# Preemptive Scheduling of Parallel Jobs on Multiprocessors [*]

Xiaotie Deng [†] Nian Gu [‡] Tim Brecht [§] KaiCheng Lu [¶] [‖]

**Abstract**

We study the problem of processor scheduling for parallel jobs. We prove that, for jobs with a single phase of parallelism, an algorithm can achieve a mean response time within $2 - \frac{2}{n+1}$ times the optimum. This is extended to jobs with multiple phases of parallelism and to interactive jobs (with phases during which the job has no CPU requirements) for a solution within $4 - \frac{4}{n+1}$ times the optimum. Comparing with previous work, our assumption that job execution times are unknown prior to their completion is more realistic, our multi-phased job model is more general, and our approximation ratio (for jobs with a single phase of parallelism) is better and cannot be improved.

## 1 Introduction

The CPU scheduling problem for computer systems distinguishes itself from general scheduling problems (e.g., job shop scheduling) in its variety of requirements of the system and variety of performance metrics. While minimizing makespan is usually a natural objective function for many general scheduling problems, a number of different possibilities exist for CPU schedulers in a general purpose multi-user computing environment [21]. Nevertheless, minimizing mean response time a common objective used in theoretical, simulation and empirical studies. Motwani, et al., show that, for uniprocessor systems, the mean response time of the Round-Robin policy is $2 - \frac{2}{n+1}$ times the optimum and that without *a priori* information about job execution time, no policy can guarantee a better factor [15] (called the competitive ratio [22][8][14]).

For parallel job execution, minimizing mean response time becomes much more difficult. Several recent analytic results have been obtained assuming that job information is completely known [26] [25] [27] [11]. These results initiated the first (theoretical) step toward understanding the general problem. Our work takes the next significant step and is distinguished from these results in that *we remove the unrealistic assumption that the job execution time is known.* Knowledge of execution times for some jobs may be obtained, but this is clearly not a valid assumption for all jobs in general purpose computing environments. Moreover, *our result cannot be further improved mathematically.* We show that the Dynamic Equipartition policy (DEQ) [24] [29] produces mean response times that are no more than $2 - \frac{2}{n+1}$ times the optimum for any set of parallel jobs, and that no policy can guarantee a better competitive ratio without *a priori* knowledge of job execution times. Although the competitive ratio turns out to be the same as in the sequential problem (not necessarily by accident), our result requires a completely different and rather difficult proof. *This provides a theoretical foundation for analyzing and understanding the performance of the DEQ policy* which, along with its various derivatives, has been shown to be superior in recent simulation and experimental studies [24] [12] [29] [10] [16] [13] [2] [17]. Furthermore, our results can be extended to jobs which may change the number of processors required during their execution, including interactive jobs, which may block and therefore, need not be assigned to a CPU while waiting for user input. We also show that DEQ is robust in presence of faulty jobs.

### 1.1 Preemptive Scheduling

Schedulers in most general purpose computer systems are preemptive, for several reasons [21]. First, job execution times may not be known prior to their completion. Thus, when non-preemptive scheduling algorithms are used, short jobs may be penalized by long jobs which utilize the CPUs for long periods of time. Second, interactive jobs require some processing and preemptive (time-sharing) scheduling policies allow them to execute by providing them with a slice of CPU time. Third, some jobs may execute infinitely, due to programming errors. If a non-preemptive scheduling policy is used they may execute forever and exclude other jobs from

[*]This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

[†]Dept. of CS, York University, North York, Canada M3J 1P3. email: deng@cs.yorku.ca

[‡]Dept. of CS, York University, North York, Canada M3J 1P3. email: gu@cs.yorku.ca

[§]Dept. of CS, York University, North York, Canada M3J 1P3. email: brecht@cs.yorku.ca

[¶]Dept. of CS, TsingHua University, Beijing, China, 100084. email:qhdata@attmail.com

[‖]A version of this paper appears in the **Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms**, (SODA), Atlanta, Georgia, January, 1996, pp. 159-167.

being processed. Obviously, the preemptive execution of jobs incurs some overhead. For multiprocessor systems, this may become more expensive. However, these overheads can be absorbed in a time-sharing scheme by choosing a scheduling quantum that is sufficiently large or by instead dynamically space-sharing processors [24] [29] [16]. This is consistent with the trend towards coarse grained machines for general purpose parallel computations, as suggested in the LogP model [1]. Setup costs can then be absorbed by pipeline routing if the size of a problem is sufficiently large in comparison with the number of processors in the system [28].Independently, there have been extensive empirical studies on the preemptive cost caused by time/space sharing scheduling policies [29] [16]. Even for some cases when the preemption cost is relatively high, simulation and experimental studies support preemptive over non-preemptive scheduling policies [29] [16].

### 1.2 Competitive Analysis

We make the assumption that job execution times are not known prior to their completion. This is quite realistic for modern general purpose multiprocessors. Since execution times are not known at the time jobs are scheduled, it is possible that any given scheduling policy may not perform very well on some specific job set. For this reason, we use competitive analysis to study policies that do not deviate, by more than a constant factor, from the optimal solution (which has and uses complete information about the job set). The competitive analysis of algorithms is a measurement of algorithms operating with incomplete information, first introduced in the study of a system memory management problem [22][8][14]. Policies for this problem are required to handle future unknown requests. The competitive ratio of a policy is defined to be the worst case ratio of the cost of the policy to the optimal cost for the same input sequence. In the CPU scheduling problem the situation is similar, in that the execution time of a job is unknown until its execution is completed. The competitive ratio of a scheduling policy $S$ is thus defined as the worst case ratio of the outcome, $S(\mathcal{J})$, of the policy on a set of jobs, $\mathcal{J}$, over the minimum outcome $OPT(\mathcal{J})$ on the same set $\mathcal{J}$: $\max_{all\ \mathcal{J}} \frac{S(\mathcal{J})}{OPT(\mathcal{J})}$. An algorithm is said to be *f(n)-competitive* if $S(\mathcal{J}) \leq f(n)OPT(\mathcal{J})$. The goal is to find an algorithm which leads to the minimum competitive ratio. Shmoys, et al., studied the optimal competitive ratio for the makespan of sequential jobs being scheduled on parallel machines [23]. For minimizing the mean response time of sequential jobs, Motwani, et al. have shown that a preemptive time-sharing policy, Round-Robin, achieves the optimal competitive ratio. It guarantees a mean response time which is within $2 - \frac{2}{n+1}$ times optimal and no policy can guarantee a better competitive ratio [15].

### 1.3 The Job Model

The most detailed description of a parallel program's execution on a multiprocessor is a data–dependency DAG, where edges represent data dependencies between the data (nodes). The DAG is revealed as the computation proceeds as a result of data-dependent conditional statements. Using a delay model introduced by Papadimitriou and Yannakakis [18], Deng and Koutsoupias show that given uniform communication delay, $\tau$, for any scheduler, there exists a DAG for which the scheduler will produce a schedule whose execution is at least $\frac{\tau}{\log \tau}$ times the optimal execution time of that DAG [3]. The same claim holds for both the BSP and the LogP models (see journal version of [3]). That is, with a parallel system of latency $L$ the competitive ratio of any scheduler is $\frac{L}{\log L}$. This work eliminates the possibility of a compiler that optimally (or near-optimally) executes all jobs and calls for the characterization of parallel jobs and the use of these characteristics in scheduling policies.

We characterize a parallel job, $J_i$, using two parameters: its execution time, $h_i$, and its parallelism, $P_i$. $P_i$ is the number of processors a job is capable of using during its execution, and $h_i$ is the time that the job needs to complete execution if it is allocated $P_i$ processors. When less than $P_i$ processors are allocated to job $J_i$, we assume that the job's execution will be prolonged proportionally. That is, if $p_i$ processors ($p_i < P_i$) are allocated to $J_i$, its actual execution time is $\frac{P_i}{p_i}h_i$. For a job, $(P_i, h_i)$, its parallelism $P_i$ is known to the scheduler but the execution time $h_i$ is unknown prior to its completion.

In general, we can use parallelism profiles to characterize parallel jobs. A parallelism profile is defined as the number of processors an application is capable of using at any point in time during its execution [9]. During execution, if the parallelism of an application varies with time, it is said to have multiple phases of parallelism. We may also formulate interactive jobs by introducing phases during which a job does not require access to a processor, because it is blocked while waiting for user input.

### 1.4 Related Results

The problem of minimizing the mean response time for parallel jobs on multiprocessors becomes much more difficult than its sequential counterpart. Recently, there have been several analytic results which assume that job information is completely known. The first significant work is that of Turek, et al., who introduce an approximation algorithm of 32 times the optimum [26]. This result has been subsequently improved and extended [25] [27] [11]. A number of different preemptive policies have been proposed and studied for scheduling parallel jobs in multiprocessors [24] [20] [29] [16] [19] [7]. In particular, experimental and simulation studies have shown that the DEQ yields low mean response times under a variety of workloads and is reported to possess desirable properties of a good scheduler [24] [12] [10] [13]. DEQ is first introduced to parallel scheduling by Tucker, et al. as a *process control* policy [24], and modified by McCann, et al. [29]. The main idea behind this approach is to distribute processors evenly among jobs, provided they have sufficient parallelism. Our proof that DEQ is $2-\frac{2}{n+1}$-competitive uses the notion of a *squashed area bound*, introduced for the non-preemptive scheduling of parallel jobs [11] [26] [27] [25]. Turek, et al. show that the minimum response time for a set of jobs, $\mathcal{J} = \{(P_1,h_1),(P_2,h_2),\cdots,(P_n,h_n)\}$, is no more than the minimum response time of the job set $\mathcal{J}_{squash} = \{(P, \frac{P_1h_1}{P}),(P, \frac{P_2h_2}{P}),\cdots,(P, \frac{P_nh_n}{P})\}$ [26]. The squashed area bound is the flow time (the product of the number of jobs and the mean response time) for $\mathcal{J}_{squash}$ under the Least Work First (LWF) policy. Sevcik shows that the LWF policy is optimal if all jobs have the same parallelism $P$ [19].

### 1.5 Outline of the Paper

In Section 2, we give a formal definition of the DEQ allocation policy. Then we establish a lower bound on the optimal flow time for parallel jobs which extends the squashed area bound and the height bound. This approach is completely different from previous work. In Section 3, we give a formal proof that the flow time of DEQ is no more than twice the optimal flow time for any job set. The mathematical induction used in this proof requires a delicate balance of the squashed area and height bounds on the work needed to be executed by each job. In Section 4, we study jobs with multiple phases of parallelism and also extend the results to interactive jobs. In Section 5, we discuss robust scheduling. We consider the case where there are faulty jobs which may execute infinitely. We show that DEQ achieves the optimal competitive ratio for the makespan. In Section 6, we conclude the paper with some remarks.

## 2 Preliminaries

Consider $n$ jobs in a system of $P$ processors. $J_i$ is characterized by the parallelism-time pair $(P_i,h_i)$, and the amount of work is $w_i = P_ih_i$, $1 \le i \le n$. Denote the job set by $\mathcal{J} = \{J_1,J_2,\cdots,J_n\}$. Suppose that under a scheduler $S$ the actual completion time of job $J_i$ is $t_i$, $1 \le i \le n$. The flow time of $\mathcal{J}$, denoted by $FT_S(\mathcal{J})$, is defined as $\sum_{i=1}^n t_i$. Then the mean response time $MRT_S(\mathcal{J})$ is defined as $\frac{FT_S(\mathcal{J})}{n}$. The height bound $H(\mathcal{J})$ is defined as $\sum_{i=1}^n h_i$. Since $J_i$ requires at least $h_i$ execution time, $1 \le i \le n$, $H(\mathcal{J})$ is an obvious lower bound on the optimal flow time. Let the jobs be ordered according to their total work $w_1 \le w_2 \le \cdots \le w_n$. The squashed area bound $A(\mathcal{J})$ is then defined as $\sum_{i=1}^n (n-i+1)\frac{w_i}{P}$. Notice that any preemptive scheduling on the job set $\mathcal{J}$ can be imitated by a preemptive scheduling on the job set $\{(\frac{w_1}{P},P),(\frac{w_2}{P},P),\cdots,(\frac{w_n}{P},P)\}$. It follows that $OPT(\mathcal{J}) \ge OPT\{(\frac{w_1}{P},P),(\frac{w_2}{P},P),\cdots,(\frac{w_n}{P},P)\}$. Since each job has the same parallelism as in the system for the latter, the short job first strategy gives the optimal solution for the flow time, easily provable as in sequential systems (see [26] [19] for details.) This is exactly the squashed area bound.

Our main result utilizes a nontrivial extension to the squashed area bound and the height bound. Suppose each job $(P_i,h_i)$ is divided into two parts: $(P_i,h_{i1})$ and $(P_i,h_{i2})$ such that $h_i = h_{i1}+h_{i2}$. Let $\mathcal{J}(1) = \{(P_i,h_{i1}) : 1 \le i \le n\}$ and $\mathcal{J}(2) = \{(P_i,h_{i2}) : 1 \le i \le n\}$. We have the following lemma for a lower bound on the optimal flow time of the job set $\mathcal{J}$.

LEMMA 2.1. $OPT(\mathcal{J}) \ge A(\mathcal{J}(1)) + H(\mathcal{J}(2))$.

**Proof of Lemma 2.1:** Consider the optimal scheduling algorithm on the input $\mathcal{J}$. Let $t_{i1}$ be the time when the remaining portion of $J_i$ is $(P_i,h_{i2})$ and $t_{i2}$ be the time when $J_i$ completes execution $1 \le i \le n$. Obviously, $t_{i2} - t_{i1} \ge h_{i2}$. The flow time for the optimal scheduler is

$$OPT(\mathcal{J}) = \sum_{i=1}^n t_{i2} \ge \sum_{i=1}^n t_{i1} + \sum_{i=1}^n h_{i2} \ge A(\mathcal{J}(1)) + H(\mathcal{J}(2)),$$

where the last inequality is derived from the height bound and the squashed area bound [26]. □

We formally define the DEQ allocation policy recursively as follows:

1. If $P_i \ge \frac{P}{n}$ for all $i : 1 \le i \le n$, each job is assigned $\frac{P}{n}$ processors.
2. Otherwise, each job $J_i$ with parallelism $P_i < \frac{P}{n}$ is allocated $P_i$ processors. Update $n$ and $P$. If $n > 0$, recursively apply DEQ.

Obviously, this schedule is valid only when $\frac{P}{n}$ is an integer. In practice, if $\frac{P}{n}$ is a rational number, and larger than 1, we can take its integer part $\lfloor\frac{P}{n}\rfloor$ and ignore its fraction part $\frac{P}{n} - \lfloor\frac{P}{n}\rfloor$. The result will be affected by a small constant factor. If $\frac{P}{n}$ is a fractional number smaller than 1, we view all the parallel jobs as sequential jobs, and apply the Round-Robin policy so that in unit time, a fraction $\frac{P}{n}$ of one processor's CPU time is assigned to one job. For the simplicity of our proof, we allow this fractional number $\frac{P}{n}$ of processors assigned to a job as long as it is no bigger than the parallelism of the job. Let $\mathcal{J}_{para}$ be the set of jobs that are allocated $P_i$ processors, and the rest of the jobs form the set $\mathcal{J}_{equi}$ (which are each assigned an equal number of processors, denoted by $\bar{p}$).

LEMMA 2.2. *If there are no idle processors, then* $\sum_{J_i\in\mathcal{J}_{para}} P_i + |\mathcal{J}_{equi}|\bar{p} = P$, $(\forall J_i \in \mathcal{J}_{para})P_i \le \bar{p}$, *and* $\bar{p} \ge \frac{P}{n}$.

Consider the execution of jobs under the DEQ policy. Each job $(P_i, h_i)$ is divided into two modes of execution: It is in *the full-parallelism mode,* if $P_i$ processors are assigned, and it is in *the equipartition mode,* if less than $P_i$ processors are assigned. It is not difficult to see that under the DEQ allocation policy, once a job enters the full-parallelism mode, it will stay in that mode until completion. Let $h(f)$ be the length of the job's execution in the full-parallelism mode, and $h(e) = h - h(f)$. Let $\mathcal{J}(f) = \{(P_i, h_i(f)) : 1 \le i \le n\}$ and $\mathcal{J}(e) = \{(P_i, h_i(e)) : 1 \le i \le n\}$. In the next section, we prove

$$FT_{DEQ}(\mathcal{J}) \le \tag{2.1}$$
$$(2 - \tfrac{2}{n+1})[A(\mathcal{J}(e)) + H(\mathcal{J}(f))] \tag{2.2}$$

Combining this with Lemma 2.1, we have

THEOREM 2.1. $FT_{DEQ}(\mathcal{J}) \le (2-\frac{2}{n+1})OPT(\mathcal{J})$.□

It is not hard to extend the same lower bound for competitive ratio of sequential jobs by Motwani, et al., to this situation [15]. Thus, this competitive ratio is optimal.

## 3 Minimizing Mean Response Time

Suppose jobs are divided into $\mathcal{J}_{para}$ and $\mathcal{J}_{equi}$ initially according to the above discussion. When one of the jobs, say $J_1$, finishes its execution under DEQ, we will re-allocate processors according to DEQ. Every job in $\mathcal{J}_{para}$ will still be assigned the same number of processors as its parallelism. Let $\mathcal{J}'_{para} \subseteq \mathcal{J}_{equi}$ be the subset of jobs in $\mathcal{J}_{equi}$ which are assigned the same number of processors as their parallelism after the re-allocation of processors. Let $\mathcal{J}'_{equi} = \mathcal{J}_{equi} - \mathcal{J}'_{para}$. Thus, jobs in $\mathcal{J}'_{equi}$ are now assigned the same number of processors.

LEMMA 3.1. *If there are no idle processors, then*

$$(n+1)P|\mathcal{J}_{equi}| \le$$
$$(n-1)P|\mathcal{J}_{para}| + n\bar{p}|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1).$$

**Proof of Lemma 3.1:**

$$\begin{aligned} RHS &= \\ &= P(|\mathcal{J}_{equi}| + |\mathcal{J}_{para}| - 1)|\mathcal{J}_{para}| \\ &+ n\bar{p}|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1) \\ &\ge P|\mathcal{J}_{equi}||\mathcal{J}_{para}| + P(|\mathcal{J}_{para}|-1)|\mathcal{J}_{para}| \\ &+ P|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1) \\ &\ge P|\mathcal{J}_{equi}|(|\mathcal{J}_{para}| + |\mathcal{J}_{equi}| + 1) \\ &= P|\mathcal{J}_{equi}|(n+1). \end{aligned}$$

In the above, the first inequality follows from Lemma 2.2, and the second inequality follows from the fact $(|\mathcal{J}_{para}|-1)|\mathcal{J}_{para}| \ge 0$. □

**Proof of Equation (2.2):** For simplicity of presentation, let $C = 2 - \frac{2}{n+1}$. In order to avoid case-by-case analysis in the proof, we prove the claim by induction on the number of jobs which have non-zero execution time, $n$.

Since $2 - \frac{2}{n+1}$ is an increasing function of $n$ and jobs of zero length would change neither the squashed area bound nor the height bound, the claim would also hold when we allow $n$ to be the number of total jobs, including jobs of zero length. Therefore, we can simply prove the claim for the case when all jobs have non-zero lengths while allowing the inductive hypothesis to include the case when jobs of zero length are present.

For the base case $n = 1$, if the parallelism, $P_1$, of job $J_1$ is less than or equal to $P$ $(P_1 \le P)$, it is assigned $P_1$ processors $(h_1(f) = h_1)$. Otherwise, $P$ processors are assigned to the job $(h_1(e) = h)$ and its execution ends in time $\frac{P_1 h_1}{P}$, which is the same as the squashed area bound.

Assume the claim holds when the number of jobs of non-zero lengths is less than $n$. Consider the case of $n$ jobs, all of non-zero lengths, $\mathcal{J} = \{(P_i, h_i) : 1 \le i \le n\}$. If there are idle processors, the claims follows immediately. So we assume there are no idle processors. Without loss of generality, let $J_1 = (P_1, h_1)$ be the first to finish and let $\tau$ denote its completion time. Therefore, the remaining portion of jobs $J_i \in \mathcal{J}_{equi}$ is $(P_i, h_i - \frac{\tau\bar{p}}{P_i})$, and the remaining portion of jobs $J_i \in \mathcal{J}_{para}$ is $(P_i, h_i - \tau)$. Therefore, the flow time is

$$FT_{DEQ}(\mathcal{J}) = \tag{3.3}$$
$$n\tau + FT_{DEQ}(\{(P_i, h_i - \tfrac{\tau\bar{p}}{P_i}) : i \in \mathcal{J}_{equi}\} \tag{3.4}$$
$$\cup\{(P_i, h_i - \tau) : i \in \mathcal{J}_{para}\}), \tag{3.5}$$

where the first term is the completion time of $J_1$ plus the time that the other $n-1$ jobs have been in the system so far, and the second term is needed in recursion for the remaining portion of the $n-1$ jobs. By the inductive hypothesis, the second term in Equation (3.3) is bounded by

$$C \cdot A(\{(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}) : i \in \mathcal{J}_{equi}\}) + \tag{3.6}$$

$$C \cdot H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \tag{3.7}$$

$$C \cdot \sum_{i \in \mathcal{J}_{para}} (h_i - \tau), \tag{3.8}$$

where $h_i(e) \geq \frac{\tau\bar{p}}{P_i}$ for each $i \in \mathcal{J}_{equi}$. DEQ will redistribute processors among jobs in $\mathcal{J}_{equi}$ after the departure of $J_1$. Let $\mathcal{J}'_{para} \subseteq \mathcal{J}_{equi}$ be the subset of $\mathcal{J}_{equi}$ such that for each $i \in \mathcal{J}'_{para}$, $J_i$ is assigned $P_i$ processors after the redistribution. Let $\mathcal{J}'_{equi}$ be the rest of the jobs in $\mathcal{J}_{equi}$.

$$\forall i \in \mathcal{J}'_{equi} \quad h_i(e) > \tfrac{\tau\bar{p}}{P_i} \tag{3.9}$$

$$\forall i \in \mathcal{J}'_{para} \quad h_i(e) = \tfrac{\tau\bar{p}}{P_i} \tag{3.10}$$

From Equation (3.10), we have

$$A(\{(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}) : i \in \mathcal{J}_{equi}\}) = \tag{3.11}$$

$$A(\{(P_i, h_i(e) - \tfrac{\tau\bar{p}}{P_i}) : i \in \mathcal{J}'_{equi}\}). \tag{3.12}$$

Let the jobs in $\mathcal{J}'_{equi}$ be ordered as $j_1, j_2, \cdots, j_k$, $k = |\mathcal{J}'_{equi}|$, according to the increasing order of the amount of remaining work $P_i(h_i(e) - \frac{\tau\bar{p}}{P_i})$, $i \in \mathcal{J}'_{equi}$, which is the same as the increasing order of $P_i h_i(e)$, $i \in \mathcal{J}'_{equi}$. Then,

$$A(\{(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}) : i \in \mathcal{J}'_{equi}\}) \tag{3.13}$$

$$= \frac{1}{P}\sum_{i=1}^{k}(k-i+1)P_{j_i}(h_{j_i}(e) - \frac{\tau\bar{p}}{P_{j_i}}) \tag{3.14}$$

$$= \frac{1}{P}\sum_{i=1}^{k}(k-i+1)P_{j_i}h_{j_i}(e) - \tag{3.15}$$

$$\frac{1}{P}\sum_{i=1}^{k}(k-i+1)\tau\bar{p} \tag{3.16}$$

$$= A(\{(P_i, h_i(e)) : i \in \mathcal{J}'_{equi}\}) - \tag{3.17}$$

$$\frac{k(k+1)}{2P}\tau\bar{p}. \tag{3.18}$$

We also have

$$H(\mathcal{J}_{equi}(f)) + \sum_{i \in \mathcal{J}_{para}} (h_i - \tau) \tag{3.19}$$

$$= H(\mathcal{J}(f)) - |\mathcal{J}_{para}|\tau. \tag{3.20}$$

Combining (3.3), (3.8), (3.12), (3.18),and (3.20), we obtain an upper bound on $FT_{DEQ}(\mathcal{J})$:

$$n\tau - C \cdot \frac{k(k+1)\tau\bar{p}}{2P} - C \cdot |\mathcal{J}_{para}|\tau$$
$$+C \cdot A(\mathcal{J}'_{equi}) + C \cdot H(\mathcal{J}(f)).$$

Since

$$A(\mathcal{J}_{equi}(e)) = A(\mathcal{J}'_{equi}(e)) + \sum_{i=1}^{|\mathcal{J}'_{para}|} \frac{(k + |\mathcal{J}'_{para}| - i + 1)\tau\bar{p}}{P},$$

the above upper bound is equal to

$$n\tau - C \cdot \frac{k(k+1)\tau\bar{p}}{2P} - C \cdot |\mathcal{J}_{para}|\tau$$
$$-C \cdot \sum_{i=1}^{|\mathcal{J}'_{para}|} \frac{(k + |\mathcal{J}'_{para}| - i + 1)\tau\bar{p}}{P}$$
$$+C \cdot (A(\mathcal{J}_{equi}(e)) + H(\mathcal{J}(f))).$$

In order to show that this is no more than $C \cdot A(\mathcal{J}(e)) + C \cdot H(\mathcal{J}(f)) = C \cdot A(\mathcal{J}_{equi}(e)) + C \cdot H(\mathcal{J}(f))$, it is sufficient to show that

$$n \leq C \cdot \frac{k(k+1)\bar{p}}{2P} + C \cdot |\mathcal{J}_{para}| + C \cdot \sum_{i=1}^{|\mathcal{J}'_{para}|} \frac{(k + |\mathcal{J}'_{para}| - i + 1)\bar{p}}{P}.$$

This inequality is the same as

$$nP \leq C \cdot |\mathcal{J}_{para}|P + \frac{C}{2}|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}| + 1)\bar{p}.$$

Equivalently,

$$|\mathcal{J}_{equi}|P \leq (C-1)|\mathcal{J}_{para}|P + \frac{C}{2}|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}| + 1)\bar{p}.$$

Since $C = 2 - \frac{2}{n+1}$, the above inequality follows from Lemma 3.1.□

## 4 Multi-phased Parallelism and Interactive Jobs

At any point in time some jobs are assigned a number of processors equal to their parallelism (those in full-parallelism mode) and others are assigned $\bar{p}$ processors (those in equipartition mode). Both the parallelism of the jobs and $\bar{p}$ may change over time. We continue to use the notation $\mathcal{J}(f)$ for the portion of jobs in $\mathcal{J}$ in full-parallelism mode, and $\mathcal{J}(e)$ for the portion of jobs in equipartition mode.

THEOREM 4.1. *For jobs with multiple phases of parallelism, we have*

$$FT_{DEQ}(\mathcal{J}) \leq (2 - \frac{2}{n+1})A(\mathcal{J}(e)) + \tag{4.21}$$

$$(2 - \frac{2}{n+1})H(\mathcal{J}(f)). \tag{4.22}$$

*Therefore, DEQ is $4 - \frac{4}{n+1}$ competitive for mean job response time.*

**Proof:** Omitted and see the appendix. □

Since we formulate interactive jobs as alternating between periods of being blocked while waiting for input from a user and periods of processing, the above result also applies to interactive jobs.

COROLLARY 4.1. *DEQ is $4 - \frac{4}{n+1}$ competitive for the mean turnaround time of interactive jobs.* □

This would immediately carry over to sequential job scheduling problems and show the same competitive ratio for the Round-Robin Policy. However, in this case we have a better result.

THEOREM 4.2. *Round-Robin is $3-\frac{2}{n+1}$ competitive for the mean turnaround time of interactive jobs on sequential machines.*

**Proof:** Omitted and see appendix. □

## 5 Robustness of DEQ

The competitive ratio approach has been successfully applied to minimize the makespan [23]. There are, however, some objections to using makespan as the only computer system performance objective. One of them is that makespan does not distinguish one policy from another. Under our model, any work-conserving policy (i.e., no processors are idle as long as there are jobs available for execution [5]), has a competitive ratio of two, which is asymptotically optimal. On the other hand, using mean response time as the performance objective, Motwani, et al. [15] have shown that no scheduling policy can achieve a performance ratio that is better than $n^{1/3}$ when there are new arrivals and the scheduler has no *a priori* information about the execution time. The mean response time is a performance metric that is too difficult to minimize in this case. As a compromise, we use the makespan as the performance objective and consider competitive scheduling policies which are robust in the presence of faulty jobs. More precisely, we assume that there are up to $K$ faulty/infinite jobs in the system but the scheduler does not know which they are. Let $T_A(\mathcal{J})$ be the completion time of the last finished non-faulty/finite job under the scheduling policy $A$. Let $OPT(\mathcal{J})$ be the optimal completion time of non-faulty/finite jobs (with full information.) From now on, $OPT(\mathcal{J})$ refers to the optimal makespan. Obviously, faulty/infinite jobs are not executed under the optimal scheduler. The competitive ratio is defined as $\max_{\mathcal{J}} \frac{T_A(\mathcal{J})}{OPT(\mathcal{J})}$.

Since the same issue arises in uniprocessor systems, we first consider this case.

THEOREM 5.1. *In a system with $K$ faulty/infinite jobs, when there are new arrivals, the competitive ratio of the makespan of the Round-Robin policy is $K+1$, which is optimal.*

**Proof:** Omitted and see appendix. □

Now consider parallel job scheduling on multiprocessors.

THEOREM 5.2. *In a multiprocessor system with new arrivals and $K$ faulty/infinite jobs, the competitive ratio for the makespan of DEQ is $K+1$, which is the best possible competitive ratio.*

**Proof:** Omitted and see appendix. □

## 6 Remarks and Discussion

We started our work on competitive analysis for the parallel scheduling problem by facing two obstacles with this approach: the lower bound of Deng and Koutsoupias on scheduling an arbitrary DAG [3], and the lower bound of Motwani, et al. for scheduling new job arrivals. While the former points out that it is impossible to obtain a general on-line strategy that schedules arbitrary jobs on parallel machines (with a given communication latency) optimally/near-optimally, the latter points out that it is impossible to obtain a general on-line preemptive strategy that schedules sequential jobs on a uniprocessor if job arrivals are unpredictable[15]. (This result can also be extended to parallel jobs.) These two results raise serious doubts about the possibility of obtaining a near-optimal scheduling strategy in realistic computing environments for parallel/real time schedulings.

We avoid the first difficulty by focusing on a special class of parallel jobs. It would be interesting to extend our positive result to larger classes of parallel jobs. The explicit definition of interactive jobs introduced in this paper and the related constant competitive ratio result resolves the second difficulty; we can treat newly arriving jobs as interactive jobs with an initial *sleep* state not requiring CPU processing. The DEQ policy also stands out among other parallel scheduling policies in the presence of faulty/infinite jobs in that it achieves the optimal competitive ratio for makespan. This presents an alternative approach to handling the second difficulty.

A central question that must be considered when developing and comparing scheduling algorithms for multiprocessors is: what is the object function being used to determine how well the algorithm is performing. For example, is it more desirable to minimize mean response time, than minimizing makespan, or should maximizing throughput or system utilization be the main goals of the scheduler. As well, real systems must also be careful to provide quick turnaround time to interactive programs. Additional consideration must also be given to the the fact that in some cases knowing or deriving a competitive ratio for an algorithm does not mean that the complexity of the algorithm is acceptable or that an algorithm can be easily constructed [4].

## References

[1] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation", *Proceedings of Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, pp. 1-12, May, 1993.

[2] S. H. Chiang, R. K. Mansharamani, and M. Vernon, "Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies", *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 33-44, 1994.

[3] X. Deng and E. Koutsoupias, " Competitive Implementation Of Parallel Programs", *The Fourth Annual ACM–SIAM Symposium on Discrete Algorithms,* 1993, pp. 455-461.

[4] X. Deng and C. H. Papadimitriou, "On the Complexity of Cooperative Game Solution Concepts", *Mathematics of Operations Research*, Vol. 19, No. 2 (1994), pp. 257–266.

[5] D.L. Eager, J. Zahorjan, and E.D. Lazowska, "Speedup Versus Efficiency in Parallel Systems", *IEEE Trans. on Computers*, Vol. 38., No. 3, pp. 408-423, 1989.

[6] M. R. Garey, R. L. Graham, "Bounds for Multiprocessor Scheduling with Resource Constraints", *SIAM Journal of Computing, Vol. 4, No. 2*, pp. 187-200, June, 1975.

[7] K. Guha, "Using Parallel Program Characteristics in Dynamic Multiprocessor Allocation Policies" *M.Sc. Thesis, Technical Report CS-95-03*, Department of Computer Science, York University, May, 1995.

[8] A.R. Karlin, M.S. Manasse, L. Rudolph, and D.D. Sleator. "Competitive Snoopy Caching", *Algorithmica* **3** pp. 79-119, 1988.

[9] M. Kumar, "Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications", *IEEE Transactions on Computers* Vol. 37, No. 9, September, 1988, pp. 1088-1098.

[10] S. T. Leutenegger and R. D. Nelson, "Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments", *Technical Report RC 17086 (No. 75594)*, IBM T. J. Watson Research Center, Yorktown Heights, NY, August, 1991.

[11] W. Ludwig and P. Tiwari, "The Power of Choice in Scheduling Parallel Tasks", *Computer Science Department, University of Wisconsin, Madison, Report CS TR1190*, Madison, WI, November, 1993.

[12] S. T. Leutenegger and M. K. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Policies", *Proceedings of the 1990 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 226-236, Boulder, CO, May, 1990.

[13] R. Mansharamani and M. K. Vernon, "Qualitative Behavior of the EQS Parallel Processor Allocation Policy", *Technical Report CS TR 1192*, Computer Sciences Department, University of Wisconsin, Madison, Madison, WI, November, 1993.

[14] M.S. Manasse, L.A. McGeoch, and D.D. Sleator, "Competitive algorithms for on-line problems", *Twentieth ACM Annual Symposium on Theory of Computing,* pp. 322-333, 1988.

[15] R. Motwani, S. Phillips, and E. Torng "Non-Clairvoyant Scheduling", *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 422-431, Austin, Texas, January, 1993.

[16] C. McCann, R. Vaswani and J. Zahorjan, "A Dynamic Processor Allocation Policy for Multiprogrammed, Shared Memory Multiprocessors", *ACM Transactions on Computer Systems, Vol. 11, No. 2*, pp. 146-178, May, 1993.

[17] C. McCann and J. Zahorjan, "Scheduling Memory Constrained Jobs on Distributed Memory Parallel Computers", *Proceedings of International Joint Conference on Measurement & Modeling of Computer Systems, ACM Sigmetrics 95 and Performance 95*, pp. 208-219, 1995.

[18] C. Papadimitriou, and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms", *Proceedings of the 20th ACM Symposium on Theory of Computing*, pp. 510-513, 1988.

[19] K. C. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Multiprocessors", *Performance Evaluation, Vol. 9, No. 2-3*, pp. 107-140, May, 1994.

[20] K. C. Sevcik, "Characterizations of Parallelism in Applications and their use in Scheduling", *Proceedings of the 1989 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 171-180, May, 1989.

[21] A. Silberschatz, J. Peterson, P. Galvin, *Operating System Concepts*, Addison-Wesley Publishing Company, New York, 1991.

[22] D. D. Sleator, and R. E. Tarjan, "Amortized Efficiency of List Update and Paging Rules", *Communications of the ACM*, Vol. 28, No. 2, pp. 202-208, 1985.

[23] D. B. Shmoys, J. Wein, and D. P. Williamson, "Scheduling Parallel Machines On-line", *Foundations of Computer Science*, pp. 131-140, 1991.

[24] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors", *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pp. 159-166, 1989.

[25] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, P. S. Yu, "Scheduling Parallelizable Tasks to Minimize Average Response Time", *Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures*, pp. 200-209, June, 1994.

[26] J. Turek, U. Schwiegelshohn, J. L. Wolf, P. S. Yu, "Scheduling Parallel Tasks to Minimize Average Response Time", *Proceedings of the 5th SIAM Symposium*

*on Discrete Algorithms*, pp. 112-121, 1994.

[27] J. Turek, U. Schwiegelshohn, J. L. Wolf, P. S. Yu, "A Significantly Smarter Bound for a Slightly Smarter SMART Algorithm", *IBM Research Division, T. J. Watson Research Center, Report RC19422 (84462)*, Yorktown Heights, NY 10598, February, 1994.

[28] L. G. Valiant, "A Bridging Model for Parallel Computation", *CACM* , Vol. 33, No. 8, pp. 103-111, August, 1990.

[29] J. Zahorjan and C. McCann, "Processor Scheduling in Shared Memory Multiprocessors", *Proceedings of the 1990 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 214-225, Boulder, CO, May, 1990.

## 7 Appendix

**Proof of Theorem 4.1:** The conclusion of $4 - \frac{4}{n+1}$ competitiveness follows immediately from the fact that both the squashed area bound and the height bound are lower bounds on the optimal flow time. Our focus is thus on Equation (2.2). We consider an inductive proof using the result for single-phased jobs as the base case. A difficulty in this case is that the order of jobs in the squashed area bound may change as execution of the jobs (according to DEQ) proceed. To deal with this problem, we divide the execution time of the jobs into intervals such that in each interval, the parallelism of all jobs does not change; and the order, according to which the squashed area bound is applied, of the total remaining work to be executed under the equipartition mode of all jobs does not change. Thus between two consecutive intervals, either some job changes its parallelism or two jobs have the same amount of remaining work to be executed under the equipartition mode. We prove our claim by induction on the number of such intervals. For the base case, the parallelism of all jobs is the same and the claim follows from our result on jobs with a single phase of parallelism.

To apply the inductive proof, consider the execution of all jobs for $\tau$ time units in the first such interval. The case with idle processors is trivial. So we assume there are no idle processors. For jobs in full-parallelism mode during this period of time (denoted by $\mathcal{J}_{para}$), their height decreases by $\tau$. For jobs in equipartition mode during this period of time (denoted by $\mathcal{J}_{equi}$), their work decreases by $\bar{p}\tau$. Applying the inductive hypothesis to the remaining portions after the first time interval, we have

$$\begin{aligned}FT_{DEQ}(\mathcal{J}) \leq \\ n\tau + C \cdot [A(\mathcal{J}(e)) - \textstyle\sum_{i\in\mathcal{J}_{equi}} \frac{(n-i+1)\tau\bar{p}}{P}] \\ +C \cdot [H(\mathcal{J}(f)) - \textstyle\sum_{i\in\mathcal{J}_{para}} \tau],\end{aligned}$$

where $C = 2-\frac{2}{n+1}$. The condition that the order of total work in $\mathcal{J}(e)$ does not change is crucial in the above formulation. At the end of the interval, it is possible that two jobs may be tied in the remaining portion of $\mathcal{J}(e)$. We can exchange their order without changing the squashed area bound. The new order would be then be used for the next interval. To obtain our proof, it is sufficient to show:

$$n\tau \leq C \cdot \sum_{i\in\mathcal{J}_{equi}} \frac{(n-i+1)\tau\bar{p}}{P} + C \cdot |\mathcal{J}_{para}|\tau.$$

This holds if we have

$$n\tau \leq C \cdot \sum_{i=1}^{|\mathcal{J}_{equi}|} \frac{i\tau\bar{p}}{P} + C \cdot |\mathcal{J}_{para}|\tau,$$

which is equivalent to

$$nP \leq C \cdot \frac{|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1)\bar{p}}{2} + C \cdot |\mathcal{J}_{para}|P.$$

This can be shown in the same way as in the proof of Theorem 2.1 by applying Lemma 3.1. □

**Proof of Theorem 4.2:** Let $W(\mathcal{J}) = \sum_{i=1}^{n}(n-i+1)t_i$ for a job set $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$, where $t_i$ is the accumulative time job $J_i$ requires CPU processing, $1 \leq i \leq n$, and $t_1 \leq t_2 \leq \cdots \leq t_n$. Let $H(\mathcal{J}) = \sum_{i=1}^{n} h_i$, where $h_i$ is the accumulative time job $J_i$ does not require CPU processing, i.e., when it is blocked. Then, we show

$$(7.23) FT_{RR}(\mathcal{J}) \leq (2 - \frac{2}{n+1})W(\mathcal{J}) + H(\mathcal{J}).$$

The theorem follows from Inequality 7.23 since both $W(\mathcal{J})$ and $H(\mathcal{J})$ are lower bounds for the optimal flow time.

Similar to the above proof for parallel jobs, we divide execution by the Round-Robin policy into a finite number of intervals in which the order of remaining accumulative CPU times of jobs does not change and each job is in the same phase (ready to execute or blocked). We apply inductive proof to the number of such intervals. The base case follows from the result of Motwani, et al.[15]. Consider one such interval of length $\tau$, let $K$ be the set of jobs ready to execute, $k = |K|$. The case $k = 0$ is trivial and we thus assume $k \geq 1$. Each such job is executed for $\frac{\tau}{k}$ time units. The accumulative blocked time for each of other jobs decrease by $\tau$. Thus we have

$$\begin{aligned}FT_{RR}(\mathcal{J}) \leq \\ n\tau + (2 - \tfrac{2}{n+1})[W(\mathcal{J}) - \textstyle\sum_{i\in K}(n-i+1)\frac{\tau}{k}] \\ +H(\mathcal{J}) - (n-k)\tau,\end{aligned}$$

which is less than or equal to $(2-\frac{2}{n+1})W(\mathcal{J})+H(\mathcal{J})$ if

$$n \leq (2-\frac{2}{n+1})\sum_{i\in K}(n-i+1)\frac{1}{k}+(n-k).$$

This last inequality holds even for the worst choice of set $K$. □

**Proof of Theorem 5.1:** Without loss of generality, assume that all $K$ faulty jobs are present in the system at time $t_0$, and there are $N$ other jobs, whose execution time is $x_1$, $x_2$, ..., $x_N$, arriving at the system at time $t_1$, $t_2$, ..., $t_N$ respectively. We consider the following two cases:

If new jobs always arrive before the last good job in the system has finished execution, the optimal scheduler will leave no processor idle. The optimal makespan will be $\sum_{i=1}^{N}x_i$. A Round-Robin scheduler will always execute at least $\frac{1}{K+1}$ of the good jobs, and all of the good jobs will complete by time $\frac{1}{K+1}\sum_{i=1}^{N}x_i$. Therefore, the competitive ratio is bounded above by $K+1$ in this case.

In case new jobs arrive some time after the last good job in the system has finished execution, we first consider the optimal solution with complete information. Let $m$ be the maximum index when a job arrives at time $t_m$ and finds no remaining jobs in the system (all previous jobs have completed their execution). Let $t^*=t_m$. In this case, the optimal completion time will be $OPT=t^*+\sum_{i=m}^{N}x_i$. Using the Round-Robin policy, at the point in time $t^*$ the total amount of time remained for good jobs $x_1$, $x_2$, $\cdots$, $x_{m-1}$ to execute is at most $\frac{K}{K+1}t^*$. From then on, the processor will always devote a fraction (at least $\frac{1}{K+1}$) of its time to good jobs. Therefore, Round-Robin will finish all the good jobs within at most $(K+1)(\frac{K}{K+1}t^*+\sum_{i=m}^{N}t_i)$ time units after $t^*$. Thus the completion time of all good jobs will be at most $(K+1)(t^*+\sum_{i=m}^{N}t_i)$, which is $(K+1)OPT$. Therefore, the Round-Robin policy achieves a competitive ratio of $K+1$.□

**Proof of Theorem 5.2:** We examine the last finished job $J_i$ according to the DEQ policy. Let $\tau_0$ be its arrival time. Again, we divide the execution of this job $J_i$ into two parts: let $t_{para}$ be total of the time during which the number of processors allocated to it is equal to its parallelism, and let $t_{equi}$ be the total total time during which it is assigned its fair share of processors according to DEQ. Let $W_{equi}$ be the total amount of work executed by $J_i$ during $t_{equi}$. Since there are at most K faulty/infinite jobs, the total amount of work performed during $t_{equi}$ on the faulty/infinite jobs is no more than $KW_{equi}$. Let $W'$ be the total work performed on good jobs during $t_{equi}$. Then the completion time of DEQ is bounded by $t_0+t_{para}+\frac{KW_{equi}+W'}{P}$. On the other hand, for the optimal completion time, we have $t_0+t_{para}+\frac{W_{equi}}{P}\leq OPT$, $\frac{(K-1)W_{equi}}{P}\leq(K-1)OPT$, and $\frac{W'}{P}\leq OPT$. This concludes our proof that DEQ has competitive ratio of $K+1$ when the system has new job arrivals. □