Fault-Tolerant Scheduling

(Extended Abstract)

*Bala Kalyanasundaram**
Computer Science Department
University of Pittsburgh
kalyan@cs.pitt.edu

Kirk R. Pruhs†
Computer Science Department
University of Pittsburgh
kirk@cs.pitt.edu

Abstract

We study fault-tolerant multiprocessor nonpreemptive scheduling under the realistic assumption that the occurrence of faults can not be predicted. The goal in these problems is to minimize the delay incurred by the jobs. Since this is an on-line problem we use competitive analysis to evaluate possible algorithms. For the problems of minimizing the make-span, and minimizing the average response time (for static release times), we give nonclairvoyant algorithms (both deterministic and randomized) that have provably asymptotically optimal competitive ratios. The main tool used by these algorithms to combat faults is redundancy. We show that randomization has the same effect as redundancy.

1 Introduction

1.1 Problem Statement

The scheduling of tasks in a multiprocessor system has been recognized as an important problem and has been extensively studied (see [2] for a survey). In a large multiprocessor system, processor faults are inevitable and fault-tolerance is a significant issue [8]. The vast majority of previous work on scheduling either assumes that there are no faults, or gives minimal analysis. In this paper we begin a theoretical investigation of the effect of processor faults on scheduling by considering several standard scheduling problems modified to allow faults. We assume that the pattern of faults can not be predicted by the on-line scheduling algorithm. We then compare the schedules produced by the on-line algorithm to the schedules produced by an omniscient algorithm.

The setting for the generic multiprocessor scheduling problem is a collection $P_1, \ldots, P_m$ of processors. Each processor P_j has a speed s_j. The processors are given a collection $J_1, \ldots J_n$ of jobs. Each job J_i has a *release time* r_i that is the time that the on-line scheduling algorithm is first aware of J_i's existence, and is the earliest time that J_i can begin execution. Furthermore, each J_i has a length x_i, and running J_i on P_j takes x_i/s_j units of time.

There are many variants of the multiprocessor scheduling problem depending on what assumptions one makes about the jobs and processors, and how one measures the desirability of a schedule. In the *static case* all release times are 0. Otherwise, the problem is said to be *dynamic*. In the *identical processors case* all the processor speeds are equal. Otherwise, it is called the *related processors case*. A scheduling algorithm is *clairvoyant* if it learns each x_i at time r_i, and is *nonclairvoyant* if x_i can not be deduced until J_i has been fully executed. A *preemptive* algorithm is allowed to suspend an executing job and later restart the job from the point of suspension. A *nonpreemptive* algorithm must begin the job afresh after suspending it. The *completion time* c_i of a job J_i is the time at which J_i has been allocated enough time to finish exe-

*Supported in part by NSF under grant CCR-9202158.
†Supported in part by NSF under grant CCR-9209283.

STOC 94- 5/94 Montreal, Quebec, Canada
© 1994 ACM 0-89791-663-8/94/0005..$3.50

cution. The *response time* of J_i is $w_i = c_i - r_i$. In the paper we primarily consider two scheduling measures, the *make-span* of a schedule, which is the maximum completion time, and the *average response time*, which is $\frac{1}{n}\sum_{i=1}^{n} w_i$.

We assume that when a processor P_j faults it is immediately detectable, and the job currently being run on P_j must be restarted from the beginning at some later point in time. A fault at P_j can be classified as either *permanent*, in which case no more jobs may be run on P_j, or *transient*, in which case P_j is inoperative for some finite period of time [8].

The *competitiveness* (or *competitive ratio*) of a deterministic (randomized) algorithm for a particular problem and measure $\mathcal{M}$ is the supremum over all instances $\mathcal{I}$ of the ratio of the (expected) value of $\mathcal{M}$ for the schedule produced by the on-line algorithm given $\mathcal{I}$ as input to the optimal value of $\mathcal{M}$ for a schedule of $\mathcal{I}$ [6]. We can assume that the optimal value of $\mathcal{M}$ was computed by an off-line algorithm with full advance knowledge of all the information about the jobs and the pattern of faults.

The on-line algorithms that we give are non-clairvoyant, although our lower bounds show that clairvoyance would not asymptotically help the on-line scheduling algorithm in the worst case. We generally disallow preemptions. Alternatively, our bounds still hold if one makes the realistic assumption that some finite amount of time is required to preempt a job. All our algorithms are simple, efficient, and easily implementable. As in most settings where fault-tolerance is an issue, the main tool available to combat faults is redundancy [8]. In this setting this means running multiple copies of the same job on different processors.

1.2 Results

The main results for the case of identical processors are summarized in the table. We use λ to denote the number of faults. In the case of permanent faults we use $\eta = m - \lambda$ to denote the number of nonfaulty processors. The constant ϵ satisfies $0 < \epsilon < 1$. So as to determine the effect of the duration of faults on competitiveness, we assume that the duration of each transient fault is 0. That is, after a processor P_j experiences a transient fault, a job may immediately be restarted on P_j. These results show that for a moderate number (say less than ϵm) of faults, the duration of the faults does not asymptotically effect the achievable competitive ratios. The results for no faults come from [7, 4]. If one expects that permanent faults are largely independent, then the number of faults is likely to be approximately ϵm, where the constant ϵ is the probability that a particular processor faults. So in practice the most relevant line of the table is probably where $\lambda = \epsilon m$. These results show that the competitive ratios are effectively bounded in practice. The competitive ratio for average response time is a constant, and the randomized competitive ratio for make-span is effectively bounded since $\log^* m$ is such a slowly growing function.

In the general related processors case, all the optimal competitive ratios listed in the table for permanent faults increase by a multiplicative factor of R, where R is the ratio of the speed of the fastest processor to the speed of the slowest processor. For no faults, [7] showed a $\Theta(\min(\log R, \log m))$ bound on the optimal deterministic competitive ratio, with respect to make-span, in the related processors case. In [4] it is implicitly shown that there exists a constant competitive algorithm for minimizing the average response time in the general related processors case. (The result is only explicitly stated for the identical processors case.) This shows that as the system becomes more heterogeneous the degradation of the optimal competitive ratio is much more rapid in the case of permanent faults.

The results for make-span with permanent faults are given in section 2, the results for average response time with permanent faults are given in section 3. In section 4, we consider transient faults for the identical processors case. In section 5, we consider the effect of not allowing the on-line algorithm to run multiple copies of the same job at the same time. It is at least possible that this may be required in some situations where the programs have side effects or use external resources. We show that this restriction really cripples deterministic algorithms in that the optimal competitive ratio for both make-span and average response time is $\Theta(R \cdot \lambda)$, for permanent faults. Interestingly enough, we show that the bounds on the competitive ratio, with respect to both make-span and av-

Optimal Competitive Ratios for Identical Processors			
	Make-span		Average Response Time (Static Case)
Number of Faults	Deterministic	Randomized	Deterministic and Randomized
$\lambda = 0$	$\Theta(1)$[7]	$\Theta(1)$ [7]	$\Theta(1)$ [4]
$\lambda > 0$	$\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$	$\Theta(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$	$\Theta(\frac{m}{\eta})$
$\lambda = \epsilon m$	$\Theta(\frac{\log m}{\log \log m})$	$\Theta(\log^* m)$	$\Theta(1)$
$\lambda > 0$ (Transient)	$\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$	$\Theta(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{\lambda}{m}))$	$\Theta(\max(\frac{\lambda}{m}, 1))$

erage response time, for randomized algorithms in the case of no redundancy are exactly the same as for deterministic algorithms that use only redundancy.

In section 6, we consider other seemingly natural scheduling measures, and show that the optimal competitive ratios for these measures are high in some situations. The *idle time* (resp. *relative response time*) of a job J_i is defined to be $d_i = w_i - x_i$ (resp. w_i/x_i). Even for a single processor, no faults, and allowing preemption we show an $\Omega(n)$ randomized lower bound on the competitive ratio for minimizing the maximum relative response time. Similar results are shown for minimizing the maximum or the average idle time.

We know of no previous theoretical investigations of this kind into fault-tolerant scheduling. Nonclairvoyant scheduling without faults is discussed in [1, 3, 4, 7]. In [4] it is shown that with dynamic release times and without faults, a nonclairvoyant deterministic (randomized) algorithm can not be better than $\Omega(n^{1/3})$-competitive ($\Omega(\log n)$-competitive), with respect to average response time, even allowing preemptions. This shows that on-line scheduling algorithms face a much more daunting task when trying to minimize the average response time of jobs with dynamic release times. For a general survey of fault tolerant scheduling see [8].

A word of warning is appropriate here. For convenience and clarity we often drop floors and ceilings in our calculations throughout this paper. This does not make an asymptotic difference in any of the results.

We use the following notation throughout this paper. In both the lower bound and upper bound proofs it is convenient to think of time as being divided into stages. We will use k to denote the number of stages. Often the exact value of k will not be determined until the end of the proof. We denote by $u(i)$ the number of unfinished jobs at the start of stage i. We use $f(i)$ to denote the number of faults during stage i. We define the *multiplicity* $\phi(J)$ of a job J at some point in time to be the number of processors on which a copy of J is being run at that time. When we speak of the multiplicity of a job during a stage, we are generally speaking about the multiplicity at the time just before the end of the stage.

2 Make-span

In this section we consider only permanent faults and allow simultaneous execution of multiple copies of a job. We use OPT to denote the optimal off-line make-span.

2.1 Lower Bounds

In this section we grant the on-line algorithm clairvoyance and assume that all release times are 0. We first state the results for identical processors, and then show how to extend the results to related processors.

Theorem 1 In the case of identical processors and static release times, the competitive ratio, with respect to the make-span, of any nonpreemptive deterministic clairvoyant algorithm $\mathcal{A}$ is $\Omega(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$, for λ permanent faults.

Proof Sketch: We first assume $\eta \geq \lambda$. The construction consists of k unit length stages. Initially, the processors are given m unit length jobs. Recall our notation that $u(i)$ is the number of unfinished jobs at the start of stage i. So, $u(1) = m$. Since all job lengths are equal it doesn't benefit the on-line algorithm to preempt a job, or to idle a processor. So we can assume that all jobs assignments to processors happen at integral times. Just before the end of each stage, we select λ/k processors to faults so as to maximize the number of unfinished jobs at the end of this stage. More precisely, assume that the $u(i)$ unfinished jobs are numbered and ordered such that $\phi(J_1) \leq \phi(J_2) \leq \ldots \leq \phi(J_{u(i)})$. Let $u(i+1)$ be the greatest integer such that $\sum_{j=1}^{u(i+1)} \phi(J_j) \leq \lambda/k$. Then all processors running copies of the jobs $J_1, \ldots, J_{u(i+1)}$ are faulted. A simple combinatorial argument shows that the on-line algorithm $\mathcal{A}$, can minimize $u(i+1)$ only when all the $\phi(J)$'s are as equal as possible. Hence, $u(i+1) \geq (\frac{\lambda}{k})/(\frac{m}{u(i)})$, or $u(i+1) \geq u(i)(\frac{\lambda}{km})$. Upon substitution, we get $u(i+1) \geq m(\frac{\lambda}{mk})^i$.

By the definition of k, $u(k) \geq 1$ and $u(k+1) = 0$. We thus solve for the largest k that satisfies $u(k) \geq 1$. Substituting for $u(k)$ we get, $k \log(\frac{km}{\lambda}) \leq \log m$. Solving this inequality we get, $k = \Omega(\frac{\log m}{\log(\frac{m \log m}{\lambda})})$. At least $m/2$ processors are functioning for the first two stages, and hence a make-span of 2 can be obtained by initially assigning 2 jobs to each of these $m/2$ processors.

The construction for $\eta < \lambda$ consists of two unit length stages. In a manner analogous to the previous argument, we fault $m/4$ processors right before the end of each of these two stages in such a way that at least $m/16$ jobs are unfinished at the end of the second stage. Notice $OPT = 2$. Now remaining $\lambda - m/2$ faulty processors fault, thus leaving with only η working processors. The result then follows, since $m/16$ jobs must be scheduled by $\mathcal{A}$ on η processors. ∎

Theorem 2 In the case of identical processors, and static release times, the competitive ratio, with respect to make-span, of any nonpreemptive randomized clairvoyant algorithm $\mathcal{A}$ is $\Omega(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$, for λ permanent faults.

Proof Sketch: Using Yao's technique [9], it suffices to bound the expected competitive ratio of any deterministic algorithm where the input is selected from some specific probability distribution. As in the case of deterministic lower bound, the construction consists of k unit length stages. Initially there are m unit length jobs. Assume for the time being that $\lambda < \eta$. Now $u(i)$ is a random variable denoting the expected number of unfinished jobs at the start of stage i.

At time t in a stage i, we say that a job J is *weak* if $\phi(J) \leq \frac{2m}{u(i)}$ at that time. Notice that at any point in time during stage i at least $u(i)/2$ jobs must be weak. Just before the end of each stage, each processor will fault independently with probability λ/mk. Hence, the expected number of faults per stage is at most λ/k, and the variance is $\Theta(\lambda/k)$. Consider jobs that are weak just before the end of stage i. The probability that such a weak job is unfinished at the end stage i is at least $(\frac{\lambda}{mk})^{\frac{2m}{u(i)}}$. Hence, the expected value of $u(i+1)$ is at least $(\frac{u(i)}{2})(\frac{\lambda}{mk})^{\frac{2m}{u(i)}}$. Furthermore, the variance of $u(i+1)$ is also $\Theta(u(i)(\frac{\lambda}{mk})^{\frac{2m}{u(i)}})$. So using Chebyshev's inequality, $u(i+1) \geq (\frac{u(i)}{4})(\frac{\lambda}{mk})^{\frac{2m}{u(i)}}$ with probability at least $(1 - 1/\Theta(u(i)(\frac{\lambda}{mk})^{\frac{2m}{u(i)}}))$.

To facilitate solving this recurrence, let $r(i) = 2m/u(i)$. Therefore, $r(1) = 2$, and $r(i+1) \leq 4r(i)(\frac{mk}{\lambda})^{r(i)}$. Notice that $r(i)'s$ form a nondecreasing sequence. Since we assumed $\lambda \leq m/2$,

$r(i+1) \leq (\frac{mk}{\lambda})^{2r(i)}$ for large enough i ($r(i) \geq 4$). As in the case of the deterministic lower bound, we solve for k in the equation $r(k) = 2m$, which is equivalent to solving for $u(k) = 1$. Upon solving the equation, we get $k = \log^* m - \log^* \frac{m}{\lambda} + O(1)$. Using Chebyshev's inequality we get that the probability that the number of faults in stage i differs from the mean by more than a multiplicative constant is $O(\frac{k}{\lambda})$. Hence, the probability that the number of faults in any stage differs from the mean by more than a multiplicative constant is $O(\frac{k^2}{\lambda})$. Note that $\frac{k^2}{\lambda} = o(1)$. Therefore, with probability $1 - o(1)$, the number of faults in the system is $\Theta(\lambda)$.

The probability that one of the inequalities for the $u(i)'s$ does not hold is bounded by $O(\sum_{i=1}^{k} 1/u(i)(\frac{\lambda}{mk})^{\frac{2m}{u(i)}}))$. Simplify this using the fact that $\lambda \leq m/2$, yields $O(\sum_{i=1}^{k} \left(\frac{1}{2k}\right)^{\frac{2m}{u(i)}})$. This sequence decreases at least geometrically and is thus bounded asymptotically by the first term $O((\frac{1}{2k})^2)$. Hence, with probability $1 - o(1)$ it is the case that all the inequalities for the $u(i)$'s hold, and the number of faults in each stage is $\Theta(\frac{\lambda}{k})$. In particular, with probability $1 - o(1)$, the number of jobs left at stage $\log^* m - \log^* \frac{m}{\lambda} + O(1)$ is more than 1.

The case where $\lambda \geq \eta$ can be handled in a manner analogous to how it was handled in theorem 1. ■

Theorem 3 **In the case of related processors, the optimal deterministic competitive ratio, with respect to make-span, is $\Omega(R \cdot \max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$, and the optimal randomized competitive ratio, with respect to make-span, is $\Omega(R \cdot \max(\log^* m - \log^* \frac{m}{\lambda}))$, for λ permanent faults.**

Proof Sketch: There are $\lambda/2$ processors with speed R and $m - \lambda/2$ with speed 1. There are exactly $\lambda/4$ jobs of length R. The adversary uniformly at random assigns the $\lambda/4$ jobs to $\lambda/4$ fast processors, and just before time 1 faults all of the other fast processors. At time 1, all of the remaining fast processors are faulted. At time 1, the on-line algorithm has $\Omega(\lambda)$ jobs left unfinished with high probability. The optimal make-span of these jobs is 1. Notice that the number of remaining faults is $\lambda/2$. Now apply the lower bound construction from the previous theorems to the unit speed processors. Note that the constructions in theorems 1 and 2 yield asymptotically the same bounds if the initial number of jobs is only $\Theta(\lambda)$ instead of m. ■

2.2 Upper Bounds

In this section we give both deterministic and randomized nonclairvoyant algorithms that allow dynamic release times and have asymptotically optimal competitive ratios. The algorithms do not know λ in advance. Note that in [7] a general reduction from the case of dynamic release times to the case of static release times is given that at most doubles the make-span. We start with the deterministic algorithm ROTARY.

ALGORITHM ROTARY: All of the jobs are initially put in a queue Q. When a processor P becomes idle then a copy of the job J on the front of the queue is assigned to P. The job J is then moved to the back of the queue. If a new job arrives then it is added to the back of Q. If some copy of a job J finishes execution then all copies of J are terminated and J is removed from Q. ■

For a set $\mathcal{J}$ of jobs and a set $\mathcal{P}$ of processors, we define a job assignment $\mathcal{G}$ to be an injective subset of $\mathcal{J} \times \mathcal{P}$.

Theorem 4 **For λ permanent faults, the competitive ratio, with respect to make-span, of ROTARY is $O(R \cdot \max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$.**

Proof Sketch: It suffices to show the bound for the identical processors case. It should be clear that the competitive ratio increases by at most a factor of R otherwise. For the purposes of this proof consider time 0 to be $2 \cdot OPT$ time units after the arrival of the last job. If there are no faults then ROTARY will complete all the jobs by time 0 (see [7]), and in general there will be at most m jobs left at time 0. Starting from time 0, we divide the job assignments into groups $G_1, \ldots, G_k$. Groups are defined with respect to $t_1 < t_2 < \ldots < t_k$ that are defined inductively as follows. As the base case $t_1 = 0$. Let t_{i+1} $(i \geq 1)$ be the first time after t_i such that every processor P is either experienced a fault before t_{i+1}, or has finished some job that P started after time t_i. Note that for all i, $t_{i+1} - t_i \leq 2 \cdot OPT$. G_i contains exactly those pairs (J, P) where P is a nonfaulty processor at time t_{i+1} and J is the last job assigned to P before time t_{i+1}.

When there is no confusion, we say that a job J is in G_i if there is pair $(J,P) \in G_i$. Let s_i be the earliest time that a job $J \in G_i$ is assignment to a processor. Notice that between time s_i and t_{i+1} the assignments made by ROTARY are exactly those in G_i. Furthermore, no job in G_i finished before time t_{i+1}. See figure 1 for an example illustrating the definitions of the t_i's, s_i's and G_i's.

We will treat each group as a stage in a manner similar to the lower bound proofs. We let $\phi(i)$ denote the minimum, over all jobs J in G_i, of the number of processors that J is assigned to in G_i. If $\phi(i) \leq 2$ then we say that G_i is *full*. If G_i is full then at least $\eta/2$ of the jobs in G_i will finish by time t_{i+2}, the time that the next stage ends. Hence there can be at most $O(\frac{m}{\eta})$ full stages. From here on we will ignore any decrease in the size of Q due to full stages, and concentrate on the decrease in nonfull stages. So let us assume that every G_i is not full.

We say that a stage G_i is *healthy* if the number of processors that experienced faults while running an instance of a job in G_i is at most $\frac{2\lambda}{k}$. At least $k/2$ of the G_i's must be healthy. So we assume that every G_i is healthy. If this assumption is not valid then the number of stages will at most double. In other words, it suffices to get a upper bound on the number of healthy stages.

In this context, we use $u(i)$ to denote the number of distinct jobs in G_i. So $u(1) \leq m$. For the moment assume $\eta \geq m/2$. Hence, $\phi(i) \geq (\frac{m}{2})/u(i)$. Since each stage is healthy, $u(i+1) \leq (\frac{2\lambda}{k})/\phi(i)$. Substituting we get $u(i+1) \leq u(i)(\frac{4\lambda}{km})$. Expanding this recurrence we get $u(i+1) \leq m(\frac{4\lambda}{km})^i$. As in the case of the lower bound proof, we now find an upper bound on k satisfying the equations $u(k) \geq 1$ and $u(k+1) = 0$. This is equivalent to solving $k \log \frac{km}{4\lambda} \leq \log m$. By solving this, we get $k \leq 2\frac{\log m}{\log(\frac{m \log m}{\lambda})}$ for large enough m.

We now consider the case where $\lambda > m/2$. Let n_i $(i \geq 1)$ be the number of stages executed by ROTARY when the total number of faults experienced is between $m - m/2^{i-1}$ and $m - m/2^i$. So the previous argument shows that $n_1 = O(\log m/\log\log m)$. Let $k_0 = 0$, and $k_i = \sum_{j=1}^{i} n_i$ for $i \geq 1$. So between time t_{k_i} and $t_{k_{i+1}}$ at most half of the processors functioning at time t_{k_i} fault. So by our previous argument, with the number of faults being at most half the available processors, $u(k_{i+1}) \leq u(k_i)(\frac{2}{n_i})^{n_i}$. Expanding this recurrence yields $u(k_{i+1}) \leq m \prod_{j=1}^{i} (\frac{2}{n_j})^{n_j}$.

So we now determine how large $k_\ell = \sum_{j=1}^{\ell} n_i$ can be subject to $m \prod_{j=1}^{\ell} (\frac{2}{n_j})^{n_j} \geq 1$. By symmetry we can assume, without loss of generality, that $n_1 \geq n_2 \geq \ldots \geq n_\ell$. Then each n_i, $1 \leq i \leq \ell/2$, must satisfy $n_i \geq k_\ell/2\ell$. So we consider the following inequality $m(\frac{2}{k_\ell/2\ell})^{k_\ell/2} \geq 1$, or equivalently $(\frac{k_\ell}{4\ell})^{k_\ell/2} \leq m$. This simplifies to $(\frac{k_\ell}{2})\log(\frac{k_\ell}{4\ell}) \leq \log m$. For the moment assume $\eta \geq m/\log m$. Then $\ell \leq \log\log m$, and $(\frac{k_\ell}{2})\log(\frac{k_\ell}{4\ell}) \leq \log m$ implies $k_\ell = O(\log m/\log\log m)$. Now assume $\eta < m/\log m$. We now prove that $k_\ell = O(\log m)$. Notice that $\ell \leq \log m$. So, without loss of generality, assume that $8\ell \leq k_\ell$, and hence $(\frac{k_\ell}{2})\log(\frac{k_\ell}{4\ell}) \leq \log m$ implies $k_\ell = O(\log m)$. ■

We now modify ROTARY to reset Q randomly on each rotation through Q.

ALGORITHM RANDOM ROTARY: Initially, we uniformly at random select a permutation of the jobs and assign these jobs to the queue Q in this order. When a processor P becomes idle then the job at the front of Q is removed and assigned to P. If Q ever becomes empty then it is reset to contain a random permutation of the remaining unfinished jobs. If some copy of a job J finishes execution then all copies of J are terminated.

Theorem 5 For λ permanent faults, the competitive ratio, with respect to make-span, of the Random Rotary is $O(R \cdot \max(\log^* m - \log^* \frac{m}{\lambda}, \frac{m}{\eta}))$.

Proof Sketch: Much of the proof is identical to the proof of theorem 4. It suffices to consider only the identical processors case. We say time 0 is $2 \cdot OPT$ units after the arrival of the last job. We define G_i, t_i, and s_i as before. We now say that a stage G_i is *full* if for all $J \in G_i$ it is the case that $\phi(J) \leq 8$. There can still be at most $O(\frac{m}{\eta})$ full stages. We continue to assume that every stage is healthy and not full. We start by assuming $\lambda \leq m/2$.

Now consider stage i. We consider only the subset G_i' of assignments made after the first time Q was reset after time s_i and before the last time it was reset before time t_{i+1}. We can partition G_i' into subclasses $F_1, \ldots, F_\ell$ where F_j represents

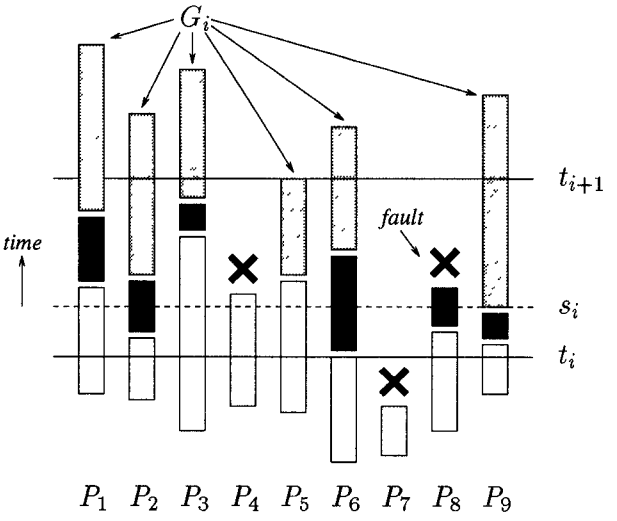

Figure 1: Job assignments in a stage

the job assignments after the jth resetting, but before the $(j+1)$st resetting, of Q within this stage. Notice that each job in G_i' occurs exactly once in each of those ℓ subclasses F_j's. We assume that the number of jobs in Q at time s_i was $u(i)$. It only benefits the on-line algorithm if this number has decreased since time t_i. Hence, $\frac{m}{2u(i)} - 2 \leq \ell$, Since G_i is not full this implies $\frac{m}{4u(i)} \leq \ell$. Among jobs and processors involved in F_j, every permutation of assignments is equally likely. If there are f_j faults within F_j, then the probability that no copy of a job J finishes during this stage is at most $\prod_{j=1}^{\ell}(\frac{f_j}{u(i)})$. (Note the probability of a job not finishing is maximized if we assume these probabilities are independent.) This product is maximized, subject to $\sum_{j=1}^{\ell} f_j \leq \frac{2\lambda}{k}$, if all the f_j are equal to $\frac{2\lambda}{k\,\ell}$. The constraint that $\sum_{j=1}^{\ell} f_j \leq \frac{2\lambda}{k}$ comes from our assumption that every stage is healthy. Hence, we can upper bound the probability that a job J is unfinished at the end of this stage by $(\frac{2\lambda}{k\,\ell\,u(i)})^{\ell}$. By substituting for ℓ and simplifying, this probability is at most $(\frac{8\lambda}{km})^{m/4u(i)}$. Therefore, the expected value of $u(i+1)$ is at most $u(i)(\frac{8\lambda}{km})^{m/4u(i)}$. Since the variance σ^2 is also $O(u(i)(\frac{8\lambda}{km})^{m/4u(i)})$ applying Chebyshev's inequality yields $u(i+1) \leq 2u(i)(\frac{8\lambda}{km})^{m/4u(i)}$ with probability $1 - O(1/\sigma^2)$. As in theorem 2 the solution to this recurrence is $O(\log^* m - \log^* \frac{m}{\lambda} + O(1))$. Also as in theorem 2, Chebyshev's inequality shows that all the inequalities involving the $u(i)$'s hold with high probability. Furthermore, the inequalities for the $u(i)$'s show that the probability that $u(i) > 1$ decreases at least exponentially after stage $\log^* m - \log^* \frac{m}{\lambda}$. Hence, the expected number of nonfull stages is $O(\log^* m - \log^* \frac{m}{\lambda})$.

If $\eta \leq m/2$ it is still the case that $2k + \log^* \frac{km}{8\lambda} + O(1) \leq \log^* m$ and hence $k = O(\log^* m)$. Hence, the total number of nonfull stages is still $O(\log^* m)$. The result then follows since the number of full stages is $O(m/\eta)$. ■

3 Average Response-Time

We present a nonclairvoyant deterministic algorithm that is $O(R\frac{m}{\eta})$-competitive with respect to average response time. We show that this is asymptotically optimal, even for randomized algorithms. We assume static release times, disallow preemption, and consider only permanent faults. Since average response time and total response time differ by exactly a multiplicative factor of n for any algorithm, the competitive ratios, with respect to these measures, will be identical. We generally compute total response times since they are more convenient. We also assume that we have a lower bound of 1 on the length of any task. This is not an unreasonable restriction since we could take our unit of length to be the time to execute one instruction on the fastest processor.

ALGORITHM GEOMETRIC ROTARY: Once again the algorithm consists of stages, and $u(i)$ denotes the number of jobs that have not yet been run to completion at the time that stage i starts. We now describe stage i ($i \geq 0$). If $u(i) \geq 2m$ then add one copy of each the $u(i)$ jobs to a queue Q_i. Else if $u(i) < 2m$ then Q_i contains $\lceil 8m/u(i) \rceil$ copies of each of these $u(i)$ jobs. When a processor P becomes idle the next job J on Q_i is removed and run from the beginning for 2^i time units (or to completion) on P (unless a fault occurs while J is being processed on P). If J has not finished after 2^i time units then it is set aside until the next stage. Stage i ends when Q_i becomes empty and every job J that was in Q_i has been run for 2^i units, or every copy of J has experienced a fault. ■

Theorem 6 **Given static release times, GEOMETRIC ROTARY is $O(R\frac{m}{\eta})$-competitive, with respect to average response time, for λ permanent faults.**

Proof Sketch: It is sufficient to prove a bound of $O(\frac{m}{\eta})$ for identical processors. At any point in time there is a lower bound to the possible length of any job, which is the longest that some copy of the job has been run by the on-line algorithm. By considering this lower bound as true length of each job, we can calculate a lower bound to the off-line minimum total response time. So for example, if all of the n jobs remain unfinished after stage i, and all these jobs were allowed to run for their full 2^i time slice in stage i, then the optimal total response time is at least $\frac{1}{m}\sum_{j=1}^{n/m} j2^i$, which is the minimum total response time for scheduling n jobs of length 2^i on the m processors. Let OPT_i be this lower bound after stage i. Note that OPT_i is monotone nondecreasing with respect to time. At the end of the stage 0 the total response time experienced by the on-line algorithm so far is $O(OPT_0)$.

Now consider a stage $i > 0$ where $u(i) \geq 2m$. Since the total number of faults is at most m, at least half of the jobs in Q_i ran for their full 2^{i-1} time slice in stage $i-1$. Hence, at least m jobs increased their minimum possible length by 2^{i-2} during stage $i-1$. The increase in OPT during stage $i-1$ (i.e., $OPT_{i-1} - OPT_{i-2}$) had to be at least $m\sum_{j=1}^{(u(i)-m)/m}(j\ 2^{i-2})$, which is the minimum total response time for packing the $(u(i)-m)$ jobs of length 2^{i-2} into m processors. This sum is at least $(u(i)-m)^2 2^{i-3}/m$. The increase in total response time experienced by the on-line algorithm during stage i is at most $u(i)^2 2^i/\eta$. Since $u(i) - m \geq u(i)/2$, the increase in OPT during stage $i-1$ (i.e., $OPT_{i-1} - OPT_{i-2}$) will be used to pay for increase in the total response time experience by the on-line algorithm in stage i.

Let k be the first stage where $u(k) < 2m$. Now consider a stage $i \geq k$. Let $s(i)$ be the number of the unfinished jobs at start of stage i with actual length at most 2^i. The increase in total response time experienced by the on-line algorithm during this stage is $O(2^i u(i)^2/\eta)$. This is $O(2^i u(i)m/\eta)$. In contrast OPT increases by at least $2^{i-1}(u(i) - s(i))$. We now consider two cases. If $s(i) \leq 7u(i)/8$ then the increase in OPT can be used to pay for this stage. We call this a paying stage. Therefore, the cumulative response time incurred by GEOMETRIC ROTARY during paying stages is $O(OPT)$. So, it suffices to consider nonpaying stages where $s(i) \geq 7u(i)/8$. If there are $f(i)$ faults in this stage then at most $f(i)/\phi(i)$ jobs will have all their copies faulted on this stage. Recall $\phi(i) = \lceil 8m/u(i) \rceil$ represents the multiplicity of the jobs. Since the number of faults is at most m, $u(i+1) \leq m/\phi(i) + (u(i) - s(i))$. Since $m/\phi(i) \leq u(i)/8$ and $(u(i) - s(i)) \leq u(i)/8$ it follows that $u(i+1) \leq u(i)/4$.

So every stage after the kth is either a paying stage, or the cost incurred by the on-line algorithm goes down by a factor of 2 from the previous stage (paying or not) since the time slice doubles and the number of jobs goes down by a factor of 4. As a consequence, the cumulative response time incurred during nonpaying stages is asymptotically equal to the response time incurred during paying stages that includes the kth stage. The result follows since the cost incurred during paying stages and during the kth stage is $O(OPT)$. ■

Theorem 7 **With static release times, and related processors, every randomized algorithm is $\Omega(R\frac{m}{\eta})$-competitive with respect to average response time.**

Proof Sketch: Similar to the proof of theorem 3. ■

4 Transient Faults

In this section we assume that the duration of all faults is 0, and disallow preemptions. Here λ de-

notes the total number of faults. Note that it is possible for the number of faults λ to exceed m.

Theorem 8 In the case of identical processors, the competitive ratio, with respect to make-span, of Rotary is $\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$, for λ transient faults. Furthermore, this competitive ratio is optimal for deterministic algorithms.

Proof Sketch: First consider the upper bound. As in the proof of theorem 4, the schedule is divided into stages. Since the total number of faults is λ, the number of stages experiencing more than $m/2$ faults each is at most $O(\frac{\lambda}{m})$. So, we can concentrate on stages with at most $m/2$ faults each. Since faults are transient, at the beginning of each stage all (i.e., m) processors are available. Therefore, following the proof of theorem 4, the number of remaining stages is at most $O(\frac{\log m}{\log(\frac{m \log m}{\lambda})})$.

We now consider the lower bound. Notice that in the proof theorem 1 we allowed the on-line algorithm m functioning processors on each stage. Thus the proof carries over here to show a bound of $\Omega(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ in the case of identical processors. ■

The proofs of the following theorems follow the same reasoning as the proof of theorem 8.

Theorem 9 In the case of identical processors, the competitive ratio, with respect to make-span, of Random Rotary is $\Theta(\max(\log^* m - \log^* \frac{m}{\lambda}, \frac{\lambda}{m}))$, for λ transient faults. Furthermore, this competitive ratio is optimal for randomized algorithms.

Theorem 10 For static release times and identical processors, the competitive ratio, with respect to average response time, of Geometric Rotary is $\Theta(\max(1, \lambda/m))$, for λ transient faults. Furthermore, this competitive ratio is optimal for deterministic and randomized algorithms.

For the related processors case, the lower bound for transient faults differ from that of permanent faults. In particular, for transient faults and related processors the lower bound for the identical processors case can not be generalized in the same manner as the permanent fault case (i.e., multiply the bound by R) when R is large. It appears that the bounds on the competitive ratio depend in a nontrivial way on the relationship among the three parameters R, λ and m, and the expression summarizing the optimal competitive ratio is significantly more complicated than the ones presented in this paper.

5 No Redundancy

In this section we look at the effects of not allowing redundancy. That is only one copy of a job may be running at any point in time. We primarily consider only identical processors. In the case of permanent faults, the following results can be extended to related processors case, as in the case of scheduling with redundancy, by multiplying each result by R.

Theorem 11 If no redundancy is allowed then the optimal deterministic competitive ratio, with respect to make-span, is $\Theta(\lambda)$, for λ permanent faults in a system of identical processors. The same bound holds for λ transient faults.

The following theorem shows that randomization has the same effect as redundancy.

Theorem 12 If no redundancy is allowed then the optimal randomized competitive ratio, with respect to make-span, is $\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{m}{\eta}))$ for permanent faults, and $\Theta(\max(\frac{\log m}{\log(\frac{m \log m}{\lambda})}, \frac{\lambda}{m}))$ for transient faults in a system of identical processors.

Theorem 13 If no redundancy is allowed then the optimal deterministic competitive ratio, with respect to average response time, is $\Theta(\lambda)$, for both permanent and transient faults.

Theorem 14 If no redundancy is allowed then the optimal randomized competitive ratio, with respect to average response time, is $\Theta(\max(\frac{m}{\eta}, 1))$ for permanent faults, and is $\Theta(\max(\frac{\lambda}{m}, 1))$ for transient faults.

6 Other Measures

In this section we consider other possible measures of delay, and show that even on a single processor the optimal competitive ratios for these measures is linear in the number of jobs.

Theorem 15 For no faults, static release times, identical processors, and allowing preemption, every randomized nonclairvoyant algorithm for minimizing the maximum relative response time has a competitive ratio that is $\Omega(n)$.

Proof Sketch: Let $m = 1$, $x_i = 2^{i-1}$, $1 \leq i \leq n$. It then takes at least $\Omega(n)$ tries to find the unit length job. A maximum relative response time of $\Theta(1)$ is possible by scheduling the jobs from shortest to longest. ■

Theorem 16 For no faults, identical processors, and disallowing preemptions, every randomized nonclairvoyant algorithm for minimizing the maximum idle time is $\Omega(n)$-competitive.

Proof Sketch: Here $m = 1$. The lower bound consists of n stages, each stage lasting $n + 1$ units of time. The first stage starts at time 0. At the start of stage i two jobs arrive in the system, one of length 1 and one of length n. If the shorter job is always run first the maximum idle time is 1. However, if the on-line algorithm picks one of these two jobs to run then with probability 1/2 it picks the longer job. It must then either incur a delay of n on the shorter job or suspend execution of the longer job. Hence at the start of the ith stage the total remaining execution time of jobs left unfinished from previous stages is $\Omega(i)$. Hence, at least by the last stage some job is idled $\Omega(n)$ time. ■

Theorem 17 For no faults, identical processors, and disallowing preemptions, every randomized nonclairvoyant algorithm for minimizing the average idle time is $\Omega(n)$-competitive.

7 Conclusion

Fault-tolerant computing in general seems like a natural area of application for on-line algorithms, and we feel that it is an area that is worthy of further investigation. The only previous application (that we are aware of) of the recently developed techniques in on-line algorithms to fault-tolerance is [5].

Acknowledgments: We thank Taieb Znati for pointing out to us the need for a theoretical analysis of fault tolerant scheduling. We also thank Joel Wein, Sundar Vishwanathan, Sunondo Ghosh and Daniel Mosse´ for helpful discussions.

References

[1] A. Feldmann, J. Sgall, S. Teng, "Dynamic scheduling on parallel machines", *Proceedings of IEEE Symposium on Foundations of Computing*, 111 – 120, 1991.

[2] R. Graham, E. Lawler, J. Lenstra, and A Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey", *Annals of Discrete Mathematics*, **5**, 287 – 326, 1979.

[3] T. Matsumoto, "Competitive analysis of the round robin algorithm', *International Symposium on Algorithms and Computation*, 71 – 77, 1992.

[4] R. Motwani, S. Phillips, E. Torng, "Nonclairvoyant scheduling", *Proceedings of the ACM/SIAM Symposium on Discrete Algorithms*, 422 – 431, 1993.

[5] K. Pruhs, "Average-case scalable on-line algorithms for fault replacement", Technical Report, Computer Science Department, University of Pittsburgh, 1993.

[6] D. Sleator, and R. Tarjan, "Amortized efficiency of list update and paging rules", *Communications of the ACM*, **28**, 202 – 208, 1985.

[7] D. Shmoys, J. Wein, and D. Williamson, "Scheduling parallel machines on-line", *Proceedings of IEEE Symposium on Foundations of Computing*, 131 – 140, 1991.

[8] J. Vytopil (ed.), *Formal Techniques in Real-time and Fault-tolerant Systems*, Kluwer Academic Publishers, 1993.

[9] A. Yao, "Probabilistic computations: towards a unified measure of complexity", *Proceedings of IEEE Symposium on Foundations of Computing*, 222 – 227, 1977.