Fast Algorithms for Testing Fault-Tolerance of Sequenced Jobs with Deadlines

Marek Chrobak* Mathilde Hurand† Jiří Sgall‡

Abstract

In queue-based scheduling systems jobs are executed according to a predefined sequential plan; faults may occur that cause jobs to re-execute thus delaying the whole schedule. It is, therefore, important to be able to determine (in real-time) whether a set of pre-ordered jobs will always meet their deadlines: it allows for instance to decide online whether to admit a new urgent job in the queue and still guaranty that the whole schedule remains fault-tolerant, i.e. that the remaining jobs still will not miss their deadlines. Our goal in this work is therefore to efficiently test whether a given set of sequenced jobs can tolerate transient faults. We consider different realistic fault models specifying which fault patterns are allowed and how soon failed jobs can be restarted. For each fault model considered we provide efficient algorithms that, given a set of sequenced jobs, decide on the feasibility of all the jobs in the schedule. Our algorithms are exact and run in time linear in the number of jobs and thus can be used to make a real-time decision.

1 Introduction

Previous work. Ghosh, Melhem and Mossé [GMM95, MMG03] (see also [EKM+99]) introduced 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 (or lengths). 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, MMG03] the authors assume that a fault can be detected only after the processing of a job is complete. We refer to such faults as *hidden faults*. The question investigated in [GMM95, MMG03] is whether all jobs in J will meet 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 [GMM95, MMG03], the authors assume the fault frequency model in which a gap between any two faults is at least Δ, where Δ is at least twice the maximum job length. For this model, they present an $O(n^2)$-time fault-tolerance testing algorithm, under the restriction that all jobs are released at the same time. In addition, they also propose a linear-time heuristic that approximates a solution for this problem. The authors also extend this linear-time heuristic to jobs with arbitrary release times, and discuss its applications and experimental results. A different fault frequency model, in which the number of faults is bounded by some constant k, has been suggested by Liberato, Melhem and Mossé [LMM00]. For this model, the authors give a $O(n^2k)$-time dynamic programming algorithm for testing fault-tolerance if jobs are ordered according to EDF and preemption is allowed. Substantial work has also been done on fault tolerant scheduling in multiprocessor systems. For example, Liberato *et al.*[LLMM99] study scheduling of periodic preemptive real-time jobs in the presence of transient faults. A different model, with processor faults and non-periodic and non-preemptive tasks was investigated by Manimaran and Siva Ram Murthy [MM98]. Pruhs and Kalyanasundaram [KP97] study fault-tolerant scheduling from the perspective of competitive analysis. (See [LLMM99, MM98, QHJ+00, QJS02, GKS04, KP97] and references therein for other work on this

*Department of Computer Science, University of California, Riverside, Research supported by NSF grants CCR-9988360, CCR-0208856, and OISE-0340752.

†Department d'Informatique (LIX), Ecole Polytechnique, Palaiseau, France.

‡Mathematical Institute, Academy of Sciences of the Czech Republic, Partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), and grant 201/05/0124 of GA ČR.f

and related topics.)

Our results. On this paper, the jobs are allready ordered, and preemption is not allowed. We consider both fault frequency models from [GMM95, MMG03, LMM00] in this paper. In addition to the hidden faults, we also consider another type of faults that we call *exposed*. Unlike hidden faults, exposed faults can be detected immediately, and therefore the job affected can be stopped and restarted right after the fault.

Our first algorithm is for the fault frequency model NUM_k, where the number of faults is bounded by k. This algorithm runs in time $O(n)$ (for both Hidden faults and exposed faults). In [Ayd04] the fault model from [LMM00] is extended so that a reexecution of a job could take time different from its processing time. Our algorithm can be adapted to handle a similar case as well.

Then we consider the fault frequency model GAP_Δ, introduced in [GMM95, MMG03], in which any two consecutive faults are separated by a gap at least Δ. For exposed faults, we give an algorithm that runs in time $O(n)$. In the case of hidden faults we present an algorithm with worst case running time $O(n^2)$. Our algorithm applies to jobs with arbitrary release times, generalizing the work from [GMM95, MMG03]. Furthermore we prove that if that for example the distribution of the jobs is strictly positive on the all interval $[0, p_{\max}]$, then this algorithm's expected running time is $O(n)$ with high probability. Our experimental study confirms that this linear-time performance for various distributions.

We eventually generalize this algorithm to take into account novel fault model, GAP^k_Δ, where two faults must be separated by at a time at least Δ, but for a fixed number of k faults that can happen at any time. Our algorithm has a worst case performance $\Omega(kn^2)$ and in practice runs in $\Omega(kn)$.

Outline We will start by some definitions and notations. Then, in Section 3 we prove that it is only necessary to restrict our attention to greedy schedules only. Then, we detail our several algorithms for testing fault-tolerance for each of the three fault model. The basic idea behind all these algorithms is similar: for each fault model we first show that one needs to consider only some model-specific worst-case fault sequencess, termed "cruel". With this restriction, using dynamic programming, we design algorithms that computes for each job its latest completion time under "cruel" fault sequences. Comparing these completion times with the deadlines, we determine whether the given set of jobs is fault-tolerant. Due to space constraints most of the proofs of lemmas and theorems are omitted.

2 Terminology and Notation

Jobs and Schedules. By J we denote the sequence of n jobs on input, 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 $0 \leq r_i < r_i + p_i \leq d_i$ for all i. By $p_{\max} = \max_j p_j$ we denote the maximum processing time. A *schedule* of J is any sequence $s = (s_1, \ldots, s_n)$, such that $s_i \geq r_i$ for all i, and $s_{i+1} \geq s_i + p_i$ for $i < n$. We refer to s_j as the *scheduled start time* of job j. Without loss of generality, throughout the paper, we assume that $r_{i+1} \geq r_i + p_i$ for all $i < n$. For any set of jobs J we can easily modify, in linear time, the release times in J to satisfy this property, without affecting job completion times. The *greedy schedule* for J is then defined simply by $s_i = r_i$, for all i.

Faults. Each fault is specified by a real number, namely the time of the fault. *Fault sequences* are denoted by letters f, g, h. We assume that the faults in these sequences are listed in increasing order.

A *fault frequency model* is a set F of potential fault sequences. F is called *sparsifiable* if for all $f \in F$, any $1 \leq a \leq b \leq |f|$, and any fault sequence g with $|g| = b - a + 1$, if $f_{a+i-1} - f_{a+i-2} \leq g_i - g_{i-1}$ for $i = 2, ..., b-a+1$, then $g \in F$ as well. Intuitively, this means that any sequence "sparser" than a sequence in F is also in F. The three particular models we consider are: GAP_Δ: the set of all sequences f in which $f_i - f_{i-1} \geq \Delta$ for each i; NUM_k: the set of all sequences f with at most k faults; GAP^k_Δ: the set of all sequences f where at most k faults f_i satisfy $f_i - f_{i-1} < \Delta$. All three models are sparsifiable. As we show later, for sparsifiable models, we can restrict ourselves to studying only greedy schedules.

Completion times. Next, we explain how a job's execution is affected when a fault occurs. This depends on the type of faults under consideration. Fix a sequence of n jobs J and a fault model 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. Informally, $S_j(s, f)$ is either s_j or the completion time of job $j - 1$, whichever is greater. If no fault oc-

curs between $S_j(s,f)$ and $S_j(s,f)+p_j$, then $C_j(s,f)$ equals $S_j(s,f)+p_j$. If a fault occurs in this interval, j will need to be reexecuted, starting either at the fault time or at $S_j(s,f)+p_j$, depending on whether we consider exposed or hidden faults. The completion time is the time when j has been fully processed without faults.

We now give a rigorous definition. Initially, set $S_1(s,f)=s_1$. Then, for $j=1,...,n$, assume that S_j has been defined, and proceed as follows:
(C) The completion time $C_j(s,f)$ depends on the fault type: (CE) For exposed-faults, $C_j(s,f)$ is the smallest $\tau \geq S_j(s,f)+p_j$ such that $f \cap (\tau - p_j, \tau] = \emptyset$, that is, the interval $(\tau - p_j, \tau]$ contains no faults; (CH) For hidden-faults, $C_j(s,f)$ is the smallest $\tau \geq S_j(s,f)+p_j$ such that $f \cap (\tau - p_j, \tau] = \emptyset$ and $\tau - S_j(s,f)$ is an integer multiple of p_j.
(S) If $j < n$, then the start time of job $j+1$ is $S_{j+1}(s,f) = \max\{s_{j+1}, C_j(s,f)\}$.

By $C_j(s,F)$ we denote the maximum completion time of job j if the faults are from F, that is $C_j(s,F) = \max_{f \in F} C_j(s,f)$. Throughout 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.

For either fault type, exposed or hidden, a schedule s of J is called *F-tolerant*, if each job completes by its deadline, that is $C_j(s,F) \leq d_j$ for all j. All algorithms we present will actually compute, for all j, the maximum completion times $C_j(s,F)$. Testing fault-tolerance, that is, whether $C_j(s,F) \leq d_j$ for all j, can then be done trivially in linear time.

3 Fault Tolerance and Greedy Schedules

In this section we show that we can restrict ourselves to greedy schedules only. For two schedules s, t, we write $s \prec t$ if $s_i \leq t_i$ for all i.

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

sketch. Let a fault f, by induction on the jobs in the sequence, we show that delaying the starting time of a job either does not affect its completion time (since the job will complete a full processing time after the date of the last fault), either delays it if the jobs waits after the last fault to start its execution. □

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

Idea of the proof. Suppose we have a fault sequence f and a schedule t that differs from the greedy schedule s by one starting date: a job b waits a time ϵ before starting in t. Then, since the fault frequency model is "spartisifiable", we can also delay the faults that would affect jobs b and after by this same amount of time, so that the faults keep hitting jobs b and after exactly at the same relative time, leading to the same completion times as in s, only shifted by ϵ to the right. If the two schedule differ by more than one starting date, we decompose the process into elementary steps where they differ only by one, and the proof generalizes. □

Lemma 3. *All three fault frequency models, NUM_k, GAP_Δ, and GAP^k_Δ, are sparsifiable.*

From the lemmas above, we can assume that the jobs are scheduled greedily, and we will use notation $S_j(f)$, $C_j(f)$, etc., for the start time and completion time in the greedy schedule. Also, we will say that a job sequence J is *F-tolerant* if the greedy schedule for J is F-tolerant.

4 Sequences with at Most k Faults

In this section we give a linear-time algorithm for testing fault tolerance when $F = \mathsf{NUM}_k$, that is, F 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.

Lemma 4. *For both exposed and hidden faults, for each $b \in J$ and $f \in \mathsf{NUM}_k$, there is $g \in \mathsf{NUM}_k$ that causes one job in J to execute $k+1$ times, and for which $C_b(g) \geq C_b(f)$.*

Idea of the proof. First note that in this case, the worst pattern for exposed faults is when the faults happen at the end of jobs, and thus the exposed fault case comes down to the hidden fault case. For hidden fault, the worst delay that can happen for a job is if the biggest job scheduled in front of it is made to restart $k+1$ times. □

Algorithm 1 — Computing the $C_j^* = C_j(\mathsf{NUM}_k)$

$C_0^* \leftarrow r_1$
for j from 1 to n **do**
$C_j^* \leftarrow \max\{ C_{j-1}^* + p_j \, , \, r_j + (k+1)p_j \}$.

Algorithm 2 : $C_j^* = C_j(\mathsf{GAP}_\Delta)$ for exposed faults

Compute the numbers α_j, π_j using algorithm 3
$C_0^* \leftarrow r_1$
for $j = 1, \ldots, n$ **do**

$$C_j^* \leftarrow \max \left\{ \begin{array}{l} C_{j-1}^* + p_j \\ r_j + 2p_j \\ C_{\alpha(j)-1}^* + \pi(j) + p_j \end{array} \right\}$$

Algorithm 1 given above will compute the latest completion time $C_j^* = C_j(\mathsf{NUM}_k)$ for each job j. To test fault-tolerance, one then only needs to check if $C_j^* \leq d_j$ for all j.

Theorem 5. *For both fault types, Algorithm 1 computes in time $O(n)$ the latest completion times for all jobs j, in the presence of up to k faults (that is, for the fault model NUM_k.)*

5 Exposed Δ-Faults

We now consider the fault model $F = \mathsf{GAP}_\Delta$, in which all fault sequences f satisfy $f_i - f_{i-1} \geq \Delta$ for all i, where Δ is some parameter of the problem. Recall that, as in [GMM97], we assume that $\Delta \geq 2p_{\max}$.

As in the previous section, the idea is to show that only some special fault sequences, the *cruel fault sequence for* GAP_Δ need to be considered. Define CEGAP_Δ^J to be the set of fault sequences $f \in \mathsf{GAP}_\Delta$ in which each fault occurs at the completion time of the first execution of some job.

Lemma 6. *In the greedy schedule, for each $b \in J$ and $f \in \mathsf{GAP}_\Delta$, there is $g \in \mathsf{CEGAP}_\Delta^J$ for which $C_b(g) \geq C_b(f)$.*

Proof intuition without release dates: Suppose we have a fault sequence f where a fault f_k hits a job k just ϵ before its first scheduled completion time. If we shift all faults after f_k (included) by ϵ the created fault sequence is still in GAP_Δ and the final completion time of job k is delayed by epsilon. Jobs after k are also delayed by ϵ since they get hit by the new faults at the same relative time as before. The complete proof is adapted to handle release dates. □

Algorithm 3 — Computing the auxiliary numbers $\alpha(j)$ and $\pi(j)$

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

For each j, we first define $\alpha(j)$ as the minimum index a such that $\sum_{i=a}^{j} p_i \leq \Delta$. (In other words, $\sum_{i=a}^{j} p_i \leq \Delta$ and either $a = 1$ or $\sum_{i=a-1}^{j} p_i > \Delta$.) Let also $\pi(j) = \sum_{i=\alpha(j)}^{j} p_i$. The numbers $\alpha(j)$ and $\pi(j)$ can be pre-computed as in Algorithm 3. By a standard amortization argument, Algorithm 3 runs in linear time. Our algorithm, Algorithm 2, uses the numbers $\alpha(j)$ and $\pi(j)$ computed above, to calculate $C_j^*(\mathsf{CEGAP}_\Delta^J)$, the worst time completion time for job j over all cruel sequences. Its linear-time complexity is obvious. Then, according to Lemma 6, for all j, $C_j^*(\mathsf{CEGAP}_\Delta^J) = C_j^*(\mathsf{GAP}_\Delta)$.

Theorem 7. *Algorithm 2 computes in linear time the maximum completion times for all jobs, for exposed faults, if all faults are separated by gaps of length at least Δ.*

6 Hidden Δ-Faults

The algorithm from [GMM97] verifies fault-tolerance if all jobs are ready at the same time. It can be shown that the method from [GMM97] does not work for arbitrary release times. Our general approach is similar to those in the previous section. We identify certain "cruel" fault sequences on which completion times of jobs are maximized. Focusing on these sequences, we derive a dynamic programming algorithm.

Let J be a set of jobs. A fault sequence $f \in \mathsf{GAP}_\Delta$ is called *cruel for* J if for all $f_i \in f$, $f_i - f_{i-1} = \Delta$,

or f_i occurs at a beginning of some job. The above conditions imply that each cruel fault sequence can be divided into *chains*, where in each chain the faults are at distance exactly Δ. We define CRUEL_J^0 to be the set of fault sequences in GAP_Δ that are cruel for J.

Lemma 8. *In the greedy schedule, for each $b \in J$ and $f \in \mathsf{GAP}_\Delta$, there is $g \in \mathsf{CRUEL}_J^0$ for which $C_b(g) \geq C_b(f)$.*

Proof intuition without the release dates. The idea is the same as in the previous case. Take a fault sequence f that is not in $\mathsf{CRUEL}_J^0$and take the first fault of the sequence that is neither at a distance Δ from the previous fault nor at the begining of a job. Then we can move this fault leftwards until one of those condition is reached, and it does not affect the completion times of the jobs. The created fault sequence is still in GAP_Δ. By repeating the process, we turn f into a fault from CRUEL_J^0that leads to the same completion times as f. □

For a fixed cruel sequence up to f_i, there are only two ways to extend it: fault again after time Δ or after a time bigger than Δ, at the starting time of a new job. This has two consequences : First, this gives us a dynamic algorithm to compute cruel sequences recursively: to know where cruel sequence f may or may not fault during the execution of job $j+1$ we really only need to keep track of when f last faulted, and at what time it made job j finish: we need to keep track of the two values $C_j(f)$ and $\delta_j(f) = C_j(f) - f_i$, where f_i is the last fault so far. Second, we can not recursively compute all cruel sequences in polynomial time since there are exponentially many of then. We then prove that we can consider only a subset of CRUEL_J^0.

Intuitively, here is how we discard some of the cruel sequences: suppose that we have computed how two cruel sequences f and g behave up to job j. Suppose that $C_j(f) > C_j(g)$, and that the last fault in f happened before the last fault in g (i.e. $\delta_j(f) \geq \delta_j(g)$). Then intuitively, we can discard g from further computation because for any extension of g there is an extension of f where the completion times are at least as large.

Formally, a pair $(\tilde{c}, \tilde{\delta})$ is said to *dominate* a pair (c, δ) iif $\tilde{c} \geq c$, $\tilde{\delta} \geq \delta$ and at least one of these two inequalities is strict. We further extend the definition of dominance to fault sequences. For two faults sequence $f, g \in \mathsf{CRUEL}_J^0$ and a job k, we say that f *k-dominates* g if $(C_k(f), \delta_k(f))$ dominates $(C_k(g), \delta_k(g))$. For each k, the k-dominance relation is a partial strict order on CRUEL_J^0. By $\mathsf{CRUEL}_J^k \subseteq \mathsf{CRUEL}_J^0$ we denote the set of cruel sequences that are not j-dominated by another sequence, for any $j = 1, 2, ..., k$. In other words, for $k > 0$, CRUEL_J^k is the set of all $f \in \mathsf{CRUEL}_J^{k-1}$ such that f is not k-dominated by any $g \in \mathsf{CRUEL}_J^{k-1}$. We now prove that, in order to compute the worst-case completion time of job k, it is sufficient to consider only the sequences in the set CRUEL_J^k.

Algorithm 4 Computing the sets H_j

```
H_0 ← {(r_0, Δ)}
for each (c, δ) ∈ H_{j-1} such that c ≥ r_j do
  if δ + p_j < Δ then
    add (c + p_j, δ + p_j) to H_j
  else
    add (c + 2p_j, δ + 2p_j − Δ) and (c + p_j, Δ) to H_j
Eliminate dominated pairs from H_j
if ∃(c, δ) ∈ H_{j-1} | c < r_j then
  H_j ← H_j ∪ {(r_j + p_j, Δ), (r_j + 2p_j, 2p_j)}.
C*_j ← max {c : (c, δ) ∈ H_j}
```

Lemma 9. *For any $f \in \mathsf{CRUEL}_J^{k-1} - \mathsf{CRUEL}_J^k$ and $b \geq k$, there exists $g \in \mathsf{CRUEL}_J^k$ such that $C_b(f) \leq C_b(g)$.*

Corollary 10. *If a job b from J is CRUEL_J^b-tolerant, then it is CRUEL_J^0-tolerant.*

The algorithm. We compute all values $C_j^* = C_j(\mathsf{GAP}_\Delta)$ for each j. By Lemma 8, for each j we have $C_j^* = \max_f C_j(f)$, where the maximum is over all cruel sequences, and according to corollary 10, $C_j^* = C_j(\mathsf{CRUEL}_J^j)$. Let $H_j = \{(C_j(f), \delta_j(f)), f \in .\mathsf{CRUEL}_J^j\}$: for any set CRUEL_J^j, H_j is the set of pairs (c, δ) such that for some fault sequence $f \in \mathsf{CRUEL}_J^j$, $c = C_j(f)$ and $\delta = \delta_j(f)$. Therefore $C_j^* = \max_{(c,\delta)\in H_j}(c)$. Initially we have $H_0 \leftarrow (0, \Delta)$, and since $\mathsf{CRUEL}_J^{j+1} \subset \mathsf{CRUEL}_J^j$, H_{j+1} can be built dynamically from set H_j . Note that in the sets H_j, we can assume that $\delta \leq \Delta$, since for $\delta \geq \Delta$ it does not matter what the exact value of δ is, so we can use the value $\delta = \Delta$ to indicate that there is no restriction on the next fault. The pseudocode for computing the sets H_j is given in Algorithm 4.

Correctness. Take $f \in \mathsf{CRUEL}_J^j$. We want to prove that $(C_j(f), \delta_j(f)) \in H_j$. Given our definition of H_0

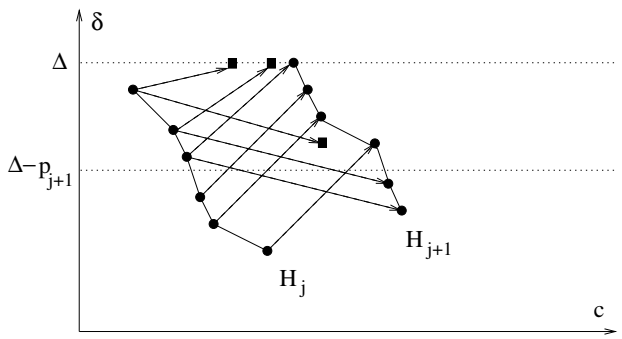

Figure 1: **Building H_{j+1} from H_j**

the basis case is trivial. Suppose that the claim is true at rank $j-1$ and take $f \in \mathsf{CRUEL}_J^j$. Since $\mathsf{CRUEL}_J^j \subset \mathsf{CRUEL}_J^{j-1}$, f is in CRUEL_J^{j-1} as well and $(C_{j-1}(f), \delta_{j-1}(f)) \in H_{j-1}$. Our algorithm considers every pair in H_{j-1} and computes the exhaustive undominated corresponding pairs in H_j. Since f is in CRUEL_J^j, $(C_j(f), \delta_j(f))$ is not dominated and will therefore be added to H_j.

For $(c, \delta) \in H_{j-1}$, several things can happen for the creation of corresponding pair(s) in H_j: If $r_j \leqslant c$, the limiting factor for the scheduling of the new job is not its release time but the current schedule. If $\delta + p_j \leq \Delta$ (line 3), no fault can occur on j. The only way of continuing the cruel sequence is to put the job back to back with the last one and change the δ and c according to this. The new pair will therefore be $(c + p_j, \delta + p_j)$ If $\delta + p_j > \Delta$ (line 5), we can continue the cruel sequence in two ways. Either we fault on j, in that case the processing time will increase by $2p_j$ and the last fault will be closer by $\Delta - 2p_j$. Or, we can decide not to fault on j. In that case the processing time will increase only by p_j but next fault might occur anywhere, so we add the pair $(c + p_j, \Delta)$ to H_j.

During the construction of H_j at least a pair (c, Δ) will be added. Also, since Δ is maximal, note that a pair $(\tilde{c}, \Delta)$ will not be discarded during the elimination process where $\tilde{c}$ is minimal over all c in H_j.

If $c < r_j$, since a job cannot start before its release date, the only way to continue the cruel sequence would be to execute the next job at its release date, with or without a fault at its beginning. But, if there is a $c < r_j \in H_{j-1}$, then by minimality of $\tilde{c}$, $\tilde{c} \leqslant c < r_j$ with $(\tilde{c}, \Delta) \in H_{j-1}$. For this pair we therefore create in H_j the pairs $(r_j + p_j, \Delta)$ and $(r_j + 2p_j, 2p_j)$. Those two pairs dominates all the ones we could create for the other $c < r_j$.

Then, since we only keep the pairs that correspond to faults from CRUEL_J^j, we need to eliminate all pairs that correspond to fault sequences dominated at rank j, i.e. pairs (c, δ) for which there exists a pair (c', δ') in H_j such that $c' \geq c$ and $\delta' \geq \delta$. This elimination can be implemented in time linear $O(|H_j|)$ by maintaining two list of ordered pairs to be added (according to the case) and merging them in the end. □

Lemma 11. *The size of H_j increases at most by 1 at each step. Therefore the procedure to build the set H_j can be implemented in time $O(j)$. So the overall algorithm works in time $O(n^2)$.*

The idea of the proof is that even though it seems that the size of H_j could double at each step, since we eliminated dominated pairs, of all pairs type (c, Δ) only the one with the biggest c remains, and the size of H_j increases at most by 1 at each step.

6.1 Experimental Results

As we showed in the previous section, the algorithm for Δ-faults runs in time $O(n^2)$, in the worst case. Note, however, that the algorithm is not data-oblivious, and that its running time depends on the size of the sets H_j. For the overall running time to be quadratic, the size of H_j would have to increase by 1 in most steps, which means in most steps no elimination would occur – a scenario that seems very unlikely in random data sequences. We confirm this intuition through experimental studies. We performed three types of experiments, for various probability distributions. In the first one, we show the expected running time to grow linearly with n. Next, we confirm this further by showing that the total size of the sets H_j is linear, in expectation. Finally, we show that, for the uniform distribution, with high probability the size of the sets H_j is constant. (Indeed, we prove this fact in the next section.)

Running time. We tested our algorithm for several random distributions of job length. The experimental running time is obviously linear, results are presented in Figure 2.

Total size of the sets H_j. The running time of the algorithm is proportional to $\sum_{j=0}^{n} |H_j|$, the total number of pairs (c, δ) in the sets H_j. In the second batch of experiments we measured the expectation of the total size of sets H_i for instances of different size n ranging from 1 to 20000. In our experiments, this value also grows linearly with n.

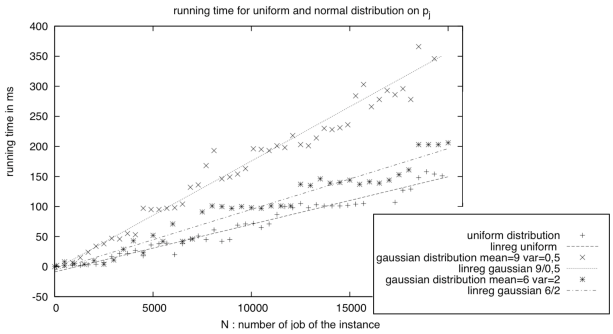

Figure 2: **Running time of Algorithm 4**

Maximal size of sets H_j. We also considered the maximum cardinality reached by sets H_j, for various values of n, ranging from 0 to 120000, and for the uniform distribution of job lengths. The results show that this quantity grows very slowly, and appears to level off at around 11. The result is represented in figure 3. Even for very large values of n, we did not find

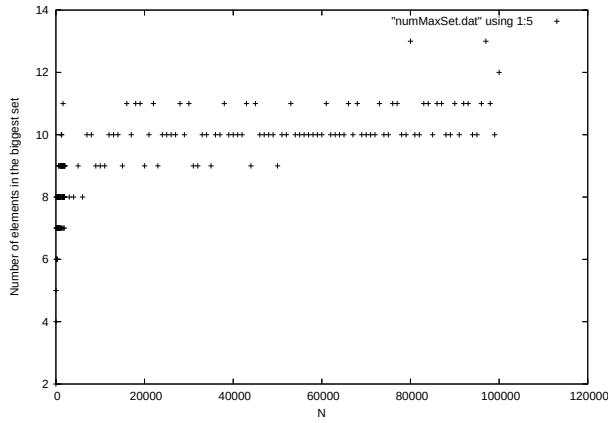

Figure 3: Maximum size of the sets H_j.

any sets H_j with more than 13 pairs. In the next section, we will prove that for the uniform distribution the expected size of the sets H_j is $O(1)$.

6.2 Probabilistic Analysis – Uniform Distribution

In this section, we are going to show that under the *uniform* distribution of the lengths p_j, our algorithm's expected running time is linear. In fact, we will prove something stronger – namely that the running time of the algorithm is $O(n)$ with very high probability. The proof is made for convenience on a uniform distribution over the p_j, it is easy to check that the same proof holds for any distribution where there is a strictly positive probability for p to fall in each of the following intervals: $[\frac{5}{26}, \frac{6}{26}]$ $]\frac{13}{52}, \frac{15}{52}[$, $[\frac{15}{52}, \frac{17}{52}]$, and $]\frac{6}{13}, \frac{1}{2}]$. Therefore, a fortiori, the proof works for a distribution that is strictly positive everywhere. Only the analysis would then result in different constants. This is consistent with our experimental results, we verify that the constant varies with the distribution. The idea of the proof is based on the intuition from the previous section. We prove that, with high probability, the size of the sets H_j remains constant throughout the computation. To simplify the analysis, we only exploit certain types of elimination in the proof. As a result, the constant bound we get is higher than the one from the empirical study.

Random sets Q_i. The idea of the proof is to define a sequence of random sets Q_i, which are essentially supersets of the sets H_i, but appropriately offset leftwards so that they are contained in the rectangle $[0, \Delta/2] \times [0, \Delta]$. The reason Q_i is not the same as H_i is that when computing Q_i we only do one type of elimination, and thus more points from Q_{i-1} may survive than when H_i is computed in the actual algorithm. Nevertheless, we still show that with high probability the size of the Q_i remains constant.

Without loss of generality, we can assume that $\Delta = 1$. Let $Z = [0, \frac{1}{2}] \times [0, 1]$. We say that $(\alpha, \beta) \in Z$ *dominates* $(\gamma, \delta) \in Z$ if and only if $\alpha \geq \gamma$ and $\beta \geq \delta$.

We define first two auxiliary functions $F(\cdot)$ and $\phi(\cdot)$. For all $p \in [0, \frac{1}{2}]$ and $\alpha, \beta \in Z$, define

$$F(p, \alpha, \beta) \quad = \quad \begin{cases} (\alpha + p, \beta + 2p - 1) & \text{if } \beta \geq 1 - p \\ (\alpha, \beta + p) & \text{if } \beta \leq 1 - p \end{cases}$$

and for $Q \subseteq Z$ and $p \in [0, \frac{1}{2}]$, let $\phi(Q, p) = \max\{\alpha : (\alpha, \beta) \in Q \ \& \ \beta \geq 1 - p\}$. Observe that, by definition, the point $(\phi_{Q,p}, 1)$ dominates all points $(\alpha, \beta) \in Q$ with $\alpha \leq \phi(Q, p)$.

In the rest of the proof we consider a random sequence $p_1, p_2, ..., p_n$ of job lengths, where each p_i is chosen uniformly from $[0, \frac{1}{2}]$, and we prove that for this sequence the size of all sets H_i remains constant with high probability. To avoid cumbersome notation, from now on we fix the values of $p_1, p_2, ..., p_n$. For each i, let $F_i(\alpha, \beta) = F(p_i, \alpha, \beta)$ and $q_i = \phi(Q_i, p_i)$. The reader needs to keep in mind though that F_i, q_i, as well as all other notions dependent on the sequence $\{p_i\}$ are actually random variables.

The sets Q_i are defined recursively. For $i = 0$, let $Q_0 = \{(0, 1)\}$. Suppose that Q_i is defined, and let $q_i = \phi(Q_i, p_i)$. Then

$$\begin{aligned} Q'_{i+1} &= F_i(Q_i) \cup \{(q_i, 1)\} \\ Q''_{i+1} &= Q'_{i+1} - \left\{ \begin{array}{ll} (\alpha, \beta) & \in F_i(Q_i) \mid (\alpha \leq q_i) \\ \text{or} & (\alpha \leq p_i \text{ and } \beta \leq 2p_i) \end{array} \right\} \\ Q_{i+1} &= Q''_{i+1} - \{(q_i, 0)\} \\ &= \{(\alpha - q_i, \beta) \mid (\alpha, \beta) \in Q"_{i+1}\} \end{aligned}$$

In other words, Q'_{i+1} is the union of $F_i(Q_i)$ and the point $(q_i, 1)$. In Q''_{i+1}, we remove the points dominated either by $F_i\big((0, 1)\big) = (p_i, 2p_i)$ or by $(q_i, 1)$. In Q_{i+1} we offset Q''_{i+1}.

Lemma 12. *For all i and for any $(\alpha', \beta') \in F_i(Q_i)$ we have $\alpha' \leq q_i + 1/2$*

Proof. Choose $(\alpha, \beta) \in Q_i$ such that $(\alpha', \beta') = F_i(\alpha, \beta)$. We have two cases. If $\beta < 1 - p_i$, then $\alpha' = \alpha \leq q_i + \frac{1}{2}$, since both $q_i, \alpha \in [0, \frac{1}{2}]$. If $\beta \geq 1 - p_i$, then $\alpha' = \alpha + p_i \leq q_i + p_i \leq q_i + \frac{1}{2}$, by the choice of q_i. □

Observe that, according to Lemma 12 and from the definition of F_i, we have $Q_i \subseteq Z$ for all i.

Theorem 13. *$|H_i| \leq |Q_i|$ for all i.*

Proof omitted. The idea is to map each point that survives in H_i to a point that survives in Q_i. □

We now count how many points from Q_i won't have an image in Q_{i+1} because of the condition $\alpha < p$ and $\beta < 2p$. We will say of these points they are "eliminated" from step i to step $i + 1$. And then we will demonstrate that with constant probability, $\frac{1}{9}^{th}$ of the points in Q_i are eliminated in Q_{i+4}. We partition Z into five zones A, B_1, B_2, C, D defined as in Fig 4:

Lemma 14. *Fix some step i. Then: [(a)] With probability at least $\frac{1}{13}$, all points in $Q_i \cap A$ will be eliminated. [(b1)] With probability at least $\frac{1}{13}$, all points in $Q_i \cap B_1$ will migrate to $Q_{i+1} \cap A$. [(b2)] With probability at least $\frac{1}{13}$, all points in $Q_i \cap B_2$ will migrate to $Q_{i+1} \cap A$. [(c)] With probability at least $\frac{1}{13}$, all points in $Q_i \cap C$ will migrate to $Q_{i+1} \cap B$. [(d)] With probability at least $\frac{1}{13}$, all points in $Q_i \cap D$ will migrate to $Q_{i+1} \cap (B \cup C)$.*

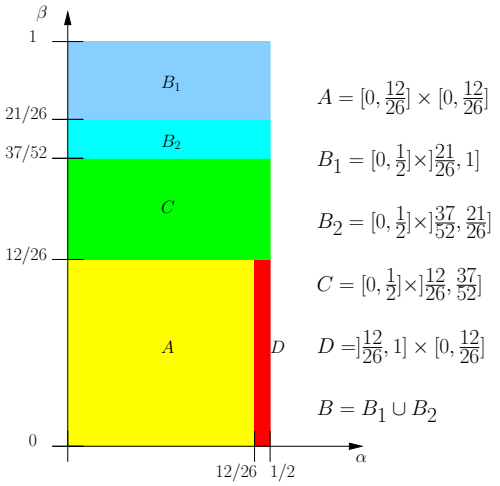

Figure 4: **Zones of elimination**

Proof. Let $p = p_{i+1}$. (a) A point $(\alpha, \beta) \in Q_i$ will be eliminated as long as $\max(\alpha, \beta) \leq p \leq \frac{1}{2}$. So for $p \in [\frac{12}{26}, \frac{1}{2}]$, all points in $Q_i \cap A$ will be eliminated. The probability that p falls in this range is $\frac{1}{13}$. (b1) A point $(\alpha, \beta) \in Q_i \cap B_1$ will migrate to $Q_{i+1} \cap A$ as long as:$\beta \geq 1 - p$, $\beta - 1 + 2 * p \leq \frac{12}{26}$ and $p \leq \frac{12}{26}$. So for $p \in [\frac{5}{26}, \frac{6}{26}]$, all points in $Q_i \cap B_1$ will migrate to $Q_{i+1} \cap A$. The probability that p falls in this range is $\frac{1}{13}$. (b2) A point $(\alpha, \beta) \in Q_i \cap B_2$ will migrate to $Q_{i+1} \cap A$ as long as: $\beta \geq 1 - p$, $\beta - 1 + 2 * p \leq \frac{12}{26}$ and $p \leq \frac{12}{26}$. So for $p \in [\frac{15}{52}, \frac{17}{52}]$, all points in $Q_i \cap B_2$ will migrate to $Q_{i+1} \cap A$. The probability that p falls in this range is $\frac{1}{13}$. (c) A point $(\alpha, \beta) \in Q_i \cap C$ will migrate to $Q_{i+1} \cap B$ as long as: $p \leq 1 - \beta$ and $\beta + p \geq \frac{37}{52}$. So for $p \in [\frac{13}{52}, \frac{15}{52}]$, all points in $Q_i \cap C$ will migrate to $Q_{i+1} \cap B$ (d) A point $(\alpha, \beta) \in Q_i \cap B_2$ will migrate to $Q_{i+1} \cap A$ as long as $\beta + p \geq \frac{12}{26}$. So for $p \in [\frac{12}{26}, \frac{1}{2}]$, all points in $Q_i \cap D$ will migrate to $Q_{i+1} \cap (B \cup C)$. The probability that p falls in this range is $\frac{1}{13}$. □

Theorem 15. *In Q_{i+4}, with probability at least $\chi = \frac{1}{13^4}$, $\frac{1}{9}$-th of the points in Q_i will be eliminated.*

proof idea. We start from the constat that we must have at least $\frac{1}{9}^{th}$ of the points of Q_i in A, B_1 or B_2, or $\frac{2}{9}^{th}$ in C or $\frac{4}{9}^{th}$ in D. And then we proceed by case and subcase analysis. The worst case being the last one, but it is easily shown that in the worst sub-case at least one fourth of those $\frac{4}{9}^{th}$ points are eliminated in four rounds (after moving to C, then to $B1$ or B_2 and finally to A), and this with probabilty at least $\frac{1}{13}^4$. In the other cases, a similar number of points

is eliminated but in less rounds and with a higher probability. □

6.3 An Upper Bound on $|H_i|$

Theorem 16. *For $t \leq n/4$, let $P_t(k) = \text{Prob}[|Q_{4t}| \geq k]$. Then $P_t(k) \leq (1 - \frac{\chi}{2})^{\frac{k-k_0}{4}}$ where $k_0 = 32\,(-ln(2)/ln(1-\chi/2)) + 4$.*

Proof. (Draft) For $k \leq k_0$ we have $(1 - \frac{\chi}{2})^{\frac{k-k_0}{4}} \geq 1$, so the condition is trivially satisfied. Assume now that $k \geq k_0$. In this case the proof is by induction on t. In the base case, for $t < \frac{k_0}{4}$ we have $k \geq 4t$ and $P_t(k) = Q_{4t}(k) = 0$ by construction and the theorem holds.

In the inductive step, let $t \geq \frac{k_0}{4}$ and suppose the property is true for all k at $t-1$. Then $P_t(k) \leq P_{t-1}(k-4)(1-\chi) + P_{t-1}\left(\frac{9}{8}(k-4)\right) * \chi \leq (1-\chi)\left(1-\frac{\chi}{2}\right)^{(k-4-k_0)/4} + \chi\left(1-\frac{\chi}{2}\right)^{(9(k-4)-8k_0)/32} \leq \left(1-\frac{\chi}{2}\right)^{(k-k_0)/4}$, and the theorem follows. □

Corollary 17. (a) *The size of each set H_i is constant with very high probability. Specifically, we have $\text{Prob}(|H_i| \geq k) \leq C_1(C_2)^k$ where C_1 and C_2 are constants, and $C_2 < 1$. (b) Consequently, the expected running time of Algorithm 4 is $O(n)$.*

Proof. (a) Since $|H_i| \leq |Q_i|$, this follows from the previous theorem with constants $C_1 = (1-\chi/2)^{-k_0/4}$ and $C_2 = (1-\chi/2)^{1/4}$, where χ and k_0 are constants defined earlier. Since χ is very small, we get $C_2 < 1$. (b) From part (a) we get $\text{Exp}(|H_j|) \leq C_1C_2/(1-C_2)^2$. Therefore each step of the algorithm takes constant time in expectation, so the overall running time is $O(n)$. As mentioned previously, if the distribution is strictly positive but not uniform, then the proof holds; the resulting C_1 and C_2 depend on the distribution. □

7 Sequences with Few non-Δ-faults

We now focuss on a more realistic model, where we allow a bounded number of faults to happen within an interval less than Δ. Such faults will be called non-Δ-faults . This fault model is denoted GAP^k_Δ if no more than k non-Δ-faults are allowed.

Idea of the algorithm (not given here). The worst sequence for a given set of jobs is such that every Δ-fault f_i is either at the beginning of a new job or exactly Δ away from the previous one, and every non-Δ-fault is at the beginning of a new job. To adapt the algorithm for $\mathsf{GAP}_\Delta$to the present problem, we augment the number of sets we are going to calculate. We now introduce H^i_j to be the set of pairs (c, δ) such that there exists a cruel sequence f_i where f_i contains exactly i non-Δ-faults up to job j, job j completes at time c with the last fault at time $c - \delta$. We then dynamically compute all these sets in worstcasee time $O(n^2k)$. Also we do not prove it formally, the intuition is that by similarity with what happens in the GAP_Δfault model, the practical running time is actually much faster, more like $O(kn)$. We present in fig 5 and fig 6 the graphics showing the actual running time of our algorithm as a function of N and k, for diverse distribution over p. Indeed this results corroborate the intuition.

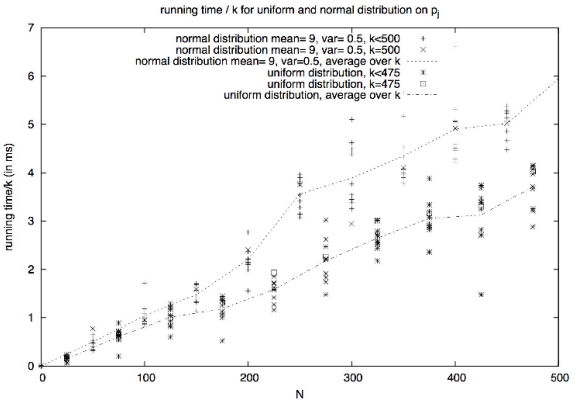

Figure 5: **Average running time increases linearly with N**

8 Final Comments and conclusion.

In this paper, we present linear-time algorithms for testing fault tolerance for the NUM_k fault model (for both hidden and exposed faults) and for the GAP_Δ fault model for exposed faults. For hidden faults in the GAP^k_Δ model, we give an exact algorithm of worst case running time $\Omega(kn^2)$, but that runs in time $\Omega(kn)$. In particular, for $k = 1$, this algorithms gives us an expected linear running time in the GAP_Δ fault-model. All our algorithms support release dates for the jobs. Whether the worst-case running time in the GAP_Δ fault model can be improved remains an open

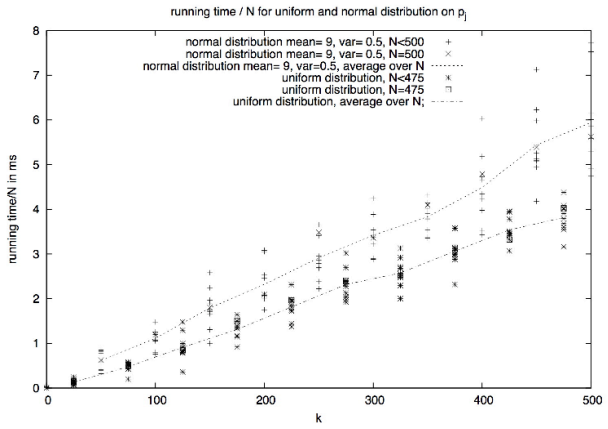

Figure 6: **Average running time increases linearly with k**

question. However, in the full version of the paper we provide evidence that such an improvement is unlikely, by showing that a slight generalization of this problem cannot be solved faster than in time $\Omega(n^2)$ in the algebraic decision tree model. We eventually want to stress that the methodology used throughout the paper, of carefully dynamically constructing instance-specific worst-case fault scenarios seems to be extremely well adapted to the problem at hand.

References

[Ayd04] Hakan Aydin. On fault-sensitive feasibility analysis of real-time task sets. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 426–434, Washington, DC, USA, 2004. IEEE Computer Society.

[EKM+99] E. Egan, D. Kutz, D. Mikulin, R. Melhem, and D. Mossé. Fault-tolerant rt-mach (ft-rt-mach) and an application to real-time train control. *Software: Practice and Experience*, 29:379–395, 1999.

[GKS04] A. Girault, H. Kalla, and Y. Sorel. A scheduling heuristics for distributed real-time embedded systems tolerant to processor and communication media failures. *International Journal of Production Research*, 42(14):2877–2898, July 2004.

[GMM95] S. Ghosh, R. Melhem, and D. Mossé. Enhancing real-time schedules to tolerate transient faults. In *Proc. IEEE Real-Time Systems Symposium*, pages 120–129, 1995.

[GMM97] S. Ghosh, R. Melhem, and D. Mossé. Fault-tolerance through scheduling of aperiodic tasks in hard-real time multiprocessor systems. *IEEE Trans. on Parallel and Distributed Systems*, 8:272–284, 1997.

[KP97] B. Kalyanasundaram and K. Pruhs. Fault-tolerant real-time scheduling. In *ESA*, pages 296–307, 1997.

[LLMM99] F. Liberato, S. Lauzac, R. Melhem, and D. Mossé. Fault tolerant real-time global scheduling on multiprocessors. In *Proc. Euromicro Workshop in Real-Time Systems*, 1999.

[LMM00] F. Liberato, R. Melhem, and D. Mosse. Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems. *IEEE Transactions on Computers*, 49:906–914, 2000.

[MM98] G. Manimaran and C. Siva Ram Murthy. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *IEEE Trans. Parallel Distrib. Syst.*, 9(11):1137–1152, 1998.

[MMG03] D. Mosse, R. Melhem, and S. Ghosh. A nonpreemptive real-time scheduler with recovery from transient faults and its implementation. *IEEE Transactions on Software Engineering*, 29:752–767, 2003.

[QHJ+00] X. Qin, Z. Han, H. Jin, L. Pang, and S. Li. Realtime fault-tolerant scheduling in heterogeneous distributed systems. In *Proc. Int. Conference on Parallel and Distributed Processing Techniques and Applications*, pages 421–427, 2000.

[QJS02] X. Qi, H. Jiang, and D.R. Swanson. An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogenous systems. In *Proc. 13th Int. Conference on Parallel Processing*, pages 360–368, 2002.