Provably Efficient
Two-Level Adaptive Scheduling

Yuxiong He[1], Wen-Jing Hsu[1], and Charles E. Leiserson[2]

[1] Nanyang Technological University, Nanyang Avenue 639798, Singapore,
yxhe@mit.edu, hsu@ntu.edu.sg
[2] Massachusetts Institute of Technology, Cambridge, MA 02139, USA,
cel@mit.edu

Abstract. Multiprocessor scheduling in a shared multiprogramming environment can be structured in two levels, where a kernel-level job scheduler allots processors to jobs and a user-level thread scheduler maps the ready threads of a job onto the allotted processors. This paper presents two-level scheduling schemes for scheduling "adaptive" multithreaded jobs whose parallelism can change during execution. The AGDEQ algorithm uses dynamic-equipartioning (DEQ) as a job-scheduling policy and an adaptive greedy algorithm (A-GREEDY) as the thread scheduler. The ASDEQ algorithm uses DEQ for job scheduling and an adaptive work-stealing algorithm (A-STEAL) as the thread scheduler. AGDEQ is suitable for scheduling in centralized scheduling environments, and ASDEQ is suitable for more decentralized settings. Both two-level schedulers achieve $O(1)$-competitiveness with respect to makespan for any set of multithreaded jobs with arbitrary release time. They are also $O(1)$-competitive for any batched jobs with respect to mean response time. Moreover, because the length of the scheduling quantum can be adjusted to amortize the cost of context-switching during processor reallocation, our schedulers provide control over the scheduling overhead and ensure effective utilization of processors.

1 Introduction

Multiprocessors are often used for multiprogrammed workloads where many parallel applications share the same machine. As Feitelson points out in his excellent survey [27], schedulers for these machines can be implemented using two levels: a kernel-level **job scheduler** which allots processors to jobs, and a user-level **thread scheduler** which maps the threads belonging to a given job onto the allotted processors. The job schedulers may implement either **space-sharing**, where jobs occupy disjoint processor resources, or **time-sharing**, where different jobs may share the same processor resources at different times. Moreover, both the thread scheduler and the job scheduler may be either **adaptive** (called "dynamic" in [19]), allowing the number of processors allotted to a job to vary

while the job is running, or **nonadaptive** (called "static" in [19]), where a job runs on a fixed number of processors over its lifetime. A **clairvoyant** scheduling algorithm may use knowledge of the jobs' execution time, whereas a **non-clairvoyant** algorithm assumes nothing about the execution time of the jobs. This paper presents two provably efficient two-level adaptive schedulers, each of which schedules jobs nonpreemptively and without clairvoyance.

With **adaptive** scheduling [4] (called "dynamic" scheduling in many other papers [27,37,41,58,60]), the job scheduler can change the number of processors allotted to a job while the job executes. Thus, new jobs can enter the system, because the job scheduler can simply recruit processors from the already executing jobs and allot them to the new jobs. Without an adequate feedback mechanism, however, both adaptive and nonadaptive schedulers may waste processor cycles, because a job with low parallelism may be allotted more processors than it can productively use.

If individual jobs provide **parallelism feedback** to the job scheduler, waste can be avoided. When a job does not require many processors, it can release the excess processors to the job scheduler to be reallotted to jobs in need. When a job needs more processors, it can make a request to the job scheduler. Based on this parallelism feedback, the job scheduler can adaptively change the allotment of processors according to the availability of processors and the system administrative policy.

A two-level scheduler communicates the parallelism feedback by each job requesting processors from a job scheduler at regular intervals, called **quanta**. The quantum length is typically chosen to be long enough to amortize the scheduling overheads, including the cost of reallotting processors among the jobs. The job scheduler uses the parallelism feedback to assign the available processors to the jobs according to its administrative policy. During the quantum, the job's allotment does not typically change. Once a job is allotted processors, the job's thread scheduler maps the job's threads onto the allotted processors, reallocating them if necessary as threads are spawned and terminated.

Various researchers [20,21,29,41,59] have proposed the use of **instantaneous parallelism** — the number of processors the job can effectively use at the current moment — as the parallelism feedback to the job scheduler. Unfortunately, using instantaneous parallelism as feedback can either cause gross misallocation of processor resources [49] or introduce significant scheduling overhead. For example, the parallelism of a job may change substantially during a scheduling quantum, alternating between parallel and serial phases. Depending on which phase is currently active, the sampling of instantaneous parallelism may lead the task scheduler to request either too many or too few processors. Consequently, the job may either waste processor cycles or take too long to complete. On the other hand, if the quantum length is set to be small enough to capture frequent changes in instantaneous parallelism, the proportion of time spent reallotting processors among the jobs increases, resulting in a high scheduling overhead.

A-Greedy [1] and A-Steal [2,3] are two adaptive thread schedulers that provide the parallelism feedback to the job scheduler. Rather than using instan-

taneous parallelism, these thread schedulers employ a single summary statistic and the job's behavior in the previous quantum to make processor requests of the job scheduler. Even though this parallelism feedback is generated based on the job's history and may not be correlated to the job's future parallelism, A-GREEDY and A-STEAL still guarantee to make effective use of the available processors.

Intuitively, if each job provides good parallelism feedback and makes productive use of available processors, a good job scheduler should ensure that *all* the jobs perform well. In this paper, we affirm this intuition for A-GREEDY and A-STEAL in the case when the job scheduler implements dynamic equipartitioning (DEQ) [41, 55]. DEQ gives each job a fair allotment of processors based on the job's request, while allowing processors that cannot be used by a job to be reallocated. DEQ was introduced by McCann, Vaswani, and Zahorjan [41] based on earlier work on equipartitioning by Tucker and Gupta [55], and it has been studied extensively [20, 21, 24, 25, 29, 36, 40–42, 45, 46, 59].

This paper shows that efficient two-level adaptive schedulers can ensure that all jobs can perform well. AGDEQ, which couples DEQ with A-GREEDY, is suitable for centralized thread scheduling, such as might be used to schedule data-parallel jobs, wherein each job's thread scheduler can dispatch all the ready threads to the allotted processors in a centralized manner. ASDEQ, which couples DEQ with A-STEAL, is suitable when each job distributes threads over the allotted processors using decentralized work-stealing [13, 16, 31, 47].

The main contributions of this paper are as follows. In a centralized environment, AGDEQ guarantees $O(1)$-competitiveness against an optimal clairvoyant scheduler with respect to makespan. For any set of batched jobs, where all jobs have the same release time, AGDEQ also achieves $O(1)$-competitiveness with respect to mean response time. In a decentralized settings where the scheduler has no knowledge of all the available threads at the current moment, ASDEQ guarantees $O(1)$-competitiveness with respect to makespan for any set of jobs with arbitrary job release time. It is also $O(1)$-competitive with respect to the mean response time for batched jobs. Unlike many previous results, which either assume clairvoyance [18, 33, 34, 38, 43, 48, 50, 56, 57] or use instantaneous parallelism [14, 21, 22], our schedulers remove these restrictive assumptions. We generate parallelism feedback after each quantum based on the job's behavior in the past quantum. Even though job's future parallelism may not be correlated with its history of parallelism, our schedulers can still guarantee constant competitiveness for both the makespan and the mean response time. Moreover, because the quantum length can be adjusted to amortize the cost of context-switching during processor reallocation, our schedulers provide control over the scheduling overhead and ensure effective utilization of processors.

The remainder of this paper is organized as follows. Section 2 describes the job model, scheduling model, and objective functions. Section 3 describes the AGDEQ algorithm. Sections 4 and 5 analyze the competitiveness of AGDEQ with respect to makespan and mean response time, respectively. Section 6 presents the ASDEQ algorithm and analyzes its performance. Section 7 gives a lower

bound on the competitiveness for mean response time. Section 9 concludes the paper by raising issues for future research.

2 Models and Objective Functions

This section provides the background formalisms for two-level scheduling, which will be used to study AGDEQ and ASDEQ. We formalize the job model, define the scheduling model, and present the optimization criteria of makespan and mean response time.

Job Model

A **two-level scheduling problem** consists of a collection of independent jobs $\mathcal{J} = \left\{ J_1, J_2, \ldots, J_{|\mathcal{J}|} \right\}$ to be scheduled on a collection of P identical processors. This paper restricts its attention to the situation where $|\mathcal{J}| \leq P$, that is, the number of jobs does not exceed the number of processors. (The situation where the parallel computer may sometimes be heavily loaded with jobs remains an interesting open problem.) Like prior work on scheduling of multithreaded jobs [8, 10–13, 26, 32, 44], we model the execution of a multithreaded job J_i as a dynamically unfolding directed acyclic graph (dag) such that $J_i = (V(J_i), E(J_i))$ where $V(J_i)$ and $E(J_i)$ represent the sets of J_i's vertices and edges, respectively. Similarly, let $V(\mathcal{J}) = \bigcup_{J_i \in \mathcal{J}} V(J_i)$. Each vertex $v \in V(\mathcal{J})$ represents a unit-time instruction. The **work** $T_1(i)$ of the job J_i corresponds to the total number of vertices in the dag, that is, $T_1(i) = |V(J_i)|$. Each edge $(u, v) \in E(J_i)$ represents a dependency between the two vertices. The precedence relationship $u \prec v$ holds if and only if there exists a path from vertex u to vertex v in $E(J_i)$. The **critical-path length** $T_\infty(i)$ corresponds to the length of the longest chain of precedence dependencies. The **release time** $r(i)$ of the job J_i is the time immediately after which J_i becomes first available for processing. For a **batched** job set $\mathcal{J}$, all jobs in $\mathcal{J}$ have the same release time. (Without loss of generality, we assume that $r(i) = 0$ for all $J_i \in \mathcal{J}$.)

Scheduling Model

Our scheduling model assumes that time is broken into a sequence of equal-sized **scheduling quanta** $1, 2, \ldots$, each of length L, where each quantum q includes the interval $[Lq, Lq + 1, \ldots, L(q + 1) - 1]$ of time steps. The quantum length L is a system configuration parameter chosen to be long enough to amortize scheduling overheads. These overheads might include the time to reallocate processors among the various jobs and the time for the thread scheduler to communicate with the job scheduler, which typically involves a system call.

The job scheduler and thread schedulers interact as follows. The job scheduler may reallocate processors between quanta. Between quantum $q - 1$ and quantum q, the thread scheduler (for example, A-GREEDY or A-STEAL) of a given job J_i determines the job's **desire** $d(i, q)$, which is the number of processors J_i wants for quantum q. The thread scheduler provides the desire $d(i, q)$ to the job scheduler as its parallelism feedback. Based on the desire of all running jobs, the job scheduler follows its processor-allocation policy (for example, dynamic equi-partitioning) to determine the **allotment** $a(i, q)$ of the job with the constraint

that $a(i,q) \leq d(i,q)$. Once a job is allotted its processors, the allotment does not change during the quantum. Consequently, the thread scheduler must do a good job in estimating how many processors it will need in the next quantum, as well as scheduling the ready threads on the allotted processors. Moreover, the thread scheduler must operate in an online and nonclairvoyant manner, oblivious to the future characteristics of the dynamically unfolding dag.

A **schedule** $\chi = (\tau, \pi)$ of a job set $\mathcal{J}$ on P processors is defined as two mappings $\tau : V(\mathcal{J}) \rightarrow \{1, 2, \ldots, \infty\}$ and $\pi : V(\mathcal{J}) \rightarrow \{1, 2, \ldots, P\}$, which map the vertices in the job set $\mathcal{J}$ to the set of time steps and to the set of processors in the machine, respectively. A valid mapping must preserve the precedence relationship of each job: for any two vertices $u, v \in V(\mathcal{J})$, if $u \prec v$, then $\tau(u) < \tau(v)$, that is, the vertex u must be executed before the vertex v. A valid mapping must also ensure that a processor is only assigned to one job at any time: for any two distinct vertices $u, v \in V(\mathcal{J})$, we have $\tau(u) \neq \tau(v)$ or $\pi(u) \neq \pi(v)$.

Objective Functions

We can now define the objective functions that a two-level scheduler should minimize.

Definition 1 Let χ be a schedule of a job set $\mathcal{J}$ on P processors. The **completion time** a job $J_i \in \mathcal{J}$ is

$$T_\chi(i) = \max_{v \in V_i} \tau(v) ,$$

and the **makespan** of $\mathcal{J}$ is

$$\mathrm{T}_\chi(\mathcal{J}) = \max_{J_i \in \mathcal{J}} T_\chi(i) .$$

The **response time** of a job $J_i \in \mathcal{J}$ is

$$\mathrm{R}_\chi(i) = T_\chi(i) - r(i) ,$$

the **total response time** of $\mathcal{J}$ is

$$\mathrm{R}_\chi(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} \mathrm{R}_\chi(i) ,$$

and the **mean response time** of $\mathcal{J}$ is

$$\overline{\mathrm{R}}_\chi(\mathcal{J}) = \mathrm{R}_\chi(\mathcal{J})/|\mathcal{J}| .$$

That is, the completion time of J_i is simply the time at which the schedule completes the execution of J_i. The makespan of $\mathcal{J}$ is the time taken to complete all jobs in the job set. The response time of a job J_i is the duration between its release time $r(i)$ and the completion time $T_\chi(i)$. The total response time of a job set is the sum of the response times of the individual jobs, and the mean response time is the arithmetic average of the jobs' response times. For batched jobs where $r(i) = 0$ for all $J_i \in \mathcal{J}$, the total response time simplifies to $\mathrm{R}_\chi(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_\chi(i)$.

Competitiveness

The competitive analysis of an online scheduling algorithm compares the algorithm against an optimal clairvoyant algorithm. Let $T^*(\mathcal{J})$ denote the makespan of the jobset $\mathcal{J}$ scheduled by an optimal clairvoyant scheduler, and $\chi(A)$ denote the schedule produced by an algorithm A for the job set $\mathcal{J}$. A deterministic algorithm A is said to be ***c-competitive*** if there exist constants $c > 0$ and $b \geq 0$ such that $T_{\chi(A)}(\mathcal{J}) \leq c \cdot T^*(\mathcal{J}) + b$ holds for the schedule $\chi(A)$ of each job set. A randomized algorithm A is said to be ***c-competitive*** if there exists constants $c > 0$ and $b \geq 0$ such that $E\left[T_{\chi(A)}(\mathcal{J})\right] \leq c \cdot T^*(\mathcal{J}) + b$ holds for the schedule $\chi(A)$ of each job set. Thus, for each job set $\mathcal{J}$, a c-competitive algorithm is guaranteed to have makespan (or expected makespan) within a factor c of that incurred in the optimal clairvoyant algorithm (up to the additive constant b). We shall show that AGDEQ and ASDEQ are c-competitive with respect to makespan, where $c > 0$ is a small constant. For the mean response time, we shall show that our algorithm is $O(1)$-competitive for batched jobs.

3 The AGDEQ Algorithm

AGDEQ is a two-level adaptive scheduler, which uses A-GREEDY [1] as its thread scheduler and DEQ [41] as its job scheduler. Given a set $\mathcal{J}$ of jobs and P processors, DEQ works at the kernel level, partitioning the P processors among the jobs. Within each job, A-GREEDY schedules threads at user level onto the allotted processors. The interactions between DEQ and A-GREEDY follow the scheduling model described in Section 2. At the beginning of each quantum q, the A-GREEDY thread scheduler for each job $J_i \in \mathcal{J}$ provides its desire $d(i, q)$ as parallelism feedback to the DEQ job scheduler. DEQ collects the desire information from all jobs and decides the allotment $a(i, q)$ for each job J_i. In this section, we briefly overview the basic properties of A-GREEDY and DEQ.

The Adaptive Greedy Thread Scheduler

A-GREEDY [1] is an adaptive greedy thread scheduler with parallelism feedback. In a two-level adaptive scheduling system, A-GREEDY performs the following functions.

- Between quanta, it estimates its job's desire and requests processors from the job scheduler using its ***desire-estimation algorithm***.

- During the quantum, it schedules the ready threads of the job onto the allotted processors using its ***thread-scheduling algorithm***.

We now describe each of these algorithms.

A-GREEDY's desire-estimation algorithm is parameterized in terms of a ***utilization parameter*** $\delta > 0$ and a ***responsiveness parameter*** $\rho > 1$, both of which can be tuned to affect variations in guaranteed bounds for waste and completion time.

Before each quantum, A-GREEDY for a job $J_i \in \mathcal{J}$ provides parallelism feedback to the job scheduler based on the J_i's history of utilization for the previous quantum. A-GREEDY classifies quanta as "satisfied" versus "deprived" and "efficient" versus "inefficient." A quantum q is ***satisfied*** if $a(i, q) = d(i, q)$,

in which case J_i's allotment is equal to its desire. Otherwise, the quantum is **deprived**. The quantum q is **efficient** if A-GREEDY utilizes no less than a δ fraction of the total allotted processor cycles during the quantum, where δ is the utilization parameter. Otherwise, the quantum is **inefficient**. Of the four possibilities of classification, however, A-GREEDY only uses three: inefficient, efficient-and-satisfied, and efficient-and-deprived.

Using this three-way classification and the job's desire for the previous quantum, A-GREEDY computes the desire for the next quantum using a simple multiplicative-increase, multiplicative-decrease strategy. If quantum $q - 1$ was inefficient, A-GREEDY decreases the desire, setting $d(i, q) = d(i, q - 1)/\rho$, where ρ is the responsiveness parameter. If quantum $q - 1$ was efficient and satisfied, A-GREEDY increases the desire, setting $d(i, q) = \rho d(i, q - 1)$. If quantum $q - 1$ was efficient but deprived, A-GREEDY keeps desire unchanged, setting $d(i, q) = d(i, q - 1)$.

A-GREEDY's thread-scheduling algorithm is based on greedy scheduling [12, 15, 28]. After A-GREEDY for a job $J_i \in \mathcal{J}$ receives its allotment $a(i, q)$ of processors from the job scheduler, it simply attempts to keep the allotted processors as busy as possible. During each time step, if there are more than $a(i, q)$ ready threads, A-GREEDY schedules any $a(i, q)$ of them. Otherwise, it schedules all of them.

The Dynamic-Equipartitioning Job Scheduler

DEQ is a dynamic-equipartitioning job scheduler [41, 55] which attempts to give each job a fair share of processors. If a job cannot use its fair share, however, DEQ distributes the extra processors across the other jobs. More precisely, upon receiving the desires $\{d(i, q)\}$ from the thread schedulers of all jobs $J_i \in \mathcal{J}$, DEQ executes the following **processor-allocation algorithm**:

1. Set $n = |\mathcal{J}|$. If $n = 0$, return.

2. If the desire for every job $J_i \in \mathcal{J}$ satisfies $d(i, q) \geq P/n$, assign each job $a(i, q) = P/n$ processors.

3. Otherwise, let $\mathcal{J}' = \{J_i \in \mathcal{J} : d(i, q) < P/n\}$. Allot $a(i, q) = d(i, q)$ processors to each $J_i \in \mathcal{J}'$. Update $\mathcal{J} = \mathcal{J} - \mathcal{J}'$. Go to Step 1.

Accordingly, for a given quantum all jobs receive the same number of processors to within 1, unless their desire is less. To simplify the analysis in this paper, we shall assume that all deprived jobs receive exactly the same number of processors, which we term the **mean deprived allotment** for the quantum. Relaxing this assumption may double the execution-time bound of a job, but our algorithms remain $O(1)$-competitive. A tighter but messier analysis retains the constants of the simpler analysis presented here.

4 Makespan of AGDEQ

This section shows that AGDEQ is c-competitive with respect to makespan for a constant $c \geq 1$. The exact value of c is related to the choice of the utilization parameter and responsiveness parameter in A-GREEDY. In this section, we first

review lower bounds for makespan. Then, we analyze the competitiveness of AGDEQ in the simple case where all jobs are released at time step 0 and the scheduling quantum length is $L = 1$. Finally, we analyze the competitiveness of AGDEQ for the general case.

Lower Bounds

Given a job set $\mathcal{J}$ and P processors, lower bounds on the makespan of any job scheduler can be obtained based on release time, work, and critical-path length. Recall that for a job $J_i \in \mathcal{J}$, the quantities $r(i)$, $T_1(i)$, and $T_\infty(i)$ represent the release time, work, and critical-path length of J_i, respectively. Let $\mathrm{T}^*(\mathcal{J})$ denote the makespan produced by an optimal scheduler on a job set $\mathcal{J}$ scheduled on P processors. Let $T_1(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_1(i)$ denote the total work of the job set. The following two inequalities give two lower bounds on the makespan [14]:

$$\mathrm{T}^*(\mathcal{J}) \geq \max_{J_i \in \mathcal{J}} \{r(i) + T_\infty(i)\} \ , \tag{1}$$

$$\mathrm{T}^*(\mathcal{J}) \geq T_1(\mathcal{J})/P \ . \tag{2}$$

Analysis of a Simple Case

To ease the understanding of the analysis, we first consider the simple case where all jobs are released at time step 0 and the quantum length $L = 1$. We show that in this case, AGDEQ is $O(1)$-competitive with respect to makespan. Afterward, we shall extend the analysis to the general case.

The next two lemmas, proved in [1], bound the satisfied steps and the waste of any single job scheduled by A-GREEDY when the quantum length is $L = 1$. We restate them as a starting point for our analysis.

Lemma 1 *[1] Suppose that* A-GREEDY *schedules a job J_i with critical-path length $T_\infty(i)$ on a machine with P processors. Let $\rho = 2$ denote* A-GREEDY*'s responsiveness parameter, $\delta = 1$ its utilization parameter, and $L = 1$ the quantum length. Then,* A-GREEDY *produces at most $2T_\infty(i) + \lg P + 1$ satisfied steps.*
□

Lemma 2 *[1] Suppose that* A-GREEDY *schedules a job J_i with work $T_1(i)$ on a machine. If $\rho = 2$ is* A-GREEDY*'s responsiveness parameter, $\delta = 1$ is its utilization parameter, and $L = 1$ is the quantum length, then* A-GREEDY *wastes no more than $2T_1(i)$ processor cycles in the course of the computation.* □

The next lemma shows that for the simple case, AGDEQ is $O(1)$-competitive with respect to makespan. Let $\chi = (\tau, \pi)$ be the schedule of a job set $\mathcal{J}$ produced by AGDEQ. For simplicity we shall use the notation $\mathrm{T}(\mathcal{J}) = \mathrm{T}_\chi(\mathcal{J})$ for the remaining of the section.

Lemma 3 *Suppose that a job set $\mathcal{J}$ is scheduled by* AGDEQ *on a machine with P processors, and suppose that all jobs arrive at time 0. Let $\rho = 2$ denote* A-GREEDY*'s responsiveness parameter, $\delta = 1$ its utilization parameter, and L the quantum length. Then, the makespan of $\mathcal{J}$ is bounded by*

$$\mathrm{T}(\mathcal{J}) \leq 5\mathrm{T}^*(\mathcal{J}) + \lg P + 1 \ ,$$

where $\mathrm{T}^*(\mathcal{J})$ *is the makespan produced by an optimal clairvoyant scheduler.*

Proof. Suppose that the job J_k is the last job completed in the execution of the job set $\mathcal{J}$ scheduled by AGDEQ. Since the scheduling quantum length is $L = 1$, we can treat each scheduling quantum as a time step. Let $S(k)$ and $D(k)$ denote the set of satisfied steps and the set of deprived steps respectively for job J_k. Since J_k is the last job completed in the job set, we have $\mathrm{T}(\mathcal{J}) = |S(k)| + |D(k)|$. We bound $|S(k)|$ and $|D(k)|$ separately.

By Lemma 1, we know that the number of satisfied steps for job J_k is $|S(k)| \leq 2T_\infty(i) + \lg P + 1$.

We now bound the number of deprived steps for J_k. If a step t is deprived for job J_k, the job gets fewer processors than it requested. On such a step $t \in D(k)$, DEQ must have allotted all the processors, and so we have $\sum_{J_i \in \mathcal{J}} a(i, t) = P$, where $a(i, t)$ denotes the allotment of the job J_i on step t. Let $a(\mathcal{J}, D(k)) = \sum_{t \in D(k)} \sum_{J_i \in \mathcal{J}} a(i, t)$ denote the total processor allotment of all jobs in $\mathcal{J}$ over J_k's deprived steps $D(k)$. We have $a(\mathcal{J}, D(k)) = \sum_{t \in D(k)} \sum_{J_i \in \mathcal{J}} a(i, t) = \sum_{t \in D(k)} P = P|D(k)|$. Since any allotted processor is either working on the ready threads of the job or wasted because of insufficient parallelism, the total allotment for any job J_i is bounded by the sum of its total work $T_1(i)$ and its total waste $w(i)$. By Lemma 2, the waste for the job J_i is $w(i) \leq 2T_1(i)$, which is at most twice its work. Thus, the total allotment for job J_i is at most $3T_1(i)$, and the total allotment for all jobs is at most $\sum_{J_i \in \mathcal{J}} 3T_1(i) = 3T_1(\mathcal{J})$. Therefore, we have $a(\mathcal{J}, D(k)) \leq 3T_1(\mathcal{J})$. Given that $a(\mathcal{J}, D(k)) \leq 3T_1(\mathcal{J})$ and $a(\mathcal{J}, D(k)) = P|D(k)|$, we have $|D(k)| \leq 3T_1(\mathcal{J})/P$.

Thus, we have $\mathrm{T}(\mathcal{J}) = |S(k)| + |D(k)| \leq 3T_1(\mathcal{J})/P + 2T_\infty(k) + \lg P + 1$. Combining this bound with Inequalities (1) and (2), we obtain $\mathrm{T}(\mathcal{J}) \leq 5\mathrm{T}^*(\mathcal{J}) + \lg P + 1$. □

Since P is the number of processors on the machine, which is an independent variable with respect to any job set $\mathcal{J}$, Lemma 3 indicates that AGDEQ is 5-competitive with respect to makespan.

Analysis of the General Case

With the intuition from the simple case in hand, we now generalize the makespan analysis of AGDEQ to job sets with arbitrary job release times and scheduled with any quantum length L. First, we state two lemmas from [1] that describe the satisfied steps and the waste of a single job scheduled by A-GREEDY. Then, we show that AGDEQ is $O(1)$-competitive with respect to makespan in the general case.

Lemma 4 *[1] Suppose that* A-GREEDY *schedules a job J_i with critical-path length $T_\infty(i)$ on a machine with P processors. Let ρ denote* A-GREEDY*'s responsiveness parameter, δ its utilization parameter, and L the quantum length,. Then,* A-GREEDY *produces at most $2T_\infty(i)/(1-\delta) + L\log_\rho P + L$ satisfied steps.* □

Lemma 5 *[1] Suppose that* A-GREEDY *schedules a job J_i with work $T_1(i)$ on a machine. Let ρ denote* A-GREEDY*'s responsiveness parameter, δ its utilization*

parameter, and L the quantum length. Then, A-GREEDY *wastes at most* $(1 + \rho - \delta)T_1(i)/\delta$ *processor cycles in the course of the computation.* $\qquad\square$

The following theorem analyzes the makespan of any job set $\mathcal{J}$ with arbitrary release times, when $\mathcal{J}$ is scheduled by AGDEQ with quantum length L. The makespan bound is based on the release time $r(i)$, critical-path length $T_\infty(i)$, and work $T_1(i)$ of individual job J_i, and the total work $T_1(\mathcal{J})$ of the job set $\mathcal{J}$.

Theorem 6 *Suppose* AGDEQ *schedules a job set* $\mathcal{J}$ *on a machine with* P *processors. Let* ρ *denote* A-GREEDY*'s responsiveness parameter,* δ *its utilization parameter, and* L *the quantum length. Then,* AGDEQ *completes the job set in*

$$\mathrm{T}(\mathcal{J}) \leq \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P} + \frac{2}{1-\delta}\max_{J_i \in \mathcal{J}}\{T_\infty(i) + r(i)\} + L\log_\rho P + 2L$$

time steps.

Proof. The proof is similar to that in the simple case for Lemma 3. Let job J_k be the last job to complete among the jobs in $\mathcal{J}$. Let $S(k)$ and $D(k)$ denote the set of satisfied steps and the set of deprived steps for J_k, respectively. The earliest that the job J_k can start its execution is at the beginning of the quantum immediately after J_k's release, which is the quantum q satisfying $Lq < r(k) \leq L(q+1)$. Thus, we have $\mathrm{T}(\mathcal{J}) < r(k) + L + |S(k)| + |D(k)|$. From Lemma 4, we know that the number of satisfied steps is $|S(k)| \leq 2T_\infty(k)/(1-\delta) + L\log_\rho P + L$. It remains to bound the quantity $|D(k)|$.

By definition, DEQ must have allotted all processors to jobs on any step $t \in D(k)$ where J_k is deprived. Thus, the total allotment of $\mathcal{J}$ over J_k's deprived steps $D(k)$ is $a(\mathcal{J}, D(k)) = \sum_{t \in D(k)}\sum_{J_i \in \mathcal{J}} a(i,t) = P|D(k)|$. Since any allotted processor is either working or wasted, the total allotment for any job J_i is bounded by the sum of its total work $T_1(i)$ and total waste $w(i)$. By Lemma 5, the waste for the job J_i is at most $(\rho - \delta + 1)/\delta$ times its work, and hence, the total allotment for job J_i is at most $T_1(i) + w(i) \leq (\rho+1)T_1(i)/\delta$, and the total allotment for all jobs is at most $\sum_{J_i \in \mathcal{J}}(\rho+1)T_1(i)/\delta = ((\rho+1)/\delta)T_1(\mathcal{J})$. Consequently, we have $a(\mathcal{J}, D(k)) \leq ((\rho+1)/\delta)T_1(\mathcal{J})$. Since $a(\mathcal{J}, D(k)) = P|D(k)|$, it follows that

$$|D(k)| < \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P}\ .$$

Combining these bounds, we obtain

$$\begin{aligned}
\mathrm{T}(\mathcal{J}) &< r(k) + L + |D(k)| + |S(k)| \\
&\leq r(k) + L + \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P} + \frac{2}{1-\delta}T_\infty(k) + L\log_\rho P + L \\
&\leq \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P} + \frac{2}{1-\delta}(r(k) + T_\infty(k)) + L\log_\rho P + 2L \\
&\leq \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P} + \frac{2}{1-\delta}\max_{J_i \in \mathcal{J}}\{T_\infty(i) + r(i)\} + L\log_\rho P + 2L\ .
\end{aligned}$$

$\square$

Since both $T_1(\mathcal{J})/P$ and $\max_{J_i \in \mathcal{J}} \{T_\infty(i) + r(i)\}$ are lower bounds of $\mathrm{T}^*(\mathcal{J})$, we obtain the following corollary.

Corollary 7 *Suppose that* AGDEQ *schedules a job set* $\mathcal{J}$ *on a machine with* P *processors. Let* ρ *denote* A-GREEDY*'s responsiveness parameter,* δ *its utilization parameter, and* L *the quantum length. Then,* AGDEQ *completes the job set in*

$$\mathrm{T}(\mathcal{J}) \leq \left(\frac{\rho+1}{\delta} + \frac{2}{1-\delta}\right) \mathrm{T}^*(\mathcal{J}) + L \log_\rho P + 2L$$

time steps, where $\mathrm{T}^*(\mathcal{J})$ *is the makespan of* $\mathcal{J}$ *produced by an optimal clairvoyant scheduler.* $\square$

When $\delta = 0.5$ and ρ is approaching 1, the competitiveness ratio $(\rho + 1)/\delta + 2/(1-\delta)$ approaches its minimum value 8. Thus, AGDEQ is $(8+\epsilon)$-competitive with respect to makespan for any constant $\epsilon > 0$.

5   Mean Response Time of AGDEQ for Batched Jobs

This section shows that AGDEQ is $O(1)$-competitive for batched jobs with respect to the mean response time, an important measure for multiuser environments where we desire as many users as possible to get fast response from the system. To analyze the mean response time of job sets scheduled by AGDEQ, we first describe lower bounds and some preliminary concepts. Then, we prove that AGDEQ is $O(1)$-competitive with respect to mean response time for batched jobs.

Lower Bounds and Preliminaries

Before stating the lower bounds on mean response time for a batched job set, we first define some terms.

Definition 2 Given a finite list $\mathcal{A} = \langle \alpha_i \rangle$ of $n = |\mathcal{A}|$ integers, define $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ to be a permutation satisfying $\alpha_{f(1)} \leq \alpha_{f(2)} \leq \cdots \leq \alpha_{f(n)}$. The *squashed sum* of $\mathcal{A}$ is defined as

$$\text{sq-sum}(\mathcal{A}) = \sum_{i=1}^{n} (n - i + 1)\alpha_{f(i)} .$$

The *squashed work area* of a job set $\mathcal{J}$ on a set of P processors is

$$\text{swa}(\mathcal{J}) = \frac{1}{P} \text{sq-sum}(\langle T_1(i) \rangle) ,$$

where $T_1(i)$ is the work of job $J_i \in \mathcal{J}$. The *aggregate critical-path length* of $\mathcal{J}$ is

$$T_\infty(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_\infty(i) ,$$

where $T_\infty(i)$ is the critical-path length of job $J_i \in \mathcal{J}$.

The research in [22,56,57] establishes two lower bounds for the mean response time:

$$\overline{R^*}(\mathcal{J}) \geq T_\infty(\mathcal{J})/|\mathcal{J}| \ , \tag{3}$$

$$\overline{R^*}(\mathcal{J}) \geq \text{swa}\,(\mathcal{J})/|\mathcal{J}| \ , \tag{4}$$

where $\overline{R^*}(\mathcal{J})$ denotes the mean response time of $\mathcal{J}$ scheduled by an optimal clairvoyant scheduler. Both the aggregate critical-path length $T_\infty(\mathcal{J})$ and the squashed work area $\text{swa}\,(\mathcal{J})$ are lower bounds for the total response time $R^*(\mathcal{J})$ under an optimal clairvoyant scheduler.

We extend the classification of "satisfied" versus "deprived" from quanta to time steps. A job J_i is **satisfied** at step $t \in [Lq, Lq+1, \ldots, L(q+1)-1]$ if J_i is satisfied at the quantum q. Otherwise, the time step t is **deprived**. At time step t, let $\mathcal{JS}(t)$ denote the set of jobs that are satisfied, and let $\mathcal{JD}(t)$ denote the set of jobs that are deprived. According to DEQ, all deprived jobs receive the mean deprived allotment.

To assist in the analysis of the mean response time, we now define some auxiliary concepts.

Definition 3 Suppose that a job set $\mathcal{J}$ is scheduled by AGDEQ on P processors. For any job $J_i \in \mathcal{J}$, let $S(i)$ and $D(i)$ denote the sets of satisfied and deprived time steps, respectively. The **total satisfied time** of $\mathcal{J}$ is

$$\text{sat}\,(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} |S(i)| \ .$$

The **accumulated allotment** of J_i is

$$a(i) = \sum_{t=1}^{\infty} a\,(i,t) \ .$$

The **accumulated deprived allotment** of J_i is

$$a\,(i, D(i)) = \sum_{t \in D(i)} a\,(i,t) \ .$$

The **squashed deprived allotment area** of $\mathcal{J}$ is

$$\text{sdaa}\,(\mathcal{J}) = \frac{1}{P}\,\text{sq-sum}(\langle a\,(i, D(i))\rangle) \ .$$

Thus, $\text{sat}\,(\mathcal{J})$ is the total number of satisfied steps of all jobs in $\mathcal{J}$, $a(i)$ is the job J_i's total allotment on all time steps, $a\,(i, D(i))$ is its total allotment during all its deprived steps, and $\text{sdaa}\,(\mathcal{J})$ is $1/P$ of the squashed sum of the accumulated deprived allotments for all jobs in $\mathcal{J}$.

Analysis

We now turn to show that AGDEQ is $O(1)$-competitive with respect to mean response time for batched jobs. Let $\chi = (\tau, \pi)$ be the schedule of a job set $\mathcal{J}$ produced by AGDEQ. For simplicity we shall use the notations $\text{R}(\mathcal{J}) = \text{R}_\chi(\mathcal{J})$ and $\overline{\text{R}}(\mathcal{J}) = \overline{\text{R}}_\chi(\mathcal{J})$. Let ρ and δ be A-GREEDY's responsiveness and utilization parameters, respectively. We shall establish the bound

$$\overline{\text{R}}(\mathcal{J}) \leq \left(2 - \frac{2}{|\mathcal{J}| + 1}\right) \left(\left(\frac{\rho + 1}{\delta} + \frac{2}{1 - \delta}\right) \overline{\text{R}^*}(\mathcal{J}) + L \log_\rho P + L\right) \,,$$

where $\overline{\text{R}^*}(\mathcal{J})$ is the mean response time produced by an optimal clairvoyant scheduler.

Our analysis comprises four major steps. First, we prove three technical lemmas concerning squashed sums. Second, we prove that

$$\text{R}(\mathcal{J}) \leq \left(2 - \frac{2}{|\mathcal{J}| + 1}\right) (\text{sdaa}(\mathcal{J}) + \text{sat}(\mathcal{J})) \,, \tag{5}$$

thereby relating the total response time $\text{R}(\mathcal{J})$ to the squashed deprived allotment area $\text{sdaa}(\mathcal{J})$ and the total satisfied time $\text{sat}(\mathcal{J})$. Third, we relate the squashed deprived allotment area $\text{sdaa}(\mathcal{J})$ and the squashed work area $\text{swa}(\mathcal{J})$. Finally, we relate the total satisfied time $\text{sat}(\mathcal{J})$ to the aggregate critical-path length $T_\infty(\mathcal{J})$. Since both $\text{swa}(\mathcal{J})$ and $T_\infty(\mathcal{J})$ are lower bounds on the total response time, we can derive an upper bound of the mean response time against the optimal.

We begin with three technical lemmas that describe properties of the squashed sum.

Lemma 8 *Let $\langle \alpha_i \rangle$ and $\langle \beta_i \rangle$ be two lists of nonnegative integers with m elements each, and suppose that $\alpha_i \leq \beta_i$ for $i = 1, 2, \ldots, m$. Then, we have $\text{sq-sum}(\langle \alpha_i \rangle) \leq \text{sq-sum}(\langle \beta_i \rangle)$.*

Proof. Let $f : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$ be the permutation satisfying $\alpha_{f(1)} \leq \alpha_{f(2)} \leq \cdots \leq \alpha_{f(m)}$, and let $g : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$ be the permutation satisfying $\beta_{g(1)} \leq \beta_{g(2)} \leq \cdots \leq \beta_{g(m)}$.

We first show that $\alpha_{f(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, m$. Suppose for the purpose of contradiction that there exists a $j \in \{1, 2, \ldots, m\}$ such that $\alpha_{f(j)} > \beta_{g(j)}$. Then, there must be at least j integers smaller than $\alpha_{f(j)}$ in $\langle \beta_i \rangle$, namely $\beta_{g(1)}, \beta_{g(2)}, \ldots, \beta_{g(j)}$. Since $\alpha_i \leq \beta_i$ for $i = 1, 2, \ldots, m$, we have $\alpha_{g(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, j$. Thus, there are at least j elements smaller than $\alpha_{f(j)}$ in $\langle \alpha_i \rangle$, namely $\alpha_{g(1)}, \alpha_{g(2)}, \ldots, \alpha_{g(j)}$. But, since $\alpha_{f(j)}$ is the jth smallest number in $\langle \alpha_i \rangle$, we obtain the contradiction that there are at most $j - 1$ integers smaller than $\alpha_{f(j)}$ in $\langle \alpha_i \rangle$, thereby establishing that that $\alpha_{f(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, m$.

Consequently, by Definition 2, we have

$$\text{sq-sum}(\langle \alpha_i \rangle) = \sum_{i=1}^{m} (m - i + 1)\alpha_{f(i)}$$

$$\leq \sum_{i=1}^{m}(m-i+1)\beta_{g(i)}$$
$$= \text{sq-sum}(\langle\beta_i\rangle) \ .$$

$\square$

Lemma 9 *Let l, h, and m be nonnegative integers such that $l \leq m$. Suppose that a list $\langle\alpha_i\rangle$ of m nonnegative integers has total value $\sum_{i=1}^{m}\alpha_i = lh$ and that each α_i satisfies $\alpha_i \leq h$. Then, the list's squashed sum satisfies $\text{sq-sum}(\langle\alpha_i\rangle) \geq hl(l+1)/2$, and its minimum occurs when $\alpha_1 = \alpha_2 = \cdots = \alpha_l = 0$ and $\alpha_{m-l+1} = \alpha_{m-l+2} = \cdots = \alpha_m = h$.*

Proof. Suppose for the purpose of contradiction that a given list $\langle\alpha_i\rangle$ of integers minimizes the function $\text{sq-sum}(\langle\alpha_i\rangle)$ but does not satisfy $\alpha_1 = \alpha_2 = \cdots = \alpha_l = 0$ and $\alpha_{m-l+1} = \alpha_{m-l+2} = \cdots = \alpha_m = h$. Then, there must exist at least one integer $\alpha_x > 0$ with index $x < m-l+1$ and at least one integer $\alpha_y < h$ with index $y \geq m-l+1$.

Define another list $\langle\alpha_i'\rangle$ of integers such that $\alpha_x' = \alpha_x - 1$, $\alpha_y' = \alpha_y + 1$, and $\alpha_i' = \alpha_i$ if $i \neq x$ and $i \neq y$. We know that $\sum_{i=1}^{m}\alpha_i' = lh$ and $\alpha_i' \leq h$ for each index $i = 1, 2, \ldots, m$. The squashed sum difference of these two lists is given by

$$\text{sq-sum}(\langle\alpha_i'\rangle) - \text{sq-sum}(\langle\alpha_i\rangle)$$
$$= (m-x+1)\alpha_x' + (m-y+1)\alpha_y' - ((m-x+1)\alpha_x + (m-y+1)\alpha_y)$$
$$= (m-x+1)(\alpha_x' - \alpha_x) + (m-y+1)(\alpha_y' - \alpha_y)$$
$$= -(m-x+1) + (m-y+1)$$
$$= x - y \ .$$

Since $x < m-l+1$ and $y \geq m-l+1$, we have $x < y$, and thus we obtain the contradiction $\text{sq-sum}(\langle\alpha_i'\rangle) < \text{sq-sum}(\langle\alpha_i\rangle)$. Since the minimum of the squashed sum occurs when $\alpha_1 = \alpha_2 = \cdots = \alpha_l = 0$ and $\alpha_{m-l+1} = \alpha_{m-l+2} = \cdots = \alpha_m = h$, the minimum value of the squashed sum is $\sum_{i=1}^{m}(m-i+1)\alpha_i = \sum_{i=m-l+1}^{m}(m-i+1)h = hl(l+1)/2$. $\square$

Lemma 10 *Let $\langle\alpha_i\rangle$ be a list of m nonnegative integers, and let $h \geq 0$ be another integer. Generate another list of integers $\langle\beta_i\rangle$ by choosing any l integers from $\langle\alpha_i\rangle$ and increasing each of their values by h. Then, we have*

$$\text{sq-sum}(\langle\beta_i\rangle) \geq \text{sq-sum}(\langle\alpha_i\rangle) + hl(l+1)/2 \ .$$

Proof. Assume that the elements in both $\langle\alpha_i\rangle$ and $\langle\beta_i\rangle$ are sorted such that $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_m$ and $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_m$. Observe that when viewed in sorted order, if an element of $\langle\beta_i\rangle$ was produced by increasing an element of $\langle\alpha_i\rangle$ by h, their indexes may now no longer correspond.

First, we show by contradiction that $\beta_i \geq \alpha_i$. If there exists an index j such that $\beta_j < \alpha_j$, there must exist at least j integers strictly less than α_j in the list $\langle\beta_i\rangle$, namely $\beta_1, \beta_2, \ldots, \beta_j$. Each β_x among these j integers corresponds to

a distinct $\alpha_y \in \langle \alpha_i \rangle$, where $\beta_x = \alpha_y$ or $\beta_x = \alpha_y + h$. Thus, there are at least j integers strictly less than α_j in the list $\langle \alpha_i \rangle$. But, there can be only at most $j-1$ integers less than α_j in the list $\langle \alpha_i \rangle$, namely $\alpha_1, \alpha_2, \ldots, \alpha_{j-1}$. Contradiction.

Second, we show by contradiction that $\beta_i \le \alpha_i + h$. If there exists an index j such that $\beta_j > \alpha_j + h$, there must exist at least $m - j + 1$ integers strictly greater than $\beta_j + h$ in the list $\langle \beta_i \rangle$, namely $\beta_j, \beta_{j+1}, \ldots, \beta_m$. Each β_x of these $m - j + 1$ integers corresponds to a distinct $\alpha_y \in \langle \alpha_i \rangle$, where $\beta_x = \alpha_y$ or $\beta_x = \alpha_y + h$. Thus, there are at least $m - j + 1$ integers strictly greater than α_j in the list $\langle \alpha_i \rangle$. But, there can be at most $m - j$ integers greater than α_j in $\langle \alpha_i \rangle$, namely $\alpha_{j+1}, \alpha_{j+2}, \ldots, \alpha_m$. Contradiction.

Now, define another list $\langle \gamma_i \rangle$ of integers by $\gamma_i = \beta_i - \alpha_i$ for $i = 1, 2, \ldots, m$. From Definition 2 we have

$$
\text{sq-sum}(\langle \beta_i \rangle) - \text{sq-sum}(\langle \alpha_i \rangle) = \sum_{i=1}^{m}(m - i + 1)(\beta_i - \alpha_i)
$$
$$
= \sum_{i=1}^{m}(m - i + 1)\gamma_i
$$
$$
= \text{sq-sum}(\langle \gamma_i \rangle) \ .
$$

Since we obtain $\langle \beta_i \rangle$ from $\langle \alpha_i \rangle$ by choosing l numbers and increasing each of them by h, we have

$$
\sum_{i=1}^{m}\gamma_i = \sum_{i=1}^{m}\beta_i - \sum_{i=1}^{m}\alpha_i
$$
$$
= lh \ .
$$

Because we have $0 \le \beta_i - \alpha_i \le h$, it follows that $0 \le \gamma_i \le h$. From Lemma 9, we know that the squashed sum of the list $\langle \gamma_i \rangle$ is $\text{sq-sum}(\langle \gamma_i \rangle) \ge hl(l+1)/2$, and its minimum occurs when $\gamma_1 = \gamma_2 = \cdots = \gamma_{m-l} = 0$ and for $\gamma_{m-l+1} = \gamma_{m-l+2} = \cdots = \gamma_m = h$. Thus, we have $\text{sq-sum}(\langle \beta_i \rangle) - \text{sq-sum}(\langle \alpha_i \rangle) \ge hl(l+1)/2$, and the minimum occurs when

$$
\beta_i = \begin{cases} \alpha_i & \text{if } i = 1, 2, \ldots, m - l, \\ \alpha_i + h & \text{if } i = m - l + 1, m - l + 2, \ldots, m. \end{cases}
$$

$\square$

The second step of our analysis bounds the total response time $\mathrm{R}(\mathcal{J})$ of AGDEQ in terms of the squashed deprived allotment area $\text{sdaa}(\mathcal{J})$ and total satisfied time $\text{sat}(\mathcal{J})$.

Lemma 11 *Suppose that a job set $\mathcal{J}$ is scheduled by* AGDEQ. *The total response time of $\mathcal{J}$ can be bounded as*

$$
\mathrm{R}(\mathcal{J}) \le \left(2 - \frac{2}{|\mathcal{J}| + 1} \right) \left(\text{sdaa}(\mathcal{J}) + \text{sat}(\mathcal{J}) \right) \ , \tag{6}
$$

where $\text{sdaa}(\mathcal{J})$ is the squashed deprived allotment area of $\mathcal{J}$ and $\text{sat}(\mathcal{J})$ is the total satisfied time of $\mathcal{J}$.

Proof. Suppose that AGDEQ produces a schedule $\chi = (\tau, \pi)$ for $\mathcal{J}$. Let $T = \mathrm{T}_\chi(\mathcal{J})$ be the completion time of the job set $\mathcal{J}$.

First, let us define some notation. For any time step t, represent set of time steps from t to the completion of $\mathcal{J}$ by $\vec{t} = \{t, t+1, \ldots, T\}$. We shall be interested in "suffixes" of jobs, namely, the portions of jobs that remain after some number of steps have been executed. To that end, define the ***t-suffix*** of a job $J_i \in \mathcal{J}$ to be the job $J_i\left(\vec{t}\right)$ induced by those vertices in $V(J_i)$ that execute on or after time t, that is,

$$J_i\left(\vec{t}\right) = \left(V\left(J_i\left(\vec{t}\right)\right), E\left(J_i\left(\vec{t}\right)\right)\right) \ ,$$

where $v \in V\left(J_i\left(\vec{t}\right)\right)$ if $v \in V(J_i)$ and $\tau(v) \geq t$, and $(u,v) \in E\left(J_i\left(\vec{t}\right)\right)$ if $(u,v) \in E(J_i)$ and $u,v \in V\left(J_i\left(\vec{t}\right)\right)$. The t-suffix of the job set $\mathcal{J}$ is

$$\mathcal{J}\left(\vec{t}\right) = \left\{J_i\left(\vec{t}\right) : J_i \in \mathcal{J} \text{ and } V\left(J_i\left(\vec{t}\right)\right) \neq \emptyset\right\} \ .$$

Thus, we have $\mathcal{J} = \mathcal{J}\left(\vec{1}\right)$, and the number of incomplete jobs at time step t is the number $\left|\mathcal{J}\left(\vec{t}\right)\right|$ of nonempty jobs in $\mathcal{J}\left(\vec{t}\right)$. Since we only consider batched jobs, the number of incomplete jobs is decreasing monotonically, and hence, we have

$$\left|\mathcal{J}\left(\overrightarrow{t+1}\right)\right| \leq \left|\mathcal{J}\left(\vec{t}\right)\right| \ . \tag{7}$$

The total response times of $\mathcal{J}\left(\vec{t}\right)$ and $\mathcal{J}\left(\overrightarrow{t+1}\right)$ can also be related using this notation. Since each incomplete job of $\mathcal{J}\left(\vec{t}\right)$ adds one time step into its total response time during step t, we have

$$\mathrm{R}\left(\mathcal{J}\left(\vec{t}\right)\right) = \mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) + \left|\mathcal{J}\left(\vec{t}\right)\right| \ . \tag{8}$$

We shall prove the lemma by induction on the remaining execution time of the job set $\mathcal{J}\left(\vec{t}\right)$.

Basis: $t = T{+}1$. Since we have $\mathcal{J}\left(\overrightarrow{T+1}\right) = \emptyset$, it follows that $\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{T+1}\right)\right) = 0$, $\mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{T+1}\right)\right) = 0$, and $\mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{T+1}\right)\right) = 0$. Thus, the claim holds trivially.

Induction: $1 \leq t \leq T$. Suppose that the lemma holds for $\mathcal{J}\left(\overrightarrow{t+1}\right)$. We shall prove that it holds for $\mathcal{J}\left(\vec{t}\right)$.

We first define some notation. At any time step t, the incomplete jobs can be partitioned as $\mathcal{J}\left(\vec{t}\right) = \mathcal{JS}(t) \cup \mathcal{JD}(t)$, representing the set of satisfied and deprived jobs at time t, respectively. For any job $J_i \in \mathcal{J}$ and time t, define

$$S\left(i, t\right) = \begin{cases} \{t\} & \text{if } J_i \in \mathcal{JS}(t) \ , \\ \emptyset & \text{if } J_i \notin \mathcal{JS}(t) \ ; \end{cases}$$

and similarly, define

$$D\left(i, t\right) = \begin{cases} \{t\} & \text{if } J_i \in \mathcal{JD}(t) \ , \\ \emptyset & \text{if } J_i \notin \mathcal{JD}(t) \ . \end{cases}$$

We can extend these definitions to suffix ranges:

$$S\left(i, \overrightarrow{t}\right) = \bigcup_{t'=t}^{T} S\left(i, t'\right) \ ,$$

$$D\left(i, \overrightarrow{t}\right) = \bigcup_{t'=t}^{T} D\left(i, t'\right) \ .$$

We now relate the total satisfied times of $\mathcal{J}\left(\overrightarrow{t}\right)$ and $\mathcal{J}\left(\overrightarrow{t+1}\right)$. By definition of total satisfied time and using the fact that $\sum_{J_i \in \mathcal{J}} |S\left(i, t\right)| = |\mathcal{JS}(t)|$, we have

$$\begin{aligned}
\mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) &= \sum_{J_i \in \mathcal{J}} \left|S\left(i, \overrightarrow{t}\right)\right| \\
&= \sum_{J_i \in \mathcal{J}} \left|S\left(i, t\right)\right| + \sum_{J_i \in \mathcal{J}} \left|S\left(i, \overrightarrow{t+1}\right)\right| \\
&= |\mathcal{JS}(t)| + \mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) \ .
\end{aligned} \tag{9}$$

We next relate the accumulated deprived allotments $a\left(i, D\left(i, \overrightarrow{t}\right)\right)$ and $a\left(i, D\left(i, \overrightarrow{t+1}\right)\right)$. Job J_i's accumulated deprived allotment on $\overrightarrow{t}$ is given by

$$a\left(i, D\left(i, \overrightarrow{t}\right)\right) = \sum_{t' \in D\left(i, \overrightarrow{t}\right)} a\left(i, t'\right) \ .$$

We consider two cases depending on whether $J_i \in \mathcal{JS}(t)$ or $J_i \in \mathcal{JD}(t)$. If $J_i \in \mathcal{JS}(t)$, we have $D\left(i, t\right) = \emptyset$ and $D\left(i, \overrightarrow{t}\right) = D\left(i, \overrightarrow{t+1}\right)$, and thus, J_i's accumulated deprived allotment is

$$\begin{aligned}
a\left(i, D\left(i, \overrightarrow{t}\right)\right) &= \sum_{t' \in D\left(i, \overrightarrow{t}\right)} a\left(i, t'\right) \\
&= \sum_{t' \in D\left(i, \overrightarrow{t+1}\right)} a\left(i, t'\right) \\
&= a\left(i, D\left(i, \overrightarrow{t+1}\right)\right) \ .
\end{aligned} \tag{10}$$

If $J_i \in \mathcal{JD}(t)$, we have $D\left(i, t\right) = \{t\}$ and $D\left(i, \overrightarrow{t}\right) = D\left(i, \overrightarrow{t+1}\right) \cup \{t\}$. Moreover, J_i has allotment $a\left(i, t\right) = p\left(t\right)$, where $p\left(t\right)$ denotes the mean deprived allotment at time step t. Thus, J_i's accumulated deprived allotment is

$$\begin{aligned}
a\left(i, D\left(i, \overrightarrow{t}\right)\right) &= \sum_{t' \in D\left(i, \overrightarrow{t}\right)} a\left(i, t'\right) \\
&= \sum_{t' \in D\left(i, \overrightarrow{t+1}\right)} a\left(i, t'\right) + a\left(i, t\right) \\
&= a\left(i, D\left(i, \overrightarrow{t+1}\right)\right) + a\left(i, t\right) \\
&= a\left(i, D\left(i, \overrightarrow{t+1}\right)\right) + p\left(t\right) \ .
\end{aligned} \tag{11}$$

Thus, going backwards from step $t+1$ to step t, the accumulated deprived allotment either stays the same or increases by $p(t)$, depending on whether step t is satisfied or deprived, respectively.

We now use Lemma 10 to relate the squashed deprived allotment areas of $\mathcal{J}\left(\overrightarrow{t}\right)$ and $\mathcal{J}\left(\overrightarrow{t+1}\right)$. Let $n = \left|\mathcal{J}\left(\overrightarrow{t}\right)\right|$ denote the number of incomplete jobs before step t. For $i = 1, 2, \ldots, n$, let $\alpha_i = a\left(i, D\left(i, \overrightarrow{t+1}\right)\right)$, and let $\beta_i = a\left(i, D\left(i, \overrightarrow{t}\right)\right)$. If $J_i \in \mathcal{JS}(t)$, Equation (10) implies that $\beta_i = \alpha_i$. If $J_i \in \mathcal{JD}(t)$, Equation (11) implies that $\beta_i = \alpha_i + p(t)$. Thus, the list $\langle\beta_i\rangle$ can be generated by choosing $l = |\mathcal{JD}(t)|$ integers from $\langle\alpha_i\rangle$ and increasing each of them by $h = p(t)$. Applying Lemma 10 and the definition of squashed deprived allotment area, we obtain

$$
\begin{aligned}
&\mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) \\
&= \frac{1}{P}\,\mathrm{sq\text{-}sum}\left(\left\langle a\left(i, D\left(i, \overrightarrow{t}\right)\right)\right\rangle\right) \\
&\geq \frac{1}{P}\left(\mathrm{sq\text{-}sum}\left(\left\langle a\left(i, D\left(i, \overrightarrow{t+1}\right)\right)\right\rangle\right) + p(t)\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)| + 1)\,/2\right) \\
&= \mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) + p(t)\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)| + 1)\,/2P\ . \qquad (12)
\end{aligned}
$$

We now can complete the proof of the lemma by using Inequality (7), Equations (8) and (9), and Inequality (12) to bound the total response time of $\mathcal{J}\left(\overrightarrow{t}\right)$:

$$
\begin{aligned}
&\mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) \\
&= \mathrm{R}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) + \left|\mathcal{J}\left(\overrightarrow{t}\right)\right| \\
&\leq \left(2 - \frac{2}{\left|\mathcal{J}\left(\overrightarrow{t+1}\right)\right| + 1}\right)\left(\mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) + \mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right)\right) + \left|\mathcal{J}\left(\overrightarrow{t}\right)\right| \\
&\leq \left(2 - \frac{2}{\left|\mathcal{J}\left(\overrightarrow{t}\right)\right| + 1}\right)\left(\mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) + \mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right)\right) + \left|\mathcal{J}\left(\overrightarrow{t}\right)\right| \\
&\leq \left(2 - \frac{2}{n+1}\right)\left(\mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - \frac{p(t)\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)| + 1)}{2P}\right) \\
&\quad + \left(2 - \frac{2}{n+1}\right)\left(\mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - |\mathcal{JS}(t)|\right) + n \\
&\leq \left(2 - \frac{2}{n+1}\right)\left(\mathrm{sdaa}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) + \mathrm{sat}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right)\right) \\
&\quad - \left(2 - \frac{2}{n+1}\right)\left(\frac{p(t)\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)| + 1)}{2P} + |\mathcal{JS}(t)|\right) + n
\end{aligned}
$$

We must show that

$$
\left(2 - \frac{2}{n+1}\right)\left(\frac{p(t)\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)| + 1)}{2P} + |\mathcal{JS}(t)|\right) - n \geq 0
$$

Using the facts that $p(t) \geq P/n$, $|\mathcal{JD}(t)| = n - |\mathcal{JS}(t)|$, $|\mathcal{JS}(t)|$ is an integer, and $0 \leq |\mathcal{JS}(t)| \leq n$, we obtain

$$
\left(2 - \frac{2}{n+1}\right)\left(\frac{p(t)\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)|+1)}{2P} + |\mathcal{JS}(t)|\right) - n
$$
$$
\geq \frac{n}{n+1}\left(\frac{p(t)}{P}\,|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)|+1) + 2\,|\mathcal{JS}(t)| - (n+1)\right)
$$
$$
\geq \frac{n}{n+1}\left(\frac{|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)|+1)}{n} + 2\,|\mathcal{JS}(t)| - n - 1\right)
$$
$$
= \frac{1}{n+1}\left(|\mathcal{JD}(t)|\,(|\mathcal{JD}(t)|+1) + 2n\,|\mathcal{JS}(t)| - n^2 - n\right)
$$
$$
= \frac{1}{n+1}\left((n - |\mathcal{JS}(t)|)\,(n - |\mathcal{JS}(t)|+1) + 2n\,|\mathcal{JS}(t)| - n^2 - n\right)
$$
$$
= \frac{1}{n+1}\,|\mathcal{JS}(t)|\,(|\mathcal{JS}(t)| - 1)
$$
$$
\geq 0 \ .
$$

$\square$

The third step of our analysis bounds the squashed deprived allotment area in terms of the squashed work area.

Lemma 12 *Suppose that a job set $\mathcal{J}$ is scheduled by* AGDEQ*, where ρ and δ are* A-Greedy*'s responsiveness and utilization parameters, respectively. The squashed deprived allotment area of $\mathcal{J}$ can be bounded as*

$$
\mathrm{sdaa}\,(\mathcal{J}) \leq \frac{\rho+1}{\delta}\,\mathrm{swa}\,(\mathcal{J}) \ ,
$$

where $\mathrm{swa}\,(\mathcal{J})$ is the squashed work area of the job set $\mathcal{J}$.

Proof. We first show that $a\,(i, D(i)) \leq cT_1(i)$ for every job $J_i \in \mathcal{J}$, where $a(i)$ and $a\,(i, D(i))$ are J_i's accumulated allotment and accumulated deprived allotment, respectively, and $c = (\rho+1)/\delta$. By Definition 3, we have $a\,(i, D(i)) = \sum_{t \in D(i)} a\,(i, t) \leq \sum_{t=0}^{\infty} a\,(i, t) = a(i)$, since $D(i) \subseteq \{1, 2, \ldots, \infty\}$ and $a\,(i, t) \geq 0$. The processor allotments to any job are either used to make progress on the total work $T_1(i)$ or wasted. According to Lemma 5, any job J_i wastes at most $w(i) = ((\rho+1-\delta)/\delta)T_1(i)$ processor cycles. For each job J_i, we have

$$
\begin{aligned}
a\,(i, D(i)) &\leq a(i) \\
&= T_1(i) + w(i) \\
&\leq ((\rho+1-\delta)/\delta)T_1(i) + T_1(i) \\
&= cT_1(i) \ .
\end{aligned}
$$

To complete the proof, we use Definition 2 and apply Lemma 8:

$$
\begin{aligned}
\mathrm{sdaa}\,(\mathcal{J}) &= (1/P)\,\mathrm{sq\text{-}sum}(\langle a\,(i, D(i))\rangle) \\
&\leq (1/P)\,\mathrm{sq\text{-}sum}(\langle cT_1(i)\rangle) \\
&= c \cdot (1/P)\,\mathrm{sq\text{-}sum}(\langle T_1(i)\rangle) \\
&= c \cdot \mathrm{swa}\,(\mathcal{J}) \ .
\end{aligned}
$$

□

The fourth step of our analysis relates the total satisfied time to the aggregate critical-path length.

Lemma 13 *Suppose that a job set $\mathcal{J}$ is scheduled by* AGDEQ, *where ρ and δ are* A-GREEDY*'s responsiveness and utilization parameters, respectively. The total satisfied time of $\mathcal{J}$ can be bounded as*

$$\mathrm{sat}\,(\mathcal{J}) \leq \frac{2}{1-\delta}T_\infty(\mathcal{J}) + |\mathcal{J}|\,(L\log_\rho P + L) \ ,$$

where $T_\infty(\mathcal{J})$ is the aggregate critical-path length of $\mathcal{J}$.

Proof.   We bound the total satisfied time using Lemma 4:

$$\mathrm{sat}\,(\mathcal{J}) = \sum_{J_i\in\mathcal{J}} |S(i)|$$

$$\leq \sum_{J_i\in\mathcal{J}} \left(\frac{2T_\infty(i)}{1-\delta} + L\log_\rho P + L\right)$$

$$= \frac{2}{1-\delta}T_\infty(\mathcal{J}) + |\mathcal{J}|\,(L\log_\rho P + L) \ .$$

□

We can now apply the results of our four-step analysis to obtain a bound on total response time.

Theorem 14 *Suppose that a job set $\mathcal{J}$ is scheduled by* AGDEQ. *Let ρ be* A-GREEDY*'s responsiveness parameter, δ its utilization parameter, and L the quantum length. The total response time $\mathrm{R}(\mathcal{J})$ of the schedule is at most*

$$\mathrm{R}(\mathcal{J}) \leq \left(2 - \frac{2}{|\mathcal{J}|+1}\right)\left(\frac{\rho+1}{\delta}\,\mathrm{swa}\,(\mathcal{J}) + \frac{2}{1-\delta}T_\infty(\mathcal{J}) + |\mathcal{J}|\,L(\log_\rho P + 1)\right) \ ,$$

where $\mathrm{swa}\,(\mathcal{J})$ is the squashed work area of $\mathcal{J}$, and $T_\infty(\mathcal{J})$ is the aggregate critical-path length of $\mathcal{J}$.

Proof.   Combine Lemmas 11, 12, and 13.   □

Since both $\mathrm{swa}\,(\mathcal{J})\,/\,|\mathcal{J}|$ and $T_\infty(\mathcal{J})/\,|\mathcal{J}|$ are lower bounds on $\overline{\mathrm{R}}(\mathcal{J})$, we obtain the following corollary.

Corollary 15 *Suppose that a job set $\mathcal{J}$ is scheduled by* AGDEQ. *Let ρ be* A-GREEDY*'s responsiveness parameter, δ its utilization parameter, and L the quantum length. The mean response time $\overline{\mathrm{R}}(\mathcal{J})$ of the schedule satisfies*

$$\overline{\mathrm{R}}(\mathcal{J}) \leq \left(2 - \frac{2}{|\mathcal{J}|+1}\right)\left(\left(\frac{\rho+1}{\delta} + \frac{2}{1-\delta}\right)\overline{\mathrm{R}^*}(\mathcal{J}) + L\log_\rho P + L\right) \ ,$$

where $\overline{\mathrm{R}^*}(\mathcal{J})$ denotes the mean response time of $\mathcal{J}$ scheduled by an optimal clairvoyant scheduler.

Proof. Combine Theorem 14 with Inequalities (3) and (4). □

Since both the quantum length L and the processor number P are independent variables with respect to any job set $\mathcal{J}$, Corollary 15 shows that AGDEQ is $O(1)$-competitive with respect to mean response time for batched jobs. Specifically, when $\delta = 1/2$ and ρ approaches 1, AGDEQ's competitiveness ratio approaches the minimum value 16. Thus, AGDEQ is $(16 + \epsilon)$-competitive with respect to mean response time for any constant $\epsilon > 0$.

The competitive ratio of 16 for AGDEQ is a worst-case bound. We expect that in practice, however, AGDEQ should perform closer to optimal. In particular, when the job set $\mathcal{J}$ exhibits reasonably large total parallelism, we have swa$(\mathcal{J}) \gg T_\infty(\mathcal{J})$, and thus, the term involving swa$(\mathcal{J})$ in Theorem 14 dominates the total response time. More importantly, the job scheduler DEQ is not actually an adversary of A-GREEDY, and simulations of A-STEAL [2] suggest that in practice A-GREEDY should produce waste closer to $(1/\delta - 1)T_1(i)$. From the proof of Lemma 12, one can determine that the coefficient on the term swa$(\mathcal{J})$ becomes $(2 - 2/(|\mathcal{J}| + 1))/\delta$ when a job's waste is no more than $(1/\delta - 1)$ times its work. That is to say, in this scenario, the mean response time of a job set scheduled by AGDEQ is about $(2/\delta)$ swa$(\mathcal{J})$. Since δ is typically in the range of 0.5 to 1, if the job set has reasonably large total parallelism, AGDEQ is likely to achieve the mean response time of less than 4 times the optimal.

6 ASDEQ Algorithm and Performance

ASDEQ is a distributed two-level adaptive scheduler that uses the A-STEAL algorithm [2,3] as its thread scheduler and DEQ as its job scheduler. A-STEAL is a decentralized thread scheduler that employs randomized work stealing [4,13, 16,31,47] to schedule and execute a job without central knowledge of all available threads. The interactions between A-STEAL and DEQ follow the scheduling model described in Section 2. In this section, we briefly overview the A-STEAL algorithm. We show that ASDEQ is $O(1)$-competitive with respect to makespan for jobs with arbitrary release time and $O(1)$-competitive with respect to mean response time for batched jobs.

The Adaptive Stealing Thread Scheduler

The A-STEAL algorithm is a decentralized adaptive thread scheduler with parallelism feedback, and like A-GREEDY, A-STEAL performs two functions. Between quanta, it estimates its job's desire and requests processors from the job scheduler. A-STEAL applies the same desire-estimation algorithm as A-GREEDY to calculate its job's desire. During the quantum, A-STEAL schedules the ready threads of the job onto the allotted processors using an adaptive work-stealing algorithm.

Each processor allotted to a job whose threads are scheduled by A-STEAL maintains a **deque** (double-ended queue) of those threads that are ready to execute. To handle an increase in allotment, A-STEAL creates an empty deque for each newly allotted processor. When the allotment decreases, A-STEAL marks the deques from deallotted processors as **muggable deques**. An allotted processor works on only one ready thread at a time. When the current thread spawns

a new thread, the processor pushes the current thread onto the top of the deque and begins working on the new thread. When the current thread completes or blocks, the processor pops the topmost thread off the deque and begins working on it. If the deque of a processor becomes empty, however, the processor becomes a ***thief***. The thief first looks for a muggable deque. If one is found, the thief ***mugs*** the deque by taking over the entire deque as its own. Otherwise, it randomly picks a ***victim*** processor and ***steals*** work from the bottom of the victim's deque. If the victim has no available work, then the steal is ***unsuccessful***, and the thief continues to steal at random from the other processors until it is ***successful*** and finds work. At all time steps, every processor is either working, stealing, or mugging.

Analysis

We now show that ASDEQ is $O(1)$-competitive with respect to both makespan and mean response time. The methods used to analyze ASDEQ are similar to those for AGDEQ. Since ASDEQ is a randomized scheduling algorithm, however, we show that its makespan (or its expected mean response time) is within a factor c of that incurred in an optimal clairvoyant algorithm in expectation, not in the worst case. Let $\chi = (\tau, \pi)$ be the schedule of a job set $\mathcal{J}$ produced by ASDEQ. For simplicity we shall use the notations $\mathrm{T}(\mathcal{J}) = \mathrm{T}_\chi(\mathcal{J})$ and $\mathrm{R}(\mathcal{J}) = \mathrm{R}_\chi(\mathcal{J})$.

The next two lemmas, proved in [3], bound the expected satisfied steps and the waste of any single job scheduled by A-STEAL. They provide a starting point for the analysis.

Lemma 16 *[3] Suppose that* A-STEAL *schedules a job J_i with critical path length $T_\infty(i)$ on a machine with P processors. Let ρ denote* A-STEAL*'s responsiveness parameter, δ its utilization parameter, and L the quantum length. Then,* A-STEAL *produces at most $48T_\infty(i)/(1-\delta) + L \log_\rho P + L$ satisfied steps in expectation.* □

Lemma 17 *[3] Suppose that* A-STEAL *schedules a job J_i with work $T_1(i)$ on a machine with P processors. Let ρ denote* A-STEAL*'s responsiveness parameter, δ is its utilization parameter, and L is the quantum length. Then,* A-STEAL *wastes at most*

$$W \leq \left(\frac{1 + \rho - \delta}{\delta} + \frac{(1+\rho)^2}{\delta(L\delta - 1 - \rho)} \right) T_1(i) \tag{13}$$

processor cycles in the course of the computation. □

The next theorem shows that ASDEQ is $O(1)$-competitive with respect to makespan for a job set $\mathcal{J}$ with arbitrary release time. The following bound is based on the release time $r(i)$, critical-path length $T_\infty(i)$, and work $T_1(i)$ of an individual job $J_i \in \mathcal{J}$, as well as on the total work $T_1(\mathcal{J})$ of the job set $\mathcal{J}$.

Theorem 18 *Suppose that* ASDEQ *schedules a job set $\mathcal{J}$ on a machine with P processors. Let ρ denote* A-STEAL*'s responsiveness parameter, δ its utilization*

parameter, and L the quantum size. Then, we expect ASDEQ *to complete $\mathcal{J}$ in*

$$\mathrm{E}\left[\mathrm{T}(\mathcal{J})\right] \leq \left(\frac{\rho+1}{\delta} + \frac{(1+\rho)^2}{\delta(L\delta-1-\rho)}\right)\frac{T_1(\mathcal{J})}{P}$$

$$+ O\left(\frac{\max_{J_i \in \mathcal{J}}\{r(i) + T_\infty(i)\}}{1-\delta}\right) + L\log_\rho P + 2L \qquad (14)$$

time steps.

Proof. The proof is similar to that of Theorem 6. Let job J_k be the last job to complete among the jobs in $\mathcal{J}$. Let $S(k)$ denote the set of satisfied steps for J_k, and let $D(k)$ denote the set of deprived steps for J_k. The earliest that the job J_k can start its execution is at the beginning of the quantum immediately after J_k's release, which is the quantum q satisfying $Lq < r(k) \leq L(q+1)$. Therefore, we have

$$\mathrm{T}(\mathcal{J}) < r(k) + L + |S(k)| + |D(k)| \ . \qquad (15)$$

Since Lemma 16 bounds the number of J_k's satisfied steps, we focus on bounding the quantity the number $|D(k)|$ of J_k's deprived steps. DEQ must allot all processors to jobs on any deprived step, and hence we have $a\left(\mathcal{J}, D(k)\right) = \sum_{t \in D(k)} \sum_{J_i \in \mathcal{J}} a\left(i, t\right) = P\left|D(k)\right|$. The allotted processor cycles are either working or wasted. Define the constant c to be

$$c = \frac{\rho+1}{\delta} + \frac{(1+\rho)^2}{\delta(L\delta-1-\rho)} \ .$$

Lemma 17 shows that the waste $w(i)$ for any job J_i is at most $(c-1)T_1(i)$. Since the total allotment $a\left(\mathcal{J}, D(k)\right)$ is at most the sum of the total work and total waste, we have $P\left|D(k)\right| = a\left(\mathcal{J}, D(k)\right) \leq \sum_{J_i \in \mathcal{J}}(T_1(i)+w(i)) \leq \sum_{J_i \in \mathcal{J}} cT_1(i) = cT_1(\mathcal{J})$, which gives us $|D(k)| \leq cT_1(\mathcal{J})/P$.

Combining this bound, the bound $\mathrm{E}\left[|S(k)|\right] \leq 48T_\infty(k)/(1-\delta)+L\log_\rho P+L$ from Lemma 16, and the bound $\mathrm{E}\left[\mathrm{T}(\mathcal{J})\right] < r(k) + L + \mathrm{E}\left[|S(k)|\right] + |D(k)|$ from Inequality (15) completes the proof. □

The next theorem shows that ASDEQ is $O(1)$-competitive with respect to mean response time for batched jobs.

Theorem 19 *Suppose that a job set $\mathcal{J}$ is scheduled by* ASDEQ. *Let ρ denote* A-STEAL*'s responsiveness parameter, δ its utilization parameter, and L the quantum length. Then, the expected response time of the schedule satisfies*

$$\mathrm{E}\left[\mathrm{R}(\mathcal{J})\right] \leq \left(2 - \frac{2}{|\mathcal{J}|+1}\right)\left(\frac{\rho+1}{\delta} + \frac{(1+\rho)^2}{\delta(L\delta-1-\rho)}\right)\mathrm{swa}\left(\mathcal{J}\right)$$

$$+ O\left(\frac{T_\infty(\mathcal{J})}{1-\delta}\right) + 2\left|\mathcal{J}\right|L(\log_\rho P + 1) \ ,$$

where $\mathrm{swa}\left(\mathcal{J}\right)$ is the squashed work area, and $T_\infty(\mathcal{J})$ is the aggregate critical-path length.

Proof. The proof of the theorem follows closely on that of Theorem 14. It turns out that Lemma 11 holds for any two-level scheduler that uses DEQ, irrespective of the thread scheduler. Lemma 12 holds with the new constant

$$c = \frac{\rho + 1}{\delta} + \frac{(1 + \rho)^2}{\delta(L\delta - 1 - \rho)} \ .$$

Lemma 13 can be adapted by using Lemma 16 in place of Lemma 4 to produce the bound

$$\mathrm{E}\left[\mathrm{sat}\left(\mathcal{J}\right)\right] = O\left(\frac{T_\infty(\mathcal{J})}{1 - \delta}\right) + L\log_\rho P + L \ .$$

Combining these bounds yields the theorem. □

Theorems 18 and 19 show that ASDEQ is $O(1)$-competitive for both makespan and, in the batch setting, mean response time. We anticipate that ASDEQ's competitive ratios would be small in practical settings, especially when many jobs have total work much larger than critical-path length and the machine is moderately or highly loaded. In this case, the term on $T_1(\mathcal{J})/P$ in Inequality (14) is much larger than the term $\max_{J_i \in \mathcal{J}} \{T_\infty(i) + r(i)\}$, which is to say, the term on $T_1(\mathcal{J})/P$ generally dominates the makespan bound. The proof of Theorem 18 calculates the coefficient of $T_1(\mathcal{J})/P$ in Inequality (14) as the ratio of the total allotment (total work plus total waste) versus the total work. When the job scheduler is DEQ, which is not a true adversary, empirical results [2] indicate that each job J_i only wastes about $(1/\delta - 1)T_1(i)$ processor cycles, which is not as large as the worst-case waste in Lemma 17. Therefore, when we use DEQ as the job scheduler, the coefficient of $T_1(\mathcal{J})/P$ seems more likely to approach $1/\delta$. In other words, the makespan of a job set $\mathcal{J}$ scheduled by ASDEQ might more typically be about $T_1(\mathcal{J})/\delta P$. Since δ is typically in the range of 0.5 to 1, ASDEQ may exhibit makespans that are only about 2 times optimal when the jobs have reasonably large parallelism and the machine is moderately or heavily loaded. Similarly, ASDEQ may exhibit only 4 times optimal with respect to mean response time for batched jobs under the same conditions.

7 Competitiveness of Mean Response Time for Nonbatched Jobs

This section studies the competitiveness of deterministic algorithms for minimizing mean response time for nonbatched job sets where jobs can be released at arbitrary times. Let $n = |\mathcal{J}|$ be the number of jobs in a job set $\mathcal{J}$, and let P be the number of processors on which the jobs are scheduled. For jobs with arbitrary release times, Motwani, Phillips, and Torng [42] study the scheduling of serial jobs on single processor, and show that every deterministic algorithm has competitiveness $\Omega(n^{1/3})$, and any randomized algorithm has competitiveness $\Omega(\log n)$ by implicitly assuming that $n > P$. We extend their result for deterministic scheduling of nonbatched jobs by showing that any deterministic algorithm is $\Omega(n^{1/3})$-competitive with respect to mean response time no matter what the relation between n and P. Thus, our results for batched job sets in

Section 5 cannot be extended to yield strong results for nonbatched job sets, except possibly if randomization is employed.

The following theorem provides the lower bound.

Theorem 20 *Suppose that a nonbatched job set $\mathcal{J}$ is scheduled on P processors. Any deterministic nonclairvoyant algorithm has competitive ratio $\Omega\left(n^{1/3}\right)$ with respect to the mean response time.*

Proof. We exhibit a job set $\mathcal{J}$ on which any deterministic clairvoyant Algorithm A must perform poorly with respect to the optimal offline clairvoyant algorithm. We construct $\mathcal{J}$ with $n = m^3 - m^2 + m$ jobs in two phases as follows. In the first phase, we allow Algorithm A to execute on m jobs released at time 0 for $m(m-1)$ time steps during which no job completes no matter how Algorithm A allocates the P processors. We give each of the m jobs the work it has executed thus far plus P additional work. In the second phase, we release the remaining jobs at times $m(m-1), m(m-1)+1, m(m-1)+2, \ldots, m(m-1)+m^3-m^2-1$, each with work P. Every job $J_i \in \mathcal{J}$ has a critical-path length of $T_\infty(i) = 1$.

We now analyze the total response time for Algorithm A. For the m jobs released in the first phase, none completes within the $m(m-1)$ time steps. Immediately after time $m(m-1)$, we have $m+1$ jobs, each with P work remaining. To minimize total response time, the best that Algorithm A can do on time step $m(m-1)+1$ is to use all P processors to complete one job. At that point, however, another job is released, and we once again have $m+1$ jobs, each with P work remaining. This process continues until all $m^3 - m^2 + m$ jobs complete. Let χ denote the schedule of the job set $\mathcal{J}$ produced by the algorithm A. By Definition 1 the total response time for Algorithm A is

$$
\begin{aligned}
\mathrm{R}_\chi(\mathcal{J}) &= \sum_{J_i \in \mathcal{J}} (T_\chi(i) - r(i)) \\
&= \sum_{J_i \in \mathcal{J}} T_\chi(i) - \sum_{J_i \in \mathcal{J}} r(i)) \\
&= \sum_{k=m(m-1)+1}^{m^3-m^2+m} k - \sum_{k=m(m-1)}^{m^3-m^2-1} k \\
&= -m(m-1) + \sum_{k=m^3-m^2}^{m^3-m^2+m} k \\
&= -m(m-1) + \frac{1}{2}(2m^3 - 2m^2 + m)(m+1) \\
&= \Omega(m^4) \ .
\end{aligned}
$$

The optimal algorithm works differently, because it knows the future. During the first $m(m-1)$ time steps, the optimal algorithm ignores the largest of the m jobs released at time 0 and works on the other $m-1$ jobs. The total work that can be accomplished in the first $m(m-1)$ time steps is $Pm(m-1)$. Since the total work of the jobs released at time 0 is $Pm(m-1) + Pm = Pm^2$, the

largest job must have at least Pm work, and thus the remaining $m - 1$ jobs have at most $Pm^2 - Pm = Pm(m - 1)$ work among them. Thus, by ignoring the largest jobs during the first phase, the optimal algorithm can complete all but the largest job. Immediately after time $m(m - 1)$, we have 2 jobs, one with Pm work remaining, and one with P work remaining. The optimal algorithm completes the smaller job in 1 time step, at which point a new job with P work is released. The process repeats, and the optimal algorithm always schedules the newly released job on all processors, which completes in just 1 time step. Finally, at time $m(m - 1) + m^3 - m^2 = m^3 - m$, only the large job remains, which completes at time $m^3 - m + (Pm)/P = m^3$, because the optimal algorithm schedules its Pm work on all P processors.

The optimal algorithm's response time for each of the $m - 1$ smaller jobs released at time 0 is at most $m(m - 1)$, for each of the $m^3 - m^2$ jobs released in the second phase is 1, and for the largest job is m^3. Thus, the total response time is

$$
\begin{aligned}
\mathrm{R}^*(\mathcal{J}) &\leq (m - 1) \cdot m(m - 1) + (m^3 - m^2) \cdot 1 + 1 \cdot m^3 \\
&= O(m^3) \ .
\end{aligned}
$$

Hence, the competitive ratio is $\mathrm{R}(\mathcal{J})/\mathrm{R}^*(\mathcal{J}) = \Omega(m^4)/O(m^3) = \Omega(m) = \Omega(n^{1/3})$. $\qquad\square$

8   Related Work

This section discusses related work on the problem of scheduling to minimize makespan and mean response time. In the offline version of the problem, all the jobs' resource requirments and release times are known in advance. In the online clairvoyant version of the problem, the algorithm knows the resource requirements of a job when it is released, but it must base its decisions only on jobs that have been released. In this paper, we have studied the online nonclairvoyant version of the problem, where the resource requirements and release times are unknown to the scheduling algorithm.

Extensive research [18,33,34,38,43,48,50,56,57] has been conducted on both the offline and online clairvoyant versions of the problem. Since both adaptive and nonadaptive task scheduling is strongly NP-hard even for a fixed number (≥ 5) of processors [23], existing work has tended to focus either on finding polynomial-time approximation algorithms or on the optimality of special cases.

The online nonclairvoyant version of the problem includes the scheduling of a single parallel job, multiple serial jobs, and multiple parallel jobs.

Prior work on scheduling a single parallel job tends to focus on nonadaptive scheduling [9, 10, 13, 15, 28, 44] or adaptive scheduling without parallelism feedback [4]. For jobs whose parallelism is unknown in advance and which may change during execution, nonadaptive scheduling is known to waste processor cycles [53], because a job with low parallelism may be allotted more processors than it can productively use. Moreover, in a multiprogrammed environment, non-adaptive scheduling may not allow a new job to start, because existing jobs may

already be using most of the processors. Although adaptive scheduling without parallelism feedback allows jobs to enter the system, jobs may still waste processor cycles if they are allotted more processors than they can use.

Adaptive thread scheduling with parallelism feedback has been studied empirically [49, 52, 54] and theoretically [1–3]. Using an adaptive thread scheduler with parallelism feedback, if a job cannot effectively use the allotted processors, the job scheduler can repurpose those processors to the other jobs that can use them. A-GREEDY and A-STEAL have been shown [1, 2] to achieve nearly linear speedup and waste a relatively small number of processor cycles for individual jobs. These algorithms model the job scheduler as the thread scheduler's adversary. An analytical technique called "trim analysis" shows that the thread scheduler can perform poorly on at most a small number of time steps while exhibiting near-optimal behavior on the vast majority. A-GREEDY and A-STEAL focus on scheduling individual jobs well with respect to both time and waste, but they do not offer any guarantee for the execution time of the overall job set.

Some researchers [5,7,17,30,35] have studied the online nonclairvoyant scheduling of serial jobs to minimize the mean response time on single or multiple processors. For jobs with arbitrary release times, Motwani, Phillips, and Torng [42] show that every deterministic algorithm has competitiveness $\Omega(n^{1/3})$ with respect to mean response time, implicitly assuming that $n > P$. Moreover, any randomized algorithm has competitiveness $\Omega(\log n)$, also assuming that $n > P$. They also show that round-robin is $(2 - 2P/(n + P))$-competitive. Becchetti and Leonardi [7] present a version of the randomized multilevel feedback algorithm (RMLF) and prove an $O(\log n \log(n/P))$-competitiveness result against any oblivious adversary on a machine with P processors. This RMLF algorithm achieves a tight $O(\log n)$ competitive ratio against an oblivious adversary on a machine with a single processor, thereby matching the lower bound for this case.

Shmoys, Wein and Williamson in [51] study the lower bounds of online nonclairvoyant scheduling of serial jobs with respect to makespan. They show that the competitive ratio is at least $(2-1/P)$ for any preemptive deterministic online algorithm, and at least $(2 - 1/\sqrt{P})$ for any nonpreemptive randomized online algorithm with an oblivious adversary.

Adaptive parallel job scheduling has been studied empirically [36,39,41,55,59] and theoretically [6,20,24,25,29,42]. McCann, Vaswani, and Zahorjan [41] study many different job schedulers and evaluated them on a set of benchmarks. They also introduce the notion of dynamic equipartitioning, which gives each job a fair allotment of processors based on the job's request, while allowing processors that cannot be used by a job to be reallocated to other jobs. Their studies indicate that dynamic equipartitioning may be an effective strategy for adaptive job scheduling. Brecht, Deng, and Gu [14] prove that dynamic equipartitioning with instantaneous parallelism as feedback is 2-competitive with respect to the makespan for jobs with multiple phases, where the parallelism of the job remains constant during the phase and the phases are relatively long compared to the length of a scheduling quantum. Their job execution model assumes that the scheduler can achieve linear speedup during each phase as long as the allotted

processors are less than the instantaneous parallelism. With similar settings and assumptions, Deng and Dymond [22] prove that DEQ with instantaneous parallelism is 4-competitive for batched multiphase jobs with respect to the mean response time.

9 Conclusion

Although the results in this paper are entirely theoretical, we are optimistic that AGDEQ and ASDEQ will perform well in the real world. The original analyses of A-GREEDY [1] and A-STEAL [2, 3] model the job scheduler as an adversary and thereby produce pessimistic bounds. A more friendly job scheduler, such as DEQ, should therefore allow jobs using A-GREEDY and A-STEAL to incur less waste and shorter execution time than predicted by the theoretical bounds. Since our analyses make use of these pessimistic bounds, we conjecture that in practice the observed makespan and mean response time will be much smaller than what the theoretical bounds predict. We are hopeful that our theoretical work will be complemented by empirical research that can shed additional light on the practicality of provably good two-level schedulers.

Acknowledgement

References

[1] Kunal Agrawal, Yuxiong He, Wen Jing Hsu, and Charles E. Leiserson. Adaptive task scheduling with parallelism feedback. In *PPoPP*, 2006.

[2] Kunal Agrawal, Yuxiong He, and Charles E. Leiserson. An empirical evaluation of work stealing with parallelism feedback. In *ICDCS*, 2006.

[3] Kunal Agrawal, Yuxiong He, and Charles E. Leiserson. Work stealing with parallelism feedback. Unpublished manuscripts, 2006.

[4] Nimar S. Arora, Robert. D. Blumofe, and C. Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. In *SPAA*, pages 119–129, Puerto Vallarta, Mexico, 1998.

[5] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, New York, NY, USA, 2003. ACM Press.

[6] Nikhil Bansal, Kedar Dhamdhere, Jochen Konemann, and Amitabh Sinha. Nonclairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004.

[7] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004.

[8] Guy Blelloch, Phil Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. *Journal of the ACM*, 46(2):281–321, 1999.

[9] Guy E. Blelloch, Phillip B. Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. In *SPAA*, pages 1–12, Santa Barbara, California, 1995.

[10] Guy E. Blelloch and John Greiner. A provable time and space efficient implementation of NESL. In *ICFP*, pages 213–225, 1996.

[11] Robert D. Blumofe. *Executing Multithreaded Programs Efficiently*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[12] Robert D. Blumofe and Charles E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, February 1998.

[13] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

[14] T. Brecht, Xiaotie Deng, and Nian Gu. Competitive dynamic multiprocessor allocation for parallel applications. In *Parallel and Distributed Processing*, pages 448 – 455. IEEE, 1995.

[15] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, pages 201–206, 1974.

[16] F. Warren Burton and M. Ronan Sleep. Executing functional programs on a virtual tree of processors. In *FPCA*, pages 187–194, Portsmouth, New Hampshire, October 1981.

[17] C. Chekuri, R. Motwani, B. Natarajan, and C. Stien. Approximation techniques for average completion time scheduling. In *SODA*, pages 609–618, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[18] Jianer Chen and Antonio Miranda. A polynomial time approximation scheme for general multiprocessor job scheduling (extended abstract). In *STOC*, pages 418–427, New York, NY, USA, 1999. ACM Press.

[19] Su-Hui Chiang and Mary K. Vernon. Dynamic vs. static quantum-based parallel processor allocation. In *JSSPP*, pages 200–223, Honolulu, Hawaii, United States, 1996.

[20] Xiaotie Deng and Patrick Dymond. On multiprocessor system scheduling. In *SPAA*, pages 82–88, 1996.

[21] Xiaotie Deng, Nian Gu, Tim Brecht, and KaiCheng Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167. Society for Industrial and Applied Mathematics, 1996.

[22] Xiaotie Deng, Nian Gu, Tim Brecht, and KaiCheng Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[23] Jianzhong Du and Joseph Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Discrete Math.*, 2(4):473–487, 1989.

[24] Jeff Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, 1999.

[25] Jeff Edmonds, Donald D. Chinn, Timothy Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *Journal of Scheduling*, 6(3):231–250, 2003.

[26] Zhixi Fang, Peiyi Tang, Pen-Chung Yew, and Chuan-Qi Zhu. Dynamic processor self-scheduling for general parallel nested loops. *IEEE Transactions on Computers*, 39(7):919–929, 1990.

[27] Dror G. Feitelson. Job scheduling in multiprogrammed parallel systems (extended version). Technical report, IBM Research Report RC 19790 (87657) 2nd Revision, 1997.

[28] R. L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal on Applied Mathematics*, pages 17(2):416–429, 1969.

[29] Nian Gu. Competitive analysis of dynamic processor allocation strategies. Master's thesis, York University, 1995.

[30] Leslie A. Hall, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line algorithms. In *SODA*, pages 142–151, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[31] Robert H. Halstead, Jr. Implementation of Multilisp: Lisp on a multiprocessor. In *LFP*, pages 9–17, Austin, Texas, August 1984.

[32] S. F. Hummel and E. Schonberg. Low-overhead scheduling of nested parallelism. *IBM Journal of Research and Development*, 35(5-6):743–765, 1991.

[33] Klaus Jansen and Lorant Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. In *SODA*, pages 490–498, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[34] Klaus Jansen and Hu Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA*, pages 86–95, New York, NY, USA, 2005. ACM Press.

[35] Bala Kalyanasundaram and Kirk R. Pruhs. Minimizing flow time nonclairvoyantly. *J. ACM*, 50(4):551–567, 2003.

[36] Scott T. Leutenegger and Mary K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *SIGMETRICS*, pages 226–236, Boulder, Colorado, United States, 1990.

[37] Steven Lucco. A dynamic scheduling method for irregular parallel programs. In *PLDI*, pages 200–211, New York, NY, USA, 1992. ACM Press.

[38] Walter Ludwig and Prasoon Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *SODA*, pages 167–176, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[39] Shikharesh Majumdar, Derek L. Eager, and Richard B. Bunt. Scheduling in multiprogrammed parallel systems. In *SIGMETRICS*, pages 104–113, Santa Fe, New Mexico, United States, 1988.

[40] Xavier Martorell, Julita Corbalán, Dimitrios S. Nikolopoulos, Nacho Navarro, Eleftherios D. Polychronopoulos, Theodore S. Papatheodorou, and Jesús Labarta. A tool to schedule parallel applications on multiprocessors: The NANOS CPU manager. In Dror G. Feitelson and Larry Rudolph, editors, *JSSPP*, pages 87–112, 2000.

[41] Cathy McCann, Raj Vaswani, and John Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, 1993.

[42] Rajeev Motwani, Steven Phillips, and Eric Torng. Non-clairvoyant scheduling. In *SODA*, pages 422–431, 1993.

[43] Gregory Mounie, Christophe Rapine, and Dennis Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *SPAA*, pages 23–32, New York, NY, USA, 1999. ACM Press.

[44] Girija J. Narlikar and Guy E. Blelloch. Space-efficient scheduling of nested parallelism. *ACM Transactions on Programming Languages and Systems*, 21(1):138–173, 1999.

[45] Emilia Rosti, Evgenia Smirni, Lawrence W. Dowdy, Giuseppe Serazzi, and Brian M. Carlson. Robust partitioning schemes of multiprocessor systems. *Performance Evaluation*, 19(2-3):141–165, 1994.

[46] Emilia Rosti, Evgenia Smirni, Giuseppe Serazzi, and Lawrence W. Dowdy. Analysis of non-work-conserving processor partitioning policies. In *IPPS*, pages 165–181, 1995.

[47] Larry Rudolph, Miriam Slivkin-Allalouf, and Eli Upfal. A simple load balancing scheme for task allocation in parallel machines. In *SPAA*, pages 237–245, Hilton Head, South Carolina, July 1991.

[48] Uwe Schwiegelshohn, Walter Ludwig, Joel L. Wolf, John Turek, and Philip S. Yu. Smart smart bounds for weighted response time scheduling. *SIAM J. Comput.*, 28(1):237–253, 1998.

[49] Siddhartha Sen. Dynamic processor allocation for adaptively parallel jobs. Master's thesis, Massachusetts Institute of technology, 2004.

[50] Kenneth C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, 19(2-3):107–140, 1994.

[51] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines online. In *FOCS*, pages 131–140, 1991.

[52] B. Song. Scheduling adaptively parallel jobs. Master's thesis, Massachusetts Institute of Technology, 1998.

[53] Mark S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. In *IPPS*, pages 219–238, 1995.

[54] Kaushik Guha Timothy B. Brecht. Using parallel program characteristics in dynamic processor allocation policies. *Performance Evaluation*, 27-28:519–539, 1996.

[55] Andrew Tucker and Anoop Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *SOSP*, pages 159–166, New York, NY, USA, 1989. ACM Press.

[56] John Turek, Walter Ludwig, Joel L. Wolf, Lisa Fleischer, Prasoon Tiwari, Jason Glasgow, Uwe Schwiegelshohn, and Philip S. Yu. Scheduling parallelizable tasks to minimize average response time. In *SPAA*, pages 200–209, 1994.

[57] John Turek, Uwe Schwiegelshohn, Joel L. Wolf, and Philip S. Yu. Scheduling parallel tasks to minimize average response time. In *SODA*, pages 112–121, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[58] Peng Yang, Dirk Desmet, Francky Catthoor, and Diederik Verkest. Dynamic scheduling of concurrent tasks with cost performance trade-off. In *CASES*, pages 103–109, New York, NY, USA, 2000. ACM Press.

[59] K. K. Yue and D. J. Lilja. Implementing a dynamic processor allocation policy for multiprogrammed parallel applications in the Solaris™operating system. *Concurrency and Computation-Practice and Experience*, 13(6):449–464, 2001.

[60] John Zahorjan and Cathy McCann. Processor scheduling in shared memory multiprocessors. In *SIGMETRICS*, pages 214–225, Boulder, Colorado, United States, May 1990.