Algorithms for Testing Fault-Tolerance of Sequenced Jobs

Marek Chrobak[1] Jiří Sgall[2]

Technical Report UCR-CS-00-06
Department of Computer Science
University of California
Riverside

[1]Department of Computer Science, University of California, Riverside, CA 92521. Email: marek@cs.ucr.edu. Research supported by NSF grant CR-9988360.

[2]Mathematical Institute, Academy of Sciences of the Czech Republic, Zitna 25, CZ-11567 Praha 1, Czech Republic. Email: sgall@math.cas.cz. Partially supported by grant A1019901 of GA AV ČR and postdoctoral grant 201/97/P038 of GA ČR.

Abstract

We study he problem of testing whether a given set of sequenced jobs can tolerate transient faults. We present efficient algorithms for this problem in several fault models.

We give an $O(n)$-time fault-tolerance testing algorithm if the number of faults does not exceed a given parameter k. Then we consider the model in which two faults are separated by a gap of length at least Δ. For the faults that can be detected immediately, we give an $O(n)$-time algorithm, and for the faults that can only be detected after a job's completion, we given an algorithm with time complexity $O(n^2)$. All our algorithms are very simple and easy to implement.

0.1 Introduction

Ghosh, Melhem and Mossé [GMM95] studied the problem of testing fault-tolerance of a collection of sequenced jobs. More specifically, we are given a sequence J of jobs, with release times, deadlines, and processing times. The jobs in J have already been sequenced, that is, their order of execution is known. Transient faults may occur when jobs are executed. If a fault occurs, the currently executed job is re-executed. In [GMM95] the authors assume that a fault can be detected only after the execution of a job is complete. We refer to such faults as *hidden*.

The question investigated in [GMM95] is whether all jobs in J will meet their their deadlines in the presence of faults. The answer is, obviously, negative when arbitrary fault patterns are allowed. However, with reasonable assumptions on the frequency of faults, the question becomes meaningful and, in some cases, non-trivial.

In [EKMMM99], the authors assume the fault model in which a gap between any two faults is at least Δ, and they present a dynamic programming, $O(n^2)$-time algorithm for testing fault-tolerance, if all jobs are released at the same time. In addition, they also present a linear-time heuristic for this problem. In [EKMMM99], the authors extend this linear-time heuristic to jobs with different release times, and describe its applications and experimental results.

A different fault model, in which the number of faults is bounded by some constant k, has been studied by Liberato, Melhem and Mossé [LMM99]. For this model, the authors give a $O(n^2k)$-time dynamic programming algorithm for testing fault-tolerance.

We consider both fault models from [GMM95, LMM99] in this paper. In addition to the hidden faults, we also consider *exposed* faults, which are faults that can be detected immediately.

In our discussion so far we have ignored the issue of scheduling, namely we assumed that each job is executed as early as possible, either at its release time or at the completion time of the previous job, whichever comes later. We refer to this as the *greedy schedule*. In the exposed-fault model, it is not difficult to see that this is the best we can do. However, for hidden faults, there are fault patterns for which it is beneficial, in some situations, to delay a job's execution. In Section 0.3 we prove that this cannot happen if the set of all possible fault sequences satisfies the following, natural *sparsifiability* property: if a certain fault sequence can occur, then any sparser sequence can occur as well (see Section 0.2 for precise definitions).

We give efficient fault-tolerance testing algorithms for several fault models. In case when the number of faults is at most k, we give an $O(n)$-time algorithm, improving the result from [LMM99]. Then we consider the faults that are separated by gaps at least Δ. For the exposed faults, we give an algorithm that runs in time $O(n)$. For hidden faults, our algorithm works in time $O(n^2)$. Unlike in [GMM95], our algorithm handles jobs with arbitrary release times. All our algorithms are simple, efficient, and easy to implement.

0.2 Terminology and Notation

Jobs and Schedules. By J we denote the set of n jobs, which are given by triples (r_j, d_j, p_j), where r_j is the *release time*, d_j is the *deadline*, and p_j is the *processing time* of job j. Without loss of generality, we assume that $0 \le r_i \le d_i$ and $r_i + p_i \le d_i$ for all i. A *schedule* of J is any sequence $s = (s_1, \ldots, s_n)$, such that $s_i \ge r_i$ for all i, and $s_{i+1} \ge s_i + p_i$ for $i < n$. We refer to s_j as the *scheduled start time* of job j. Thus job j occupies the machine during time interval $(s_j, s_j + p_j)$. We say that job j *meets the deadline* in schedule s if $s_j + p_j \le d_j$.

Without loss of generality, throughout the paper, we assume that $r_{i+1} \ge r_i + p_i$ for all $i < n$. For any set of jobs J we can modify, in linear time, the release times in J to satisfy this property, without affecting job completion times. The *greedy schedule* for J is defined by $s_i = r_i$, for all i.

Faults. By f we denote a fault sequence, $f = (f_1, f_2, \ldots)$. A *fault model* is a set F of potential fault sequences. F is called *sparsifiable* if for each $f \in F$, if $f_i - f_{i-1} \le g_i - g_{i-1}$ for each i, then $g \in F$ as well. Two particular sets of fault sequences that we consider are GAP_Δ, that consists of all sequences f in which $f_i - f_{i-1} \ge \Delta$ for each i, and $FAULTS_k$, that consists of sequences with at most k faults. Both models are sparsifiable.

Completion times. Fix J and F. By $S_j(s, f)$ and $C_j(s, f)$ we denote the *start time* and *completion time* of job j, if we execute the jobs according to schedule s and the fault sequence is f. The formal definition is recursive, and depends on the fault type. The start times are defined by $S_1(s, f) = s_1$ and $S_j(s, f) = \max\{s_j, C_{j-1}(s, f)\}$ for $j > 1$. The completion times are defined by

- For exposed-faults, $C_j(s, f)$ is the smallest $\tau \ge S_j(s, f) + p_j$ such that $f \cap [\tau - p_j, \tau) = \emptyset$, that is, the interval $[\tau - p_j, \tau)$ contains no faults.
- For hidden-faults, $C_j(s, f)$ is the smallest $\tau \ge S_j(s, f) + p_j$ such that $f \cap [\tau - p_j, \tau) = \emptyset$ and $\tau - S_j(s, f)$ is an integer multiple of p_j. Ghosh, Melhem and Mosse [GMM95] assume hidden faults.

By $C_j(s, F)$ we denote the maximum completion time of job j if the faults are from F, that is $C_j(s, F) = \max_{f \in F} C_j(s, F)$. Througout the paper, we will simplify notation by omitting the arguments that are understood from context, for example $S_j(s)$, $C_j(F)$, C_j, etc.

In either model, schedule s is called *F-tolerant*, if each job completes by its deadline, that is $C_j(s, F) \le d_j$ for all j. All algorithms we present will compute the maximum completion times C_j, for all j. Testing fault-tolerance, that is, whether $C_j \le d_j$ for all j, can then be done trivially in linear time.

0.3 Fault Tolerance and Greedy Schedules

Lemma 1 *For exposed faults, for any fault model F, if J has any F-tolerant schedule then the greedy schedule for J is F-tolerant.*

Proof: Fix any fault sequence f and schedules s, t such that $s_j \leq t_j$ for all j. It is enough to show that $C_j(s,f) \leq C_j(t,f)$ for all j. We show this inequality by induction on j. For $j = 1$, $C_0(s,f) \leq C_0(t,f)$ is obvious. Suppose that $j > 1$ and $C_{j-1}(s,f) \leq C_{j-1}(t,f)$. Then $\tau = C_j(t,f)$ satisfies $f \cap [\tau - p_j, \tau) = \emptyset$ and $\tau \geq S_j(t,f) + p_j = \max\{t_j, C_{j-1}(t,f)\} + p_j \geq \max\{s_j, C_{j-1}(s,f)\} + p_j = S_j(s,f) + p_j$. Thus $C_j(s,f) \leq C_j(t,f)$ as well. $\Box$

In the hidden-fault model, it is possible that J has an F-tolerant schedule even though the greedy schedule is not F-tolerant. For example, take $J = \{(0,5,3)\}$ (just one job) and $F = \{(1)\}$, one fault sequence. Schedule $s = (2)$ is F-tolerant, but the greedy schedule $s = (0)$ is not.

Lemma 2 *For hidden faults, for any sparsifiable fault model F, if J has any F-tolerant schedule then the greedy schedule for J is F-tolerant.*

Proof: It is sufficient to prove the following claim: If $s_j \leq t_j$ for all j, then for any job $b \in J$ and $f \in F$ there is $g \in F$ such that $C_b(s,f) \leq C_b(t,g)$. For if this claim holds, then t being F-tolerant implies that s is F-tolerant. It is also sufficient to prove the lemma if s and t differ in just one start time, say $t_m = s_m + \epsilon$ and $t_j = s_j$ for $j \neq m$, where $\epsilon > 0$ is sufficiently small so that $\epsilon < s_j - C_{j-1}(s,h)$, for all j and $h \in F$ for which $s_j - C_{j-1}(s,h) > 0$. Since, for a fixed s, there are only a finite number of possible values of $s_j - C_{j-1}(s,h)$, over all j and $h \in F$, such ϵ exists.

Pick the largest $a \leq b$ such that job a starts on time, that is $S_a(s,f) = s_a$. Thus jobs $a, a+1, \ldots, b$ execute back-to-back, some possibly twice. Without loss of generality we can assume that all faults in f occur between s_a and $C_b(s,f)$, since we can remove other faults without changing the value of $C_b(s,f)$.

If $m < a$, take $g = f$. Since $\epsilon \leq s_a - C_{a-1}(s,f)$, then $C_{a-1}(t,g) \leq C_{a-1}(t,f) \leq C_{a-1}(s,f) + \epsilon$. This means that jobs $a, a+1, \ldots, b$ are not affected by the delay of job m, so $C_b(s,f) = C_b(t,g)$. If $m > b$, we also take $g = f$, and $C_b(s,f) = C_b(t,g)$ as well. The last case is when $a \leq m \leq b$. In this case set $g_i = f_i$ for all $f_i \leq C_{m-1}(s,f)$ and $g_i = f_i + \epsilon$ for all $f_i \geq C_{m-1}(s,f) = S_m(s,f)$. In g, each fault will hit each job at exactly the same time (relative to its start time) as in f. Thus $C_b(s,f) \leq C_b(t,g)$, and we are done. $\Box$

From the two lemmas below, throughout the rest of the paper we can assume that the jobs are scheduled greedily. Also, we will say that J is F-tolerant if the greedy schedule for J is F-tolerant.

0.4 Sequences with at Most k Faults

In this section we give a linear-time algorithm for testing fault tolerance when $F = FAULTS_k$ consists of all sequences with at most k faults, where k is a given parameter. By the results from the previous section, we can assume that the jobs are scheduled according to the greedy schedule. The general idea of the algorithm is that in the worst case all faults will affect just one "critical" job and that each fault occurs at the end of an execution of a job.

Lemma 3 *For both fault types, if J tolerates all sequences of k faults that cause one job to execute completely $k+1$ times (that is, the k faults occur at the end of its first k executions), then it tolerates all sequences with k faults.*

Proof: It is sufficient to show that, in the greedy schedule, for each $b \in J$ and $f \in FAULTS_k$ there is $g \in FAULTS_k$ that satisfies the condition in the lemma and $C_b(g) \geq C_b(f)$.

Pick the smallest a such that the jobs $a, \ldots, b$ are executed back-to-back, that is $S_a(f) = r_a$ and $S_j(f) = C_{j-1}(f)$ for $j = a+1, \ldots, b$. Let e, $a \leq e \leq b$, be the job in this block that has the largest processing time p_e, and take g to be the sequence $g_i = r_e + ip_e$, $i = 1, \ldots, k$. Then g causes job e to execute completely $k+1$ times and, by the maximality of p_e, $C_b(g) \geq C_b(f)$. $\square$

The algorithm given below will compute the latest completion time C_j for each job j.

Algorithm 1. For $j = 1, \ldots, n$ do $C_j \leftarrow \max\{\, C_{j-1} + p_j \,,\, r_j + (k+1)p_j \,\}$.

That Algorithm 1 works in linear time is obvious. We show that the completion times are computed correctly. That the computed C_j are the lower bounds on the completion times is obvious. Now we show that each job j is completed no later than at time C_j. By Lemma 3, it is sufficient to consider fault sequences f that force one job e to re-execute k times. If $j < e$ then job j starts at r_j (recall that $r_{i+1} \geq r_i + p_i$ for all i) and executes once, so it is completed no later than C_j. For $j > e$, job j is completed at time $\max r_j, C_{j-1} + p_j \leq C_j$. For $e = j$, job j completes at time $r_j + (k+1)p_j \leq C_j$. In conclusion, we obtain the following theorem.

Theorem 1 *Algorithm* 1 *computes the latest completion times for all jobs in the presence of up to k faults, and it works in linear time.*

0.5 Δ-Faults

In this section the fault model is $F = GAP_\Delta$ consisting all fault sequences f such that $f_i - f_{I-1} \geq \Delta$ for all i, where Δ is some parameter of the problem. As in [GMM95], we assume that the processing times of all jobs are at most $\Delta/2$.

For each j, we define $\alpha(j)$ as the index a such that $\sum_{i=a}^{j} p_i < \Delta$, and either $a = 1$ or $\sum_{i=a-1}^{j} p_i \geq$

Δ. Let also $\pi(j) = \sum_{i=\alpha(j)}^{j} p_i$. Note that if a fault occurs at the end of job j and jobs $\alpha(j), \ldots, j$ are executed back-to-back then no faults could have occurred on these jobs. The numbers $\alpha(j)$ and $\pi(j)$ can be pre-computed as follows:

$\alpha(1) \leftarrow 1$
$\pi(1) \leftarrow p_1$
for $j \leftarrow 2$ **to** n **do**
    $a \leftarrow \alpha(j-1)$
    $x \leftarrow \pi(j-1) + p_j$
    while $x \geq \Delta$ **do**
        $x \leftarrow x - p_a$
        $a \leftarrow a + 1$
    $\alpha(j) \leftarrow a$
    $\pi(j) \leftarrow x$

By standard amortization, the above algorithm runs in linear time.

0.5.1 Exposed Faults

We present first a linear-time algorithm for the exposed-fault model. We start with the following lemma.

Lemma 4 *Suppose that J tolerates all faults sequences in* GAP_Δ *in which each fault occurs at the completion time of the first execution of some job. Then J is* GAP_Δ*-tolerant.*

Proof: It is sufficient to show that, in the greedy schedule, for each $b \in J$ and $f \in GAP_\Delta$, there is $g \in GAP_\Delta$ that satisfies the condition in the lemma and $C_b(g) \geq C_b(f)$.

Pick the largest $a \leq b$ such that job a starts on time, that is $S_a(s, f) = r_a$. Thus jobs $a, a+1, \ldots, b$ execute back-to-back, some possibly twice. Without loss of generality we can assume that all faults in f occur between r_a and $C_b(f)$, since we can remove all other faults without affecting the value of $C_b(f)$.

If f satisfies the condition in the lemma, we take $g = f$ and we are done. Otherwise, let f_l be the last fault in f that does not satisfy the condition in the lemma. Let also e, $a \leq e \leq b$, be the job affected by f_l, that is $S_e(f) \leq f_l \leq s_e(f) + p_e$. For $\delta = S_e(f) + p_a - f_l$, define a new fault sequence $h_i = f_i$ if $i < l$ and $h_i = f_i + \delta$ for $i \geq l$. Then $C_j(h) \geq C_j(f)$ for all $j = a, \ldots, b$, and h has more faults satisfying the condition in the lemma than f. So after repeating this process we transform f into a desired fault sequence g. □

Our algorithm uses the numbers $\alpha(j)$ and $\pi(j)$ computed above. Assume that $C_0 = s_1$.

Algorithm 2. for $j = 1, \ldots, n$ do $C_j \leftarrow \max\left\{ C_{j-1} + p_j \,,\, r_j + 2p_j \,,\, C_{\alpha(j)-1} + \pi(j) + p_j \right\}$

The linear-time complexity is obvious. We show that the numbers C_j are computed correctly. For the current job j we consider cases depending on whether the last fault occured. If j is executed without faults, then there are no restrictions on where was the previous fault, so $C_j = \max\{r_j, C_{j-1}\} + p_j$. Suppose that a fault occurs at job j. Without loss of generality, this fault is at the end of its first execution. Job j will be executed twice, so it is sufficient to show that its latest possible start time is at $\max\left\{r_j, C_{\alpha(j)} + \pi_j\right\}$. The only way S_j can be greater than r_j is when job j was delayed because of a previous fault. So the jobs $\alpha(j), \ldots, j-1$ must have been executed back-to-back with jobs $\alpha(j)+1, \ldots, j-1$ executing without faults. Therefore $C_j = C_{\alpha(j)} + \pi(j)$. We conclude that the algorithm is correct.

Theorem 2 *Algorithm* 2 *computes in linear time the maximum completion times for all jobs if all faults are separated by gaps of length at least* Δ.

0.5.2 Hidden Faults

The algorithm from [GMM95] verifies fault-tolerance if all jobs are ready at the same time. Their method is to divide the sequence of jobs into blocks of length at most Δ, where each block includes an additional unallocated recovery interval whose length is at least the longest processing time of the jobs. Then they compute the partition that minimizes the total execution time. This method does not work for jobs with different release times, because some job sequences can be executed fault-tolerantly but do not have such partition into blocks. Consider, for example, a sequence J of jobs in which job j has start time $3j-3$, deadline $3j+1$ and all jobs have execution time equal 2. Let $\Delta = 6$. With the greedy schedules all jobs in J will meet their deadlines, but it will not be possible if we postpone the execution of any job.

The idea of our algorithm is this: the worst sequence for a given set of jobs is such that each fault f_i is either at a beginning of a job or at distance exactly Δ from the previous fault. Thus the sequences of faults form chains separated by Δ, each sequence starting at a beginning of some job. So we can compute the earliest completion time for each job if a fault occurs at its beginning, and then using the chain structure and the completion times of the previous jobs we can determine its worst completion time.

We now formalize this idea. Fix a set of jobs J. We call a fault sequence $f \in GAP_\Delta$ *cruel* for J if, for each i, we have

(c1) either $f_i - f_{i-1} = \Delta$ or f_i occurs at a beginning of some job, and

(c2) if $f_i - f_{i-1} = \Delta$ and $f_{i-1} \le S_j(f) \le f_i$ then $S_j(f) = C_{j-1}(f)$.

The above conditions imply that each fault sequence can be divided into chains, where in each chain the faults are at distance exactly Δ, and the jobs affected by these chains are executed back-to-back.

Lemma 5 *If J tolerates all cruel fault sequences from* GAP_Δ *then J is* GAP_Δ*-tolerant.*

Proof: It is sufficient to show that, in the greedy schedule, for each $b \in J$ and $f \in GAP_\Delta$, there is a cruel $g \in GAP_\Delta$ for which $C_b(g) \geq C_b(f)$.

Pick the largest $a \leq b$ such that job a starts on time, that is $S_a(s, f) = r_a$. Thus jobs $a, a+1, \ldots, b$ execute back-to-back, some possibly twice. Without loss of generality we can assume that all faults in f occur between r_a and $C_b(f)$, since we can remove all other faults without affecting the value of $C_b(f)$.

Take the first fault f_i that does not satisfy the definition of the cruel sequence. Suppose that f_i occurs when a job j is executed. We can move f_i earlier by a small $\epsilon > 0$ without affecting the completion time of jobs $j, j+1, \ldots, b$. In this way, we eventually will make f_i equal to either $f_{i-1} + \Delta$ or to S_j. By repeating this process, we turn f into a cruel fault sequence. □

We use two auxiliary arrays $E(i,j)$ and $E^\circ(i,j)$ defined as follows. Assume that we start job i at time 0, execute all jobs back-to-back (ignoring release times), with the fault sequence $f_i = (i-1)\Delta$, $i = 1, \ldots, m$.

$E(i,j)$ is the maximum time to complete job j,

$E^\circ(i,j)$ is the maximum time to complete job j, if the last fault f_m is at least Δ before the completion time of its (first) execution.

Intuitively, we distinguish $E^\circ(i,j)$ from $E(i,j)$, becuase we need $E^\circ(i,j)$ to tell us how long it will take to execute jobs $i, \ldots, j$ if another fault chain starts at the beginning of job $j+1$.

For a fixed i, array $E(i,j)$ can be computed by adding jobs $j = i, i+1, \ldots$, possibly with re-executions. We keep track of where the last fault ϕ occurred, and we update the array and ϕ at each step. Array $E^\circ(i,j)$ can be computed in a similar manner. In this case, we do not add another execution of the job affected by ϕ until when ϕ will be at least Δ away.

Algorithm 3. Initialize $C_0^0 = 0$. Then

for $j = 1, \ldots, n$ **do**
  $C_j^0 \leftarrow \max_{i \leq j} \{r_i + E^\circ(i,j),\, C_{i-1}^0 + E^\circ(i,j)\}$;
  $C_j' \leftarrow \max_{i \leq j} \{r_i + E(i,j),\, C_{i-1}^0 + E(i,j)\}$;
  $C_j \leftarrow \max\left\{C_j^0, C_j'\right\}$

The quadratic complexity of the algorithm is obvious.

For correctness, we claim that C_j^0 is the completion time of job j if it does not belong to a fault chain, and that C_j is its worst-case completion time. The proof is by induction. That C_j^0 and C_j

are lower bounds is obvious. We need to show that job j cannot terminated later than $C_j'^0$ if j is not in a fault chain, and not later than C_j overall. Let i be the beginning of the last fault chain. Job i started at $\max\{r_i, C_{i-1}^0\}$, and jobs $i, \ldots, j$ take time $E^\circ(i,j)$ to execute.

Theorem 3 *Algorithm 3 computes the maximum completion times for all jobs if all faults are separated by at least time Δ, and it works in quadratic time.*

0.5.3 Another Method

For simplicity, assume here that all jobs are released at time 0. We give a different dynamic programming algorithm.

Define H_j to be the set of pairs (c, δ) such that for any cruel $f \in GAP_\Delta$, job j can be completed at time c with last fault at $\leq c - \delta$. Note that if $(c \leq c'$ and $\delta \geq \delta'$, then we can eliminate (c', δ') from H_j.

The sets H_j can be updated as follows. For each (c, δ):

- If $\delta + p_{j+1} < \Delta$, add $(c + p_{j+1}, \delta + p_{j+1})$ to H_{j+1}
- If $\delta + p_{j+1} \geq \Delta$, add $(c + p_{j+1}, \Delta)$ and $(c + 2p_{j+1}, \delta + 2p_{j+1} - \Delta)$ to H_{j+1}.

It is easy to see that we only need to add at most one pair $(c+p_{j+1}, \Delta)$ at each step. Further, if we keep H_j ordered, the list H_{j+1} can be obtained by merging two sorted sequences. So the algorithm will run in time $O(n^2)$.

Bibliography

[EKMMM99] E. Egan, D. Kutz, D. Mikulin, R. Melhem, D. Mossé, "Fault-tolerant RT-Mach (FT-RT-Mach) and an application to real-time train control", *Software: Practice and Experience*, to appear, 1999.

[GMM95] S. Ghosh, R. Melhem and D. Mosse, "Enhancing real-time schedules to tolerate transient faults", *Proc. of the 16th IEEE Real-Time Systems Symposium*, Pisa, Italy (1995).

[LMM99] F. Liberato, R. Melhem and D. Mosse, "Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems", manuscript, 1999.