SIAM J. COMPUT.
Vol. 30, No. 1, pp. 145–160

PREEMPTIVE SCHEDULING OF PARALLEL JOBS ON MULTIPROCESSORS*

XIAOTIE DENG†, NIAN GU‡, TIM BRECHT§, AND KAICHENG LU¶

Abstract. We study the problem of processor scheduling for n parallel jobs applying the method of competitive analysis. We prove that for jobs with a single phase of parallelism, a preemptive scheduling algorithm without information about job execution time can achieve a mean completion time within $2 - \frac{2}{n+1}$ times the optimum. In other words, we prove a competitive ratio of $2 - \frac{2}{n+1}$. The result is extended to jobs with multiple phases of parallelism (which can be used to model jobs with sublinear speedup) and to interactive jobs (with phases during which the job has no CPU requirements) to derive solutions guaranteed to be within $4 - \frac{4}{n+1}$ times the optimum. In comparison with previous work, our assumption that job execution times are unknown prior to their completion is more realistic, our multiphased job model is more general, and our approximation ratio (for jobs with a single phase of parallelism) is tighter and cannot be improved. While this work presents theoretical results obtained using competitive analysis, we believe that the results provide insight into the performance of practical multiprocessor scheduling algorithms that operate in the absence of complete information.

Key words. processor scheduling, parallel programs, competitive analysis, unknown information

AMS subject classifications. 68M20, 68Q22

PII. S0097539797315598

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 (the time at which the last job completes execution) is usually a natural objective function for many general scheduling problems, a number of different possibilities exist for CPU schedulers in a general purpose multiuser computing environment [31]. Nevertheless, minimizing the mean completion time (the sum of the times at which each job completes, divided by the number of jobs) is a commonly used objective function [22], [18], [34], [35], [27]. We can equivalently just consider the sum of the completion times. In this paper, the phrase *completion times* is used to imply that all jobs are available for execution at time zero, while the phrase *response time* implies that there are new job arrivals.

Several recent analytic results have been obtained for the problem of minimizing mean completion times using nonpreemptive scheduling algorithms which assume that

*Received by the editors January 27, 1997; accepted for publication (in revised form) May 27, 1999; published electronically April 25, 2000. A preliminary version of this paper appears in *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January, 1996, pp. 159–167. This research was funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and a strategic research grant and an RGC CERG grant from the City University of Hong Kong.

http://www.siam.org/journals/sicomp/30-1/31559.html

†Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR, P.R. China (deng@cs.cityu.edu.hk).

‡Department of Computer Science, York University, Toronto, Canada M3J 1P3 (nian_gu@yahoo.com).

§Department of Computer Science, University of Waterloo, Waterloo, ON Canada N2L 3G1 (brecht@cs.uwaterloo.ca).

¶Department of Computer Science, TsingHua University, Beijing, China, 100084 (lkc-dcs@mail.tsinghua.edu.cn).

job information is completely known [35], [34], [27], [18]. 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.

Using the terminology of Feitelson et al. [9], the work by Turek et al. [35] and Schwiegelshohn et al. [27] examines nonpreemptive scheduling policies using a rigid job model. That is, the number of processors required by a job is defined by the job and must be assigned to the job for its lifetime. Subsequent work [34], [18] relaxes the job model to consider moldable jobs. A moldable job [9] is one which can be run on any number of processors provided that once the processors have been allocated they remain allocated to that job for the duration of its execution. In this case the scheduler is free to determine the number of processors to allocate to each job. However, once a job is assigned processors it cannot be preempted. Additionally, some of these studies [34], [18] explicitly model jobs with sublinear speedup.

By comparison, in this work we model malleable jobs in order to examine dynamic preemptive scheduling policies. That is, we model jobs that are capable of executing with a changing number of processors and scheduling algorithms that can modify the number of processors allocated to jobs during their execution in order to adjust to changing requirements or system loads. Note that in using this model, threads of a job can migrate (i.e., they can be suspended on one processor and resumed at a later time on another processor). We also explicitly study jobs with multiple phases of parallelism in order to approximate jobs with sublinear speedups. During each phase of execution a job is capable of executing with perfect speedup until a maximum degree of parallelism is reached. The addition of extra processors beyond this limit neither increases nor decreases the execution time of the job. While each phase executes with linear speedup (up to the maximum degree of parallelism), multiple phases of execution with different maximum degrees of parallelism can be combined to produce an overall model of sublinear speedup.

In this paper we show that the dynamic equipartition (DEQ) policy [33], [38] produces mean completion times that are no more than $2 - \frac{2}{n+1}$ times the optimum for any set of n parallel jobs with one phase of parallelism, 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. In fact, *the ratio of* $2 - \frac{2}{n+1}$ *cannot be further improved mathematically* for jobs with a single phase of parallelism.

This result 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 to nonpreemptive algorithms in recent simulation and experimental studies [33], [17], [38], [16], [20], [21].

The remainder of the paper is organized as follows. We complete section 1 with a further description of the problem and a discussion of related work. In section 2, we give a formal definition of the DEQ allocation policy. Then we establish a lower bound on the optimal total completion time for parallel jobs by extending the squashed area bound and the height bound [34], using a completely different approach from those used previously. In section 3, we give a formal proof that the total completion time of DEQ is no more than twice the optimal total completion 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. Furthermore, in section 4, we show that 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. In section 5, we present theoretical results which demonstrate that DEQ is robust in the presence of faulty jobs. We consider the case where there are faulty jobs which may execute infinitely and show that, in this case, DEQ achieves the optimal competitive ratio for makespan. In section 6, we conclude the paper.

1.1. Preemptive scheduling. Schedulers in most general purpose computer systems are preemptive for several reasons [31]. First, job execution times may not be known prior to their completion. Thus, when nonpreemptive 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 nonpreemptive scheduling policy is used they may execute forever and exclude other jobs from 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 dynamically space-sharing processors instead [33], [38], [20], [26]. This is consistent with the trend toward coarse grained machines for general purpose parallel computations, as suggested in the LogP model [3]. 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 [36]. Independently, there have been extensive empirical studies on the preemptive cost caused by time/space-sharing scheduling policies [38], [20], [37]. Even for some cases when the preemption cost is relatively high, simulation and experimental studies support preemptive over nonpreemptive scheduling policies [38], [20], [23], [24].

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 from the optimal solution (which has and uses complete information about the job set) by more than a constant factor. The competitive analysis of algorithms is a measure of algorithms operating with incomplete information, first introduced in the study of system memory management [32], [13], [19]. 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 (which is different for different problems) 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 mean completion time (or makespan), $S(\mathcal{J})$, of the policy on a set of jobs, $\mathcal{J}$, over the minimum mean completion time (or makespan) $OPT(\mathcal{J})$ on the same job set $\mathcal{J}$: $\max_{all\ \mathcal{J}} \frac{S(\mathcal{J})}{OPT(\mathcal{J})}$. An algorithm is said to be *f(n)-competitive* in mean completion time (or makespan) 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, Wein, and Williamson studied the optimal competitive ratio in the makespan of sequential jobs being scheduled on parallel machines [30]. For minimizing the mean completion time of sequential jobs, Motwani, Phillips, and Torng have shown that a preemptive time-sharing policy, round-robin, achieves the optimal competitive ratio. It guarantees a mean completion time which is within $2 - \frac{2}{n+1}$ times optimal, and no policy can guarantee a better competitive ratio [22]. As further evidence that preemptive policies should be preferred to nonpreemptive policies, it is not hard to see that any nonpreemptive policy can result in a mean completion time that is $\Omega(n)$ times the optimum when n jobs are scheduled.

1.3. The job model. The most detailed description of a parallel program's execution on a multiprocessor is a data-dependency directed acyclic graph (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 [25], Deng and Koutsoupias show that given uniform communication delay, τ, 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 [4]. The same claim holds for both the bulk synchronous parallelism (BSP) and the LogP models. That is, for a parallel system with communication latency L between processors, the competitive ratio of any scheduler is at least $\frac{L}{\log L}$. This work shows that it is not possible for a compiler to optimally (or near-optimally) execute all jobs for distributed memory parallel systems, and it 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. Therefore, jobs are considered malleable and the scheduling algorithms are dynamic and preemptive, since they can adjust the number of processors allocated to a job during its execution [9].

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 [15]. During execution, if the parallelism of an application varies with time, it is said to have multiple phases of parallelism. Note that although our job model assumes linear speedup within each phase of parallelism, jobs with multiple phases of parallelism will execute with sublinear speedup. We also consider 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. Motwani, Phillips, and Torng [22] show that, for uniprocessor systems, the mean completion time of the round-robin scheduling policy is $2 - \frac{2}{n+1}$ times the optimum and that without a priori information about job execution times, no policy can guarantee mean completion times within a better approximation factor of the optimum (called the competitive ratio [32], [13], [19]).

The problem of minimizing the mean completion time of parallel jobs executing on multiprocessors is also of interest, and here a number of positive results exist.

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. [35] who introduce an approximation algorithm of 32 times the optimum for a nonpreemptive scheduling model. This result for nonpreemptive algorithms has been subsequently improved and extended [34], [27], [18].

A number of different preemptive policies have been proposed and studied for scheduling parallel jobs in multiprocessors [33], [29], [38], [20], [28], [26], [23], [24], [2]. In particular, experimental and simulation studies have shown that the DEQ algorithm yields low mean completion times under a variety of workloads and is reported to possess desirable properties of a good scheduler [33], [17], [16]. DEQ was first introduced to parallel scheduling by Tucker and Gupta as a *process control* policy [33] and was modified by Zahorjan and McCann [38]. The main idea behind this approach is to distribute processors evenly among jobs, provided they have sufficient parallelism.

One of the main drawbacks of DEQ, when compared with other approaches like gang scheduling, is that it requires each job to be implemented in such a way that the number of processes allocated to the job can change during its execution. It also requires significant coordination between the operating system and the run-time system. While these requirements might seem restrictive, studies have shown that it is relatively simple to write malleable applications, that coordination between the scheduler and run-time system is not prohibitive, and that performance is improved significantly when compared with other techniques [20], [10], [23], [24]. In addition, the DEQ algorithm is simple to implement and requires no information about job execution times. Therefore, this work examines the DEQ scheduling algorithm.

Our proof that DEQ is $2 - \frac{2}{n+1}$-competitive uses the notion of a *squashed area bound*, introduced for the nonpreemptive scheduling of parallel jobs [18], [35], [27], [34]. Turek et al. show that the minimum completion time for a set of jobs, $\mathcal{J} = \{(P_1, h_1), (P_2, h_2), \ldots, (P_n, h_n)\}$, is no more than the minimum completion time of the job set $\mathcal{J}_{squash} = \{(P, \frac{P_1 h_1}{P}), (P, \frac{P_2 h_2}{P}), \ldots, (P, \frac{P_n h_n}{P})\}$ [35]. The squashed area bound is the total completion time (the product of the number of jobs and the mean completion 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 [28].

2. Preliminaries. Consider n jobs in a system of P processors. Job J_i is characterized by the parallelism-time pair (P_i, h_i), and the amount of work is $w_i = P_i h_i$, $1 \leq i \leq n$. Denote the job set by $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Suppose that under a scheduler S the actual completion time of job J_i is t_i, $1 \leq i \leq n$. The total completion time of $\mathcal{J}$, denoted by $TC_S(\mathcal{J})$, is defined as $\sum_{i=1}^{n} t_i$. Then the mean completion time $MC_S(\mathcal{J})$ is defined as $\frac{TC_S(\mathcal{J})}{n}$. The height bound $H(\mathcal{J})$ is defined as $\sum_{i=1}^{n} h_i$ [35]. Since J_i requires at least h_i units of execution time, $1 \leq i \leq n$, $H(\mathcal{J})$ is an obvious lower bound on the optimal total completion time. Let the jobs be ordered according to their total work $w_1 \leq w_2 \leq \cdots \leq w_n$. The squashed area bound $A(\mathcal{J})$ is then defined as $\sum_{i=1}^{n} (n-i+1)\frac{w_i}{P}$ [35]. Notice that any preemptive scheduling of the job set $\mathcal{J}$ can be obtained by a preemptive scheduling of the job set $\{(P, \frac{w_1}{P}), (P, \frac{w_2}{P}), \ldots, (P, \frac{w_n}{P})\}$. It follows that $OPT(\mathcal{J}) \geq OPT\{(P, \frac{w_1}{P}), (P, \frac{w_2}{P}), \ldots, (P, \frac{w_n}{P})\}$. Since each job has the same parallelism, the shortest job first (or LWF) strategy gives the optimal solution for the total completion time, easily provable as in sequential systems (see [35], [28] for details). This gives 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 $\mathcal{J}(1) = \{(P_i, h_{i1}) : 1 \leq i \leq n\}$ 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 total completion 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 total completion 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 [35]. □

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. To simplify our proof, we allow a fractional number of processors to be assigned to a job, $\frac{P}{n}$, as long as that number is smaller 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 *full-parallelism mode* if P_i processors are assigned, and it is in *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 full-parallelism mode, it will stay in that mode until completion. Let $h_i(f)$ be the length of execution of job i in full-parallelism mode, and $h_i(e) = h_i - h_i(f)$. 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

$$TC_{DEQ}(\mathcal{J}) \le \left(2 - \frac{2}{n+1}\right)[A(\mathcal{J}(e)) + H(\mathcal{J}(f))]. \tag{2.1}$$

Combining this with Lemma 2.1, we have the following theorem.

THEOREM 2.3. $TC_{DEQ}(\mathcal{J}) \le (2 - \frac{2}{n+1})OPT(\mathcal{J})$. □

It is not hard to extend the lower bound for the competitive ratio of sequential jobs by Motwani, Phillips, and Torng [22] to this situation. In fact, we can replace each job in their proof with a parallel job of the same execution time with parallelism P, the number of processors in the system. Thus, this competitive ratio is optimal.

3. Minimizing mean completion time. In this section, we prove that (2.1) holds. Suppose jobs are initially divided into $\mathcal{J}_{para}$ and $\mathcal{J}_{equi}$ according to the discussion in section 2. We need the following lemma.

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 $(|\mathcal{J}_{para}|-1)|\mathcal{J}_{para}| \ge 0$. □

When one of the jobs, say J_1, finishes its execution under DEQ, we will reallocate 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 reallocation 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.

Proof of (2.1). 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 nonzero 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 nonzero lengths while allowing the induction 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 nonzero length is less than n. Consider the case of n jobs, all of nonzero length, $\mathcal{J} = \{(P_i, h_i) : 1 \le i \le n\}$. If there are idle processors, the claim 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 τ 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 total completion time is

$$TC_{DEQ}(\mathcal{J}) = n\tau + TC_{DEQ}\left(\left\{\left(P_i, h_i - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\} \cup \{(P_i, h_i - \tau) : i \in \mathcal{J}_{para}\}\right), \tag{3.1}$$

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 induction hypothesis,the second term

in (3.1) is bounded by C times

$$A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\}\right) + H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \sum_{i\in\mathcal{J}_{para}} (h_i - \tau), \tag{3.2}$$

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 $\mathcal{J}_{equi}$. Since jobs in $\mathcal{J}'_{para}$ will stay in full parallelism mode, we have

$$\forall i \in \mathcal{J}'_{para} h_i(e) = \frac{\tau\bar{p}}{P_i}. \tag{3.3}$$

Similarly,

$$\forall i \in \mathcal{J}'_{equi} h_i(e) > \frac{\tau\bar{p}}{P_i}. \tag{3.4}$$

From (3.3), we have

$$A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\}\right) = A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}'_{equi}\right\}\right). \tag{3.5}$$

Let the jobs in $\mathcal{J}'_{equi}$ be ordered as $j_1, j_2, \ldots, j_k$, $k = |\mathcal{J}'_{equi}|$, 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,

$$\begin{aligned} A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}'_{equi}\right\}\right) &= \frac{1}{P}\sum_{i=1}^{k}(k-i+1)P_{j_i}\left(h_{j_i}(e) - \frac{\tau\bar{p}}{P_{j_i}}\right) \\ &= \frac{1}{P}\sum_{i=1}^{k}(k-i+1)P_{j_i}h_{j_i}(e) - \frac{1}{P}\sum_{i=1}^{k}(k-i+1)\tau\bar{p}. \end{aligned}$$

Therefore,

$$A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}'_{equi}\right\}\right) = A(\{(P_i, h_i(e)) : i \in \mathcal{J}'_{equi}\}) - \frac{k(k+1)}{2P}\tau\bar{p}. \tag{3.6}$$

We also have

$$H(\mathcal{J}_{equi}(f)) + \sum_{i\in\mathcal{J}_{para}} (h_i - \tau) = H(\mathcal{J}(f)) - |\mathcal{J}_{para}|\tau. \tag{3.7}$$

Combining (3.1) and (3.2), we know that $TC_{DEQ}(\mathcal{J})$ is no more than

$$n\tau + C \cdot \left(A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\}\right) + H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \sum_{i\in\mathcal{J}_{para}} (h_i - \tau)\right).$$

By (3.5), this is the same as

$$n\tau + C \cdot \left(A\left(\left\{ \left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i} \right) : i \in \mathcal{J}'_{equi} \right\} \right) + H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \sum_{i \in \mathcal{J}_{para}} (h_i - \tau) \right).$$

Now by substituting (3.6) and (3.7), we have the following upper bound for $TC_{DEQ}(\mathcal{J})$:

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

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

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

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

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

This inequality is the same as

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

Equivalently,

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

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

4. Multiphased 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*

$$TC_{DEQ}(\mathcal{J}) \leq \left(2 - \frac{2}{n+1}\right) A(\mathcal{J}(e)) + \left(2 - \frac{2}{n+1}\right) H(\mathcal{J}(f)). \tag{4.1}$$

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

Proof. 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 total completion time. Our focus is thus on (4.1). 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 in section 3. To apply the inductive proof, consider the execution of all jobs for τ time units in the interval during which no job changes its parallelism. 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 τ. For jobs in equipartition mode during this period of time (denoted by $\mathcal{J}_{equi}$), their work decreases by $\bar{p}\tau$. Applying the induction hypothesis to the remaining portions after the first time interval, we have

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

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 then be used for the next interval. To obtain our proof, it is sufficient to show

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

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 2.3 by applying Lemma 3.1. In fact, Lemma 3.1 holds independently of the fact that jobs contain a single phase of parallelism or multiple phases of parallelism. □

Interactive jobs can be formulated as jobs alternating between periods of being blocked while waiting for input from a user (requiring no processor), and periods of processing. Our result above also applies to such interactive jobs. We assume that users respond to each request for input in a finite amount of time. The degenerate case, where a user may not respond to an input request, is considered in detail in section 5. Thus, we have the following corollary.

COROLLARY 4.2. *DEQ is* $4 - \frac{4}{n+1}$ *competitive for the mean completion time of interactive jobs.* □

This result immediately carries over to sequential job scheduling problems and produces the same competitive ratio for the round-robin policy. However, in this case we have a better result.

THEOREM 4.3. *Round-robin is* $3 - \frac{2}{n+1}$ *competitive for the mean completion time of interactive jobs on sequential machines.*

Proof. Let $W(\mathcal{J}) = \sum_{i=1}^{n}(n-i+1)t_i$ for a job set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, where t_i is the cumulative 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 cumulative time job J_i does not require CPU processing (i.e., when it is blocked). We now show

$$TC_{RR}(\mathcal{J}) \leq \left(2 - \frac{2}{n+1}\right) W(\mathcal{J}) + H(\mathcal{J}). \tag{4.2}$$

The theorem follows from (4.2) since both $W(\mathcal{J})$ and $H(\mathcal{J})$ are lower bounds for the optimal total completion time.

In a fashion similar to the above proof of Theorem 4.1, we use the round-robin policy to divide the execution of jobs into a finite number of intervals in which the order of remaining cumulative CPU times of jobs does not change and each job is in the same phase (ready to execute or be blocked). We apply an inductive proof to the number of such intervals. The base case follows from the result of Motwani, Phillips, and Torng [22]. Consider one such interval of length τ; 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 cumulative blocked time for each of the other jobs is decreased by τ. Thus we have

$$TC_{RR}(\mathcal{J}) \leq n\tau + \left(2 - \frac{2}{n+1}\right)\left[W(\mathcal{J}) - \sum_{i \in K}(n-i+1)\frac{\tau}{k}\right] + H(\mathcal{J}) - (n-k)\tau,$$

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

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

This last inequality holds even for the worst choice of set K. Therefore, the theorem follows. □

5. Robustness of DEQ. Competitive analysis has been successfully applied in multiple processor scheduling problems to minimize the makespan [30]. There are, however, some objections to using makespan as a computer system's performance measure. 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 [6]) has a competitive ratio of two, which is asymptotically optimal [1]. On the other hand, when there are new arrivals and the scheduler has no a priori information about the execution time, using mean response time as the performance objective, Motwani, Phillips, and Torng [22] have shown that no scheduling policy can achieve a performance ratio that is better than $n^{1/3}$. In this case, mean response time is a performance metric that is extremely difficult to minimize without any a priori information, since the adversary is able to choose arrival times and job sizes that can defeat any scheduler. Rather than using competitive analysis one might consider using mathematical analysis or simulation to compare various scheduling algorithms. In addition, Kellerer, Tautenhahn, and Woeginger [14] have shown that mean response time is also difficult to minimize even in the presence of complete job information. Unfortunately, these approaches require either knowing or making assumptions about job arrival and execution times. The power of competitive analysis is that a positive result applies without the need to understand or justify the workload model. That is, it applies for all workloads.

Since it is not possible to obtain a positive result for minimizing the competitive ratio for mean response times when there are new arrivals, as a compromise, we examine competitive scheduling policies for minimizing makespan. These policies are robust in the presence of infinite (possibly faulty) jobs. More precisely, we assume that there are up to K infinite jobs in the system but the scheduler does not know which jobs they are. Let $T_A(\mathcal{J})$ be the completion time of the last finished finite job under the scheduling policy A. Let $OPT(\mathcal{J})$ be the optimal completion time with full information about finite jobs (and information about which are infinite jobs). Notice that from now on, $OPT(\mathcal{J})$ refers to the optimal makespan instead of mean completion time. Obviously, 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 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. To show that (K+1) is a lower bound on the competitive ratio, consider a case of $K+1$ jobs including K infinite jobs and one finite job with execution time t. No matter what scheduling algorithm is used, the adversary always assigns the finite job to execute last. Thus, the total time required to complete the finite job is at least $(K+1)t$. An optimal schedule with complete information will take only t time units, running only the finite job in t time units and not even running the infinite jobs. Therefore, the competitive ratio is at least $K+1$ for any scheduling algorithm.

For the upper bound, without loss of generality, we 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 $x_1, x_2, \ldots, x_N$, arriving at time $t_1, t_2, \ldots, t_N$, respectively. We consider the following two cases.

Under the optimal scheduling algorithm with complete information, if new jobs always arrive before the last finite job in the system has finished, no processor will be left idle, and 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 finite jobs, and all of the finite 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.

The second case to consider under the optimal scheduling algorithm is when new jobs arrive some time after the last finite job in the system has finished execution. Let indices be defined according to job arrival orders. In this case, denote by m 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 $OPT = t^* + \sum_{i=m}^{N} x_i$. Consider the last consecutive interval $[\tau_1, \tau_2]$, $0 \le \tau_1 < \tau_2 \le t^*$, during which there is at least one finite job executing under the round-robin policy. The total length of finite jobs arriving during this interval will be no more than $\tau_2 - \tau_1$, since an optimal scheduler (with complete information) will finish all of them. Since the round-robin policy will assign at least $\frac{1}{K+1}$ of the total processing power to one finite job whenever there is one in the system, by time τ_2 the remaining total length of jobs is no more than $\frac{K}{K+1}(\tau_2 - \tau_1) \le \frac{K}{K+1}t^*$. From then on, the processor will always devote a fraction (at least $\frac{1}{K+1}$) of its time to finite jobs. Therefore, round-robin will finish all of the finite jobs within at most $(K+1)(\frac{K}{K+1}t^* + \sum_{i=m}^{N} x_i)$ time units after t^*. Thus the completion time of all finite jobs will be at most $(K+1)(t^* + \sum_{i=m}^{N} x_i)$, which is $(K+1)OPT$. Therefore, the round-robin policy achieves a competitive ratio of $K+1$. □

A similar argument can be applied to parallel job scheduling on multiprocessors.

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

Proof. Without loss of generality, assume that all K faulty jobs are present in the system at time $t_0 = 0$, and there are N other finite jobs, $J_1, J_2, \ldots, J_N$, which arrive at time $t_1, t_2, \ldots, t_N$, respectively. Similarly, we examine the last finished finite job J_i according to the DEQ policy. Again, we divide the execution of this job J_i into two parts: let t_{para} be the total time during which the number of processors allocated to the job is equal to its parallelism, and let t_{equi} be the 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 the period it is assigned its fair share of processors (i.e., during t_{equi}). Since there are at most K infinite jobs, the total amount of work performed on the infinite jobs during the period when J_i is assigned its fair share of processors is no more than KW_{equi}. Let W' be the total work performed on finite jobs during the same period. Then the completion time of DEQ is bounded by $t_i + t_{para} + \frac{KW_{equi}+W'}{P}$. On the other hand, for the optimal completion time, we have $t_i + t_{para} + \frac{W_{equi}}{P} \le OPT$, which is the minimum time to complete J_i, ignoring all other jobs. Furthermore, $\frac{(K-1)W_{equi}}{P} \le (K-1)OPT$, and $\frac{W'}{P} \le OPT$. This concludes our proof that DEQ has a competitive ratio of $K+1$ when the system has new arrivals of jobs with multiple phases of parallelism. □

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 [4] and the lower bound of Motwani, Phillips, and Torng for scheduling with 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) near-optimally, the latter points out that it is impossible to obtain a general online preemptive strategy that schedules sequential jobs on a uniprocessor to minimize mean response time if job arrivals are unpredictable [22]. (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 jobs with new arrivals.

In this paper, we avoid the first difficulty by utilizing a special class of parallel jobs. This class includes jobs with sublinear speedups by modeling them using multiple phases of parallelism. With Edmonds and Chinn we have also recently extended some of our positive results to larger classes of jobs and to models of systems which more explicitly account for communication delays [7]. It seems that the second difficulty may be much harder to overcome since it holds even for single processor scheduling. If the objective of computer system scheduling is indeed to minimize mean response time, more empirical workload studies will help to understand arrival times and job sizes in typical computer systems. (Recent work has begun to examine workload characteristics on multiprocessor systems [8], [11].) Workload information could be used to more accurately reflect typical arrival times and job sizes and to perhaps avoid scenarios that prevent competitive ratios from being obtained for algorithms that perform well in practice. Alternatively, we may study other objective functions. The explicit definition of interactive jobs introduced in this paper and the related constant competitive ratio (for minimizing makespan) is an effort in this direction. Under these conditions (including the presence of infinite jobs), the DEQ policy stands out among other parallel scheduling policies in that it achieves the optimal competitive ratio for makespan. An alternative approach to this second difficulty has been proposed by Kalyanasundaram and Pruhs [12]. They show that a moderate increase in the speed of the processor used by a nonclairvoyant scheduler can effectively give this processor power equal to clairvoyance.

A central question that must be considered when developing and comparing scheduling algorithms for multiprocessors is, What is the objective function being used to determine how well the algorithm is performing? For example, is it more desirable to minimize mean response time than to minimize makespan, or should maximizing throughput be the main goal 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 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. These issues are quite similar to solution concepts in the game theoretical framework for the study of sharing economic resources, and it might be interesting to explore possible links between them [5].

Acknowledgment. We would like to thank the anonymous referees whose comments helped to significantly improve this paper.

REFERENCES

[1] T. Brecht, X. Deng, and N. Gu, *Competitive dynamic processor allocation for parallel applications*, Parallel Process. Lett., 7 (1997), pp. 89–100.

[2] T. Brecht, and K. Guha, *Using parallel program characteristics in dynamic processor allocation policies*, Performance Evaluation, 27–28 (1996), pp. 519–539.

[3] 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*, in Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, New York, 1993, pp. 1–12.

[4] X. Deng and E. Koutsoupias, *Competitive implementation of parallel programs*, in Proceedings of the Fourth Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1993, pp. 455–461.

[5] X. Deng and C. Papadimitriou, *On the complexity of cooperative game solution concepts*, Math. Oper. Res., 19 (1994), pp. 257–266.

[6] D. Eager, J. Zahorjan, and E. Lazowska, *Speedup versus efficiency in parallel systems*, IEEE Trans. Comput., 38 (1989), pp. 408–423.

[7] J. Edmonds, D. Chinn, T. Brecht, and X. Deng, *Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics*, in Proceedings of the 22nd ACM Symposium on the Theory of Computing, ACM, New York, 1997, pp. 120–129.

[8] D.G. Feitelson and B. Nitzberg, *Job characteristics of a production parallel scientific workload on the NASA AMES iPSC/*860, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 1291, Springer-Verlag, Berlin, New York, Heidelberg, 1997, pp. 1–34.

[9] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong, *Theory and practice in parallel job scheduling*, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 949, Springer-Verlag, Berlin, New York, Heidelberg, 1995, pp. 337–360.

[10] A. Gupta, A. Tucker, and S. Urushibara, *The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications*, in Proceedings of the ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, New York, 1991, pp. 120–132.

[11] S. Hotvoy, *Workload evolution on the Cornell theory center IBM SP*2, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 1162, Springer-Verlag, Berlin, New York, Heidelberg, 1996, pp. 27–40.

[12] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, in Proceedings of the 36th Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 214–221.

[13] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, *Competitive Snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[14] H. Kellerer, T. Tautenhahn, and G.J. Woeginger, *Approximability and nonapproximability results for minimizing total flowtime on a single machine*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1996, pp. 418–426.

[15] M. Kumar, *Measuring parallelism in computation-intensive scientific/engineering applications*, IEEE Trans. Comput., 37 (1988), pp. 1088–1098.

[16] S. Leutenegger and R. Nelson, *Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments*, Technical Report RC 17086 75594, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1991.

[17] S. Leutenegger and M. Vernon, *The performance of multiprogrammed multiprocessor scheduling policies*, in Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1990, pp. 226–236.

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

[19] M. Manasse, L. McGeoch, and D. Sleator, *Competitive algorithms for on-line problems*, in Proceedings of the 30th ACM Annual Symposium on the Theory of Computing, ACM, New York, 1988, pp. 322–333.

[20] C. McCann, R. Vaswani, and J. Zahorjan, *A dynamic processor allocation policy for multiprogrammed shared memory multiprocessors*, ACM Trans. Comput. Systems, 11 (1993), pp. 146–178.

[21] C. McCann and J. Zahorjan, *Scheduling memory constrained jobs on distributed memory parallel computers*, in Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 95 and Performance 95, ACM, New York, 1995, pp. 208–219.

[22] R. Motwani, S. Phillips, and E. Torng *Non-clairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.

[23] T. Nguyen, R. Vaswani, and J. Zahorjan, *Using runtime measured workload characteristics in parallel processor scheduling*, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 1162, Springer-Verlag,

Berlin, New York, Heidelberg, 1996, pp. 175–199.

[24] T. Nguyen, R. Vaswani, and J. Zahorjan, *Maximizing speedup through self-tuning of processor allocation*, in Proceedings of the 10th International Parallel Processing Symposium, Honolulu, HI, 1996, pp. 463–468.

[25] C. Papadimitriou and M. Yannakakis, *Towards an architecture-independent analysis of parallel algorithms*, in Proceedings of the 20th ACM Symposium on the Theory of Computing, ACM, New York, 1988, pp. 510–513.

[26] E. Parsons and K. Sevcik, *Multiprocessor scheduling for high-variability service time distributions*, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 949, Springer-Verlag, Berlin, New York, Heidelberg, 1995, pp. 127–145.

[27] U. Schwiegelshohn, W. Ludwig, J.L. Wolf, J. Turek, and P.S. Yu, *Smart SMART bounds for weighted response time scheduling*, SIAM J. Comput., 28 (1999), pp. 237–253.

[28] K. Sevcik, *Application scheduling and processor allocation in multiprogrammed multiprocessors*, Performance Evaluation, 9 (1994), pp. 107–140.

[29] K. Sevcik, *Characterizations of parallelism in applications and their use in scheduling*, in Proceedings of the 1989 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1989, pp. 171–180.

[30] D.B. Shmoys, J. Wein, and D.P. Williamson, *Scheduling parallel machines on-line*, SIAM J. Comput., 24 (1995), pp. 1313–1331.

[31] A. Silberschatz and P. Galvin, *Operating System Concepts*, Addison–Wesley, New York, 1994.

[32] D. Sleator and R. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. Assoc. Comput. Mach., 28 (1985), pp. 202–208.

[33] A. Tucker and A. Gupta, *Process control and scheduling issues for multiprogrammed shared-memory multiprocessors*, in Proceedings of the 12th ACM Symposium on Operating Systems Principles, ACM, New York, 1989, pp. 159–166.

[34] J. Turek, W. Ludwig, J.L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. Yu, *Scheduling parallelizable tasks to minimize average response time*, in Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1994, pp. 200–209.

[35] J. Turek, U. Schwiegelshohn, J.L. Wolf, and P. Yu, *Scheduling parallel tasks to minimize average response time*, in Proceedings of the Fifth ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1994, pp. 112–121.

[36] L. Valiant, *A bridging model for parallel computation*, Comm. Assoc. Comput. Mach., 33 (1990), pp. 103–111.

[37] C. Wu, *Processor Scheduling in Multiprogrammed Shared-Memory NUMA Multiprocessors*, M.Sc. thesis, University of Toronto, Toronto, ON, Canada, October, 1993.

[38] J. Zahorjan and C. McCann, *Processor scheduling in shared memory multiprocessors*, in Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1990, pp. 214–225.