Competitive Two-Level Adaptive Scheduling Using Resource Augmentation

Hongyang Sun[1], Yangjie Cao[2], and Wen-Jing Hsu[1]

[1] School of Computer Engineering, Nanyang Technological University, Block N4, Nanyang Avenue, Singapore 639798
{sunh0007, hsu}@ntu.edu.sg
[2] School of Electronic and Information Engineering, Xi'an Jiaotong University, No.28, Xianning West Road, Xi'an, Shaanxi, 710049, P.R. China
caoyj@stu.xjtu.edu.cn

Abstract. As multi-core processors proliferate, it has become more important than ever to ensure efficient execution of parallel jobs on multiprocessor systems. In this paper, we study the problem of scheduling parallel jobs with arbitrary release time on multiprocessors while minimizing the jobs' mean response time. We focus on non-clairvoyant scheduling schemes that adaptively reallocate processors based on periodic feedbacks from the individual jobs. Since it is known that no deterministic non-clairvoyant algorithm is competitive for this problem, we focus on resource augmentation analysis, and show that two adaptive algorithms, AGDEQ and ABGDEQ, achieve competitive performance using $O(1)$ times faster processors than the adversary. These results are obtained through a general framework for analyzing the mean response time of any two-level adaptive scheduler. Our simulation results verify the effectiveness of AGDEQ and ABGDEQ by evaluating their performances over a wide range of workloads consisting of synthetic parallel jobs with different parallelism characteristics.

Key words: Malleable jobs, Mean response time, Non-clairvoyant algorithm, Online scheduling, Resource augmentation analysis, Two-level adaptive scheduling, Simulations

1 Introduction

With the proliferation of multi-core computers and the increasing use of multiprocessor systems, more software developers have started programming in parallel and migrating the existing sequential applications to the parallel platforms. One imminent challenge for the operating system is thus to schedule the parallel applications to fully exploit the multiprocessor resources.

In this paper, we study the problem of scheduling a set of parallel jobs with arbitrary release time on multiprocessors. The objective is to minimize the jobs' mean response time, where the response time of a job is defined to be the duration between its release and its completion. We consider *malleable* jobs [17] that have changing degrees of parallelism and can execute with a varying number

of allocated processors [12, 13, 18]. The task of the operating system scheduler is to decide the number of processors allocated to each job at any time. Since information about the jobs' characteristics is generally unavailable to the system, we assume that the schedulers are *online non-clairvoyant*, that is, they know nothing about the job's release time, work and future parallelism when making scheduling decisions.

To date, many excellent results for online scheduling have been obtained using *competitive analysis* [9], in which the performance of a scheduling algorithm is described in terms of its competitive ratio against the optimal scheduler. However, since it has been shown in [25] that any deterministic online non-clairvoyant algorithm is $\Omega(n^{1/3})$-competitive with respect to the mean response time even for scheduling sequential jobs on a single processor, some recent studies along this line have focused on *resource augmentation analysis* [21, 26], in which the online algorithm is augmented with extra resources as compared to the adversary, either in the form of faster processors or more processors. In this case, the online algorithm is said to be s-speed c-competitive if its performance with s times of extra resources is no worse than c times that of the optimal. The rationale for resource augmentation is that the traditional competitive analysis for an online algorithm can lead to a large competitive ratio because the online algorithm, being non-clairvoyant, cannot recover from sometimes even a small mistake made on certain worst-case job instances. The extra resources for the online algorithm compensates for their non-clairvoyance on these worst-case scenarios. Hence, if an algorithm achieves competitive result with moderate increase in processor resources, then it is likely to perform comparably to the optimal on most practical workloads. The readers may wish to refer to [21, 26, 28, 27] for more elaborate interpretations of resource augmentation. Basically, the goal is to achieve competitive performance for an algorithm with minimal extra resources.

Perhaps the simplest online non-clairvoyant scheduler for parallel jobs is EQUI (Equi-Partitioning)[14, 13], which divides the total number of processors evenly among all active jobs at any time. Using a sophisticated analysis, Edmonds [13] proved that EQUI is $(2+\epsilon)$-speed $O(1)$-competitive with respect to the mean response time of any set of jobs. Recently, Edmonds and Pruhs [16] proposed LAPS (Latest Arrival Processor Sharing), a variant of EQUI that divides the total number of processors among a certain portion of the latest released jobs. They showed that LAPS is $(1+\epsilon)$-speed $O(1)$-competitive with sufficiently large ϵ. The analysis of EQUI and LAPS employs a technique called *amortized local competitive argument*, which bounds the amortized performance of an algorithm at any local time through a carefully designed potential function, and it has become a useful technique for analyzing scheduling algorithms (see, e.g. [28, 27, 5, 4, 23, 11]). In this paper, we extend the amortized local competitive argument and provide a simple framework to analyze the mean response time for a set of perhaps less well-known but also quite effective schedulers called *two-level adaptive schedulers* [1, 18, 32].

The theoretical study of two-level adaptive schedulers was initiated by Agrawal, et al. [1]. Unlike EQUI, which allocates processors to jobs without considering

the jobs' utilization of the allocated resources, the two-level adaptive schedulers take a corrective approach by collecting statistics from jobs' *past* executions and using them to guide the future processor allocations. Since no knowledge about the future is assumed, they are also non-clairvoyant. Specifically, the scheduling of jobs by a two-level adaptive scheduler can be decomposed into two parts: at the system level, an *OS allocator* decides the processor allocations for jobs; at the job level, a *task scheduler* schedules the tasks of each job with the allocated processors. In order to allocate processors more effectively, each task scheduler provides feedback to the OS allocator indicating the job's future processor desire. The processors are reallocated periodically by the OS allocator after each *scheduling quantum* based on the feedback from the jobs. The length of the scheduling quantum is usually set to be long enough to amortize the overheads incurred by the processor reallocations and bookkeepings for scheduling, but it should not be too long to make the feedback relevant.

Using the above two-level adaptive scheduling framework, Agrawal et al. [1] proposed AG (Adaptive Greedy) task scheduler, which calculates the processor desire for a job in each scheduling quantum with a simple multiplicative-increase multiplicative-decrease strategy based on the execution statistics of the job in the immediate previous quantum. They analyzed the performance of AG in terms of an individual job's running time and processor utilization. He et al. [18] combined AG task scheduler with DEQ (Dynamic Equi-Partitioning) [33, 24] OS allocator, which is a variant of EQUI that never allocates more processors to a job than the job's processor desire. They called the resulting two-level scheduler AGDEQ, and showed that it is $O(1)$-competitive with respect to the mean response time of any set of *batched* parallel jobs (i.e., all jobs are released at time 0). Furthermore, they showed that AGDEQ simultaneously guarantees $O(1)$-competitiveness for the makespan of arbitrarily released jobs. Sun and Hsu [32] later proposed a task scheduler ABG (Adaptive B-Greedy), which directly utilizes the job's past parallelism to calculate the processor desire and improves upon AG in terms of its desire stability. They also showed similar mean response time and makespan performances for the two-level scheduler ABGDEQ.

In this paper, we show that, for parallel jobs with *arbitrary* release time, AGDEQ and ABGDEQ are competitive with respect to the mean response time with $O(1)$ times faster processors. Compared to EQUI, which is competitive for the mean response time [14, 13], but not competitive for the makespan [29], the results of AGDEQ and ABGDEQ show that the two-level adaptive schedulers achieve both fairness and efficiency for executing parallel applications on multiprocessor systems. Moreover, we provide a framework for analyzing the mean response time of any algorithm that can be formulated as a two-level adaptive scheduler. The analysis given in this paper and in [18] offers a convenient technique for analyzing the mean response time of a wide spectrum of scheduling algorithms that utilize parallelism feedbacks from the jobs, while the analysis in the previous results [13, 15, 29, 30, 16] are applied more specifically to EQUI and its variants.

In addition, we also conduct simulations and compare the mean response time of AGDEQ and ABGDEQ with that of EQUI. The simulation results verify the effectiveness of the two-level adaptive schedulers over a wide range of workloads consisting of synthetic parallel jobs with different parallelism characteristics.

The rest of this paper is organized as follows. Section 2 formally introduces the job model and the objective function. Section 3 discusses two-level adaptive scheduling in general, and describes AGDEQ and ABGDEQ algorithms in details. Section 4 provides the mean response time analysis framework, and its applications to AGDEQ and ABGDEQ. Our simulation results are presented in Section 5. Section 6 discusses some related work, and Section 7 concludes the paper.

2 Job Model and Objective Function

We adopt the job model used by Edmonds et al. [13, 16], which allows a parallel job to have time-varying parallelism modeled by multiple phases of speedup functions. However, unlike in [13, 16], where each phase of a job admits an arbitrary non-decreasing but sub-linear speedup, we consider a simpler model where each phase has a linear speedup function up to a certain degree of parallelism, beyond which no further speedup can be gained.[3] Specifically, we consider a set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of n jobs to be scheduled on P processors. Each job J_i in the job set contains p_i phases $\langle J_i^1, J_i^2, \ldots, J_i^{p_i}\rangle$, and each phase J_i^p has an amount of *work* w_i^p, and a linear *speedup function* Γ_i^p up to a certain degree of parallelism h_i^p, where $h_i^p \geq 1$. The *span* l_i^p of phase J_i^p is therefore $l_i^p = w_i^p / h_i^p$. The phase is *parallelizable* if $h_i^p = \infty$, and it is *sequential* if $h_i^p = 1$. The total work of job J_i is denoted by $w_i = \sum_{p=1}^{p_i} w_i^p$, and the total span of the job is $l_i = \sum_{p=1}^{p_i} l_i^p$. At any time t, suppose that job J_i is in its p-th phase and is allocated $a_i(t)$ processors of speed s, the effective speedup or execution rate of the job is thus given by $\Gamma_i^p(a_i(t)) = a_i(t) \cdot s$ if $a_i(t) \leq h_i^p$ and $\Gamma_i^p(a_i(t)) = h_i^p \cdot s$ if $a_i(t) > h_i^p$.

A scheduling algorithm ALG for any set $\mathcal{J}$ of jobs specifies the number $a_i(t)$ of processors allocated to each job J_i at any time t. In order for the schedule to be valid, we require that at any time t the total processor allocation is not more than the total number of processors, i.e., $\sum_{i=1}^{n} a_i(t) \leq P$. Let r_i denote the *release time* of job J_i. Let c_i^p denote the completion time of the p-th phase of job J_i, and let $c_i = c_i^{p_i}$ denote the *completion time* of job J_i. We also require that a valid schedule must complete all jobs in finite amount of time and can not begin to execute a phase of a job unless it has completed all its previous phases, i.e., $r_i = c_i^0 < c_i^1 < \ldots < c_i^{p_i} < \infty$, and $\int_{c_i^{p-1}}^{c_i^p} \Gamma_i^p(a_i(t))dt = w_i^p$ for all $1 \leq p \leq p_i$.

The job J_i is said to be *active* at time t if it is released but not completed at t, i.e., $r_i < t < c_i$. The *response time* R_i of the job is the duration between the completion time and the release time of the job, i.e., $R_i = c_i - r_i$. The *total response time* $R(\mathcal{J})$ of the entire job set $\mathcal{J}$ is thus given by $R(\mathcal{J}) = \sum_{i=1}^{n} R_i$, or alternatively can be expressed as $R(\mathcal{J}) = \int_0^\infty n_t dt$, where n_t is the number of

[3] See Section 7 for our discussion on the chosen job model.

active jobs at time t. The *mean response time* $\overline{R}(\mathcal{J})$ of the job set is therefore $\overline{R}(\mathcal{J}) = R(\mathcal{J})/n$.

Our objective is to minimize the mean response time $\overline{R}(\mathcal{J})$ of the job set $\mathcal{J}$, for which we use *resource augmentation analysis* [21, 26]. Specifically, we equip the online algorithm ALG with processors of speed s, where $s > 1$, while the optimal algorithm is only given processors of unit speed. In this case, ALG is said to be s-speed c-competitive with respect to the mean response time if there exists a constant b such that it satisfies $\overline{R}_{\text{ALG}}(\mathcal{J}) \leq c \cdot \overline{R}^*(\mathcal{J}) + b$ for any job set $\mathcal{J}$, where $\overline{R}^*(\mathcal{J})$ denotes the mean response time of the optimal scheduler.

Let $l(\mathcal{J})$ denote the total span of job set $\mathcal{J}$, i.e., $l(\mathcal{J}) = \sum_{i=1}^{n} l_i$. Then a simple lower bound for the total response time of job set $\mathcal{J}$ is its total span, that is, $R^*(\mathcal{J}) \geq l(\mathcal{J})$, since it takes at least l_i time to complete job J_i using any scheduler on unit-speed processors.

3 Two-Level Adaptive Scheduling: from AGDEQ to ABGDEQ and beyond

In this section, we present how two-level adaptive scheduling can be used to schedule parallel jobs. We first introduce the basic concept of two-level adaptive scheduling, followed by detailed descriptions of two specific two-level algorithms, namely AGDEQ and ABGDEQ. We end this section with a remark on the general applicability of two-level adaptive schedulers.

3.1 Two-Level Adaptive Scheduling

In two-level adaptive scheduling, the execution of jobs is decomposed into two parts: a system-level *OS allocator* decides the processor allocations for jobs; and a *task scheduler* for each job in the user level executes the job with the allocated processors. The processors are reallocated by the OS allocator after each *scheduling quantum*, which is usually set to be a fixed amount of time, say L time units. In order for the OS allocator to allocate processors to jobs more effectively, the task scheduler should provide feedback to the OS allocator indicating the job's processor *desire* for each quantum, typically based on the execution statistics of the job in previous quanta. For a scheduling quantum q, let $d_i(q)$ and $a_i(q)$ denote the processor desire and the processor allocation for job J_i, respectively. We assume that the task scheduler always executes the job based on the model given in Section 2. Hence, as far as the task schedulers are concerned, their only difference lies in the strategies for estimating the processor desires. In [19], the interaction between task scheduler and the OS allocator is referred to as the *processor request-allocation protocol*. At each quantum in this protocol, we say that a job J_i is *satisfied* if its processor allocation is no less than its processor desire, i.e., $a_i(q) \geq d_i(q)$. Otherwise, the job is *deprived* if $a_i(q) < d_i(q)$. In addition, the notions of satisfied and deprived are extended from quantum to time. A job is said to be satisfied (deprived) at time t if t is

within a satisfied (deprived) quantum for the job. Finally, to ease our analysis, when a new job is released in the middle of a quantum, it is not scheduled until the beginning of the next quantum.

3.2 AGDEQ

AGDEQ (Adaptive-Greedy Dynamic Equi-Partitioning) is a two-level adaptive scheduler proposed by He et al. [18] that combines the task scheduler AG [1] and the OS allocator DEQ [33, 24]. In this subsection, we will describe the two parts in detail.

The task scheduler AG estimates the processor desire for a job in a scheduling quantum based on the job's execution characteristics in the immediate previous quantum. Specifically, let t_q denote the time when quantum q starts, then the work $w_i(q)$ completed for job J_i in quantum q is given by $w_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{p_t}(a_i(q))dt$, where L is the length of the scheduling quantum and p_t is the phase job J_i is executing at time t. The execution of job J_i is said to be *efficient* in quantum q if the work $w_i(q)$ completed is at least δ fraction of the maximum amount of work that can be done in the quantum, i.e., $w_i(q) \geq \delta a_i(q)sL$, where $0 < \delta < 1$ is called the *utilization threshold*; otherwise it is *inefficient* if $w_i(q) < \delta a_i(q)sL$. Based on the efficient and inefficient classification as well as the satisfied and deprived classification for quantum q, the processor desire for the next quantum $q+1$ is calculated using a multiplicative-increase multiplicative-decrease strategy as shown in Algorithm 1, assuming that the OS allocator never allocates more processors than its desire.[4]

Algorithm 1 AG(δ)

1: **if** $w_i(q) < \delta a_i(q)sL$ **then**
2: $\quad d_i(q+1) = d_i(q)/2$ //inefficient
3: **else if** $a_i(q) = d_i(q)$ **then**
4: $\quad d_i(q+1) = d_i(q) \cdot 2$ //efficient and satisfied
5: **else**
6: $\quad d_i(q+1) = d_i(q)$ //efficient and deprived

The rationale of the AG algorithm is as follows [1]. If the allocated processors in quantum q are not utilized efficiently, then the parallelism of the job may not be as high. Therefore, the processor desire will be reduced by a factor of 2 for the next quantum $q+1$ (line 1 and line 2). If the allocated processors are utilized

[4] In this paper, we simplify AG algorithm by setting its multiplicative factor to 2, while in [1], a tuning parameter ρ is defined that can take on any value greater than 1. The simplification is justified by the fact that the multiplicative factor is mainly related to the processor waste of the job in deductible quanta [1], which we show in this paper is actually irrelevant to the jobs' mean response time, provided that the initial processor desire is set sufficiently high.

efficiently and the processor desire is satisfied, then the parallelism of the job could be even higher. Thus, the processor desire will be increased by a factor of 2 (line 3 and line 4). Lastly, if the allocated processors are utilized efficiently but the desire is deprived, then it is not known whether the processors could still be efficiently utilized had the desire been satisfied. Therefore, the processor desire is not changed for the next quantum (line 5 and line 6). The processor desire for the initial quantum when the job is first scheduled is set to be the total number P of processors.[5] Following the terminologies in [1], a job is referred to as *accounted* in quantum q if the job is both deprived and efficient. Otherwise, the job is *deductible*.

Now, we describe the OS allocator DEQ, which is a variants of EQUI that partitions the total number of processors equally among all active jobs. In DEQ, however, if a job desires for less processors than the equal share, it will not be allocated more processors than its desire, and the remaining processors will instead be given to the other jobs with higher desires. Let $\mathcal{J}(t)$ denote the set of active jobs at time t when a new quantum begins. Based on the processor desires from all jobs in $\mathcal{J}(t)$, DEQ allocates the processors as shown in Algorithm 2.

Algorithm 2 DEQ$(\mathcal{J}(t), P)$

```
1: if J(t) = ∅ then
2:    return
3: S = {J_i ∈ J(t) : d_i(t) ≤ P/|J(t)|}
4: if S = ∅ then
5:    for each J_i ∈ J(t) do
6:       a_i(t) = P/|J(t)|//deprived jobs get current equal share
7:    return
8: else
9:    for each J_i ∈ S do
10:      a_i(t) = d_i(t)//satisfied jobs get their desires
11:   DEQ(J(t) − S, P − Σ_{J_i∈S} a_i(t))
```

As can be seen in the pseudocode, if a job's processor desire is not more than the equal share $P/|\mathcal{J}(t)|$ of processors, the job will be satisfied (line 3 and line 10). After that, the equal share will be recalculated excluding the jobs already satisfied and the processors already allocated. The remaining processors will then be allocated to the rest of the jobs by recursively calling the main

[5] Note that in [1], the initial desire is set to be 1. This more conservative strategy is to ensure that jobs do not waste a lot of processors, especially in deductible quanta, which intuitively could affect the mean response time of the job set, since the wasted processors of a job could have been well utilized by the other jobs to speed up their executions [18]. However, we show in this paper that the mean response time of the jobs is actually independent of their deductible waste. This phenomenon can also be observed in the analysis of EQUI, which fares poorly in terms of its processor utilization, yet it still achieves good performance in terms of mean response time [13].

procedure (line 11) until all jobs' processor desires are satisfied or they exceed the equal share. In the latter case, each remaining job will be deprived and get the current equal share of processors (lines 4-7). As was shown in [12, 18], if there are deprived jobs for a quantum, then all P processors must have been allocated by DEQ, and each deprived job will have the same number of allocated processors, which is higher than the initial equal share $P/|\mathcal{J}(t)|$. Note that in this paper, as in [12, 13, 18, 16], we assume that the number of processors allocated to a job can be non-integral. The fractional allocation can be considered as time-sharing a processor among the jobs.

3.3 ABGDEQ

ABGDEQ (Adaptive B-Greedy Dynamic Equi-Partitioning) was proposed by Sun and Hsu [32] and it combines OS allocator DEQ with task scheduler ABG, which directly calculates the average parallelism of the job in a quantum, and uses it as the processor desire for the next quantum. Specifically, let t_q denote the time when quantum q starts, then the work $w_i(q)$ completed for job J_i in quantum q is $w_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{p_t}(a_i(q))dt$, and the span $l_i(q)$ reduced for job J_i in quantum q is $l_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{p_t}(a_i(q))/h_i^{p_t}dt$, where L is the length of the scheduling quantum and p_t is the phase job J_i is executing at time t. Although the instantaneous parallelism of job J_i at any time during quantum q may vary, its average parallelism $A_i(q)$ in the quantum is given by $A_i(q) = w_i(q)/l_i(q)$. ABG directly sets the processor desire for quantum $q+1$ to the average parallelism of quantum q, i.e., $d_i(q+1) = A_i(q)$.[6] This strategy makes the processor desire more representative of the job's average processor requirement, and eliminates the desire instability problem of AG when the parallelism of the job stays constant for sufficiently long time (see Section 5). Again, the initial desire is set to be the total number P of processors. In addition, job J_i is said to be *under-allocated* in quantum q if the average parallelism is at least the processor allocation, i.e., $A_i(q) \geq a_i(q)$, otherwise it is *over-allocated* if $A_i(q) < a_i(q)$. Following the terminologies from AG, a job is *accounted* if it is both deprived and under-allocated, and otherwise it is *deductible*.

3.4 Remark

Beyond AGDEQ and ABGDEQ, the general concept of two-level adaptive scheduling can represent a rich class of scheduling algorithms with various other feedback mechanisms and allocation policies. For instance, EQUI can be considered as a special type of two-level adaptive scheduler with variable quantum length (a quantum only expires if a job completes or a new job is released) and an

[6] In [32], the processor desire for quantum $q+1$ is set to be a linear combination of the average parallelism and the processor desire of quantum q, i.e., $d_i(q+1) = (1-v)A_i(q) + vd_i(q)$, where v is called the *convergence rate*. In this paper, we set the convergence rate to be $v = 0$, hence make the processor desire achieve one-quantum convergence towards the job's average parallelism when the latter is constant.

oblivious OS allocator (which always divides the processors equally among the active jobs regardless of each job's processor desire). In the following section, we will present a general framework for analyzing the mean response time on this class of schedulers.

4 Mean Response Time Analysis

In this section, we present a general framework for the mean response time analysis of the two-level adaptive schedulers, and apply it to AGDEQ and ABGDEQ algorithms. We begin this section by introducing a few concepts and notations.

4.1 Preliminaries

At any time t, a job J_i executed by the online algorithm can be characterized by certain properties. For example, a job can be classified according to "satisfied", "deprived", "accounted", or "deductible" as described in the preceding section. Our analysis relies on identifying two such properties A and B for the set of active jobs. Let $\mathcal{J}(t)$ denote the set of active jobs at time t, and let $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the sets of active jobs at time t that satisfy property A and property B, respectively. Throughout the execution of job J_i, let $a_A(J_i)$ denote the total processor allocation when job J_i satisfies property A, i.e., $a_A(J_i) = \int_0^\infty a_i(t)s \cdot [J_i(t) \in \mathcal{J}_A(t)]dt$, and let $t_B(J_i)$ denote the total amount of time when job J_i satisfies property B, i.e., $t_B(J_i) = \int_0^\infty s \cdot [J_i(t) \in \mathcal{J}_B(t)]dt$, where s denotes the processor speed of the online algorithm and $[x]$ is 1 if proposition x is true and 0 otherwise. In addition, we also require the notion of total amount of time for the entire job set $\mathcal{J}$ that satisfies property B, which is defined to be $t_B(\mathcal{J}) = \sum_{i=1}^{n} t_B(J_i)$. To simplify our notations, we let $n_t = |\mathcal{J}(t)|$ denote the number of active jobs at time t, and let $n_t^A = |\mathcal{J}_A(t)|$ and $n_t^B = |\mathcal{J}_B(t)|$ denote the number of active jobs at time t that satisfy property A and property B, respectively. As far as this paper is concerned, the two properties A and B are chosen such that $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ are disjoint, i.e., $\mathcal{J}_A(t) \bigcap \mathcal{J}_B(t) = \emptyset$, and they cover the whole set of active jobs, i.e., $\mathcal{J}_A(t) \bigcup \mathcal{J}_B(t) = \mathcal{J}(t)$. Hence, we have $n_t^A + n_t^B = n_t$.

We also introduce the notions of *t-prefix* and *t-suffix* for jobs to ease our analysis. For an online algorithm, define the t-prefix $J_i(\overleftarrow{t})$ of job J_i to be the portion of the job executed before time t, and the t-suffix $J_i(\overrightarrow{t})$ to be the portion executed after time t. Specifically, if the online algorithm is executing the p-th phase of job J_i at time t, then $J_i(\overleftarrow{t})$ consists of the first $p-1$ phases $\langle J_i^1, J_i^2, \ldots, J_i^{p-1} \rangle$ of job J_i, followed by part of the p-th phase with work $\int_{c_i^{p-1}}^{t} \Gamma_i^p(a_i(t))dt$; and $J_i(\overrightarrow{t})$ begins with the rest of the p-th phase with work $\int_t^{c_i^p} \Gamma_i^p(a_i(t))dt$, followed by the remaining phases $\langle J_i^{p+1}, J_i^{p+2}, \ldots, J_i^{p_i} \rangle$ of job J_i. In addition, we extend the definitions of t-prefix and t-suffix from a job to a job set such that $\mathcal{J}(\overleftarrow{t}) = \{J_1(\overleftarrow{t}), J_2(\overleftarrow{t}), \ldots, J_{n_t}(\overleftarrow{t})\}$ and $\mathcal{J}(\overrightarrow{t}) = \{J_1(\overrightarrow{t}), J_2(\overrightarrow{t}), \ldots, J_{n_t}(\overrightarrow{t})\}$. We let $\mathcal{J}^*(\overleftarrow{t}) = \{J_1^*(\overleftarrow{t}), J_2^*(\overleftarrow{t}), \ldots, J_{n_t^*}^*(\overleftarrow{t})\}$

and $\mathcal{J}^*(\overrightarrow{t}) = \{J_1^*(\overrightarrow{t}), J_2^*(\overrightarrow{t}), \ldots, J_{n_t^*}^*(\overrightarrow{t})\}$ denote the t-prefix and t-suffix of job set $\mathcal{J}$ executed by the optimal scheduler, respectively, where n_t^* is the number of active jobs at time t under the optimal scheduler.

4.2 Analysis Framework

Our analysis framework adopts the *amortized local competitive argument* [27], which bounds the amortized performance of an online algorithm at any time through a potential function. In addition, we also extend the *two-step analysis* used in [18] for bounding the mean response time of batched parallel jobs (i.e., when all jobs arrive at time 0). In the case of two-level adaptive schedulers, we first analyze the task scheduler by bounding two specific properties of it on an individual job. We then apply the amortized local competitive argument to the OS allocator. Finally, by combining the analysis of the task scheduler and the OS allocator, we can obtain the mean response time performance of the two-level algorithm. Specifically, our analysis develops as follows.

Step (1): For the task scheduler, choose two properties A and B. Then bound the processor allocation $a_A(J_i)$ to any job J_i in terms of the total work w_i of the job, as well as the amount of time $t_B(J_i)$ for any job J_i in terms of the total span l_i of the job, on processors of any speed s, where $s > 0$. That is, find coefficients γ_1, γ_2 and constant γ_3 such that

$$a_A(J_i) \le \gamma_1 \cdot w_i \ , \tag{1}$$

$$t_B(J_i) \le \gamma_2 \cdot l_i + \gamma_3 \ . \tag{2}$$

Step (2): For the OS allocator, with properties A and B chosen in Step (1) in mind, find a potential function $\Phi(t)$, a processor speed s', and coefficients c_1 and c_2 such that on processors of speed $s = s' + \epsilon$ for any $\epsilon > 0$, the execution of the job set satisfies the following

- *Boundary Condition*: $\Phi(0) = 0$ and $\Phi(\infty) \ge 0$;
- *Arrival Condition*: $\Phi(t)$ does not increase when new jobs arrive;
- *Completion Condition*: $\Phi(t)$ does not increase when jobs complete;
- *Running Condition*:

$$\frac{dR(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \le c_1 \cdot \frac{dR^*(\mathcal{J}^*(t))}{dt} + c_2 \cdot \frac{dt_B(\mathcal{J}(t))}{dt} \ , \tag{3}$$

where $\frac{dR(\mathcal{J}(t))}{dt} = \lim_{\Delta t \to 0} \frac{R(\mathcal{J}(\overleftarrow{t+\Delta t})) - R(\mathcal{J}(\overleftarrow{t}))}{dt}$ denotes the change of total response time under the online algorithm in an infinitesimal interval of time Δt, and apparently we have $\frac{dR(\mathcal{J}(t))}{dt} = n_t$. Similarly, the change of total response time under the optimal algorithm satisfies $\frac{dR^*(\mathcal{J}^*(t))}{dt} = n_t^*$, and the change of total amount of time for the job set satisfying property B under the online algorithm is given by $\frac{dt_B(\mathcal{J}(t))}{dt} = sn_t^B$. In addition, $\frac{d\Phi(t)}{dt} = \lim_{\Delta t \to 0} \frac{\Phi(t+\Delta t) - \Phi(t)}{dt}$ denotes the change of potential function in interval Δt. Note that we assume Δt is infinitesimally small such that no new job arrives, and no job completes, or

makes a transition between two phases, or experiences processors reallocation under both the online algorithm and the optimal.

The form of the potential function $\Phi(t)$ may not be unique, but it usually depends on the processor allocation $a_A(J_i)$ and the coefficient γ_1 given in Inequality (1). In addition, coefficient γ_1 can also affect the choice of processor speed s' for Inequality (3) to be satisfied. In the next subsection, we will provide a concrete example on choosing the potential function and the processor speed for the analysis of the OS allocator DEQ. Now, combining the results of Step (1) and Step (2), we can obtain the performance for the two-level algorithm. First, summing over all jobs for Inequality (2), we get $t_B(\mathcal{J}) \le \gamma_2 \cdot l(\mathcal{J}) + \gamma_3 n$. Since $l(\mathcal{J})$ is a lower bound on the total response time of job set $\mathcal{J}$, integrating Inequality (3) over time, we have

$$\begin{aligned} R(\mathcal{J}) &\le c_1 \cdot R^*(\mathcal{J}) + c_2 \cdot t_B(\mathcal{J}) \\ &\le c_1 \cdot R^*(\mathcal{J}) + c_2\gamma_2 \cdot l(\mathcal{J}) + c_2\gamma_3 n \\ &\le (c_1 + c_2\gamma_2) \cdot R^*(\mathcal{J}) + c_2\gamma_3 n \ . \end{aligned}$$

The mean response time $\overline{R}(\mathcal{J})$ thus satisfies $\overline{R}(\mathcal{J}) \le (c_1 + c_2\gamma_2) \cdot \overline{R}^*(\mathcal{J}) + c_2\gamma_3$, which suggests that the two-level algorithm is $(s' + \epsilon)$-speed $(c_1 + c_2\gamma_2)$-competitive with respect to the mean response time for any job set $\mathcal{J}$, provided that c_2 and γ_3 are constants.

4.3 Performance of AGDEQ

We now apply the framework outlined in the preceding subsection to analyze the mean response time of two-level adaptive scheduler AGDEQ. We choose property A and property B for AGDEQ to be "accounted" and "deductible", respectively.

We first focus on the task scheduler AG, whose total accounted allocation and the total deductible time have been bounded in [1] on unit-speed processors. Using the same argument [1], we can show similar results for AG on speed s processors. The following lemma gives the performance bounds.

Lemma 1. *Suppose that* AG *schedules a job J_i with work w_i and span l_i on speed s processors. Then the total accounted allocation to the job satisfies $a_A(J_i) \le w_i/\delta$, and the total deductible time of the job satisfies $t_B(J_i) \le 2l_i/(1-\delta) + 2L$, where $\delta < 1$ is* AG*'s utilization threshold and L is the length of the scheduling quantum.*

Proof sketch. The total accounted allocation $a_A(J_i)$ to job J_i can be directly inferred from the definition of accounted quantum. Specifically, since an accounted quantum q for job J_i is also efficient, we have $a_i(q)sL \le w_i(q)/\delta$. Let AC denote the set of accounted quanta for the job. The total accounted allocation thus satisfies $a_A(J_i) = \sum_{q \in AC} a_i(q)sL \le \sum_{q \in AC} w_i(q)/\delta \le w_i/\delta$.

The total deductible time $t_B(J_i)$ of job J_i can be bounded by considering the total inefficient time and the total efficient-and-satisfied time, separately. The former is no more than $l_i/(1-\delta)$ by considering the reduction of the job's span

in each inefficient quantum. The latter can be related to the former, because there exists a correspondence between the set of inefficient quanta and the set of efficient-and-satisfied quanta due to the multiplicative-increase multiplicative-decrease strategy [3]. By setting the initial processor desire to P, the total efficient-and-satisfied time turns out to be no more than $l_i/(1-\delta)+L$. The total deductible time of the job is thus bounded by summing up the two terms as well as the additional waiting time of the job after its arrival, which is at most the quantum length L. □

We now turn to the analysis of the OS allocator DEQ using amortized local competitive argument. We adopt the potential function used by Lam et al. [23] in the context of online speed scaling and tailor it to suit the mean response time analysis of two-level adaptive schedulers. Specifically, at any time t, let $n_t(z)$ denote the number of jobs whose remaining accounted allocation is at least $\gamma_1 z$ under AGDEQ, i.e., $n_t(z) = \sum_{i=1}^{n_t}[a_A(J_i(\overrightarrow{t})) \geq \gamma_1 z]$, and let $n_t^*(z)$ denote the number of jobs whose remaining work is at least z under the optimal, i.e., $n_t^*(z) = \sum_{i=1}^{n_t}[w(J_i^*(\overrightarrow{t})) \geq z]$. Apparently, $n_t(z)$ and $n_t^*(z)$ are staircase-like decreasing functions of z, and Figure 1(a) shows an example of $n_t(z)$ and $n_t^*(z)$ at a given time t. The potential function is defined to be

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i \right) - n_t(z) n_t^*(z) \right] dz \ , \tag{4}$$

where $\eta = \frac{2\gamma_1}{\epsilon P}$. For convenience, define $\phi_t(z) = \left(\sum_{i=1}^{n_t(z)} i\right) - n_t(z)n_t^*(z)$. We can now check the four conditions for Step (2) of the analysis framework given in the preceding subsection.

- Boundary Condition: at time 0, no job exists in the system. The terms $n_t(z)$ and $n_t^*(z)$ are both 0 for all z. Therefore, we have $\Phi(0) = 0$. At time ∞, no job remains in the system, so again we have $\Phi(\infty) = 0$. Hence, the boundary condition holds.

- Arrival Condition: suppose that a new job with work w' arrives at time t. Let t^- and t^+ denote the instances right before and after the job arrives. Hence, we have $n_{t^+}^*(z) = n_{t^-}^*(z) + 1$ for $z \leq w'$ and $n_{t^+}^*(z) = n_{t^-}^*(z)$ for $z > w'$. Similarly, $n_{t^+}(z) = n_{t^-}(z) + 1$ for $z \leq a'/\gamma_1$ and $n_{t^+}(z) = n_{t^-}(z)$ for $z > a'/\gamma_1$, where a' is the total accounted allocation to the job. Figure 1(b) illustrates the changes of $n_t(z)$ and $n_t^*(z)$ in this case. Note that $a'/\gamma_1 \leq w'$ from Step (1) of the analysis. Thus, it is obvious that for $z > w'$, we have $\phi_{t^+}(z) = \phi_{t^-}(z)$. For $z \leq w'$, we consider two cases.

Case 1: for $z \leq a'/\gamma_2$, we have $\phi_{t^+}(z) - \phi_{t^-}(z) = \left(\sum_{i=1}^{n_{t^-}(z)+1} i\right) -$
$(n_{t^-}(z)+1)\left(n_{t^-}^*(z)+1\right) - \left(\sum_{i=1}^{n_{t^-}(z)} i\right) + n_{t^-}(z)n_{t^-}^*(z) = -n_{t^-}^*(z) \leq 0.$

Case 2: for $a'/\gamma_2 \leq z \leq w'$, we have $\phi_{t^+}(z) - \phi_{t^-}(z) = \left(\sum_{i=1}^{n_{t^-}(z)} i\right) -$
$n_{t^-}(z)\left(n_{t^-}^*(z)+1\right) - \left(\sum_{i=1}^{n_{t^-}(z)} i\right) + n_{t^-}(z)n_{t^-}^*(z) = -n_{t^-}(z) \leq 0.$

Hence, $\Phi(t^+) = \eta \int_0^\infty \phi_{t^+}(z)dz \leq \eta \int_0^\infty \phi_{t^-}(z)dz = \Phi(t^-)$, and the arrival condition holds.

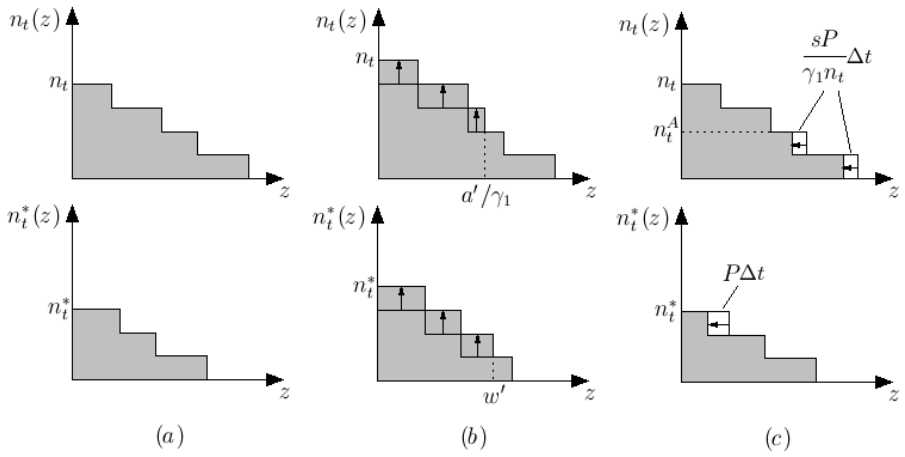

Fig. 1. (a) An example of $n_t(z)$ and $n_t^*(z)$ at a given time t. (b) The changes of $n_t(z)$ and $n_t^*(z)$ after a new job arrives. (c) The changes of $n_t(z)$ and $n_t^*(z)$ in an infinitesimally small interval of time Δt.

- Completion Condition: when a job completes under AGDEQ or the optimal algorithm, the potential function $\Phi(t)$ remains unchanged, because in such cases, $n_t(z)$ or $n_t^*(z)$ only reduces by 1 for $z = 0$. Hence, the completion condition also holds.

At this point, the first three conditions hold true regardless of the OS allocator used and the properties A and B chosen. It remains to check the running condition, which typically depends on the specific OS allocator as well as the properties A and B. The following lemma shows the running condition of DEQ with property A and property B being "accounted" and "deductible", respectively.

Lemma 2. *Suppose that* DEQ *schedules a job set $\mathcal{J}$ on speed s processors with* AG*. Then the running condition in Inequality (3) is satisfied with potential function in Equation (4), processor speed $s = 2\gamma_1 + \epsilon$, and coefficients $c_1 = 2s/\epsilon$ and $c_2 = 2/\epsilon$, provided that property A and property B are chosen to be "accounted" and "deductible", respectively.*

Proof. As mentioned in Section 3.2, DEQ ensures that accounted jobs, which are deprived by definition, get at least P/n_t processors at any time t. In the worst case, these n_t^A accounted jobs at time t have the most remaining accounted processor allocation among the n_t active jobs, while the optimal scheduler executes the job with the least remaining work using all P processors. As a result, which can be seen from Figure 1(c), each of the bottom n_t^A horizontal stripes of $n_t(z)$ shrinks by $sP\Delta t/(\gamma_1 n_t)$, and the top horizontal stripe of $n_t^*(z)$ shrinks by $P\Delta t$

in interval Δt. The change of the potential function can then be bounded by

$$\begin{aligned}
\frac{d\Phi(t)}{dt} &= \frac{\eta}{\Delta t}\int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i\right) - n_{t+\Delta t}(z)n^*_{t+\Delta t}(z) - \left(\sum_{i=1}^{n_t(z)} i\right) + n_t(z)n^*_t(z)\right] dz \\
&\le \frac{\eta}{\Delta t}\int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i\right) - \left(\sum_{i=1}^{n_t(z)} i\right)\right] dz \\
&\quad + \frac{\eta}{\Delta t}\int_0^\infty \left[n_t(z)\left(n^*_t(z) - n^*_{t+\Delta t}(z)\right) + n^*_t(z)\left(n_t(z) - n_{t+\Delta t}(z)\right)\right] dz \\
&\le \frac{2\gamma_1}{\epsilon P\Delta t}\left(-\frac{n^A_t(n^A_t+1)}{2}\cdot\frac{sP}{\gamma_1 n_t}\Delta t + n_t P\Delta t + n^*_t\frac{sPn^A_t}{\gamma_1 n_t}\Delta t\right) \\
&\le \frac{2\gamma_1}{\epsilon}\left(1 - \frac{x_t^2 s}{2\gamma_1}\right)n_t + \frac{2s}{\epsilon}n^*_t\,, \qquad (5)
\end{aligned}$$

where $x_t = n^A_t/n_t$, and $0 \le x_t \le 1$. Since a job is either accounted or deductible, we have $n^B_t = (1-x_t)n_t$. It can be easily verified that the running condition holds for all values of x_t by substituting Inequality (5), $s = 2\gamma_1 + \epsilon$, $c_1 = 2s/\epsilon$ and $c_2 = 2/\epsilon$ into Inequality (3). □

Now, we can establish the mean response time performance of two-level adaptive scheduler AGDEQ in the following theorem.

Theorem 1. AGDEQ *is* $(\frac{2}{\delta}+\epsilon)$*-speed* $\left(2 + \frac{4}{\delta(1-\delta)\epsilon}\right)$*-competitive with respect to the mean response time of any job set, where* $\delta < 1$ *is* AG*'s utilization threshold.*

Proof. The theorem follows by combining the analysis given in Section 4.2 and the results of task schedulers AG in Lemma 1 and the result of OS allocator DEQ in Lemma 2. □

4.4 Performance of ABGDEQ

In this subsection, we show the mean response time of two-level adaptive scheduler ABGDEQ. Again, we choose property A and property B to be "accounted" and "deductible", respectively.

The performance of task scheduler ABG relies on a certain characteristic of the job, which is called *transition factor* in [32] and denoted as C_L for a given quantum length L. Roughly speaking, the transition factor of a job characterizes how fast the job's parallelism changes with time in the worst case, and hence reflects the degree of difficulty to schedule it in an adaptive fashion. The following lemma bounds the performance of ABG on speed s processors. The proof follows closely that of Lemma 1 and can be found in [32].

Lemma 3. *Suppose that* ABG *schedules a job* J_i *with work* w_i *and span* l_i *on speed* s *processors. Then the total accounted allocation to the job satisfies*

$a_A(J_i) \leq 2w_i$, and the total deductible time of the job satisfies $t_B(J_i) \leq (C_L + 1)l_i + 2L$, where C_L is the transition factor of the job and L is the length of the scheduling quantum. □

The following theorem gives the mean response time performance of ABGDEQ.

Theorem 2. ABGDEQ *is $(4+\epsilon)$-speed $\left(2 + \frac{10+2C_L}{\epsilon}\right)$-competitive with respect to the mean response time of any job set, where $C_L \geq 1$ is the maximum transition factor of the jobs in the job set.*

Proof. Since we can apply the analysis of DEQ to ABGDEQ as well, combining the results of Lemma 3 and Lemma 2, the theorem follows. □

4.5 Discussions

As suggested in Section 3.4, we can formulate EQUI as a two-level adaptive scheduler, where EQUI serves as the OS allocator itself that interacts with an arbitrary task scheduler using variable quantum length. To analyze its mean response time, we choose properties A and B as follows. At any time t, a job J_i satisfies property A if its processor allocation is no more than the maximum parallelism of the phase the job is currently executing, i.e., $a_i(t) \leq h_i^{p_t}$. Otherwise, the job satisfies property B if $a_i(t) > h_i^{p_t}$. In this case, we can easily show that the coefficients γ_1 and γ_2 are both equal to 1, which when combined with a similar analysis in Lemma 2, can lead to a matching mean response time performance of $(2+\epsilon)$-speed $(2 + \frac{6}{\epsilon})$-competitiveness as obtained in [13]. This demonstrates the generality of two-level adaptive scheduling as well as the usefulness of our analysis framework.

It is worth noting that the extra resources required by AGDEQ and ABGDEQ as well as their competitive ratios are more than that of EQUI, which implies that the two-level adaptive schedulers have inferior mean response time performance in the worst case. The same phenomenon can be observed when comparing the competitive ratios of AGDEQ, ABGDEQ and EQUI for scheduling batched parallel jobs [14, 18, 32]. The reason is because two-level adaptive schedulers only utilize the history of the job to generate feedbacks and we assume that the job's future parallelism need not be correlated to its past. Hence, in the worst case, the adversary can always make the future parallelism of the job deviate from its processor desire, e.g., by forcing the job to have high parallelism when its processor desire is low or vice versa. Thus, compared to EQUI, the OS allocator of AGDEQ and ABGDEQ can be tricked into making poorer decisions, resulting in worse processor distributions.

In practice, however, the worst-case scenario is not likely to occur. Therefore, we expect that AGDEQ and ABGDEQ should perform comparably to or even better than EQUI, especially when the parallelism of the job does not change frequently, hence the correlation between the future parallelism and the past can be well exploited by the adaptive strategies of AGDEQ and ABGDEQ. Moreover, the practical performances of the two-level adaptive schedulers may also depend

upon the specific parallelism characteristics of the jobs, the length of the scheduling quantum selected and the amount of system overhead incurred, etc., which are omitted in the theoretical analysis. We will evaluate the impacts of these factors in the next section by carrying out simulations.

5 Simulations

In this section, we conduct simulations on two-level adaptive schedulers AGDEQ and ABGDEQ using synthetic parallel jobs with various parallelism characteristics. To better understand adaptive scheduling, we first study how task schedulers AG and ABG respond to these parallelism characteristics in terms of their processor desire estimation. We then focus on the mean response time of AGDEQ and ABGDEQ by comparing them with EQUI on various workloads and by studying the impacts of different quantum length and system overhead. Since resource augmentation employed in the previous sections only serves as an analysis tool for deriving the performance bound of an algorithm, for a fair comparison, we assign unit-speed processors to all algorithms in our simulations instead of giving them different extra-speed processors.

5.1 Synthetic Parallel Jobs

We construct synthetic parallel jobs with different types of parallelism characteristics. Specifically, we identify five distinct parallelism variation curves, which are specified by Step, Impulse, Ramp, Poly(I) and Poly(II) functions, respectively, and they describe precisely how the parallelism changes with time. Figure 2(a) shows these five parallelism variation curves with each one containing the same underlining work and span, hence the same average parallelism. Among them, the Step function can represent part of a data-parallel job that contains constant and stable parallelism. The Impulse function, with drastically increased parallelism after a sequential phase, can approximate part of an irregular parallel job containing transient and spiky parallelism profile. The Ramp, Poly(I) and Poly(II) functions, which are constructed by polynomials of degree 1, 3 and 1/3 respectively in our simulation, can represent parts of a parallel job whose parallelism increases at different levels of steepness. Since the exact parallelism characteristics of the actual applications are generally unknown, we believe that these types of parallelism variation curves can represent a wide range of parallelism structures, which are useful for evaluating scheduling algorithms.

Besides the increasing parallelism curves as shown in Figure 2(a), we can also have parallelism structures specified by the corresponding decreasing curves, which together with the increasing curves form the basic building blocks for our parallel job construction. In our simulations, each *block* contains a pair of increasing and decreasing curves of the same type with the average parallelism chosen uniformly from 1 to 200 and the length fixed at 250. In addition, to study the impacts of different parallelism variations on scheduling algorithms, we only generate homogeneous jobs, where each job contains over 500 blocks of

the same type interconnected by sequential phases with the same length. Note that the concept of a block used here should be distinguished from that of a phase introduced in Section 2. While a block describes the parallelism structure of a job over a period of time, a phase refers to a segment of the job in which the parallelism is constant.

5.2 Transient Response

To better understand the behavior of two-level adaptive scheduling algorithms, we first focus on task schedulers AG and ABG in this subsection by studying their *transient response* to different parallelism variation curves.[7]

Figures 2(b)-2(e) demonstrate the transient response of AG and ABG on four parallelism variations given in the previous subsection. (The response of the Impulse function is similar to that of the Step function and is not shown.) The length of the scheduling quantum is set to 100, which is scaled in the figure to restore the original parallelism variation. We assume that the desires for both schedulers start and end at a steady state with value of 1, and are satisfied by the OS allocator at all time. As shown in these figures, both adaptive schedulers can efficiently adjust the processor desires based on the parallelism changes, although AG and ABG exhibit different transient responses with respect to different parallelism variations. For the Step function, AG is able to gradually catch up with the parallelism change but suffers from desire instability even when the parallelism remains constant. In contrast, ABG rapidly approaches the parallelism within a quantum, and thereafter provides stable desires by directly utilizing the average parallelism of the job. For the other functions, both AG and ABG respond gradually to the parallelism variations with ABG in general following more closely the changes of the parallelism and thus taking shorter time to reach steady state. This is due to ABG's more effective processor desire calculation.

To confirm the quality of feedbacks observed in the transient responses, we also measure the response time of AG and ABG on five parallel jobs with the same average parallelism but different parallelism variations. Figure 2(f) shows the performances of AG and ABG on each of the five jobs in terms of the job's response time normalized by its span. We can see clearly that ABG indeed outperforms AG for all parallelism variations. This is especially true on the Step and Impulse functions, where ABG shows clear advantage over AG with more stable and efficient feedbacks.

5.3 Mean Response Time

In this subsection, we study the mean response time of two-level adaptive schedulers AGDEQ and ABGDEQ. We simulate a system with 1000 processors, and

[7] The term "transient response" stems from electrical/control engineering, and often refers to the response of a system to changes in the input signal from a steady state. In this case, we use transient response to describe how a task scheduler responds to different parallelism variations in terms of its processor desire estimation.

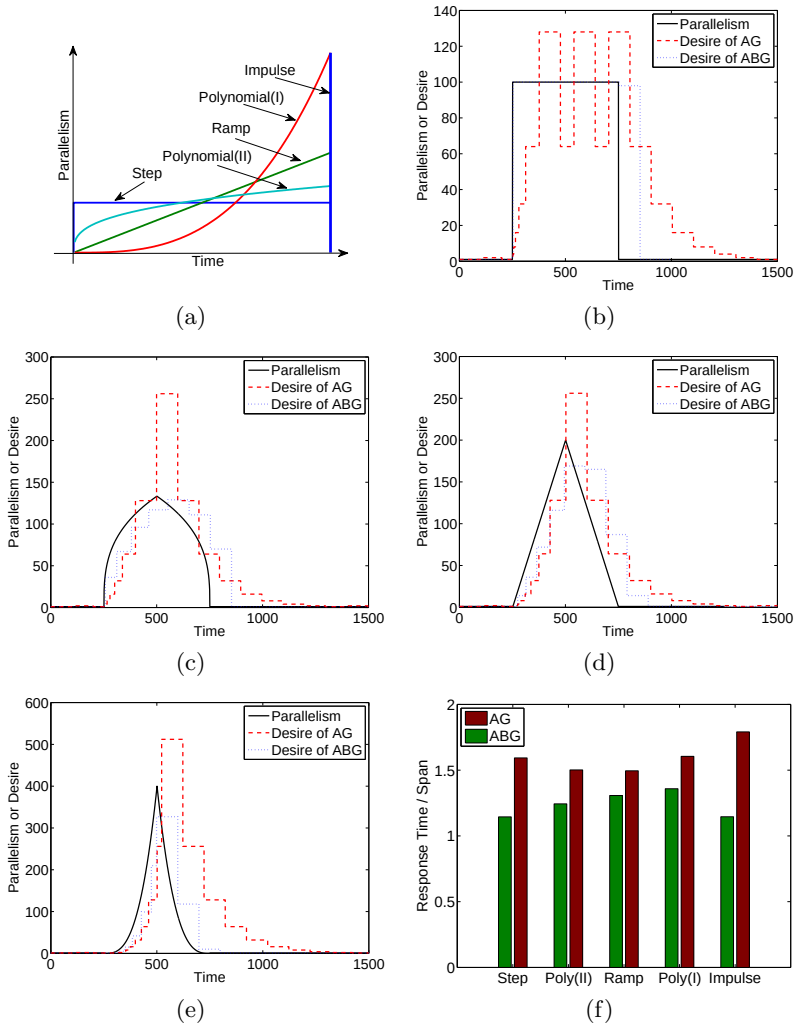

Fig. 2. (a) Five different parallelism variation curves, represented by the Step, Poly(II), Ramp, Poly(I) and Impulse functions. (b)-(e) Transient responses of AG and ABG on the Step, Poly(II), Ramp, and Poly(I) functions. (f) The response time ratios of AG and ABG on five parallel jobs with same average parallelism but different parallelism variations.

generate a wide range of workloads by varying the number of jobs and their parallelism variations. In each experiment, jobs are released according to the Poisson process within the span of the first arrived job so that all jobs would be released before any could complete. Hence, the load of the system will increase with the number of jobs used for each experiment. As with [19], we define the load of the system to be the sum of the average parallelism of all jobs normalized by the total number of processors. In our simulations, the number of jobs is varied from 1 to 100 for each parallelism variation. The mean response time of AGDEQ and ABGDEQ are compared to that of EQUI. In addition, we also study the impacts of quantum length and system overhead on the performances of the two-level adaptive schedulers.

(1) Performance comparison.

As can be observed in Figure 3, AGDEQ and ABGDEQ generally achieve better performances than EQUI on jobs with all parallelism variations. When the system has light workload with a small number of jobs, EQUI performs better because in this case all jobs can be easily satisfied on the given processors. With increased workload, however, both AGDEQ and ABGDEQ outperform EQUI, and eventually tend to converge to EQUI at extremely heavy workloads, where each job gets very few processors most of the time and hence the advantage of adaptive scheduling is diminished. This suggests that two-level adaptive scheduling is more effective under moderate workloads with many parallel jobs competing for but not overwhelmed by the limited processor resources. Moreover, Figure 3 also shows that the performance of ABGDEQ is always better than that of AGDEQ, which is again due to task scheduler ABG's more effective processor desire feedbacks.

(2) Impact of parallelism variations.

The impact of different parallelism variations on the performances of AGDEQ, ABGDEQ, and EQUI are shown in Figure 3(f), which gives the average performances of the three algorithms over our entire workload range in terms of their mean response time normalized by the theoretical lower bound. Roughly speaking, the performances of all three algorithms are closely related to the degree at which the parallelism varies. Specifically, the Impulse function contains the most drastic parallelism variation and therefore incurs the worst performance for all algorithms. The other functions present better performances for the algorithms with smoother parallelism variations. Furthermore, we can also see that the performance of ABGDEQ is relatively insensitive to different parallelism variations, while EQUI is affected the most as the parallelism variations change from Step to Impulse.

(3) Impacts of quantum length and overhead.

In the preceding simulations, we fixed the quantum length to 100 and ignored the system overhead. However, in two-level adaptive scheduling, the length of the scheduling quantum is an important system parameter, which together with the scheduling overhead can significantly affect the mean response time performance. To better understand their impacts, we conduct a set of simulations by changing the quantum length and the scheduling overhead. Specifically, the

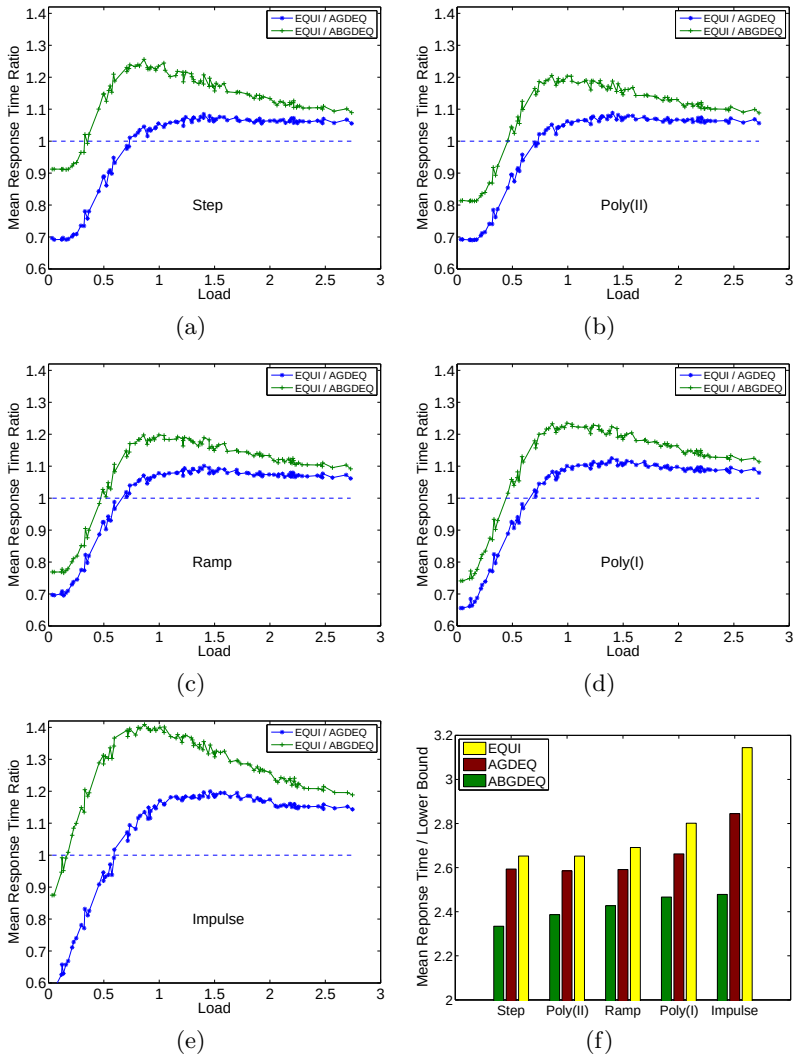

Fig. 3. (a)-(e) Mean response time ratios of EQUI over AGDEQ and EQUI over ABGDEQ on different workloads and parallel jobs with Step, Poly(II), Ramp, Poly(I), and Impulse parallelism curves. (f) Average mean response time normalized by the theoretical lower bound for EQUI, AGDEQ and ABGDEQ over the entire workload range on jobs with the five parallelism variations.

quantum length is varied from 50 to 600 in steps of 50. The scheduling overhead is changed from zero (i.e., OH= 0%) up to 36% in terms of the smallest quantum length 50 at an increment of 4% each time. Figure 4 shows the simulation results on medium to heavy workloads using jobs whose parallelism variation follows the Step function. Similar outcomes are observed using jobs with the other types of parallelism variations. We can see that as the scheduling overhead increases, both AGDEQ and ABGDEQ have significantly worse performances when the quantum length is small, while the impact of the overhead becomes less severe with larger quantum length. When the quantum length is large enough, the performances of AGDEQ and ABGDEQ become generally stable and are slightly worse than that of EQUI, which is hardly affected by the overhead. However, when the system has relatively small overhead, both adaptive schedulers do outperform EQUI with suitably chosen quantum length. In this sense, two-level adaptive schedulers are quite sensitive to the length of scheduling quantum and the amount of system overhead. Hence, when implementing these algorithms on different platforms, attentions should be paid to choosing an appropriate quantum length based on the scheduling overhead of the system in order to offer desirable performances.

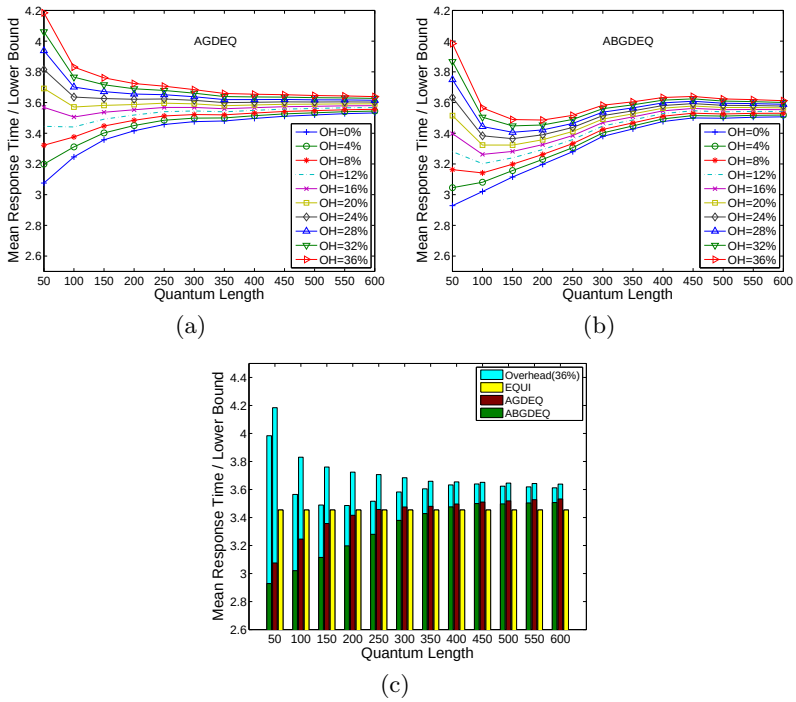

Fig. 4. (a)-(b) Impacts of different quantum length and scheduling overhead on the performances of AGDEQ and ABGDEQ. (c) Performance comparison among EQUI, AGDEQ, and ABGDEQ on medium to heavy workloads with and without overhead (36%) for different quantum length.

6 Related Work

The problem of scheduling a set of fully parallelizable jobs on multiprocessors is equivalent to scheduling sequential jobs on a single processor. For the latter problem, Motwani et al. [25] showed that, for batched jobs, RR (Round Robin) is $(2 - 2/(n + 1))$-competitive with respect to the mean response time. When jobs can have arbitrary release time, however, they showed that every deterministic non-clairvoyant algorithm is $\Omega(n^{1/3})$-competitive and every randomized non-clairvoyant algorithm is $\Omega(\log n)$-competitive. Using resource augmentation analysis, Kalyanasundaram and Pruhs [21] proved that the deterministic non-clairvoyant algorithm SETF (Shortest Elapsed Time First) is $(1 + \epsilon)$-speed $(1 + 1/\epsilon)$-competitive, which was later improved by Berman and Coulston [7] to $2/s$ when $s \geq 2$. Kalyanasundaram and Pruhs [22] also showed that the randomized non-clairvoyant algorithm RMLF (Randomized Multi-Level Feedback) is $O(\log n \log \log n)$-competitive against an adaptive adversary. Becchetti and Leonardi [6] improved the competitive ratio of RMLF to $O(\log n)$ when the adversary is oblivious, hence matching the lower bound in this case. In addition, it is well-known that the clairvoyant algorithm SRPT (Shortest Remaining Processing Time) is optimal for this problem [10].

For parallel jobs with changing execution characteristics, Edmonds [13] proved that EQUI is $(2 + \epsilon)$-speed $O(1)$-competitive with respect to the mean response time of the jobs. Edmonds and Pruhs [16] recently proposed LAPS (Latest Arrival Processor Sharing), which in a sense combines EQUI and SETF, and proved that it is $(1 + \epsilon)$-speed $O(1)$-competitive for sufficiently large ϵ, hence achieving *almost fully scalable* [28, 27], i.e., the least possible extra resources required to be competitive. For the relatively easier case, where jobs are released in a batched fashion, Edmonds et al. [14] showed that EQUI is $(2 + \sqrt{3})$-competitive. Deng et al. [12] showed that DEQ with jobs' instantaneous parallelism as feedback is 2-competitive for parallel jobs with single phase and 4-competitive for multiple-phase jobs. The latter ratio was recently improved to 3 by He et al. [20]. In addition, Edmonds et al. [15] also extended the analysis of EQUI to the TCP protocol in Internet congestion control. Robert and Schabanel [29, 30] applied variations of EQUI to other job models and objective functions.

For two-level adaptive schedulers, modeling a parallel job as a directed acyclic graph (dag), Agrawal et al. [1, 3] proposed two algorithms, namely AG (Adaptive Greedy) and AS (Adaptive Work-Stealing), which are based on centralized scheduling and distributed work stealing, respectively. They proved that AG and AS achieve nearly linear speedup and waste a relatively small number of processor cycles for each individual job. He et al. [18] later combined task schedulers AG and AS with the OS allocator DEQ to form two-level schedulers. They proved that the resulting algorithms AGDEQ and ASDEQ are both $O(1)$-competitive with respect to the mean response time for batched parallel jobs. In addition, He et al. [19] also showed that when the system is heavily-loaded, the two-level algorithms can be coupled with RR to achieve similar results. Observing that AG can cause unstable processor desires although the parallelism of the job is constant, Sun and Hsu [32] proposed ABG (Adaptive B-Greedy) task scheduler, which

guarantees stability of the processor desires along with other control-theoretic properties. They also proved the mean response time of batched parallel jobs for the two-level scheduler ABGDEQ in terms of the jobs' parallelism transition.

Several empirical studies on two-level adaptive scheduling are also known in the literature. Sen [31] presented experimental results on a dynamic desire estimation algorithm for the Cilk work-stealing scheduler [8], which inspired the research presented in [1, 3]. Agrawal, He and Leiserson [2] compared ASDEQ with EQUI through simulations, and confirmed that the former has superior performance. He, Hsu and Leiserson [19] evaluated the performance AGDEQ under a wide range of workloads, and revealed that it actually performs much better in practice than predicted by the theoretical bounds. Sun and Hsu [32], also through simulations, confirmed that ABGDEQ does improve upon AGDEQ for batched parallel jobs.

7 Conclusion

In this paper, we have analyzed the mean response time of two-level adaptive schedulers AGDEQ and ABGDEQ on parallel jobs with arbitrary release time and changing degrees of parallelism. We have shown through a general analysis framework that both AGDEQ and ABGDEQ are competitive with respect to the mean response time using $O(1)$ times faster processors. In addition, we have also conducted simulations over a wide range of workloads using parallel jobs with different parallelism variations. The simulation results have verified the effectiveness of AGDEQ and ABGDEQ with appropriately chosen quantum length.

Compared to the job model used in this paper, Edmonds et al. [13, 16] have assumed a more general model, in which each phase of a job can admit an arbitrary non-decreasing and sub-linear speedup. However, to analyze the mean response time of EQUI and LAPS, Edmonds et al. reduced any set $\mathcal{J}$ of jobs with non-decreasing and sub-linear speedups to a set $\mathcal{J}'$ of jobs that consist of only fully parallelizable and strongly sequential phases, where a phase is *fully parallelizable* if its speedup function Γ satisfies $\Gamma(a) = a$ for all $a \geq 0$ and it is *strongly sequential* if $\Gamma(a) = 1$ for all $a \geq 0$. One implicit assumption used in this reduction is that the online algorithm is not able to distinguish a newly constructed phase from the original phase because it is non-clairvoyant. Thus, the same number of processors will be allocated to $\mathcal{J}'$ and $\mathcal{J}$ at any time. However, such reduction does not directly apply to the type of adaptive schedulers considered in this paper because their future processor allocations do depend on the past parallelism of the jobs. It will be interesting to see if similar reductions are possible for these adaptive schedulers.

Another problem with existing task schedulers AG and ABG is that they both require comprehensive statistics about a job's execution in the current quantum in order to estimate its processor desire for the next quantum. Collecting such statistics can be difficult in real systems, and even if possible, might incur significant overheads, which as have been shown in our simulations can have adverse effect on the system performance. It will be useful to design task schedulers

that use incomplete information about a job's execution (e.g., obtained through samplings) to estimate its processor desires while still guaranteeing desirable performances.

Acknowledgments

The first author would like to thank Yuxiong He for many discussions and much help on two-level adaptive scheduling.

References

[1] K. Agrawal, Y. He, W.-J. Hsu, and C. E. Leiserson. Adaptive scheduling with parallelism feedback. In *PPoPP*, 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 *ICDCS*, pages 19 – 29, Lisbon, Portugal, 2006.
[3] K. Agrawal, Y. He, and C. E. Leiserson. Adaptive work stealing with parallelism feedback. In *PPoPP*, pages 112–120, San Jose, CA, USA, 2007.
[4] N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. In *ICALP*, pages 409–420, Reykjavik, Iceland, 2008.
[5] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *SODA*, pages 805–813, New Orleans, LA, USA, 2007.
[6] L. Becchetti and S. Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *Journal of the ACM*, 51(4):517–539, 2004.
[7] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, 6(2):181–193, 1999.
[8] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.
[9] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.
[10] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., 2001.
[11] H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 409–420, Freiburg, Germany, 2009.
[12] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167, Philadelphia, PA, USA, 1996.
[13] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, Atlanta, GA, USA, 1999.
[14] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. In *STOC*, pages 120–129, El Paso, TX, USA, 1997.
[15] J. Edmonds, S. Datta, and P. Dymond. TCP is competitive against a limited adversary. In *SPAA*, pages 174–183, San Diego, CA, USA, 2003.
[16] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, pages 685–692, New York, NY, USA, 2009.
[17] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems (extended version). *IBM Research Report RC19790(87657) 2nd Revision*, 1997.
[18] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, Saint-Malo, France, 2006.

[19] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient online non-clairvoyant adaptive scheduling. In *IPDPS*, pages 1–10, Long Beach, CA, USA, 2007.

[20] Y. He, H. Sun, and W.-J. Hsu. Adaptive scheduling of parallel jobs on functionally heterogeneous resources. In *ICPP*, Xi'an, China, 2007.

[21] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *FOCS*, pages 214–221, Milwaukee, WI, USA, 1995.

[22] B. Kalyanasundaram and K. Pruhs. Minimizing flow time nonclairvoyantly. In *FOCS*, page 345, Miami Beach, FL, USA, 1997.

[23] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *ESA*, pages 647–659, Karlsruhe, Germany, 2008.

[24] 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.

[25] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *SODA*, pages 422–431, Austin, TX, USA, 1993.

[26] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *STOC*, pages 140–149, El Paso, TX, USA, 1997.

[27] K. Pruhs. Competitive online scheduling for server systems. *ACM SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[28] K. Pruhs, E. Torong, and J. Sgall. Online scheduling. In handbook of scheduling: Algorithms, models, and performance analysis, chapter 15, CRC Press. 2004.

[29] J. Robert and N. Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *ESA*, pages 741–753, 2007.

[30] J. Robert and N. Schabanel. Non-clairvoyant scheduling with precedence constraints. In *SODA*, pages 491–500, San Francisco, CA, USA, 2008.

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

[32] H. Sun and W.-J. Hsu. Adaptive B-Greedy (ABG): A simple yet efficient scheduling algorithm. In *SMTPS in conjunction with IPDPS*, pages 1–8, Miami, FL, USA, 2008.

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