We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Provably-Efficient Online Adaptive Scheduling of Parallel Jobs Based on Simple Greedy Rules

Yuxiong He[1,2,3] and Wen-Jing Hsu[1,2]
[1]*Singapore-MIT Alliance*
[2]*Nanyang Technological University*
[3]*Sun Microsystems*
Singapore

1. Introduction

Scheduling competing jobs on multiprocessors has always been an important issue for parallel and distributed systems. The challenge is to ensure overall system efficiency while offering a level of fairness to user jobs. Although various degrees of successes have been achieved over the past decades, few existing schemes address both efficiency and fairness over a wide range of work loads. Moreover, in order to obtain analytical results, many known results [22, 24, 7, 8, 17, 20, 23, 25, 33] require prior information about jobs such as jobs' release time, amount of work, parallelism profile, etc, which may be difficult to obtain in real applications. This chapter describes a scheduling algorithm - GRAD, which offers provable efficiency in terms of makespan and mean response time by allotting each job a fair share of processor resources. Our algorithm is *non-clairvoyant* [10, 6, 18, 12], i.e. it assumes nothing about the release time, the execution time, and the parallelism profile of jobs.

A parallel job can be classified as adaptive or non-adaptive. An *adaptively parallel job* [34] may change its parallelism, and it allows the number of the allotted processors to vary during its execution. A job is *nonadaptive* if it runs on a fixed number of processors over its lifetime. With adaptivity, new jobs can enter the system by simply recruiting processors from the already executing jobs. Moreover, in order to improve the system utilization, schedulers can shift processors from jobs that do not require many processors to the jobs in need. However, since the parallelism of adaptively parallel jobs can change during the execution and the future parallelism is usually unknown, how a scheduler decides the processor allotments for jobs is a challenging problem. We describe GRAD that effectively addresses such an adaptive scheduling problem.

Scheduling parallel jobs on multiprocessors can be implemented in two levels [14]: a kernel-level *job scheduler* which allots processors to jobs, and a user-level *thread scheduler* which maps the threads of a given job to the allotted processors. The processor reallocation occurs periodically between *scheduling quanta*. The thread scheduler provides *parallelism feedback* to the job scheduler. The feedback is an estimation of the number of processors that its job can effectively use during the next quantum. The job scheduler follows some processor allocation policy to determine the *allotment* to the job. It may implement a policy that is either *space-sharing*, where jobs occupy disjoint processor resources, or *time-sharing*, where

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

different jobs may share the same processor resources at different points in time. Once a job is allotted its processors, the allotment does not change within the quantum.

GRAD is a two-level scheduling algorithm that uses simple, greedy-like rules. The thread-level scheduler called A-GREEDY [1] provides feedback based on two simple indicators acquired from the past quantum, namely, whether its request was satisfied and whether the allotted processors are well utilized. Based on the feedbacks from all jobs, the OS allocator RAD [19] partitions processors as equally as possible. Once given the processors, A-GREEDY greedily maps the ready threads of the job onto its allotted processors. If the number of ready threads is less than or equal to the number of allotted processors, all ready threads are scheduled to execute. Otherwise, each allotted processor is assigned with one ready thread. The thread mapping in greedy manner ensures that the allotted processors always make useful work unless there are insufficient number of ready threads to work on.

Based on the "equalized allotment" scheme for processor allocation, and by using the history-based feedback, we show that GRAD is provably efficient. The performance is measured in terms of both makespan and mean response time. GRAD achieves $O(1)$-competitiveness with respect to makespan for job sets with arbitrary release times, and $O(1)$-competitiveness with respect to mean response time for batched job sets where all jobs are released simultaneously. Unlike many previous results, which either assume clairvoyance [29, 21, 31] or use instantaneous parallelism [10, 6], GRAD removes these restrictive assumptions. Moreover, because the quantum length can be adjusted to amortize the cost of context-switching during processor reallocation, GRAD provides effective control over the scheduling overhead and ensures efficient utilization of processors.

Our simulation results also suggest that GRAD performs well in practice. For job sets with arbitrary release time, their makespan scheduled by GRAD is no more than 1.39 times of the optimal on average (geometric mean). For batched job sets, their mean response time scheduled by GRAD is no more than 2.37 times of the optimal on average.

The remainder of this chapter is organized as follows. Section 2 describes the job model, scheduling model, and objective functions. Section 3 describes the GRAD algorithm. Section 4 analyzes the competitiveness of GRAD with respect to makespan. Section 5 shows the competitiveness of GRAD with respect to mean response time for batched jobs, while its detailed analysis is presented in Appendix A. Section 6 presents the empirical results. Section 7 discusses the related work, and Section 8 gives some concluding remarks.

2. Scheduling and analytical model

Our scheduling input consists of a collection of independent jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_{|\mathcal{J}|}\}$ to be scheduled on a collection of P identical processors. Time is broken into a sequence of equal-sized *scheduling quanta* 1, 2, …, each of length L, where each quantum q includes the interval $[L \cdot q, L \cdot q + 1, \ldots, L(q + 1) - 1]$ of time steps. The quantum length L is a system configuration parameter chosen to be long enough to amortize scheduling overheads. In this section, we formalize the job model, define the scheduling model, and present the optimization criteria of makespan and mean response time.

We model the execution of a multithreaded job J_i as a dynamically unfolding directed acyclic graph (*DAG*, for short). Each vertex of the DAG represents a unit-time instruction. The *work* $T_1(J_i)$ of the job J_i corresponds to the total number of vertices in the dag. Each edge represents a dependency between the two vertices. The *span* $T_\infty(J_i)$ corresponds to the

number of nodes on the longest chain of the precedence dependencies. The *release time* $r(J_i)$ of the job J_i is the time at which J_i becomes first available for processing. Each job is handled by a dedicated thread scheduler, which operates in an online manner, oblivious to the future characteristics of the dynamically unfolding DAG.

The job scheduler and the thread schedulers interact as follows. The job scheduler may reallocate processors between scheduling quanta. Between quantum q - 1 and quantum q, the thread scheduler of a given job J_i determines the job's *desire* $d(J_i, q)$, which is the number of processors J_i wants for quantum q. Based on the desire of all running jobs, the job scheduler follows its processor-allocation policy to determine the *allotment* $a(J_i, q)$ of the job with the constraint that $a(J_i, q) \leq d(J_i, q)$. Once a job is allotted its processors, the allotment does not change during the quantum.

A schedule $\mathcal{X} = (\tau, \pi)$ of a job set $\mathcal{J}$ is defined as two mappings $\tau : \cup_{J_i \in \mathcal{J}} V_i \rightarrow \{1, 2, \ldots, 1\}$, and $\pi : \cup_{J_i \in \mathcal{J}} V_i \rightarrow \{1, 2, \ldots, P\}$, which map the vertices of the jobs in the job set $\mathcal{J}$ to the set of time steps, and the set of processors on the machine respectively. A valid mapping must preserve the precedence relationship of each job. For any two vertices $u, v \in V_i$ of the job J_i, if $u \prec v$, then $\tau(u) < \tau(v)$, i.e. the vertex u must be executed before the vertex v. A valid mapping must also ensure that one processor can only be assigned to one job at any given time. For any two vertices u and v, both $\tau(u) = \tau(v)$ and $\pi(u) = \pi(v)$ are true iff $u = v$.

Our scheduler uses makespan and mean response time as the performance measurement.

Definition 1 The *makespan* of a given job set $\mathcal{J}$ is the time taken to complete all the jobs in $\mathcal{J}$, i.e. $T(\mathcal{J}) = \max_{J_i \in \mathcal{J}} T(J_i)$, where $T(J_i)$ denotes the completion time of job J_i.

Definition 2 The *response time* of a job J_i is $T(J_i) - r(J_i)$, which is the duration between its release time $r(J_i)$ and the completion time $T(J_i)$. The *total response time* of a job set $\mathcal{J}$ is given by $R(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} (T(J_i) - r(J_i))$ and the *mean response time* is $\overline{R}(\mathcal{J}) = R(\mathcal{J}) / |\mathcal{J}|$.

The goal of the chapter is to show that our scheduler optimizes the makespan and mean response time, and we use competitive analysis as a tool to evaluate and compare the scheduling algorithm. The competitive analysis of an online scheduling algorithm is to compare the algorithm against an optimal clairvoyant algorithm. Let $T^*(\mathcal{J})$ denote the makespan of an arbitrary jobset $\mathcal{J}$ scheduled by an optimal scheduler, and $T(\mathcal{J})$ denote the makespan produced by an algorithm A for the job set $\mathcal{J}$. A deterministic algorithm A is said to be *c-competitive* if there exists a constant b such that $T(\mathcal{J}) \leq c \cdot T^*(\mathcal{J}) + b$ holds for the schedule of any job set. We will show that our algorithm is c-competitive in terms of the makespan, where c is a small constant. Similarly, for the mean response time, we will show that our algorithm is also constant-competitive for any batched jobs.

3. Algorithms

This section presents the job scheduler - RAD, and overviews the thread scheduler - A-GREEDY [1].

RAD Job Scheduler

The job scheduler RAD unifies the space-sharing job scheduling algorithm DEQ [35, 27] with the time-sharing RR algorithm. When the number of jobs is greater than the number of processors, GRAD schedules the jobs in a batched, round-robin fashion, which allocates one processor to each job with an equal share of time. When the number of jobs is not more than the number of processors, GRAD uses DEQ as the job scheduler. DEQ gives each job an equal share of spatial allotments unless the job requests for less.

When a batch of jobs are scheduled in the round-robin fashion, RAD maintains a queue of jobs. At the beginning of each quantum, if there are more than P jobs, it pops P jobs from the top of the queue, and allots one processor to each of them during the quantum. At the end of the quantum, RAD pushes the P jobs back to the bottom of the queue if they are uncompleted. The new jobs can be put into the queue once they are released.

DEQ attempts to give each job a fair share of processors. If a job requires less than its fair share, however, DEQ distributes the extra processors to the other jobs. More precisely, upon receiving the desires $\{d(J_i, q)\}$ 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 of every job $J_i \in \mathcal{J}$ satisfies $d(J_i, q) \geq P/n$, assign each job $a(J_i, q) = P/n$ processors.
3. Otherwise, let $\mathcal{J}' = \{J_i \in \mathcal{J}: d(J_i, q) < P/n\}$. Assign $a(J_i, q) = d(J_i, q)$ processors to each $J_i \in \mathcal{J}'$. Update $\mathcal{J} = \mathcal{J} - \mathcal{J}'$, and $P = P - \sum_{J_i \in \mathcal{J}'} d(J_i, q)$. Go to Step 1.

Note that, at any quantum where the number of jobs is equal to the number of processors, DEQ and RR give exactly the same processor allotment, and allocate each of P jobs with one processor.

Adaptive Greedy Thread Scheduler

A-GREEDY [1] is an adaptive greedy thread scheduler with parallelism feedback. Between quanta, it estimates its job's desire, and requests processors from the job scheduler. During the quantum, it schedules the ready threads of the job onto the allotted processors greedily [15, 5]. If there are more than $a(J_i, q)$ ready threads, A-GREEDY schedules any $a(J_i, q)$ of them. Otherwise, it schedules all of them.

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 adjusted for different levels of guarantees for waste and completion time.

Before each quantum, A- GREEDY y for a job $J_i \in \mathcal{J}$ provides parallelism feedback to the job scheduler based on the J_i's history of utilization in the previous quantum. A- GREEDY classifies quanta as "satisfied" versus "deprived" and "eficient" versus "inefficient." A quantum q is *satisfied* if $a(J_i, q) = d(J_i, q)$, in which case J_i's allotment is equal to its desire. Otherwise, the quantum is *deprived*.[1] 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*. Under the four-way 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 as follows:

- If quantum $q - 1$ was inefficient, decrease the desire, setting $d(J_i, q) = d(J_i, q - 1)/\rho$, where ρ is the responsiveness parameter.
- If quantum $q - 1$ was efficient-and-satisfied, increase the desire, setting $d(J_i, q) = \rho d(J_i, q - 1)$.
- If quantum $q - 1$ was efficient-and-deprived, keep desire unchanged, setting $d(J_i, q) = d(J_i, q - 1)$.

[1] We can extend the classification of "satisfied" versus "deprived" from quanta to time steps. A job J_i is *satisfied* (or *deprived*) at step $t \in [L \cdot q, L \cdot q + 1, \ldots, L(q + 1) - 1]$ if J_i is satisfied (resp. deprived) at the quantum q.

4. Makespan

This section shows that GRAD is *c*-competitive with respect to makespan, where *c* denotes a constant. The exact value of *c* is related to the choice of A-GREEDY's utilization and responsiveness parameter, as will be explained shortly.

We first review the lower bounds of makespan. 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 span. Recall that, for a job $J_i \in \mathcal{J}$, the quantities $r(J_i)$, $T_1(J_i)$, and $T_\infty(J_i)$ represent the release time, work, and span of J_i, respectively. Let $T^*(\mathcal{J})$ denote the makespan produced by an optimal scheduler on a job set $\mathcal{J}$ on P processors. Let $T_1(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_1(J_i)$ denote the total work of the job set. The following two inequalities give two lower bounds on the makespan [6]:

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

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

To facilitate the analysis, we state a lemma from [1] that bounds the satisfied steps and the waste of a single job scheduled by A-GREEDY. Recall that, the parameter $\rho > 1$ denotes A-GREEDY's responsiveness parameter, $\delta > 0$ its utilization parameter, and L the quantum length.

Lemma 1 *[1] For a job J_i with work $T_1(J_i)$ and span $T_\infty(J_i)$ on a machine with P processors,* A- GREEDY *produces at most* $2T_\infty(J_i)/(1 - \delta)+L\log_\rho P + L$ *satisfied steps, and it wastes at most* $(1+\rho - \delta)T_1(J_i)/\delta$ *processor cycles in the course of the computation.* □

The following theorem analyzes the makespan of a job set $\mathcal{J}$ scheduled by GRAD.

Theorem 2 *Let ρ denote* A-GREEDY's *responsiveness parameter, δ its utilization parameter, and L the quantum length. Then,* GRAD *completes a job set $\mathcal{J}$ on P processors in*

$$T(\mathcal{J}) \leq \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P} + \frac{2}{1-\delta}\max_{J_i \in \mathcal{J}}\{T_\infty(J_i) + r(J_i)\} + L\log_\rho P + 2L \quad (3)$$

time steps.

Proof. Suppose job J_k is the last job completed among the jobs in $\mathcal{J}$. Let $S(J_k)$ denote the set of satisfied steps for J_k, and $D(J_k)$ denote its set of deprived steps. The job J_k is scheduled to start its execution at the beginning of the quantum q where $Lq < r(J_k) \leq L(q + 1)$, which is the quantum immediately after J_k's release. Therefore, we have $T(\mathcal{J}) \leq r(J_k) + L + |S(J_k)| + |D(J_k)|$. We now bound $|S(J_k)|$ and $|D(J_k)|$ respectively.

From Lemma 1, we know that the number of satisfied steps attributed to J_k is at most $|S(J_k)| \leq 2T_\infty(J_k)/(1 - \delta) + L\log_\rho P + L$.

We now bound the total number of deprived steps $D(J_k)$ of job J_k. For each step $t \in D(J_k)$, GRAD applies either DEQ or RR as job scheduler. RR always allots all processors to jobs. By definition, DEQ must have allotted all processors to jobs whenever J_k is deprived. Thus, the total allotment on such a step t is always equal to the total number of processors P. Moreover, the total allotment of $\mathcal{J}$ over J_k's deprived steps $D(J_k)$ is $a(\mathcal{J}, D(J_k)) = \sum_{t \in D(J_k)} \sum_{J_i \in \mathcal{J}} a(J_i, t) = P|D(J_k)|$. Since any allotted processor is either working productively or wasted, the total allotment for any job J_i is bounded by the sum of its total

work $T_1(J_i)$ and total waste $w(J_i)$. By Lemma 1, the waste for the job J_i is at most $(\rho - \delta + 1)/\delta$ times of its work. Thus, the total number of allotted processor cycles for job J_i is at most $T_1(J_i) + w(J_i) \le (\rho + 1)T_1(J_i)/\delta$. The total number of allotted processor cycles for all jobs is at most $\sum_{J_i \in \mathcal{J}} (\rho + 1)T_1(J_i)/\delta = ((\rho + 1)/\delta)T_1(\mathcal{J})$. Given $a(\mathcal{J}, D(J_k)) \le ((\rho + 1)/\delta)T_1(\mathcal{J})$ and $a(\mathcal{J}, D(J_k)) = P|D(J_k)|$, we have $|D(J_k)| \le \frac{\rho+1}{\delta}\frac{T_1(\mathcal{J})}{P}$.
Therefore, we can get

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

□

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

Corollary 3 GRAD *completes a job set* $\mathcal{J}$ *in*

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

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

Since both the quantum length L and the processor number P are independent variables with respect to any job set $\mathcal{J}$, Corollary 3 shows that GRAD is $O(1)$-competitive with respect to makespan.

To better interpret the bound, let's substitute $\rho = 1.2$ and $\delta = 0.6$, we have $\mathrm{T}(\mathcal{J}) \le 8.67\mathrm{T}^*(\mathcal{J}) + L\lg P/\lg 1.2 + 2L$. Since both the quantum length L and the processor number P are independent variables with respect to any job set $\mathcal{J}$, GRAD is 8.67-competitive given $\rho = 1.2$ and $\delta = 0.6$.

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

5. Mean response time

Mean response time is an important measure for multiuser environments where we desire as many users as possible to get fast response from the system. In this section, we first introduce the lower bounds. Then, we show that GRAD is $O(1)$-competitive for batched jobs with respect to the mean response time.

Lower Bounds and Preliminaries

We first introduce some definitions.

Definition 3 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 $\alpha_{f(1)} \le \alpha_{f(2)} \le \ldots \le \alpha_{f(n)}$. The *squashed sum* of 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

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

where $T_1\,(J_i)$ is the work of job $J_i \in \mathcal{J}$. The *aggregate span* of $\mathcal{J}$ is

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

where $T_\infty(J_i)$ is the span of job $J_i \in \mathcal{J}$.
The research in [36, 37, 10] establishes two lower bounds for the mean response time:

$$\overline{\mathrm{R}^*}(\mathcal{J}) \quad \geq \quad T_\infty(\mathcal{J})/\,|\mathcal{J}|\ , \tag{4}$$

$$\overline{\mathrm{R}^*}(\mathcal{J}) \quad \geq \quad \mathrm{swa}\,(\mathcal{J})\,/\,|\mathcal{J}|\ , \tag{5}$$

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

Analysis

The proof is divided into two parts. In the first part where $|\mathcal{J}| \leq P$, GRAD always uses DEQ as job scheduler. In this case, we apply the result in [18], and show that GRAD is $O(1)$-competitive. In the second part where $|\mathcal{J}| > P$, GRAD uses both RR and DEQ. Since we consider batched jobs, the number of incomplete jobs decreases monotonically. When the number of incomplete jobs drops to P, GRAD switches its job scheduler from RR to DEQ. Therefore, we prove the second case based on the properties of round robin scheduling and the results of the first case. The following theorem shows the total response time bound for the batched job sets scheduled by GRAD. Please refer to Appendix A for the complete proof.

Theorem 4 *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 a job set* $\mathcal{J}$ *produced by* GRAD *is at most*

$$\mathrm{R}(\mathcal{J}) = \left(2 - \tfrac{2}{|\mathcal{J}|+1}\right)\left(\tfrac{\rho+1}{\delta}\,\mathrm{swa}\,(\mathcal{J}) + \tfrac{2}{1-\delta}T_\infty(\mathcal{J})\right) + O(|\mathcal{J}|\,L\log_\rho P)\ , \tag{6}$$

where swa $(\mathcal{J})$ *denotes the squashed work area of* $\mathcal{J}$, *and* $T_\infty(\mathcal{J})$ *denotes the aggregate span of* $\mathcal{J}$. □

Since both swa $(\mathcal{J})$ / $|\mathcal{J}|$ and $T_\infty(\mathcal{J})$/ $|\mathcal{J}|$ are lower bounds on $\mathrm{R}(\mathcal{J})$, we obtain the following corollary. It shows that GRAD is $O(1)$-competitive with respect to mean response time for batched jobs.

Corollary 5 *The mean response time* $\overline{\mathrm{R}}(\mathcal{J})$ *of a batched job set* $\mathcal{J}$ *produced by* GRAD *satisfies*

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

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

6. Experimental results

To evaluate the performance of GRAD, we conducted four sets of experiments, which are summarized below.

- The **makespan experiments** compares the makespan produced by GRAD against the theoretical lower bound for over 10000 runs of job sets.
- The **mean response time experiments** investigate how GRAD performs with respect to mean response time for over 8000 batched job sets.
- The **load experiments** investigate how the system load affects the performance of GRAD.
- The **proactive RAD experiments** compare the performance of RAD against its variation - proactive RAD. The proactive RAD always allots all processors to jobs even if the overall desire is less than the total number of processors.

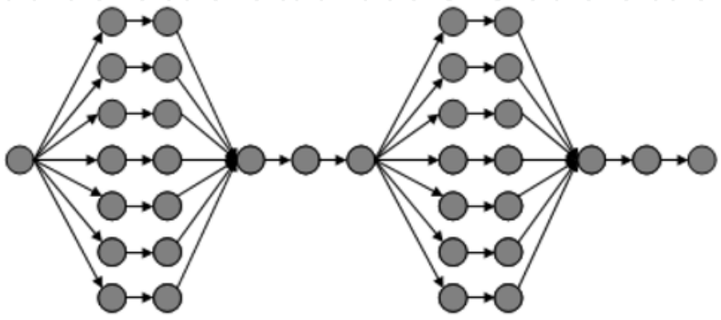


Fig. 1. The DAG of a fork-join job used in the simulation. This job has start-up length $w_0 = 1$, serial phase length $w_1 = 3$, parallel phase length $w_2 = 2$, parallelism $h = 7$, and the number of iterations $iter = 2$.

6.1 Simulation setup

To study GRAD, we build a Java-based discrete-time simulator using DESMO-J [11]. Our simulator models four major entities - processors, jobs, thread schedulers, and job schedulers, and simulates their interactions in a two-level scheduling environment. As described in Section 2, we model the execution of a multithreaded job as a dag. When a job is submitted to the simulated multiprocessor system, an instance of a thread scheduler is created for the job. The job scheduler allots processors to the job, and the thread scheduler executes the job using A-GREEDY. The simulator operates in discrete time steps, and we ignore the overheads incurred in the reallocation of processors.

Our benchmark application is the Fork-Join jobs, whose task graphs are typically as shown in Figure 1. Each job alternates between a *serial phase* of length w_1 and a *parallel phase* (with h-way parallelism) of length w_2, while the initial serial phase has length w_0. The parallelism of job's parallel phase is the *height* h of the job, and the number of iterations is denoted as *iter* . Fork-Join jobs arise naturally in jobs that exhibit "data parallelism", and apply the same computation to a number of different data points. Many computationally intensive applications can be expressed in a data-parallel fashion [30]. The repeated fork-join cycle in the job reflects the often iterative nature of these computations. The average parallelism of the job is approximately $(w_1 + hw_2)=(w_1 + w_2)$. By varying the values of w_0, w_1, w_2, h, and the number of iterations, we can generate jobs with different work, spans, and phase lengths.

GRAD requires some parameters as input. We set the responsiveness parameter to be $\rho = 2.0$, and the utilization parameter $\delta = 0.8$ unless otherwise specified. GRAD is designed for moderate-scale and large-scale multiprocessors, and we set the number of processors to be $P = 128$. The quantum length L represents the time between successive reallocations of

processors by the job scheduler, and is selected to amortize the overheads due to the communication between the job scheduler and the thread scheduler, and the reallocation of processors. In conventional computer systems, a scheduling quantum is typically between 10 and 20 milliseconds. The execution time of a task is decided by the granularity of the job. If a task takes approximately 0.5 to 5 microseconds, then the quantum length L should be set to values between 10^3 and 10^5 time steps. Our theoretical bounds indicate that as long as $T_\infty \gg L\log P$, the length of L should have little effect on our results. In our experiments, we set L = 1000.

6.2 Makespan experiments

The competitive ratio of makespan derived in Section 4, though asymptotically strong, has a relatively large constant multiplier. The makespan experiments were designed to evaluate the constants that would occur in practice and compare GRAD to an optimal scheduler. The experiments are conducted on more than 10, 000 runs of job sets using many combinations of jobs and different loads.

Figure 2 shows how GRAD performs compared to an optimal scheduler. The makespan of a job set $\mathcal{J}$ has two lower bounds $\max_{J_i \in \mathcal{J}}(r(J_i) + T_\infty(J_i))$ and $T_1(\mathcal{J}) = P$. The makespan produced by an optimal scheduler is lower-bounded by the larger of these two values. The makespan ratio in Figure 2 is defined as the makespan of a job set scheduled by GRAD divided by the theoretical lower bounds. Its X-axis represents the range of the makespan ratio, while the histogram shows the percentage of the job sets whose makespan ratio falls into the range. Among more than 10, 000 runs, 76.19% of them use less than 1.5 times of the theoretical lower bound, 89.70% use less than 2.0 times, and none uses more than 4.5 times. The average makepsan ratio is 1.39, which suggests that, in practice, GRAD has a small competitive ratio with respect to the makespan.

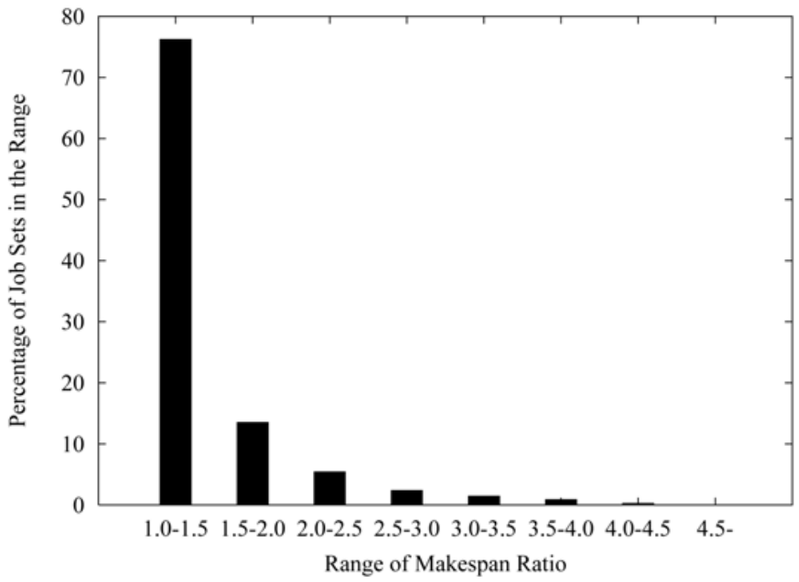


Fig. 2. Comparing the makespan of GRAD with the theoretical lower bound for job sets with arbitrary job release time.

We now interpret the relation between the theoretical bounds and experimental results as follows. When ρ = 2 and δ = 0.8, from Theorem 2, GRAD is 13.75-competitive in the worst case. However, we anticipate that GRAD's makespan ratio would be small in practical settings, especially when the jobs have total work much great than the span and with the machine moderately- or highly- loaded. In this case, the term on $T_1(\mathcal{J})/P$ in Inequality (3) of Theorem 2 is much larger than the term $\max_{J_i \in \mathcal{J}} \{T_\infty(i) + r(i)\}$, i.e. the term $T_1(\mathcal{J})/P$ generally dominates the makespan bound. The proof of Theorem 2 calculates the coefficient of $T_1(\mathcal{J})/P$ as the ratio of the total allotment (total work plus total waste) versus the total work. When the job scheduler is RAD, which is not a true adversary, our simulation results indicate that the ratio of the waste versus the total work is only about 1/10 of the total work. Thus, the coefficient of $T_1(\mathcal{J})/P$ in Inequality (3) is about 1.1. It explains why the makespan produced by GRAD is less than 2 times of the lower bound on average as shown in Figure 2.

6.3 Mean response time experiments

This set of experiments is designed to evaluate the mean response time of the batch job sets scheduled by GRAD. Figure 3 shows the distribution of the mean response time normalized w.r.t. the larger of the two lower bounds { the squashed work bound swa $(\mathcal{J})/|\mathcal{J}|$ and the aggregated critical path bound $T_\infty(\mathcal{J})/|\mathcal{J}|$. The histogram in Figure 3 shows that, among more than 8, 000 runs, 94.65% of them use less than 3 times of the theoretical lower bound, and none of them uses more than 5:5 times. The average mean response time ratio is 2.37.

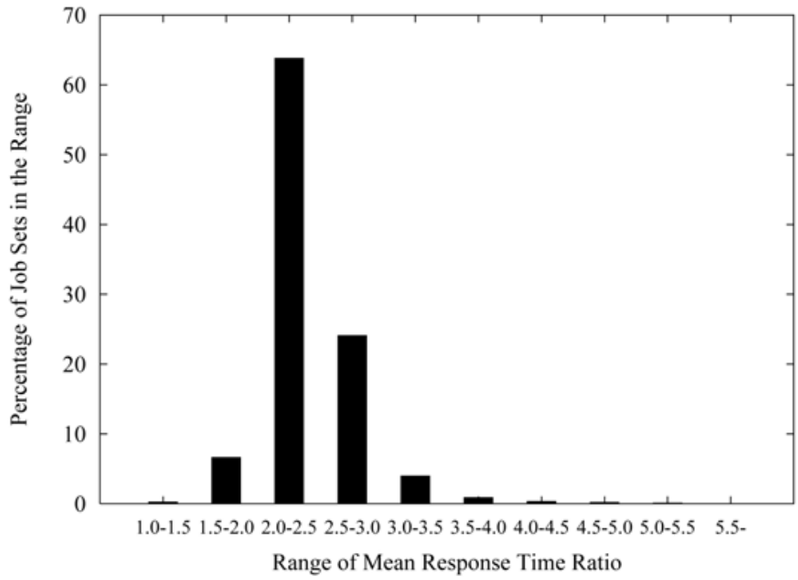


Fig. 3. Comparing the mean response time of GRAD with the theoretical lower bound for batched job sets.

Similar to the discussion in Section 6.2, we can relate the theoretical bounds for mean response time to the experimental results. When ρ = 2 and ρ = 0.8, from Theorem 4, GRAD is 27.60-competitive. However, we expect that GRAD should perform closer to optimal in practice. In particular, when the job set 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 4 dominates the total response time. More importantly, RAD is not an adversary of A-GREEDY, as mentioned

before, the waste of a job is only about 1/10 of the total work in average for over 100, 000 job runs we tested. Based on this waste, the squashed area bound swa ($\mathcal{J}$) in Inequality (6) of Theorem 4 has a coefficient to be around 2.2. It explains that the mean response time produced by GRAD is less than 3 times of the lower bound as shown in Figure 3.

6.4 Load experiments

This set of experiments is designed to investigate how the load affects the performance of GRAD. The *load* of a job set J on a machine with *P* processors indicates how heavily the jobs compete for processors on the machine, which is calculated as follows

$$load = \frac{T_1(\mathcal{J})}{P \cdot \left(\max_{J_i \in \mathcal{J}} r(J_i) - \min_{J_i \in \mathcal{J}} r(J_i) + T_\infty(\mathcal{J}) / |\mathcal{J}| \right)}.$$

For a batched job set, the load is just the average parallelism of the set divided by the total number of processors.

Figure 4 shows how GRAD performs against the theoretical lower bound with respect to makespan by varying system load. The makespan ratio in this figure is defined as the makespan of a job set scheduled by GRAD divided by the larger of the two lower bounds. Each data point represents the makespan ratio of a job set. The testing results suggest that the makespan ratio becomes smaller when the load gets heavier. Specifically, the makespan generated by GRAD is very close to the lower bound when the load is greater than 4; it never exceeds 1.5 times of the makespan produced when the system load is greater than 3. However, when the load is less than 2, the makespan ratio spreads in the range from 1 to 4.

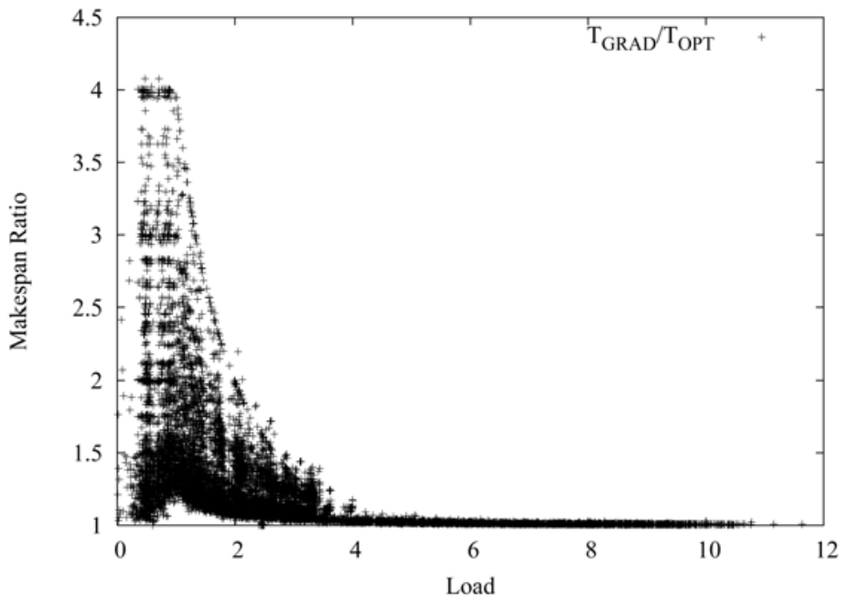


Fig. 4. Comparing GRAD against the theoretical lower bound for makespan with varying load.

Figure 5 shows the performance of GRAD with respect to mean response time for batched jobs by varying system load. It compares the mean response time incurred by GRAD with

the theoretical lower bound. Under heavy load, the mean response time produced by GRAD concentrates on about 2 times of the lower bound, while under light load, the ratio spreads in the range from 1 to 4.

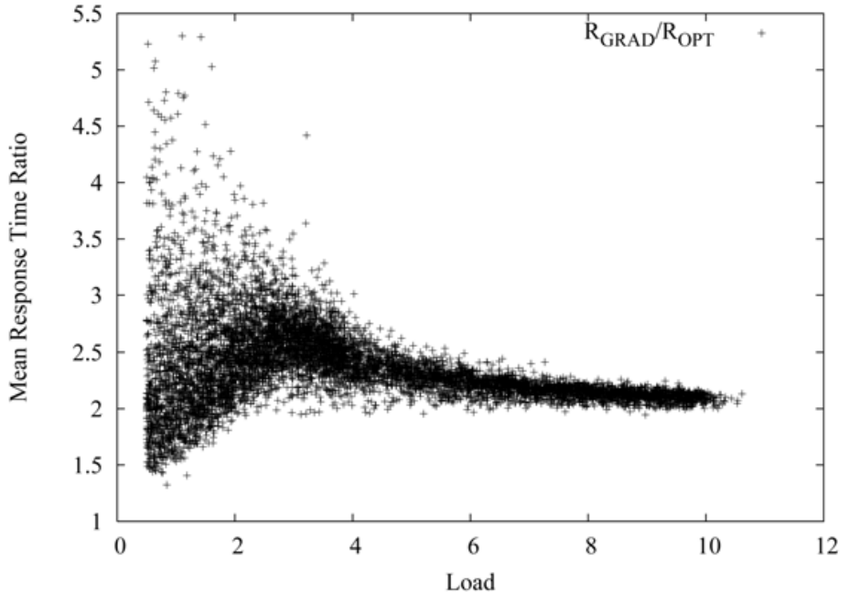


Fig. 5. Comparing GRAD against the theoretical lower bound for mean response time with varying load for batched jobs.

The load experiments bring up a question of how to improve the performance of GRAD under light load. The job scheduler RAD makes conservative decision on the allocation of processors to jobs. When the system is lightly loaded where the total demand is less than the total number of processors, RAD keeps some processors idle without allocating them to any jobs. Since a greedy thread scheduler executes a job faster with more processors allotted, a job scheduler that always allots all processors to jobs should perform better under light load. We will explore such a variation of the job scheduler RAD in the next set of the experiments.

6.5 Proactive RAD experiments

Proactive RAD always allocates all processors to jobs even if the total requests are less than the total number of processors. At a quantum q, when the total requests $d(\mathcal{J}, q) = \Sigma_{J_i \in \mathcal{J}} d(J_i, q)$ are greater than or equal to the total number P of processors, the proactive RAD works exactly the same as the original one. However, if $d(\mathcal{J}, q) < P$, the proactive RAD evenly allots the remaining $P - d(\mathcal{J}, q)$ processors to all the jobs.

Figure 6 shows the makespan ratio of proactive RAD against its original algorithm by varying system load. Each data point in the figure represents a job set's makespan ratio, defined as the makespan produced by the proactive RAD divided by that of the original. We can see that the makespan ratio is less than 1 for most of the runs, indicating that the proactive RAD out-performs the original one in most of these job sets. Moreover, the difference between them becomes more pronounced under light load, and diminishes with the increase of the system load. The reason is that the proactive RAD generally allocates more processors to jobs, especially when the load is light. The increased allotment allows

faster execution of jobs which shortens the makespan of the job set. Figure 6 gives evidences that the proactive RAD improves the performance of our scheduling algorithm under light load.

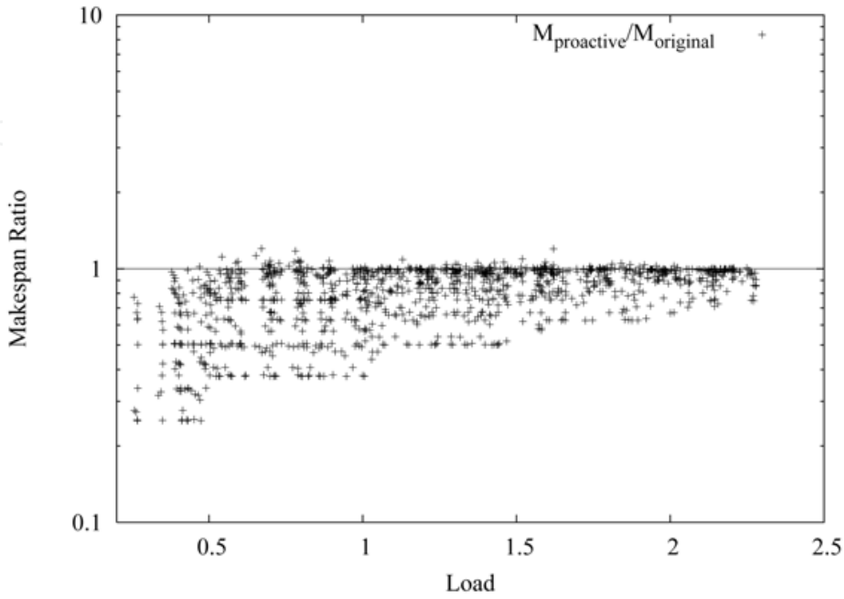


Fig. 6. Comparing the proactive RAD against the original for makespan with varying load. The X-axis represents the load of the system. The Y- axis represents the makespan ratio between the proactive and original RAD.

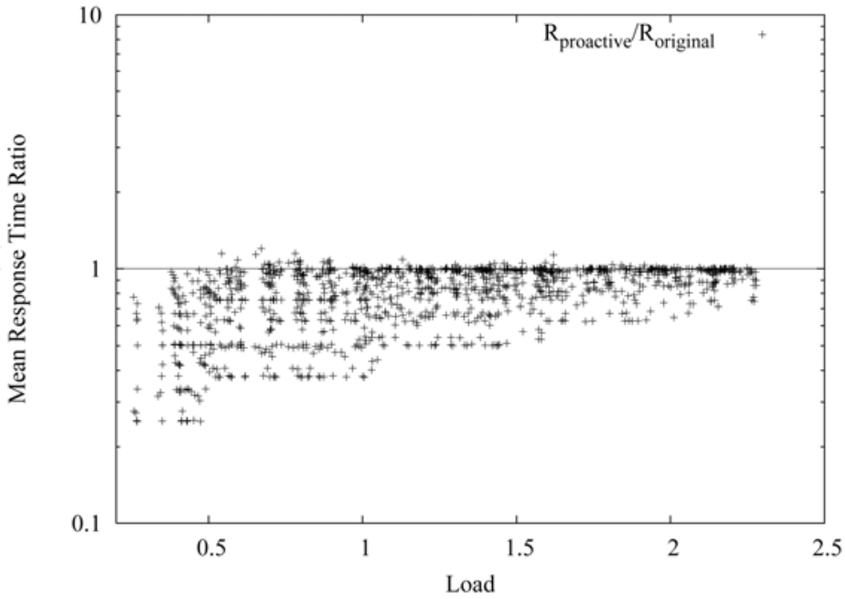


Fig. 7. Comparing the proactive RAD against the original for mean response time with varying load . The X-axis represents the load of the system. The Y-axis represents the mean response time ratio between the proactive and original RAD.

7. Related work

Adaptive parallel job scheduling has been studied both empirically [27, 38, 35, 26] and theoretically [16, 9, 28, 12, 13, 4]. McCann, Vaswani, and Zahorjan [27] introduce the notion of dynamic equipartitioning (DEQ), 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 the other jobs. Brecht, Deng, and Gu [6] prove that DEQ with instantaneous parallelism as feedback is 2-competitive with respect to the makespan. Later, Deng and Dymond [10] prove that DEQ with instantaneous parallelism is also 4-competitive for batched jobs with respect to the mean response time.

Even though using instantaneous parallelism as feedback is intuitive, it can either cause gross misallocation of processor resources [32] 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.

Our previous work in [18] presents a two-level adaptive scheduler AGDEQ, which uses DEQ as the job scheduler, and A-GREEDY as the thread scheduler. Instead of using instantaneous parallelism, AGDEQ uses the job's utilization in the past as feedback. AGDEQ is $O(1)$-competitive for makespan, and in a batched setting, $O(1)$-competitive for mean response time. However, as with other prior work [6, 10] that uses DEQ as the job scheduler, AGDEQ can only be applied to the case where the total number of jobs in the job set is less than or equal to the number of processors.

8. Conclusions

We have presented a non-clairvoyant adaptive scheduling algorithm GRAD that ensures provable efficiency, fairness and minimal overhead.

The history-based feedback mechanism of GRAD can be applied to not only greedy-based thread schedulers, but many other thread schedulers. For example, GRAD using greedy rules to map ready threads to allotted processors is suitable for scheduling jobs in more centralized setting such as data parallel applications. In the centralized setting, the scheduler has the information of all ready threads at any moment such that it can apply greedy rules to make effective assignment of ready threads. However, for applications using many processors and executed with more distributed setting, it can be costly for a scheduler to collect the ready threads information before making each scheduling decision. In this case, other than using a greedy thread scheduler, it is more practical to apply a distributed thread scheduler such as A-STEAL [2, 3] that uses randomized work stealing. A-STEAL performs as well as A-GREEDY asymptotically [3] in terms of both job completion time and waste, however, A-STEAL has slightly larger coefficients because it does not have the complete information on ready threads to make full utilization of the allotted processors. Therefore, a greedy scheduler like A-GREEDY could be a good choice in the centralized setting, while A-STEAL can be applied in the distributed setting where a greedy thread scheduler is no

longer applicable. Analogously, one can develop a two-level scheduler by applying the feedback mechanism in GRAD, and application-specific thread schedulers. Such a two-level scheduler can be developed to provide both system-wide performance guarantees such as minimal makespan and mean response time, and optimization of individual applications.

9. Acknowledgements

The preliminary version of GRAD algorithm was published in our paper [19] coauthored with Charles E. Leiserson. The authors would like to thank Charles for many helpful discussions on formalizing the analysis and advices on revising the write-up.

10. References

[1] K. Agrawal, Y. He, W. J. Hsu, and C. E. Leiserson. Adaptive task scheduling with parallelism feedback. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 100 - 109, New York City, NY, USA, 2006.

[2] K. Agrawal, Y. He, and C. E. Leiserson. An empirical evaluation of work stealing with parallelism feedback. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 19 - 29, Lisboa, Portugal, 2006.

[3] K. Agrawal, Y. He, and C. E. Leiserson. Work stealing with parallelism feedback. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Jose, CA, USA, 2007.

[4] N. Bansal, K. Dhamdhere, J. Konemann, and A. Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305-318, 2004.

[5] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202-229, 1998.

[6] T. Brecht, X. Deng, and N. Gu. Competitive dynamic multiprocessor allocation for parallel applications. In *Parallel and Distributed Processing*, pages 448 - 455, San Antonio, TX, 1995.

[7] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In *In the Proceedings of Automata, Languages and Programming*, pages 646-657, Paderborn, Germany, 1996.

[8] B. Chen and A. P. A. Vestjens. Scheduling on identical machines: How good is lpt in an on-line setting? *Operations Research Letters*, 21:165-169, 1998.

[9] X. Deng and P. Dymond. On multiprocessor system scheduling. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 82-88, Padua, Italy, 1996.

[10] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 159-167, Philadelphia, PA, USA, 1996.

[11] DESMO-J: A framework for discrete-event modelling and simulation. http://asi-www.informatik.uni-hamburg.de/desmoj/.

[12] J. Edmonds. Scheduling in the dark. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 179-188, Atlanta, Georgia, United States, 1999.

[13] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *Journal of Scheduling*, 6(3):231-250, 2003.

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

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

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

[17] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 142-151, Philadelphia, PA, USA, 1996.

[18] Y. He, W. J. Hsu, and C. E. Leiserson. Provably e±cient two-level adaptive scheduling. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Saint-Malo, France, 2006.

[19] Y. He, W. J. Hsu, and C. E. Leiserson. Provably efficient online non-clairvoyant scheduling. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, 2007.

[20] K. S. Hong and J. Y. T. Leung. On-line scheduling of real-time tasks. *IEEE Transactions on Computers*, 41(10):1326-1331, 1992.

[21] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 86-95, New York, NY, USA, 2005.

[22] D. Karger, C. Stein, and J. Wein. *Handbook of Algorithms and Theory of Computation*, chapter 35 - Scheduling Algorithms. CRC Press, 1997.

[23] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonaproximability results for minimizing total flow time on single machine. In *Proceedings of the ACM Symposium on the Theory of Computing*, Philadelphia, Pennsylvania, USA, 1996.

[24] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *Sequencing and Scheduling: Algorithms and Complexity*, pages 445-552. Elsevier Science Publishers, 1997.

[25] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 110-119, El Paso, Texas, USA, 1997.

[26] S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *SIGMETRICS*, pages 226-236, Boulder, Colorado, United States, 1990.

[27] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146-178, 1993.

[28] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *Proceedings of the ACM- SIAM Symposium on Discrete Algorithms*, pages 422-431, Austin, Texas, United States, 1993.

[29] G. Mounie, C. Rapine, and D. Trystram. E±cient approximation algorithms for scheduling malleable tasks. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 23-32, New York, NY, USA, 1999.

[30] L. S. Nyland, J. F. Prins, A. Goldberg, and P. H. Mills. A design methodology for data-parallel applications. *IEEE Transactions on Software Engineering*, 26(4):293-314, 2000.

[31] U. Schwiegelshohn, W. Ludwig, J. L.Wolf, J. Turek, and P. S. Yu. Smart smart bounds for weighted response time scheduling. *SIAM Journal of Computing*, 28(1):237-253, 1998.

[32] S. Sen. Dynamic processor allocation for adaptively parallel jobs. Master's thesis, Massachusetts Institute of technology, 2004.
[33] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines online. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 131-140, San Juan, Puerto Rico, 1991.
[34] B. Song. Scheduling adaptively parallel jobs. Master's thesis, Massachusetts Institute of Technology, 1998.
[35] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 159-166, New York, NY, USA, 1989.
[36] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response time. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 200-209, Cape May, New Jersey, United States, 1994.
[37] J. Turek, U. Schwiegelshohn, J. L.Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 112-121, Philadelphia, PA, USA, 1994.
[38] 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.

Appendix A. Proof of Theorem 4

The proof is divided into two cases - when $|\mathcal{J}| \le P$ and when $|\mathcal{J}| > P$.

Case 1: when $|\mathcal{J}| \le P$

For the first case where $|\mathcal{J}| \le P$, GRAD always use DEQ as job scheduler. In our previous work [18], we show that AGDEQ (the combination of DEQ and A-GREEDY) is O(1)-competitive with respect to mean response time for batched jobs when $|\mathcal{J}| \le P$. The following lemma from [18] bounds the mean response time of a batched job set with $|\mathcal{J}| \le P$.

Lemma 7 *[18] A job set $\mathcal{J}$ is scheduled by* GRAD *on P processors where $|\mathcal{J}| \le P$. The total response time* $\mathrm{R}(\mathcal{J})$ *of the schedule is at most*

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

where $c = 2 - 2/|\mathcal{J}| + 1$).

Case 2: when jJ j > *P*

We now derive the mean response time of GRAD for batched jobs for the second case where $|\mathcal{J}| > P$. Since all jobs in the job set J arrive at time step 0, the number of uncompleted jobs decreases monotonically. When the number of uncompleted jobs drops down to *P* or below, GRAD switches its job scheduler from RR to DEQ. We divide the analysis into three parts. In **Part (a)**, we prove two technical lemmas (Lemmas 8 and 9) which show the properties of round robin as the job scheduler. In **Part (b)**, we analyze the completion time of the jobs which are scheduled by RR during their entire execution. In **Part (c)**, we combine results and give response time of GRAD in general.

A batched job set $\mathcal{J}$ can be divided into two subsets - *RR set* and *DEQ set*. The RR set, denoted as $\mathcal{J}_{RR}$, includes all the jobs in $\mathcal{J}$ which are entirely scheduled by RR for their execution. The DEQ set, denoted as $\mathcal{J}_{DEQ}$, includes all the jobs in $\mathcal{J}$ which are scheduled by RR at the beginning, and by DEQ eventually. There exists a unique quantum q called the

final RR quantum such that q is the last quantum scheduled by RR, and from quanta q+1 onwards are all scheduled by DEQ. According to RAD, there must be greater than P uncompleted jobs at the beginning of q, and less than or equal to P uncompleted jobs immediately after the execution of q. Let σ denote the total number of uncompleted jobs immediately after the execution of the final RR quantum. We know that $\sigma = |\mathcal{J}_{DEQ}|$, and $\sigma \le P$. Let π denote a permutation that lists the jobs according to the ascending order of their completion time, i.e. $T(J_{\pi(1)}) \le T(J_{\pi(2)}) \le \ldots \le T(J_{\pi(|\mathcal{J}|)})$. We have $\mathcal{J}_{RR} = \{\mathcal{J}_{\pi(i)} \mid 1 \le i \le |\mathcal{J}| - \sigma\}$ and $\mathcal{J}_{DEQ} = \{\mathcal{J}_{\pi(i)} \mid i > |\mathcal{J}| - \sigma\}$, i.e. $\mathcal{J}_{DEQ}$ includes the σ jobs that are completed last, and $\mathcal{J}_{RR}$ includes the other $|\mathcal{J}| - \sigma$ jobs.

We define two notations - *t-suffix* and *t-preffix*, and use them to simplify the notations. For any time step t, *t-suffix* denoted as $\overrightarrow{t}$ represents the set of time steps from t to the completion of $\mathcal{J}$ by $\overrightarrow{t} = \{t, t+1, \ldots, T(\mathcal{J})\}$, while *t-preffix* denoted as $\overleftarrow{t}$ represents set of time steps from 1 to t by $\overleftarrow{t} = \{1, 2, \ldots, t\}$. We shall be interested in the suffixes of jobs. Define the t-suffix of a job $J_i \in \mathcal{J}$ to be the job $J_i(\overrightarrow{t})$, which is the portion of job J_i that remains after t - 1 number of time steps have been executed. The t-suffix of the job set $\mathcal{J}$ is

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

Thus, we have $\mathcal{J} = \mathcal{J}(\overrightarrow{1})$, and the number of uncompleted jobs at time step t is the number $|\mathcal{J}(\overrightarrow{t})|$ of nonempty jobs in $\mathcal{J}(\overrightarrow{t})$. Similarly, we can define the t-prefix of a job J_i as $J_i(\overleftarrow{t})$, and the t-prefix of a job set $\mathcal{J}$ as $\mathcal{J}(\overleftarrow{t})$.

Case 2 - Part (a)

The following two technical lemmas present the properties of round robin as a job scheduler. The first lemma shows that jobs make almost the same progress on the execution of their work when they are scheduled by RR. The second lemma relates the work of jobs to their completion time.

Lemma 8 *A batched job set $\mathcal{J}$ is scheduled by* GRAD *on a machine with P processors where $|\mathcal{J}| > P$. At any time step t scheduled by RR, for any two uncompleted jobs J_i and J_j, we have $|T_1(J_i(\overleftarrow{t})) - T_1(J_j(\overleftarrow{t}))| \le L$, where L is the length of the scheduling quantum.*

Proof. Since RR gives an equal share of processors to all uncompleted jobs, for any two jobs that arrive at the same time, their allotments differ by at most L at any time. When a job's allotment is 1, its allotted processor is always making useful work. Then the work done for any two uncompleted jobs differs by at most L at any time before their completion. □

Lemma 9 *A batched job set $\mathcal{J}$ is scheduled by* GRAD *on a machine with P processors where $|\mathcal{J}| > P$. The following two statements are true*:

1. *If $J_i \in \mathcal{J}_{RR}$, $J_j \in \mathcal{J}_{RR}$, and $T_1(J_i) < T_1(J_j)$, then $T(J_i) \le T(J_j)$.*
2. *If $J_i \in \mathcal{J}_{RR}$, and $J_j \in \mathcal{J}_{DEQ}$, then $T_1(J_i) \le T_1(J_j)$.*

Proof. We now prove the first statement. Let $t = T(J_i)$. At time step t, job J_i completes work $T_1(J_i)$. From Lemma 8, we know that $T_1(J_j(\overleftarrow{t})) \ge T_1(J_i(\overleftarrow{t})) - L = T_1(J_i) - L$. Since job J_j completes after job J_i, job J_j takes at least one more scheduling quantum than J_i to complete its execution. Thus the work done for J_j during the period from t to $T(J_j)$ is at least L. Therefore, we have $T_1(J_j) = T_1(J_i(\overleftarrow{T(J_j)})) \ge T_1(J_i(\overleftarrow{t})) + L \ge T_1(J_i)$.

For any two jobs $J_i \in \mathcal{J}_{RR}$, and $J_j \in \mathcal{J}_{DEQ}$, we have $T(J_i) < T(J_j)$. By using a similar analysis, we can prove the second statement.

Lemma 9 relates the work of jobs to their completion time. Its second statement tells us that only the σ jobs with largest work are scheduled by DEQ eventually, and the other $|\mathcal{J}|$ - σ jobs are scheduled by RR for their overall execution. Moreover, according to its first statement, under the schedule of RR, the jobs with less work are completed more quickly than those with more work. Consider the jobs according to their work such that $T_1(J_1) \le T_1(J_2) \le \ldots \le T_1(\mathcal{J}_{|\mathcal{J}|})$. From Lemma 9, we have $\mathcal{J}_{RR} = \{J_i \mid 1 \le i \le |\mathcal{J}| - \sigma\}$ and $\mathcal{J}_{DEQ} = \{J_i \mid i > |\mathcal{J}| - \sigma\}$.

Case 2 - Part (b)

The following lemma bounds the completion time of the jobs in $\mathcal{J}_{RR}$ where $T_1(J_i)$ denotes the work of a job J_i.

Lemma 10 GRAD *schedules a batched job set* $\mathcal{J}$ *on a machine with P processors where* $|\mathcal{J}| > P$. *Consider the jobs according to their work such that* $T_1(J_1) \le T_1(J_2) \le \ldots \le T_1(\mathcal{J}_{|\mathcal{J}|})$. *For* $1 \le i \le |\mathcal{J}| - \sigma$, *the completion time* $T(J_i)$ *of a job* J_i *is* $T(J_i) \le ((|\mathcal{J}| - i + 1)\, T_1(J_i) + \sum_{1 \le j < i} T_1(J_j))/P + L$.

Proof. Since we consider the jobs according to their work, from Lemma 9, we have $J_i \in J_{RR}$ where $1 \le i \le |\mathcal{J}| - \sigma$. Such a job J_i completes its overall execution under the schedule of RR as job scheduler.

We first evaluate $T_1(\mathcal{J}(\overleftarrow{t}))$, which is the work done for $\mathcal{J}$ up to a time step t. Suppose that the job J_i terminates at the end of a quantum q where $T(J_i) = q(L + 1) - 1$. Let $t = qL - 1$ be the end of the quantum q - 1, which is L steps before the completion of J_i. The work done for J_i in interval $\overleftarrow{t}$ is $T_1(J_i(\overleftarrow{t})) = T_1(J_i) - L$. According to Lemma 8, no job completes more than $T_1(J_i(\overleftarrow{t})) + L$ amount of work in interval $\overleftarrow{t}$. Therefore, for any job J_j with $j > i$, we have

$$\begin{aligned} T_1\left(J_j\left(\overleftarrow{t}\right)\right) &\le T_1\left(J_i\left(\overleftarrow{t}\right)\right) + L \\ &= T_1(J_i) \ . \end{aligned} \tag{7}$$

For each job J_j where $j < i$, by definition, we always have

$$T_1\left(J_j\left(\overleftarrow{t}\right)\right) \le T_1(J_j) \ . \tag{8}$$

Thus, at time step t, from Inequalities (7) and (8), the total work done for the job set $\mathcal{J}$ is

$$\begin{aligned} & T_1\left(\mathcal{J}\left(\overleftarrow{t}\right)\right) \\ &= \sum_{1 \le j < i} T_1\left(J_j\left(\overleftarrow{t}\right)\right) + T_1\left(J_i\left(\overleftarrow{t}\right)\right) + \sum_{i < j \le |\mathcal{J}|} T_1\left(J_j\left(\overleftarrow{t}\right)\right) \\ &\le (|\mathcal{J}| - i + 1) T_1(J_i) + \sum_{1 \le j < i} T_1(J_j) \ . \end{aligned} \tag{9}$$

Since RR always allots all processors to jobs, and all allotted processors are making useful work, RR executes P ready threads at any time step. Thus, the total work done for job set $\mathcal{J}$ increases by P at each time step. From Inequality (9), we have

$$\begin{aligned} t &= T_1\left(\mathcal{J}\left(\overleftarrow{t}\right)\right)/P \\ &\le \left((|\mathcal{J}| - i + 1)\, T_1(J_i) + \sum_{1 \le j < i} T_1(J_j)\right)/P \ . \end{aligned}$$

Since $T(J_i) = t + L$, we complete the proof. □

Case 2 - Part (c)
The following lemma bounds the total response time of job sets scheduled by GRAD when $|\mathcal{J}| > P$ where swa ($\mathcal{J}$) denotes squashed work area, and $T_\infty(\mathcal{J})$ denotes the aggregate span.
Lemma 11 Suppose that a job set $\mathcal{J}$ is scheduled by GRAD on a machine with P processors where $|\mathcal{J}| > P$. The response time R($\mathcal{J}$) of $\mathcal{J}$ is bounded by

$$\mathrm{R}(\mathcal{J}) = \left(2 - \tfrac{2}{|\mathcal{J}|+1}\right)\left(\tfrac{\rho+1}{\delta}\,\mathrm{swa}\,(\mathcal{J}) + \tfrac{2}{1-\delta}T_\infty(\mathcal{J})\right) + |\mathcal{J}|\,L + O(LP\log_\rho P)\ . \quad (10)$$

Proof. The jobs in $\mathcal{J}$ can be divided into RR set $\mathcal{J}_{RR}$ and DEQ set $\mathcal{J}_{DEQ}$. Let $n = |\mathcal{J}|$ denote the number of jobs in $\mathcal{J}$. Recall that σ denotes the number of jobs in $\mathcal{J}_{DEQ}$, i.e. $\sigma \le P$. Consider the jobs in the ascending order of their completion time such that $T(J_1) \le T(J_2) \le \ldots \preceq T(J_n)$. From Lemma 9, we have $\mathcal{J}_{RR} = \{J_i \mid 1 \le i \le n - \sigma\}$ and $\mathcal{J}_{DEQ} = \{J_i \mid i > n - \sigma\}$. We will calculate the total response time of the jobs in $\mathcal{J}_{RR}$ and $\mathcal{J}_{DEQ}$ respectively.
Step 1: To calculate R($\mathcal{J}_{RR}$), we apply Lemma 10. For any job $J_i \in \mathcal{J}_{RR}$, its completion time is $T(J_i) \le (1/P)((n - i+1)T_1(J_i)+\sum_{1 \le j < i} T_1(J_j))+L$ according to Lemma 10. Thus, the total response time of the jobs in $\mathcal{J}_{RR}$ is

$$\mathrm{R}(\mathcal{J}_{RR}) \le \tfrac{1}{P}\textstyle\sum_{1\le i\le n-\sigma}(2n-\sigma-2i+1)T_1(J_i) + Ln. \quad (11)$$

Step 2: We now calculate R($\mathcal{J}_{DEQ}$). The σ jobs in $\mathcal{J}_{DEQ}$ are scheduled by RR until the time step $t = T(J_{n-\sigma})$ at which the job $J_{n-\sigma}$ completes, and scheduled by DEQ afterwards. The total response time of $\mathcal{J}_{DEQ}$ is

$$\mathrm{R}\left(\mathcal{J}_{DEQ}\right) = \mathrm{R}\left(\mathcal{J}_{DEQ}\left(\overrightarrow{t+1}\right)\right) + \sigma \cdot t\ . \quad (12)$$

From Lemma 10, we know that the completion time of the job $J_{n-\sigma}$ is

$$t \le \left((\sigma+1)T_1(J_{n-\sigma}) + \textstyle\sum_{1\le i<n-\sigma} T_1(J_i)\right)/P + L\ . \quad (13)$$

To get R($\mathcal{J}_{DEQ}$), we only need to calculate R($\mathcal{J}_{DEQ}(\overrightarrow{t+1})$).
Since the job set $\mathcal{J}_{DEQ}$ is scheduled by DEQ as the job scheduler from time step t onwards, we can apply the total response time bound in Lemma 7 to calculate R($J_{DEQ}(\overrightarrow{t+1})$). During the interval $\overleftarrow{t}$, job $J_{n-\sigma}$ completes $T_1(J_{n-\sigma})$ amount of work. From Lemma 8, we know that each job J_i with $i > n - \sigma$ has completed at least $T_1(J_{n-\sigma}) - L$ amount of work. Thus, such a job J_i has remaining work $T_1(J_i(\overrightarrow{t+1}) \le T_1(J_i) - T_1(J_{n-\sigma}) + L$. The squashed work of $\mathcal{J}_{DEQ}(\overrightarrow{t+1})$ is

$$\begin{aligned}
&\mathrm{swa}\left(\mathcal{J}_{DEQ}\left(\overrightarrow{t+1}\right)\right)\\
&= \frac{1}{P}\,\mathrm{sq\text{-}sum}\left(\left\langle T_1\left(J_i\left(\overrightarrow{t+1}\right)\right) \mid n-\sigma+1 \le i \le n\right\rangle\right)\\
&\le \frac{1}{P}\,\mathrm{sq\text{-}sum}\left(\left\langle T_1(J_i) - T_1(J_{n-\sigma}) + L \mid n-\sigma+1 \le i \le n\right\rangle\right) \quad (14)\\
&= \tfrac{1}{P}\textstyle\sum_{n-\sigma+1\le i\le n}(n-i+1)\left(T_1(J_i) - T_1(J_{n-\sigma}) + L\right)\\
&\le \tfrac{1}{P}\textstyle\sum_{n-\sigma+1\le i\le n}(n-i+1)T_1(J_i) - \tfrac{(1+\sigma)\sigma}{2P}T_1(J_{n-\sigma}) + PL\ .
\end{aligned}$$

Let the constant $c = 2 - 2/(1 + P) < 2$. According to Lemma 7, we have

$$\mathrm{R}(\mathcal{J}_{DEQ}(\overrightarrow{t+1})) \le c \cdot \frac{\rho+1}{\delta} \operatorname{swa}\left(\mathcal{J}_{DEQ}(\overrightarrow{t+1})\right) + E_1 \ , \tag{15}$$

where $E_1 = c \cdot \frac{2}{1-\delta} T_\infty(\mathcal{J}) + cPL(\log_\rho P + 1)$.

We will now calculate the response time of $\mathcal{J}_{DEQ}$. Since we know $c = 2 - 2/(1 + P) > 1$, the responsiveness parameter $\rho > 1$, and the utilization parameter $\delta \le 1$, we have $c(\rho + 1)=\delta > 2$. Given Equation (12), and Inequalities (13), (14) and (15), the response time of $\mathcal{J}_{DEQ}$ is

$$\begin{aligned} & \mathrm{R}(\mathcal{J}_{DEQ}) \\ &= \mathrm{R}(\mathcal{J}_{DEQ}(\overrightarrow{t+1})) + \sigma \cdot t \\ &\le c \cdot \tfrac{\rho+1}{\delta} \operatorname{swa}\left(\mathcal{J}_{DEQ}(\overrightarrow{t+1})\right) + E_1 + \sigma \cdot t \\ &\le c \cdot \tfrac{\rho+1}{\delta P} \textstyle\sum_{n-\sigma+1 \le i \le n} (n-i+1) T_1(J_i) \\ &\quad + \tfrac{\sigma}{P} \textstyle\sum_{1 \le i < n-\sigma} T_1(J_i) + E_2 \ , \end{aligned} \tag{16}$$

where $E_2 = E_1 + (c \cdot \frac{\rho+1}{\delta} + 1)PL$.

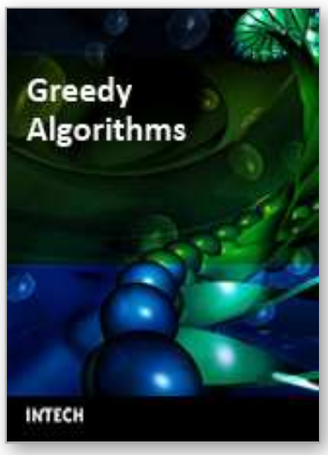
Step 3: Given $\mathrm{R}(\mathcal{J}_{RR})$ in Inequality (11), $\mathrm{R}(\mathcal{J}_{DEQ})$ in Inequality (16), and $c(\rho+1)=\delta > 2$, the response time of $\mathcal{J}$ is the sum of them as follows:

$$\begin{aligned} & \mathrm{R}(\mathcal{J}) \\ &= \mathrm{R}(\mathcal{J}_{RR}) + \mathrm{R}(\mathcal{J}_{DEQ}) \\ &< \frac{1}{P} \sum_{1 \le i \le n-\sigma} (2n - \sigma - 2i + 1) T_1(J_i) + Ln + \ c \cdot \frac{\rho+1}{\delta P} \sum_{n-\sigma+1 \le i \le n} (n-i+1) T_1(J_i) \\ &\quad + \frac{\sigma}{P} \sum_{1 \le i < n-\sigma} T_1(J_i) + E_2 \\ &= \frac{1}{P} \sum_{1 \le i \le n-\sigma} (2n - 2i + 1) T_1(J_i) + E_2 + Ln + \ c \cdot \frac{\rho+1}{\delta P} \sum_{n-\sigma+1 \le i \le n} (n-i+1) T_1(J_i) \\ &\quad + \frac{\sigma}{P} \sum_{1 \le i < n-\sigma} T_1(J_i) - \frac{\sigma}{P} \sum_{1 \le i \le n-\sigma} T_1(J_i) \\ &\le c \cdot \frac{\rho+1}{\delta P} \sum_{J_i \in \mathcal{J}} (n-i+1) T_1(J_i) + \ c \cdot \frac{2}{1-\delta} T_\infty(\mathcal{J}) + Ln + E_2 \\ &= \left(2 - \frac{2}{n+1}\right) \left(\frac{\rho+1}{\delta} \operatorname{swa}(\mathcal{J}) + \frac{2}{1-\delta} T_\infty(\mathcal{J})\right) + \ Ln + O(PL \log_\rho P) \ . \end{aligned}$$

□

Lemmas 7 and 11 bound the total response time of a batched job set $\mathcal{J}$ when $|\mathcal{J}| \leq P$ and $|\mathcal{J}| > P$ respectively. Combining them, we have completed the proof of Theorem 4.

□



Greedy Algorithms

Edited by Witold Bednorz

ISBN 978-953-7619-27-5
Hard cover, 586 pages
Publisher InTech
Published online 01, November, 2008
Published in print edition November, 2008

Each chapter comprises a separate study on some optimization problem giving both an introductory look into the theory the problem comes from and some new developments invented by author(s). Usually some elementary knowledge is assumed, yet all the required facts are quoted mostly in examples, remarks or theorems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yuxiong He, and Wen-Jing Hsu (2008). Provably-Efficient Online Adaptive Scheduling of Parallel Jobs Based on Simple Greedy Rules, Greedy Algorithms, Witold Bednorz (Ed.), ISBN: 978-953-7619-27-5, InTech, Available from: http://www.intechopen.com/books/greedy_algorithms/provably-efficient_online_adaptive_scheduling_of_parallel_jobs_based_on_simple_greedy_rules

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen