval.* For example, in Figure 5, $J_{i,3}$ is the critical job of $J_{i,5}$, and

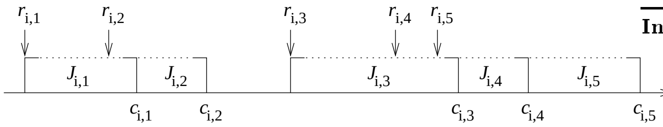

Fig. 5. Critical Job in a Schedule

interval $(r_{i,3}, c_{i,5}]$ is the critical interval. The following two lemmas state facts that help us to bound the duration of the critical interval $(r_{i,c(j)}, c_{i,j}]$.

Lemma 2: At most one job blocks a job in the critical interval $(r_{i,c(j)}, c_{i,j}]$.

Proof: Clearly a job with a priority lower than $J_{i,c(j)}$ that is in a critical section at $r_{i,c(j)}$ can block $J_{i,c(j)}$ and execute in the critical interval. That the subsequent jobs $J_{i,c(j)+1}, \ldots, J_{i,j}$ are never blocked (and hence no other lower priority jobs can execute in $(r_{i,c(j)}, c_{i,j}]$) follows from the fact that each of these jobs are ready immediately after its immediate predecessor completes. ■

Lemma 3: Any job that is in a job chain other than $\mathbf{J}_i$ and executes in the critical interval $(r_{i,c(j)}, c_{i,j}]$ must either have a priority higher than or equal to the priority of $J_{i,low}$, where job $J_{i,low}$ ($c(j) \leq low \leq j$) is the lowest priority job among jobs $J_{i,c(j)}$, $J_{i,c(j)+1}, \ldots, J_{i,j}$, or is a lower priority job that can block $J_{i,c(j)}$.

Proof: A job $J_{k,l}$ that is in another job chain $\mathbf{J}_k$, $k \neq i$, and has a priority lower than $J_{i,low}$ cannot preempt any of the jobs $J_{i,c(j)}, J_{i,c(j)+1}, \ldots, J_{i,j}$. According to Lemma 2, only $J_{i,c(j)}$ can be blocked and it can be blocked by at most one job. ■

According to Lemmas 2 and 3, the jobs of equal or higher priority that are in job chains other than $\mathbf{J}_i$ and can execute in the critical interval $(r_{i,c(j)}, c_{i,j}]$ are the same set of equal or higher priority jobs that are in job chains other than $\mathbf{J}_i$ and can execute in the interval $(y_{i,low}, c_{i,low}]$. Consequently, their total execution time can be bounded by $totalInter(J_{i,low}, \mathbf{J})$. The duration of the critical interval is never larger than this amount plus the maximum execution times of $J_{i,c(j)}, J_{i,c(j)+1}, \ldots, J_{i,j}$ and $block(J_{i,c(j)}, \mathbf{J})$, the longest critical section of jobs that have lower priorities than $J_{i,c(j)}$ and are in job chains other than $\mathbf{J}_i$. In other words,

$$c_{i,j} \leq r_{i,c(j)} + \sum_{l=c(j)}^{j} e^{+}_{i,l} + totalInter(J_{i,low}, \mathbf{J}) + block(J_{i,c(j)}, \mathbf{J}) \qquad (6)$$

Because it is possible for a job $J_{k,l}$, $k \neq i$, to block $J_{i,c(j)}$ and a subsequent interference block in $\mathbf{J}_k$ to preempt one of the jobs among $J_{i,c(j)+1}, \ldots, J_{i,j}$, we cannot tighten the bound as we did in Algorithm ERT by subtracting the minimum of $minInter(J_{i,low}, \mathbf{J})$ and $block(J_{i,c(j)}, \mathbf{J})$.

In the above critical job analysis, we assume that we know the critical job $J_{i,c(j)}$ of each target job $J_{i,j}$ in the worst-case schedule. This assumption is not true in general. To get around this problem, Algorithm CJA computes a bound on the completion time of $J_{i,j}$ by assuming that each of its predecessors, including $J_{i,j}$ itself, is the critical job. Since in the worst-case schedule there must exist a critical job, one of the bounds thus computed must be a correct one, and the maximum of these bounds must be a correct bound as well. The pseudo code of Algorithm CJA is listed in Figure 6. Its complexity is $O(N^3)$. We again use $\hat{c}_{i,j}$ to denote the upper bound on the completion time of $J_{i,j}$.

For example, we apply Algorithm CJA to bound the completion time of $J_{1,3}$ in Example 1 and obtain the following results. In the description, we treat each predecessor $J_{i,k}$ of the target job and the target job as the critical job in turn and compute the upper bound $b_{i,k}$ of the completion time of the target job according to Step 1(b) of Figure 6.

1. Let $J_{1,1}$ be the critical job. Job $J_{1,1}$ has the lowest priority among $J_{1,1}$, $J_{1,2}$ and $J_{1,3}$, so $b_{1,1}$ is equal to 190.
2. Let $J_{1,2}$ be the critical job. Job $J_{1,3}$ has the lowest priority among $J_{1,2}$ and $J_{1,3}$, so $b_{1,2}$ is

---

Input:

A set $\mathbf{J}$ of jobs where each job $J_{i,j}$ has a release time $r_{i,j}$, a priority $\phi_{i,j}$, a range $[e^{-}_{i,j}, e^{+}_{i,j}]$ of execution time, and a maximum duration $D_{i,j}$ of critical sections in the job.

Output: The bound $\hat{c}_{i,j}$ on the completion time of each job $J_{i,j}$.

Algorithm:

For each job $J_{i,j}$,

1. for each possible critical job $J_{i,k}$ ($1 \leq k \leq j$),
   (a) find the job $J_{i,low}$ such that $J_{i,low}$ has the lowest priority among job $J_{i,k}, J_{i,k+1}, \ldots, J_{i,j}$;
   (b) compute
   $$b_{i,k} = r_{i,k} + \sum_{l=k}^{j} e^{+}_{i,l} + block(J_{i,k}, \mathbf{J}) + totalInter(J_{i,low}, \mathbf{J}).$$
2. Let $\hat{c}_{i,j} = \max_{1 \leq k \leq j}\{b_{i,k}\}$.

Fig. 6. Algorithm CJA

TABLE III

Bounds Computed by Algorithm ERT and CJA

	$J_{1,1}$	$J_{1,2}$	$J_{1,3}$	$J_{1,4}$	$J_{2,1}$	$J_{2,2}$	$J_{2,3}$
ERT	100	170	260	370	90	180	380
CJA	150	160	215	265	100	160	320

equal to 170.

3. Let $J_{1,3}$ be the critical job. Job $J_{1,3}$ is also the lowest priority job, so $b_{1,3}$ is equal to 215.
4. The bound $\hat{c}_{1,3}$ is equal to $\max\{b_{1,1}, b_{1,2}, b_{1,3}\} = 215$.

We note that when computing the bounds $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$, and hence the final bound, the delay from every job in $\mathbf{J}_2$ is counted only once. As a result, the bound obtained by Algorithm CJA for $J_{1,3}$ is tighter than that obtained by Algorithm ERT. For the same reason, the bounds obtained by Algorithm CJA for $J_{1,2}$, $J_{1,4}$, $J_{2,2}$, and $J_{2,3}$ are tighter than those computed by Algorithm ERT. Table III lists the bounds on the completion times computed by Algorithm CJA for all the jobs in Example 1. For the sake of comparison, the bounds computed by Algorithm ERT are also listed in the table.

When no job has a critical section every bound obtained by Algorithm CJA is tighter than the corresponding bound obtained by Algorithm ERT. To compare the bounds we let $\hat{c}_{i,j}$ denote the bound on the completion time of $J_{i,j}$ computed by Algorithm ERT. From Eqs. (1), (4), and (5), we can write $\hat{c}_{i,j}$ as

$$\hat{c}_{i,j} \geq \hat{c}_{i,j-1} + e^{+}_{i,j} + \mathit{totalInter}(J_{i,j}, \mathbf{J}) \qquad (7)$$

and

$$\hat{c}_{i,j} \geq r_{i,j} + e^{+}_{i,j} + \mathit{totalInter}(J_{i,j}, \mathbf{J}) \qquad (8)$$

for the special case where the delay due to blocking is zero for every job. Given any k $(1 \leq k \leq j)$, we expand $\hat{c}_{i,j}$ recursively using the above two inequalities to obtain

$$\hat{c}_{i,j} \geq r_{i,k} + \sum_{l=k}^{j} e^{+}_{i,l} + \sum_{l=k}^{j} \mathit{totalInter}(J_{i,l}, \mathbf{J}) \qquad (9)$$

On the other hand, Step 1(b) of Algorithm CJA in Figure 6 states that

$$b_{i,k} = r_{i,k} + \sum_{l=k}^{j} e^{+}_{i,l} + \mathit{totalInter}(J_{i,low}, \mathbf{J}) \qquad (10)$$

where $1 \leq k \leq j$ and $k \leq low \leq j$. Comparing Eqs. (9) and (10) we can see that in this special case the bound computed by Algorithm ERT is greater than or equal to every $b_{i,k}$ computed by Algorithm CJA and hence is greater than or equal to the maximum of $b_{i,k}$'s, which is the final bound computed by Algorithm CJA.

Table III shows that in general the bounds computed by Algorithm CJA are not always tighter than those computed by Algorithm ERT. In this example, the length of the critical sections are relatively large compared with the execution time of the jobs. (We choose these numbers to illustrate the effect of blocking.) The critical sections of actual systems are likely to be relatively short. Since Algorithm CJA considers subchains of the target job, it is expected to perform better than Algorithm ERT on average. The simulation results in Section VI support this conclusion.

V. Algorithm ITR

An obvious drawback of the previous algorithms is that the release times of jobs are not taken into account. For example, when the maximum delay suffered by $J_{1,1}$ in Example 1 is computed, the critical section of $J_{2,3}$ is counted in the delay. However, we notice that $J_{2,3}$ will not be released until time 120, by which time $J_{1,1}$ should have completed even when $J_{1,1}$ has its maximum execution time and is preempted by $J_{2,1}$ and $J_{2,2}$. Hence one improvement is to remove from consideration the jobs (such as $J_{2,3}$ in this example) that cannot possibly delay with the execution of the target job. In other words, in Step 1(b) of Algorithm CJA, if we can prune some jobs from the job set $\mathbf{J}$ that cannot possibly execute in the critical interval, we can obtain a tighter bound on the maximum possible delay the job $J_{i,j}$ might suffer. Clearly, for each target job the pruning process must be done for every assumed critical job because different jobs may be pruned for different combinations.

The next question is how to obtain the information we need to prune jobs properly. One approach is called *pessimistic iteration*. First we use Algorithm CJA to obtain an initial bound on the completion time of every job. We then iteratively apply the modified Algorithm CJA to obtain a new bound on the completion time of each job. When bounding the duration of the critical interval for each pair of target job $J_{i,j}$ and assumed critical job $J_{i,c(j)}$, the modified Algorithm CJA excludes from consideration any job $J_{k,l}$ whose interval $(r_{k,l}, \hat{c}_{k,l}]$ does not overlap with the interval $(r_{i,c(j)}, \hat{c}_{i,j}]$, where $\hat{c}_{k,l}$ and $\hat{c}_{i,j}$ are the bounds on the completion times of $J_{k,l}$ and $J_{i,j}$, respectively, obtained in the initial step or the previous iteration step. This pruning process is safe because $\hat{c}_{i,j}$'s computed in the initial step and each of the previous iteration step are correct upper bounds on the completion times of jobs and, hence, all the pruned jobs cannot execute in the critical interval $(r_{i,c(j)}, c_{i,j}]$. The iteration will terminate when all the new bounds obtained in the current step are equal to the corresponding bounds obtained in the previous step. Obviously, during each iteration before termination, at least one bound on the completion time of a job is strictly smaller than its corresponding previous one. Since bounds cannot be arbitrarily small, the iteration will terminate in a finite number of steps.

Although the pessimistic iteration approach improves the bounds in general, it does not provide a tight bound in our example. We notice that job $J_{1,1}$ in Example 1 is both the target job and the critical job. The initial bound on the completion time of $J_{1,1}$ is 150. Based on this bound, interval $(r_{1,1}, \hat{c}_{1,1}]$ overlaps with critical interval $(r_{2,2}, \hat{c}_{2,2}]$ of job $J_{2,2}$. Consequently $J_{2,2}$ and $J_{2,3}$ will not be pruned by the pessimistic iteration approach.

A more aggressive approach is called *optimistic iteration.* Contrary to pessimistic iteration, optimistic iteration starts with an optimistic bounds on the completion times of jobs, obtained by computing the completion time as if the chain containing the target job is executed in isolation. During each subsequent iteration, we use the modified CJA algorithm to obtain a new bound on the completion time of each job $J_{i,j}$ based on bounds obtained in either the previous iteration step or the initial step. Like pessimistic iteration, for each pair of critical job $J_{i,c(j)}$ and target job $J_{i,j}$, we prune any job $J_{k,l}$ whose interval $(r_{k,l}, \hat{c}_{k,l}]$ does not overlap with the interval $(r_{i,c(j)}, \hat{c}_{i,j}]$. The iteration will terminate when all the new bounds are equal to the corresponding bounds obtained in the previous step.

Figure 7 lists the pseudo code of Algorithm ITR, which uses the optimistic iteration approach. It is essentially a loop which is preceded by an initial step. Inside the loop, Algorithm CJA is applied but is preceded by two extra steps. Step 2(biA) and Step 2(biB) are inserted to prune the jobs $J_{u,v}$ whose intervals $(r_{u,v}, \hat{c}_{u,v}]$ do not overlap with the interval $(r_{i,c(j)}, \hat{c}_{i,j}]$. Because of the extra pruning steps, the bounds obtained at the end of the loop body are always no larger than the corresponding bounds obtained by Algorithm CJA without any pruning. So are the final bounds when the iteration terminates.

The correctness of Algorithm ITR is stated formally by the following two theorems. Their proofs are in the appendix.

Theorem 1: Algorithm ITR terminates after a finite number of iterations.

Input:

A set $\mathbf{J}$ of jobs where each job $J_{i,j}$ has a release time $r_{i,j}$, a priority $\phi_{i,j}$, a range $[e^-_{i,j}, e^+_{i,j}]$ of execution times, and a maximum duration $D_{i,j}$ of critical sections in the job.

Output: A bound $\hat{c}_{i,j}$ on the completion time of each job $J_{i,j}$.

Algorithm:

1. For each job $J_{i,j}$,
   (a) if $j = 1$, $\hat{c}_{i,j} = r_{i,j} + e^+_{i,j}$;
   (b) otherwise, $\hat{c}_{i,j} = \max\{\hat{c}_{i,j-1}, r_{i,j}\} + e^+_{i,j}$;
   (c) $\hat{c'}_{i,j}$, the bound on the completion time of $J_{i,j}$ computed in the previous iteration, is 0.
2. Repeat until $(\hat{c}_{i,j} = \hat{c'}_{i,j})$ for every job $J_{i,j}$,
   (a) for each job $J_{i,j}$, $\hat{c'}_{i,j} = \hat{c}_{i,j}$;
   (b) for each target job $J_{i,j}$,
      i. for each possible critical job $J_{i,k}$ $(1 \le k \le j)$,
         A. $\mathbf{J}' = \mathbf{J}$;
         B. purge from $\mathbf{J}'$ any job $J_{u,v}$ $(u \neq i)$ for which the interval $(r_{u,v}, \hat{c'}_{u,v}]$ does not overlap with the interval $(r_{i,k}, \hat{c'}_{i,j}]$;
         C. find the job $J_{i,low}$ such that $J_{i,low}$ has the lowest priority among job $J_{i,k}$, $J_{i,k+1}$, ..., $J_{i,j}$;
         D. compute the bound $b_{i,k}$ by
         $$b_{i,k} = r_{i,k} + \sum_{l=k}^{j} e^+_{i,l} + block(J_{i,k}, \mathbf{J}') + totalInter(J_{i,low}, \mathbf{J}').$$
      ii. let $\hat{c}_{i,j} = \max_{1 \le k \le j}\{b_{i,k}\}$.

Fig. 7. Algorithm ITR

Theorem 2: The bounds obtained in the last iteration of Algorithm ITR are correct upper bounds on the completion times of jobs.

From the proof of Theorem 1, we see that there can be no more than $O(N^3)$ number of iteration steps. Since each iteration has the same complexity as Algorithm CJA, which is $O(N^3)$, the complexity of Algorithm ITR is thus $O(N^6)$.

As an example, we apply Algorithm ITR to bound the completion time of $J_{1,1}$ in Example 1. The initial optimistic bound is 40, which is equal to the release time of $J_{1,1}$ plus its maximum execution time. During the first iteration, the interval $(r_{2,1}, \hat{c}_{2,1}]$ overlaps with $(r_{1,1}, \hat{c}_{1,1}]$, and therefore $J_{2,1}$ is retained in $\mathbf{J}'$ at Step 2(biB). The intervals $(r_{2,2}, \hat{c}_{2,2}]$ and $(r_{2,3}, \hat{c}_{2,3}]$, however, do not overlap with the interval $(r_{1,1}, \hat{c}_{1,1}]$.

TABLE IV
BOUNDS COMPUTED BY THE THREE ALGORITHMS AND THE ACTUAL WORST-CASE COMPLETION TIMES

	$J_{1,1}$	$J_{1,2}$	$J_{1,3}$	$J_{1,4}$	$J_{2,1}$	$J_{2,2}$	$J_{2,3}$
ERT	100	170	260	370	90	180	380
CJA	150	160	215	265	100	160	320
ITR	50	60	205	255	50	110	290
Worst	50	60	130	240	50	110	250

Therefore $J_{2,2}$ and $J_{2,3}$ are pruned at Step 2(biB). The new bound on the completion time of $J_{1,1}$ becomes 50. During the second and later iterations, the intervals $(r_{2,2}, \hat{c}_{2,2}]$ and $(r_{2,3}, \hat{c}_{2,3}]$ still do not not overlap with the interval $(r_{1,1}, \hat{c}_{1,1}]$, and jobs $J_{2,2}$ and $J_{2,3}$ are always pruned. Consequently the bound on the completion time of $J_{1,1}$ remains 50.

The final bounds on the completion times of all jobs in Example 1 obtained by (optimistic) Algorithm ITR are listed in Table IV. We also list the bounds obtained by Algorithm ERT and Algorithm CJA, as well as the actual worst case completion times of the jobs. We note that although Algorithm ITR gives fairly tight bounds compared with the other two algorithms, it may still fail to find the actual worst-case completion times. Take job $J_{1,3}$ for example. Although Algorithm ITR has correctly determined that the completion of $J_{1,3}$ is delayed only by $J_{2,2}$, it fails to see that the maximum delay caused by $J_{2,2}$ is less than the maximum execution time of $J_{2,2}$ because $J_{2,2}$ is released 15 time units earlier than $J_{1,3}$. It also fails to see that in the worst-case $J_{1,3}$ will be ready by time 100 and that the critical section of $J_{2,3}$ will never be able to block it.

VI. Performance of the Algorithms

From the previous discussion, we know that bounds yielded by Algorithm ITR are tighter than those yielded by Algorithm CJA, which in turn may be tighter than those yielded by Algorithm ERT, but we do not know by how much. To quantify their relative merits and to determine how their relative performance depends on the characteristics of jobs, we perform a series of simulation experiments. This section discusses the criterion used to evaluate their performance, the method used to generate the workload, and finally the simulation results.

A. Performance Criterion

The performance criterion we use to compare two algorithms, say A and B, is the *bound ratio* or the *average bound ratio* of A over B. The ratios are defined as follows. For a given system of jobs, the bound ratio of (algorithm) A over (algorithm) B for a job is the ratio of the upper bound on the response time of the job obtained by A over the corresponding bound obtained by B. The bound ratio of the system is the average of the bound ratios of all the jobs in the system. In our experiment, we generate many synthetic systems and compute the bound ratio for each system. The average bound ratio is the average of the bound ratios of all the systems with the same characteristics examined in the experiment. Obviously the smaller the average bound ratio of A over B, the better algorithm A is compared with algorithm B, provided that the ratio is less than 1.

B. The generation of workload

Through preliminary experiment, we found that performance of the algorithms depends almost entirely on three factors. They are the number of job chains in the system, the number of jobs in each job chain and the *density of the schedule*, or the *schedule density*. Intuitively, the density of a schedule indicates how "sparse" the schedule is. It can be quantified by the *density factor*, which is the total maximum execution time of all jobs divided by the range of release times of jobs. For example, if the release times of jobs are distributed in the range of $[1, 1000]$ and the total maximum execution time of all jobs is equal to 1500, then the schedule density factor is equal to 1.5. The smaller the schedule density factor, the "sparser" the schedule.

A *configuration* is a unique combination of values of the above three factors. We say that synthetic systems have the same configuration when they have the same number of job chains, number of jobs per job chain, and schedule density. In our simulation experiment, we examined configurations with the number of job chains being 5, 10, or 15, the number of jobs per job chain being 1, 2, 5, or 10, and the schedule density being 0.5, 1, or 2. We thus have 36 configurations. For each configuration, we generated 1000 systems to yield negligibly small confidence intervals for all the average values presented below.

Each system of a configuration with x job chains, y jobs per chain and schedule density z, is generated as follows. For each of the x job chains and each of the y jobs in the chain, we choose the release time of the job from a uniform distribution in the range [1,1000000]. We then sort the jobs in each job chain in increasing order by their release times and add a precedence constraint to each pair of adjacent jobs in the job chain.

To choose the execution times of the jobs, we first compute the total maximum execution time of all jobs by multiplying the schedule density z by the range of job release times, 1000000. We then randomly divide the total maximum execution time among the xy jobs. This is done by first generating an *execution factor* for each job, which is uniformly distributed in range $[0.01, 1]$. We obtain the normalized execution factor for each job by dividing its execution factor by the sum of execution factors of all jobs. The maximum execution time of each job is then equal to its normalized execution factor times the total maximum execution time. We let the minimum execution time of every job to be 0. Finally, the priority of every job is randomly distributed in range $[1, 10000]$.

The maximum duration of a critical section of a job $J_{i,j}$ is obtained by multiplying the maximum execution time $e^+_{i,j}$ by a *blocking factor*, which is uniformly distributed in the range $[0, d]$. Because preliminary experiments showed that the bound ratios are not sensitive to the value of d, its value was fixed at 1. Thus the blocking factor of each job is uniformly distributed in the range $[0, 1]$, and the maximum duration of a critical section ranges from 0 to the maximum execution time.

C. Comparison of Algorithm ERT and Algorithm CJA

In this subsection, by "bound ratio" we mean the average bound ratio of Algorithm CJA over Algorithm ERT. The simulation results show that bound ratios are not sensitive to the number of job chains in the system and the critical section density. For this reason, we present in Figure 8 the bound ratio as a function of the number of jobs in each chain and the schedule density. Each value in the figure is the average value of the bound ratios of all systems of a configuration. The overall average bound ratio for all configurations is 0.77 which indicates that on the average the bounds on the job response times computed by Algorithm CJA are 23% shorter than the bounds computed by Algorithm ERT.

In the figure, we notice that the average bound ratio decreases as the number of jobs in each job chain increases. A closer examination reveals that this is because the bound ratio for an individual job is strongly correlated with the position of the job in the job chain. Figure 9 depicts the average bound ratio of jobs as a function of their position in the job chains. The value on the horizontal-axis is the position of jobs in their corresponding job chains, and the corresponding value on the vertical-axis is the average bound ratio for all jobs in that position. We notice that for the first job on every job chain, Algorithm CJA and Algorithm ERT yield the same bound on the average. If Algorithm ERT used Eq.(1), both algorithms would in fact do the same

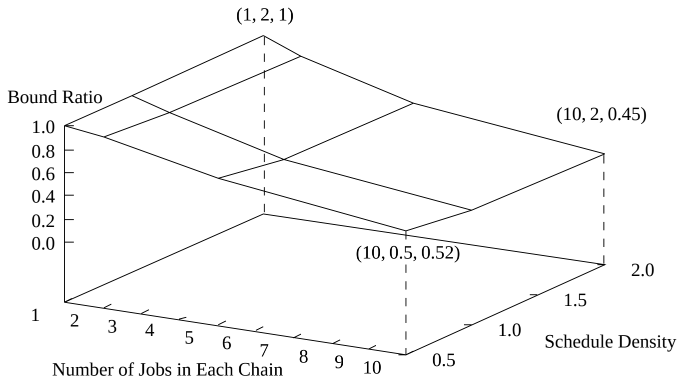

Fig. 8. Bound Ratio of Algorithm CJA over Algorithm ERT

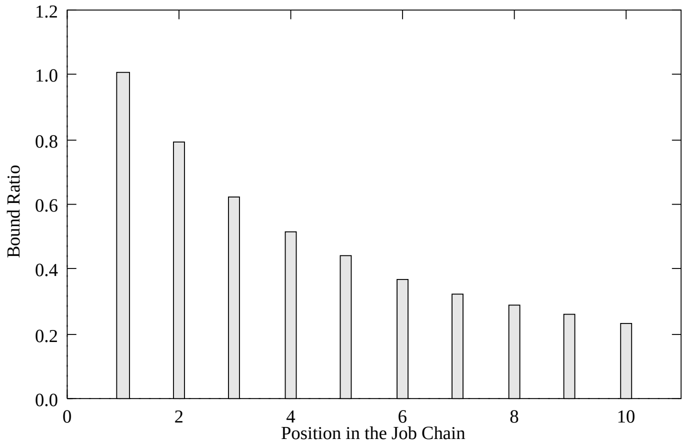

Fig. 9. Bound Ratio as Function of the Position of Jobs

computation for these jobs. However, the fact is that Algorithm ERT uses Eq.(3), which can produce a tighter bound. This result indicates that instances where the bound produced by Algorithm ERT is tighter than the corresponding bound produced by Algorithm CJA are rare. The average bound ratio decreases as the number of predecessors of the target job increases, largely due to the fact that Algorithm ERT sometimes counts the interference and blocking of jobs multiple times. The later a job is in a job chain, the more likely Algorithm ERT is to do so. As a result, a system with longer job chains has a smaller average bound ratio. Figure 8 shows that the bound ratio also decreases a little as the schedule density increases, for a similar reason.

D. Comparison of Algorithm CJA and Algorithm ITR

In this subsection we focus on Algorithms CJA and ITR, and by "bound ratio" we mean the average bound ratio of Algorithm ITR over Algorithm CJA. The overall average bound ratio is 0.51 for all config-

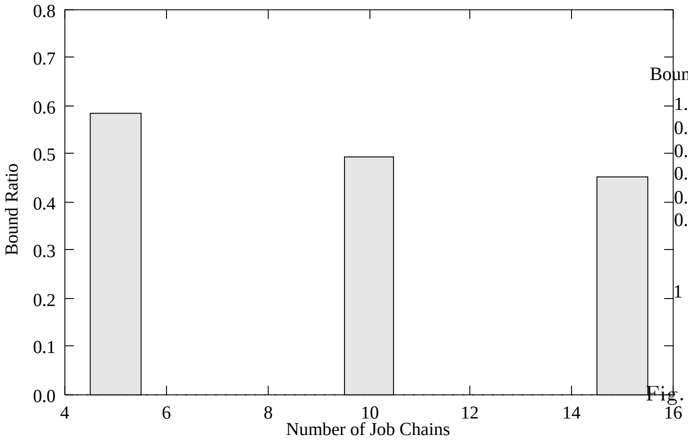

Fig. 10. Bound Ratio as a Function of the Number of Job Chains in the System

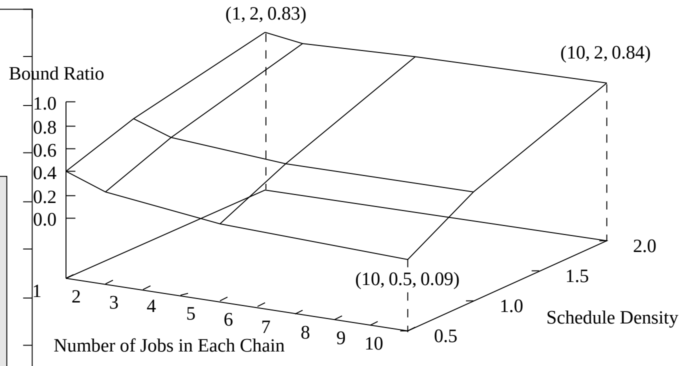

Fig. 11. Bound Ratio of Algorithm ITR over Algorithm CJA

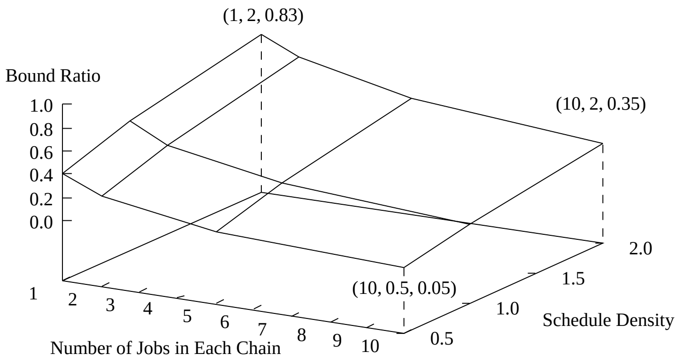

Fig. 12. Bound Ratio of Algorithm ITR over Algorithm ERT

urations, which indicates that on average the bounds on job response times computed by Algorithm ITR are about half the bounds computed by Algorithm CJA.

Figure 10 depicts the average bound ratio as a function of number of job chains in the system and shows that the bound ratio of Algorithm ITR over Algorithm CJA varies slightly but noticeably with the number of job chains in the system. When the number of job chains increases while the other two parameters remain constant, the delay due to interference and blocking from jobs in different chains increases. Due to the pruning step, Algorithm ITR can better accommodate the effect of the increase than Algorithm CJA. As a result, Algorithm ITR obtains tighter bounds as the number of job chains in a system increases.

Figure 11 shows the average bound ratio as a function of number of jobs in each job chain and the schedule density. The bound ratios are much smaller when the schedule densities are smaller, indicating that Algorithm ITR is much more effective for sparse schedules. When the schedule is sparse, many jobs execute in isolation and do not delay each other. The pruning step in Algorithm ITR can correctly detect this and obtain tighter bounds, while Algorithm CJA does not have this capability.

E. Summary of the Simulation Results

Figure 12 shows the bound ratio of Algorithm ITR over Algorithm ERT as a function of the number of jobs in each job chain and the schedule density. As we expect, the bounds yielded by Algorithm ITR are much tighter than those by Algorithm ERT. In summary, we see a great reduction in the upper bounds on job response times by Algorithm ITR over Algorithms CJA and ERT. Furthermore, Algorithm ITR is more effective when the schedule is sparse and the number of jobs in the system is large. When the schedule is "dense", the performance of Algorithm CJA is close to that of Algorithm ITR.

VII. Extensions and Future Work

One obvious extension is to combine Algorithm ERT and Algorithm CJA, selecting the smaller of the two bounds for each job. A combined algorithm would yield better bounds than Algorithm CJA in Example 1 yet have the same complexity as Algorithm CJA (i.e., $O(N^3)$). However, the results presented in Section VI suggest that little improvement can be expected on the average.

Thus far we have assumed that the release times of all the jobs are fixed. In real systems, the release time of each job is often known to lie within a range, while the actual release time is not known. The three algorithms presented above can be modified to deal with jittered release times, i.e., the actual release time of each job is in a range of $[r^-_{i,j}, r^+_{i,j}]$. In the case of Algorithm ERT, we simply replace all $r_{i,j}$'s with corresponding $r^+_{i,j}$'s. The bounds computed by the modified Algorithm ERT are correct.

In the case of Algorithm CJA, we first need to redefine the critical job of the target job as follows: For a target job $J_{i,j}$, the critical job $J_{i,c(j)}$ is the last job in $\mathbf{J_i}$ before and including $J_{i,j}$ whose ready time is in its release time range $[r^-_{i,c(j)}, r^+_{i,c(j)}]$. By the new definition, the critical job analysis remains correct in bounding the duration of interval $(r^+_{i,c(j)}, c_{i,j}]$. Consequently, the pseudo code of Algorithm CJA remains correct if we replace $r_{i,k}$ with $r^+_{i,k}$ at Step 1(b) in Figure 6.

To see how to take into account release time jitter in the case of Algorithm ITR, we note that in the initial steps, Step 1(a) and 1(b) in Figure 7, we should replace the release time $r_{i,j}$ with the latest possible release time $r^+_{i,j}$. As a consequence, the initial bounds are conservative. In Step 2(biB), we need to prune non-interfering jobs from $\mathbf{J}'$. Due to jittered release, the interval of a job $J_{u,v}$ becomes $(r^-_{u,v}, \hat{c'}_{u,v}]$. Similarly, the critical interval between the assumed critical job $J_{i,k}$ and the target job $J_{i,j}$ becomes $(r^-_{i,k}, \hat{c'}_{i,j}]$. We can thus test if a job $J_{u,v}$ delays the target job based on whether these two intervals overlap. Lastly, when we compute each individual $b_{i,k}$ for each assumed critical job $J_{i,k}$ at Step 2(biD), we simply replace $r_{i,k}$ with $r^+_{i,k}$ for a correct final bound.

A problem related to this work is to find the exact worst-case completion time. Specifically, the pruning technique used in Algorithm ITR can effectively reduce a large number of combinations when exhaustive searching is used to find the worst-case completion time.

In the algorithms above, we have assumed that access to resources was controlled by the NPS protocol. This protocol has the virtue of being simple to implement. However, all higher priority jobs may be blocked even if they do not contend for any resource. The Priority-Ceiling Protocol (PCP) [10] and the Stack-Based Protocol (SBP) [11] do not suffer from this shortcoming. To take advantage of this desirable property, we are refining Algorithms ERT, CJA and ITR to use either PCP or SBP. The results of this work are reported in [12] which also describes modifications to these algorithms to achieve tighter bounds.

VIII. Conclusions

We have described three algorithms that bound the completion times of jobs in independent chains when the jobs have variable execution times, arbitrary release times, fixed priorities, and when access to shared resources must be controlled. The algorithms have different complexities and yield different performance. Our simulation results show that Algorithm ITR consistently produces tighter bounds than the other two algorithms. The example presented here suggests that the bounds obtained by Algorithm ITR are close to the actual worst-case completion times. The complexity of Algorithm ITR is $O(N^6)$, where N is the total number of jobs in a system. This complexity is not a problem for off-line analysis. When this complexity is too high, e.g., for the purpose of on-line admission control, Algorithm CJA is a good alternative choice, especially when the schedule is expected to be "dense".

Acknowledgments

This work was supported in part by NSF Grant No. NSF CCR 92-24269 and by NASA Contract NAG 1-613.

Appendix A
Proof of Theorem 1

The bound of the completion time of each job obtained in the first iteration is no smaller than the corresponding bound obtained in the initial step, due to the optimistic estimation used in the initial step. As a consequence, for each pair of target job and critical job considered at Step 2(bi), fewer or the same number of jobs are pruned at Step 2(biB) in the second iteration step, making the value of $b_{i,k}$ greater than or equal the corresponding one obtained in the first iteration step. Hence every bound obtained in the second iteration step is no smaller than the corresponding bound obtained in the first iteration step. In general, the bounds obtained in each iteration step are monotonically non-decreasing.

The iteration continues if and only if for at least one job the new bound is greater than the corresponding bound obtained in the previous step. For a new bound of a job to be greater than its previous bound, fewer jobs must have been pruned from set $\mathbf{J}'$ at Step 2(biB) in the current iteration step than in the previous step. Overall, in each iteration step, the total number of jobs pruned at Step 2(biB) must decrease at least by one. Since the total number of jobs that can be pruned at Step 2(biB) cannot exceed $N^2(N-1)$ in the first iteration step, the iterative procedure will terminate in a finite number of iterations. ∎

Appendix B
Proof of Theorem 2

We prove this theorem by an induction over the jobs in the increasing order of their release times. By convention, let $\hat{c}_{i,j}$ denote the bound on the completion time of $J_{i,j}$ obtained by Algorithm ITR, i.e., the bound obtained by the last two iteration steps, and let $c_{i,j}$ denote the actual completion time of $J_{i,j}$ in the schedule.

Induction basis: Because the release times of all jobs are consistent with their precedence constraints, the job with the earliest release time among all jobs must be $J_{i,1}$ for some i. Suppose that the actual completion time $c_{i,1}$ of $J_{i,1}$ is greater than the bound $\hat{c}_{i,1}$. Let Γ denote the total amount of execution times of all jobs, excluding $J_{i,1}$, that execute in interval $(r_{i,1}, \hat{c}_{i,1}]$. We must have $\Gamma + e_{i,1} > \hat{c}_{i,1} - r_{i,1}$ (Otherwise job $J_{i,1}$ would have completed by time $\hat{c}_{i,1}$). Hence $\Gamma + e^{+}_{i,1} > \hat{c}_{i,1} - r_{i,1}$.

Now let us focus on Steps 2(bi) during the last (outer-most) iteration step in Algorithm ITR, specifically when $J_{i,1}$ is the target job and the critical job. In the last iteration step, for every job $J_{x,y}$, the bound $\hat{c}_{x,y}$ obtained is the same as the bound obtained in the second last iteration step, which is copied to $\hat{c'}_{x,y}$. Consequently the jobs pruned at Step 2(biB) are those whose release times are later than $\hat{c}_{i,1}$. Job set $\mathbf{J}'$ obtained at this step thus gives all the jobs that can execute in interval $(r_{i,1}, \hat{c}_{i,1}]$.

Obviously, every job that executes in interval $(r_{i,1}, \hat{c}_{i,1}]$ must have priority higher than or equal to $J_{i,1}$ or must block $J_{i,1}$. Thus $\mathit{totalInter}(J_{i,1}, \mathbf{J}') + \mathit{block}(J_{i,1}, \mathbf{J}')$ gives an upper bound on Γ. By Step 2(biD) and 2(bii), we have

$$\begin{aligned}\Gamma + e^{+}_{i,1} \quad &\leq \quad \mathit{totalInter}(J_{i,1}, \mathbf{J}') + \mathit{block}(J_{i,1}, \mathbf{J}') \\ &\quad + e^{+}_{i,1} = b_{i,1} - r_{i,1} = \hat{c}_{i,1} - r_{i,1}\end{aligned}$$

This is a contradiction to the conclusion stated above. Therefore the hypothesis must be wrong, and for job $J_{i,1}$ Algorithm ITR yields a correct upper bound on its completion time.

Induction: Now we let $J_{i,j}$ be the job whose release time is later than the release times of k other jobs. As an induction hypothesis, we assume that the completion time of every job released before $J_{i,j}$ is no larger than the upper bound on its completion time obtained by Algorithm ITR. We will now prove that $c_{i,j}$ is no larger than $\hat{c}_{i,j}$ either.

We, again, prove this by contradiction. Suppose that $c_{i,j}$ is larger than $\hat{c}_{i,j}$. Let $J_{i,c}$ $(1 \leq c \leq j)$ be the critical job for $J_{i,j}$ in this schedule, and Γ be the total amount of execution times of all jobs from chains other than $\mathbf{J}_i$ that execute in interval $(r_{i,c}, \hat{c}_{i,j}]$. Since job $J_{i,j}$ is not completed by $\hat{c}_{i,j}$, we must have

$$\Gamma + \sum_{l=c}^{j} e^{+}_{i,l} \geq \Gamma + \sum_{l=c}^{j} e_{i,l} > \hat{c}_{i,j} - r_{i,c}$$

Now let us focus on Steps 2(bi) during the last (outer-most) iteration step in Algorithm ITR, specifically when $J_{i,j}$ is the target job and $J_{i,c}$ is the critical job. If job $J_{u,v}$ is pruned at Step 2(biB), then either (1) $\hat{c}_{u,v} < r_{i,c}$, or (2) $r_{u,v} > \hat{c}_{i,j}$. If a job $J_{u,v}$ is pruned due to the first reason, its release time must be earlier than that of $J_{i,j}$. By induction hypothesis, the bound $\hat{c}_{u,v}$ is a correct upper bound on the completion time. Hence we are certain that job $J_{u,v}$ cannot execute in interval $(r_{i,c}, \hat{c}_{i,j}]$. On the other hand, if $J_{u,v}$ is pruned due to the second reason, it cannot execute in interval $(r_{i,c}, \hat{c}_{i,j}]$ either. Thus the new job set $\mathbf{J}'$ obtained at Step 2(biB) contains all the possible jobs that can execute in interval $(r_{i,c}, \hat{c}_{i,j}]$.

Since no job with priority lower than $J_{i,low}$, obtained at Step 2(biC), can execute in interval $(r_{i,c}, \hat{c}_{i,j}]$ unless it blocks the critical job $J_{i,c}$, $\mathit{totalInter}(J_{i,low}, \mathbf{J}') + \mathit{block}(J_{i,c}, \mathbf{J}')$ will give an upper bound on Γ, the total execution time of jobs that can execute in interval $(r_{i,c}, \hat{c}_{i,j}]$. By Step 2(biD) and Step 2(bii), we have

$$\begin{aligned}\Gamma + \sum_{l=c}^{j} e^{+}_{i,l} \quad &\leq \quad \mathit{totalInter}(J_{i,low}, \mathbf{J}') + \mathit{block}(J_{i,c}, \mathbf{J}') \\ &\quad + \sum_{l=c}^{j} e^{+}_{i,l} = b_{i,c} - r_{i,c} \leq \hat{c}_{i,j} - r_{i,c}\end{aligned}$$

This contradicts the conclusion we obtained in the previous paragraph. The hypothesis must be wrong; we must have $c_{i,j} \leq \hat{c}_{i,j}$. By induction, we know that for every job in this schedule its completion time is no longer than the corresponding bound computed by Algorithm ITR. ■

References

[1] A. K. Mok, *Fundamental design problems of distributed systems for the hard real-time environment*, Ph.D. thesis, MIT, 1983.

[2] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors," in *9th IEEE Real-Time Systems Symposium*, Dec. 1988, pp. 259–269.

[3] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[4] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *10th IEEE Real-Time Systems Symposium*, Dec. 1989, pp. 166–171.

[5] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *11th IEEE Real-Time Systems Symposium*, Dec. 1990, pp. 201–209.

[6] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[7] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Timing analysis for fixed-priority scheduling of hard real-time systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 1, pp. 13–28, Jan. 1994.

[8] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda, "Fixed priority scheduling theory for hard real-time sys-

tems," in *Foundations of Real-Time Computing, Scheduling and Resource Management*, A. M. Tilborg and G. M. Koob, Eds., chapter 1. Kluwer Academic Publishers, 1991.

[9] R. Ha, *Validating Timing Constraints in Multiprocessor and Distributed Systems*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, 1995.

[10] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, Sept. 1990.

[11] T. P. Baker, "A stack-based resource allocation policy for real-time processes," in *11th IEEE Real-Time Systems Symposium*, 1990, pp. 191–200.

[12] M. K. Gardner, "Resource sharing among linear job chains," Tech. Rep. UIUCDCS-R-97-2010, University of Illinois at Urbana-Champaign, Department of Computer Science, Aug. 1997.

Jun Sun received his BS degree from Shanghai Jiao Tong University, China, in 1989 and received his Ph.D. degree in computer science from University of Illinois at Urbana-Champaign, 1997. He is now a Design Engineer at Geoworks Inc. in California. His research interest includes real-time scheduling, embedded systems, distributed computing and operating systems. He is the recipient of the Outstanding Paper Award in the 16th International Conference on Distributed Computing Systems. Dr. Sun is a member of IEEE and Upsilon Pi Epsilon.

Mark K. Gardner received his B.S. degree with honors in mechanical engineering and his M.S. in computer science from Brigham Young University (Provo, UT) in 1986 and 1994, respectively. He is currently a Ph.D. student in computer science at the University of Illinois at Urbana-Champaign. Before returning to school to pursue his M.S. degree, he worked as an aerodynamic engineer for Allied-Signal Aerospace, Garrett Auxiliary Power Division (Phoenix, AZ). His research interests include real-time systems, programming languages, operating systems, performance analysis, and software engineering. He is a student member of the IEEE and ACM. He is also a member of the American Society of Mechanical Engineers and Tau Beta Pi.

Jane W.S. Liu received her BSEE degree in 1959 from the Cleveland State University, Ohio. She received her MS and ScD degrees in 1966 and 1968, respectively, from the Massachusetts Institute of Technology. She is currently a professor of computer science at the University of Illinois at Urbana-Champaign. Before joining the University of Illinois in 1973, she worked as an electronics engineer for the U.S. Department of Transportation, the Mitre Corporation, and the Radio Corporation of America. Her research interests are in the areas of real-time systems, distributed systems, and networks. She served as chair of the IEEE Computer Society Technical Committee on Data Base Engineering in 1981 and 1982, and chair of the Technical Committee on Distributed Processing in 1989 and 1990. She is the editor-in-chief of the IEEE Transactions on Computers and an associate editor of the International Real-Time Systems Journal. She is a fellow of the IEEE and a member of the ACM.