Fair and Efficient Online Adaptive Scheduling for Multiple Sets of Parallel Applications

Hongyang Sun*, Yangjie Cao†, Wen-Jing Hsu*
*School of Computer Engineering, Nanyang Technological University, Singapore
†School of Electronic and Information Engineering, Xi'an Jiaotong University, China
{sunh0007, hsu}@ntu.edu.sg; caoyj@stu.xjtu.edu.cn

***Abstract*—Both fairness and efficiency are crucial measures for the performance of parallel applications on multiprocessor systems. In this paper, we study online adaptive scheduling for multiple sets of such applications, where each set may contain one or more jobs with time-varying parallelism profile. This scenario arises naturally when dealing with several applications submitted simultaneously by different users in a large parallel system, where both user-level fairness and system-wide efficiency are important concerns. To achieve fairness, we use the equi-partitioning algorithm, which evenly splits the available processors among the active job sets at any time. For efficiency, we apply a feedback-driven adaptive scheduler, which periodically adjusts the processor allocations within each set by consciously exploiting the jobs' execution history. We show that our algorithm is competitive for the objective of minimizing the set response time. For sufficiently large jobs, this theoretical result improves upon an existing algorithm that provides only fairness but lacks efficiency. Furthermore, we conduct simulations to empirically evaluate our algorithm, and the results confirm its improved performance using malleable workloads consisting of a wide range of parallelism variation structures.**

***Index Terms*—Adaptive scheduling; Online algorithms; Feedback-driven scheduling; Parallel applications; Multiprocessor systems; Set response time; Fairness; Efficiency**

I. Introduction

Scheduling parallel applications on multiprocessor systems has been a fundamental area of research in computer science for decades [8], [11], [14]. Recently, as more parallel systems have been deployed to support high-performance computing infrastructures, such as various cloud computing services in large clusters and data centers, efficient scheduling on these platforms will play an even more important role in boosting application performance and increasing system utilization.

Since many parallel systems nowadays are shared by multiple users, a natural scenario arises when each user simultaneously submits several applications to the system. An crucial scheduling goal in this scenario is to achieve efficient execution for the applications while at the same time offering a level of fairness among different users. In this paper, we consider such a scenario, in which a collection of parallel job sets needs to be scheduled on a multiprocessor system and each job set corresponds to the set of applications submitted by a particular user of the system. We are interested in the *response time* of a job set, which is defined to be the duration between when the job set is submitted and when all jobs in the job set are completed. The objective is to minimize the overall response time of all job sets, or the *set response time*.

As pointed out by Robert and Schabanel [15], the metric of set response time benchmarks both fairness and efficiency of a scheduling algorithm. In fact, it represents a more general performance measure that incorporates two widely used metrics, namely *total response time* and *makespan*, as special cases. Suppose that each job set in the collection contains only a single job, the set response time becomes the total response time of all jobs in the collection. At the other extreme, if the collection contains only a single job set, the set response time is simply the makespan. To schedule a collection of job sets, an algorithm needs to allocate processors at both the *job-set level* and the *job level*. That is, it needs to specify the number of processors allocated to each job set, as well as the processor allocation for each job within the job set.

We adopt the *online adaptive* scheduling model [9], which requires an algorithm to make scheduling decisions in an online manner without any knowledge of the jobs' future characteristics, such as their release time and remaining work, etc. This is a natural assumption since such information is indeed generally not available to the operating system schedulers. However, the online scheduler is allowed to be adaptive, i.e., it can dynamically adjust the jobs' processor allocations at runtime. This is in contrast to the traditional *static scheduling* [11], which restricts the processor allocation of a job to be constant throughout its execution. Since modern parallel applications often exhibit irregular and time-varying parallelism structures, static scheduling may either under-utilize system resources or cause job execution delays. With flexible runtime support [17], [18], *adaptive scheduling* is able to benefit from the malleable behavior of the jobs' processor requirements and hence appears to be a more promising approach to scheduling modern parallel applications. The performance of an online adaptive scheduler is measured using *competitive analysis* [2], which compares the online algorithm with an optimal offline scheduler.

A well-known online adaptive scheduler is Equi-partitioning (EQUI) [22], which at any time divides the total available processor resources equally among all jobs present in the system. This algorithm, although simple, is able to ensure fairness by automatically adjusting the processor allocations whenever a new job is admitted into the system or an existing job is completed and thus leaves the system. In fact, such simple notion of fairness is sufficient to guarantee satisfying performance when each user submits only one job. In particular, Edmonds et al. [7] showed that EQUI is $(2+\sqrt{3})$-competitive with respect to the total response time of all jobs if they are released at the same time. Using resource augmentation analysis [10], Edmonds [6] also showed that EQUI achieves $O(1)$-competitive for arbitrarily released jobs when it is augmented with $O(1)$ times more resources than the optimal. However,

despite its excellent performance for the total response time, EQUI fares poorly in terms of the makespan, which to a certain extent reflects the system efficiency when there is only one user in the system. Since EQUI does not consider how efficiently each job is able to utilize the allocated processors, it may under-utilize the system resources particularly when different jobs can have very different processor requirements. In [15], Robert and Schabanel showed that EQUI is $\Theta(\frac{\ln n}{\ln \ln n})$-competitive with respect to the makespan even if all jobs are batch released, where n is the total number of jobs in the system.

To schedule a collection of job sets, both user-level fairness and system-wide efficiency turn out to be critical. In [15], Robert and Schabanel applied EQUI to both scheduling levels by equally dividing the total available processors among all active job sets and within each job set equally dividing the allocated processors to its active jobs. They showed that the resulting algorithm EQUI∘EQUI achieves a competitive ratio of $(2+\sqrt{3}+o(1))\frac{\ln n}{\ln \ln n}$ with respect to the set response time when all job sets are batch released, where n is the maximum number of jobs in a set. This result suggests that the set response time ratio of a scheduling algorithm actually combines the total response time ratio and the makespan ratio of the corresponding algorithms at the job-set level and the job level, respectively. Hence, it is important to retain both fairness and efficiency in order to achieve satisfying performance for this more general scheduling metric.

To improve application efficiency, feedback-driven adaptive schedulers [1], [9], [20] were recently introduced. Unlike EQUI, which obliviously allocates processors to jobs regardless of their actual resource requirements, feedback-driven schedulers periodically adjust processors among the jobs by consciously exploiting the jobs' execution history. In particular, Agrawal et al. [1] introduced the A-GREEDY scheduler that periodically collects the resource utilization of each job, and based on this information estimates the job's future processor requirement. It has been shown that A-GREEDY wastes at most a constant fraction of a job's allocated processors, and thus indeed achieves efficient processor utilization [1]. Furthermore, by combining A-GREEDY with a conservative resource allocator, such as Dynamic Equi-partitioning (DEQ) [12], He et al. [9] showed that the feedback-driven algorithm AGDEQ achieves $O(1)$-competitive with respect to the makespan regardless of the number of jobs, provided that the jobs under consideration are sufficiently large. Recently, Sun et al. [20] proposed another adaptive scheduler ACDEQ, which uses a control-theoretical approach to estimate the jobs' processor requirements and it has been shown to improve upon AGDEQ in terms of both feedback stability and system efficiency.

In this paper, aiming at both user-level fairness and system-wide efficiency, we bring together the benefit of the EQUI algorithm and that of the feedback-driven algorithm AGDEQ, and propose a fair and efficient online adaptive scheduler EQUI∘AGDEQ for any collection of job sets. Using competitive analysis, we show that the set response time ratio of our algorithm indeed combines the total response time ratio and the makespan ratio of the respective algorithms in a non-trivial manner. Specifically, we prove that EQUI∘AGDEQ possesses the following desirable properties:

- EQUI∘AGDEQ achieves $O(1)$-competitive with respect to the set response time when all job sets are batch released. This result improves the competitive ratio $\Theta(\frac{\ln n}{\ln \ln n})$ of EQUI∘EQUI [15] for sufficiently large jobs, where n is the maximum number of jobs in a set.
- EQUI∘AGDEQ achieves $O(1)$-speed $O(1)$-competitive with respect to the set response time for arbitrarily released job sets. This result extends the same ratio obtained in [16] from jobs with specific parallelism structure to sufficiently large jobs with any parallelism profile.

Furthermore, we also conduct simulations to empirically evaluate the performance of EQUI∘AGDEQ, and to compare it with EQUI∘EQUI and another algorithm EQUI∘ACDEQ which combines EQUI with the feedback-driven scheduler ACDEQ. The simulations are carried out using *malleable* jobs [8] generated from Downey's parallel workload model [5] and augmented with a wide range of internal parallelism structures [3]. The results confirm that the algorithms which utilize feedback-driven schedulers indeed achieve better performances than EQUI∘EQUI in terms of set response time as well as processor utilization. The improvement is a direct result that these algorithms exhibit both fairness and efficiency while EQUI∘EQUI provides only fairness but lacks efficiency.

The rest of this paper is organized as follows. Section II formally defines the scheduling problem. Section III describes the EQUI∘AGDEQ algorithm, followed by its analysis for both batched and arbitrarily released job sets in Section IV. The simulation results are presented in Section V. Finally, Section VI concludes the paper.

II. PROBLEM DEFINITIONS

A. Job Model and Scheduling Model

We adopt the job model used in [19], [20] to represent a parallel application that consists of a series of phases with different degrees of parallelism. Specifically, we consider a collection $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \cdots, \mathcal{J}_m\}$ of m job sets, which correspond to m different users. Each job set $\mathcal{J}_i = \{J_{i1}, J_{i2}, \cdots, J_{in_i}\}$ contains n_i jobs, which correspond to the applications submitted by the i-th user. Each job $J_{ij} = \langle J_{ij}^1, J_{ij}^2, \cdots, J_{ij}^{k_{ij}} \rangle$ contains k_{ij} phases with each phase J_{ij}^k having an amount of *work* w_{ij}^k, where $w_{ij}^k > 0$, and a maximum parallelism h_{ij}^k, where $h_{ij}^k \geq 1$. Suppose that at time t job J_{ij} is in its k-th phase and is allocated a_{ij} processors, then its effective *speedup* is given by $\Gamma_{ij}^k(t) = \min\{a_{ij}, h_{ij}^k\}$. Hence, on processors of speed s for any $s > 0$, the *execution rate* of the job at time t is given by $s\Gamma_{ij}^k(t)$. The *span* l_{ij}^k of phase J_{ij}^k, which represents the amount of time to execute the phase with h_{ij}^k or more processors of unit speed, is therefore $l_{ij}^k = w_{ij}^k / h_{ij}^k$. The *work* $w(J_{ij})$ of job J_{ij} is given by $w(J_{ij}) = \sum_{k=1}^{k_{ij}} w_{ij}^k$, and the *span* $l(J_{ij})$ of the job is $l(J_{ij}) = \sum_{k=1}^{k_{ij}} l_{ij}^k$. Moreover, for each job set $\mathcal{J}_i$, we define its *set work* to be $w(\mathcal{J}_i) = \sum_{j=1}^{n_i} w(J_{ij})$ and define its *set span* to be $l(\mathcal{J}_i) = \max_{j=1\cdots n_i} l(J_{ij})$.

At any time t, given a total number P of processors, a scheduling algorithm needs to decide the processor allocation $a(\mathcal{J}_i, t)$ for each job set $\mathcal{J}_i$, as well as the processor allocation $a(J_{ij}, t)$ for each job J_{ij} within job set $\mathcal{J}_i$, where $1 \leq i \leq m$ and $1 \leq j \leq n_i$. We require that the total processor allocation cannot exceed the total number of available processors, i.e.,

$\sum_{i=1}^{m}\sum_{j=1}^{n_i} a(J_{ij},t) \le P$. Let r_{ij} denote the *release time* of job J_{ij} and let r_i denote the *release time* of job set $\mathcal{J}_i$. In this paper, we assume that all jobs in a job set are released at the same time, i.e., $r_{ij} = r_i$ for all $1 \le j \le n_i$. Moreover, if all job sets are released in a single *batch*, their release times are equal to 0. Otherwise, we can assume without loss of generality that the first released job set arrives at time 0. Let c_{ij}^k denote the completion time of the k-th phase of job J_{ij}, and let $c_{ij} = c_{ij}^{k_{ij}}$ denote the *completion time* of job J_{ij}. The *completion time* c_i of job set $\mathcal{J}_i$ is given by $c_i = \max_{j=1\cdots n_i} c_{ij}$. We also require that a valid schedule cannot begin to execute a phase of a job unless it has completed all its previous phases. To simplify analysis as in many previous work [6], [9], [15], [19], [20], we allow the processor allocation of a job to take non-integer values. The fractional allocation can be considered as time-sharing a processor with other jobs.

B. Objective Function

The *response time* or *flow time* f_{ij} of job J_{ij} is the duration between the completion time and the release time of the job, i.e., $f_{ij} = c_{ij} - r_{ij}$, and the *response time* f_i of job set $\mathcal{J}_i$ is given by $f_i = c_i - r_i$. The *total response time* $F(\mathcal{J})$ of all jobs in $\mathcal{J}$ is $F(\mathcal{J}) = \sum_{i=1}^{m}\sum_{j=1}^{n_i} f_{ij}$ and the *makespan* $M(\mathcal{J})$ of all jobs in $\mathcal{J}$ is $M(\mathcal{J}) = \max_{i=1\cdots m, j=1\cdots n_i} c_{ij}$. Our objective is to minimize the total response time of all job sets, or the *set response time* $H(\mathcal{J})$, which is given by $H(\mathcal{J}) = \sum_{i=1}^{m} f_i$. A job J_{ij} is said to be *active* at time t if it has been released but not completed at t, i.e., $r_{ij} \le t \le c_{ij}$. A job set $\mathcal{J}_i$ is said to be *active* at time t if it contains at least one active job at t. An alternative expression for the set response time is thus given by $H(\mathcal{J}) = \int_0^\infty m_t dt$, where m_t denotes the number of active job sets at time t. As pointed out in [15], the set response time of a job set collection $\mathcal{J}$ has the following interesting property: if $\mathcal{J} = \{\mathcal{J}_1\}$ contains a single job set, then the set response time is simply the makespan of all jobs in $\mathcal{J}$; if $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \cdots, \mathcal{J}_m\}$ contains a collection of singleton job sets, where for each $i = 1 \cdots m$, we have $\mathcal{J}_i = \{J_{i1}\}$, then the set response time is the total response time of all jobs in $\mathcal{J}$. Hence, the objective of set response time in some sense combines both makespan and total response time.

We use *competitive analysis* [2] to compare the set response time of an online algorithm with that of an optimal offline scheduler. An online algorithm is said to be *c-competitive* if its set response time satisfies $H(\mathcal{J}) \le c \cdot H^*(\mathcal{J})$ for any job set collection $\mathcal{J}$, where $H^*(\mathcal{J})$ denotes the set response time of $\mathcal{J}$ under an optimal offline scheduler. Since an online algorithm does not have any prior knowledge about the jobs while the optimal offline scheduler knows everything in advance, it is generally not possible to get good competitive ratios when the job sets can have arbitrary release time [6], [16]. In such case, we use the *resource augmentation analysis* [10], which gives the online algorithm extra resources and in a sense limits the power of the adversary. An online algorithm is then said to be *s-speed c-competitive* if its set response time $H_s(\mathcal{J})$ on processors of speed s, where $s > 1$, satisfies $H_s(\mathcal{J}) \le c \cdot H_1^*(\mathcal{J})$ for any job set collection $\mathcal{J}$, where $H_1^*(\mathcal{J})$ denotes the set response time of $\mathcal{J}$ under the optimal offline scheduler using unit-speed processors.

C. Lower Bounds for Set Response Time

For any job set collection $\mathcal{J}$, we define its *total span* to be $l(\mathcal{J}) = \sum_{i=1}^{m} l(\mathcal{J}_i)$, and define its *squashed work* to be $\hat{w}(\mathcal{J}) = \frac{1}{P}\sum_{i=1}^{m} i \cdot w(\mathcal{J}_{\pi(i)})$, where $\pi(\cdot)$ denotes a permutation of the job sets sorted in non-increasing work order, i.e., $w(\mathcal{J}_{\pi(1)}) \ge w(\mathcal{J}_{\pi(2)}) \ge \cdots \ge w(\mathcal{J}_{\pi(m)})$. Lemma 1 below states that the total span and the squashed work can serve as two lower bounds for the set response time of any job set collection, where the latter one only applies to batched job sets. In fact, these two lower bounds resemble the well-known lower bounds for the total response time of a single job set [7], [6], [9]. Due to space constraint, the detailed proofs for all lemmas of this paper are omitted, and can be found in the full version.

Lemma 1: To schedule any job set collection $\mathcal{J}$ on P processors of unit speed, the optimal set response time is at least the total span of $\mathcal{J}$, i.e., $H_1^*(\mathcal{J}) \ge l(\mathcal{J})$; if all job sets are batch released, the optimal set response time also satisfies $H_1^*(\mathcal{J}) \ge \hat{w}(\mathcal{J})$, where $\hat{w}(\mathcal{J})$ is the squashed work of $\mathcal{J}$. ■

III. The EQUI∘AGDEQ Algorithm

In this section, we introduce an online adaptive algorithm EQUI∘AGDEQ, which combines the well-known algorithm EQUI [22] with the feedback-driven scheduler AGDEQ [9] for scheduling any collection of job sets.

A. EQUI Algorithm

EQUI (Equi-partitioning) algorithm [22] simply divides the total number of processors evenly among all active job sets at any time. Suppose that at time t there are m_t active job sets, then each active job set $\mathcal{J}_i$ receives an allocation $a(\mathcal{J}_i, t) = P/m_t$ of processors. Hence, EQUI only reallocates the processors whenever a new job set is released or an existing job set is completed. Although simple, this algorithm does ensure absolute fairness among the competing job sets by giving each of them the same processor resources.

B. AGDEQ Algorithm

Within each job set, the efficiency of the allocated processors are guaranteed by consciously exploiting the jobs' execution history and periodically adjusting the processor allocations among the jobs. This is realized by the feedback-driven algorithm AGDEQ [9], which works based on the interaction between the A-GREEDY [1] scheduler and the DEQ [12] allocator after each *scheduling quantum*. Specifically, A-GREEDY collects the execution statistics of a job in each quantum, based on which it calculates the job's *processor desire*, that is, how many processors the job needs, for the next quantum. The DEQ allocator then decides a *processor allocation* for each job in the next quantum according to the processor desires of all jobs. As this process only involves the execution history of the jobs but not their future characteristics, AGDEQ can be considered as a *non-clairvoyant* algorithm [13], [9]. In the following, we will describe the desire calculation strategy of A-GREEDY and the processor allocation policy of DEQ separately.

1) A-GREEDY *Scheduler:* The A-GREEDY scheduler [1] calculates the processor desires of a job using a simple *multiplicative-increase multiplicative-decrease* strategy. For each job J_{ij} in a scheduling quantum q, let $d(J_{ij}, q)$ and $a(J_{ij}, q)$ denote its processor desire and processor allocation,

respectively. We say that job J_{ij} is *satisfied* in quantum q if its processor allocation is at least its processor desire, i.e., $a(J_{ij}, q) \geq d(J_{ij}, q)$. Otherwise, the job is *deprived* if $a(J_{ij}, q) < d(J_{ij}, q)$. Let t_q denote the time when quantum q starts and let L denote the duration of the quantum. A-GREEDY collects the amount of work $w(J_{ij}, q)$ completed for job J_{ij} in quantum q, i.e., $w(J_{ij}, q) = \int_{t_q}^{t_q+L} s\Gamma_{ij}^{k_t}(t)dt$, where k_t is the phase job J_{ij} is executing at time t and s is the processor speed. Given that the total allocated processor cycles for job J_{ij} in quantum q is $a(J_{ij}, q)sL$, job J_{ij} is said to be *efficient* if its completed work is at least δ fraction of the total allocated processor cycles, i.e., $w(J_{ij}, q) \geq \delta \cdot a(J_{ij}, q)sL$, where $\delta \leq 1$ is called the *utilization parameter*. Otherwise, the job is *inefficient* if $w(J_{ij}, q) < \delta \cdot a(J_{ij}, q)sL$. The processor desire $d(J_{ij}, q+1)$ of job J_{ij} for the next quantum $q+1$ is then calculated based on whether the job is satisfied or deprived and whether it is efficient or inefficient in quantum q as shown below:

$$d(J_{ij}, q+1) = \begin{cases} d(J_{ij}, q) \cdot \rho & \text{if efficient and satisfied in } q, \\ d(J_{ij}, q)/\rho & \text{if inefficient in } q, \\ d(J_{ij}, q) & \text{if efficient and deprived in } q, \end{cases}$$

where $\rho > 1$ is called the *responsiveness parameter*. The initial desire of the job when it first entered the system is simply set as 1 to start with.

2) DEQ *Allocator:* The DEQ (Dynamic Equi-partitioning) allocator [12] distributes the processors among the jobs in a set based on the processor desires of all jobs. Similar to EQUI, DEQ attempts to ensure fairness among the active jobs by giving each of them an equal processor allocation. However, DEQ never allocates more processors to a job than the job's desire, since otherwise the extra processors are likely to be wasted in the following quantum. The surplus processors will instead be shared among the other jobs with higher desires. Suppose that at time t when a quantum starts, there are n_t active jobs in set $\mathcal{J}_i$. If the total processor desire of these jobs is not more than $a(\mathcal{J}_i, t)$, the processor allocation for set $\mathcal{J}_i$ at time t, then all active jobs in $\mathcal{J}_i$ will be satisfied. Otherwise, DEQ calculates the fair share $a(\mathcal{J}_i, t)/n_t$ of processors, and the jobs whose desires are less than this fair share will be satisfied. The fair share is then recomputed by excluding the jobs already satisfied and the processors already allocated. The same process continues until no job can be satisfied with the fair share, in which case the remaining processors will be split evenly among the deprived jobs. In general, DEQ tends to satisfy those jobs with low desires while the jobs with high desires are likely to be deprived and share an equal processor allocation.

C. Simplifying Assumption

To ease analysis, we assume that a job set is only released or completed at the beginning of a scheduling quantum. Since EQUI is not quantum-based but rather reallocates the processors based on the release and the completion of job sets, the assumption ensures that the scheduling decision made by EQUI at the job-set level and the decision by AGDEQ at the job level are well synchronized. This assumption can be easily justified since most computation-intensive workloads take much longer to execute compared to any realistic quantum size, which is normally in the order of milliseconds. Hence, the effect of a few extra quanta on the execution time of a job can be practically ignored.

IV. PERFORMANCE ANALYSIS

A. Preliminaries

To analyze the performance of EQUI∘AGDEQ, we need to define some preliminary concepts. First of all, we extend the notions of "satisfied" and "deprived" from quantum to time: a job is said to be satisfied (deprived) at time t if t is within a satisfied (resp., deprived) quantum for the job. In addition, we extend these notions from an individual job to a job set as follows: a job set $\mathcal{J}_i$ is said to be *satisfied* at time t if *all* jobs in $\mathcal{J}_i$ are satisfied at t; otherwise, $\mathcal{J}_i$ is said to be *deprived* if it contains *at least one* deprived job at t. Let $\mathcal{J}_i(t)$ denote job set $\mathcal{J}_i$ at time t, and let $\mathcal{J}(t)$ denote the set of all active job sets at t. Moreover, let $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the set of deprived job sets and the set of satisfied job sets in $\mathcal{J}(t)$, respectively. Throughout the execution of job set $\mathcal{J}_i$, we define $a_A(\mathcal{J}_i)$ to be the amount of processor allocation $\mathcal{J}_i$ receives when it is deprived, or its *deprived processor allocation*, i.e., $a_A(\mathcal{J}_i) = \int_0^\infty a(\mathcal{J}_i, t)s \cdot [\mathcal{J}_i(t) \in \mathcal{J}_A(t)]dt$, and define $t_B(\mathcal{J}_i)$ to be the amount of processor time for $\mathcal{J}_i$ when it is satisfied, or its *satisfied processor time*, i.e., $t_B(\mathcal{J}_i) = \int_0^\infty s \cdot [\mathcal{J}_i(t) \in \mathcal{J}_B(t)]dt$, where s is the processor speed of EQUI∘AGDEQ, and $[x]$ is 1 if proposition x is true and 0 otherwise. To simplify notations, let $m_t^A = |\mathcal{J}_A(t)|$ and $m_t^B = |\mathcal{J}_B(t)|$ denote the number of deprived job sets and the number of satisfied job sets at time t, respectively. Since an active job set is either satisfied or deprived, we have $m_t^A + m_t^B = m_t$, where $m_t = |\mathcal{J}(t)|$ is the total number of active job sets at t.

We now introduce the concepts of *squashed deprived processor allocation* $\hat{a}_A(\mathcal{J})$ and *total satisfied processor time* $t_B(\mathcal{J})$ for the entire job set collection $\mathcal{J}$ as follows:

$$\hat{a}_A(\mathcal{J}) = \frac{1}{P}\sum_{i=1}^{m} i \cdot a_A(\mathcal{J}_{\gamma(i)}), \quad (1)$$

$$t_B(\mathcal{J}) = \sum_{i=1}^{m} t_B(\mathcal{J}_i), \quad (2)$$

where $\gamma(\cdot)$ denotes a permutation of the job sets sorted in non-increasing order of deprived processor allocation, i.e., $a_A(\mathcal{J}_{\gamma(1)}) \geq a_A(\mathcal{J}_{\gamma(2)}) \geq \cdots \geq a_A(\mathcal{J}_{\gamma(m)})$. It is not difficult to see that $\gamma(\cdot)$, among all permutations of the job sets, gives the minimum value for the squashed formulation, i.e., $\sum_{i=1}^{m} i \cdot a_A(\mathcal{J}_{\gamma(i)}) \leq \sum_{i=1}^{m} i \cdot a_A(\mathcal{J}_{\pi(i)})$ for any permutation $\pi(\cdot)$ of the job sets. The following lemma derives the upper bounds for the squashed deprived processor allocation and the total satisfied processor time respectively in terms of the squashed work and the total span of a job set collection. These bounds will be used later in the analysis.

Lemma 2: Suppose that EQUI∘AGDEQ schedules a collection $\mathcal{J}$ of m job sets on P processors of speed s. Then the squashed deprived processor allocation $\hat{a}_A(\mathcal{J})$ and the total satisfied processor time $t_B(\mathcal{J})$ for the collection $\mathcal{J}$ satisfy

$$\hat{a}_A(\mathcal{J}) \leq \frac{1+\rho}{\delta} \cdot \hat{w}(\mathcal{J}), \quad (3)$$

$$t_B(\mathcal{J}) \leq \frac{2}{1-\delta} \cdot l(\mathcal{J}) + msL(\log_\rho P + 1), \quad (4)$$

where $\hat{w}(\mathcal{J})$ and $l(\mathcal{J})$ denote the squashed work and the total span of $\mathcal{J}$, δ and ρ denote A-GREEDY's utilization and responsiveness parameters, and L is the quantum length. ■

Finally, we introduce the notions of t-prefix and t-suffix. For EQUI∘AGDEQ, we define the *t-prefix* $\mathcal{J}_i(\overleftarrow{t})$ for job set $\mathcal{J}_i$ to be the portion of the job set executed before and at time t, and define its *t-suffix* $\mathcal{J}_i(\overrightarrow{t})$ to be the portion executed after time t. We then extend the notions of t-prefix and t-suffix to the job set collection as follows: the t-prefix of job set collection $\mathcal{J}$ is defined to be $\mathcal{J}(\overleftarrow{t}) = \{\mathcal{J}_i(\overleftarrow{t}) : \mathcal{J}_i \in \mathcal{J} \text{ and } r_i \geq t\}$ and the t-suffix of $\mathcal{J}$ is $\mathcal{J}(\overrightarrow{t}) = \{\mathcal{J}_i(\overrightarrow{t}) : \mathcal{J}_i \in \mathcal{J} \text{ and } r_i \geq t\}$. Similarly, we define $\mathcal{J}^*(\overleftarrow{t})$ and $\mathcal{J}^*(\overrightarrow{t})$ to be the t-prefix and the t-suffix for the job set collection $\mathcal{J}$ executed by the optimal offline scheduler.

B. Analysis for Batched Job Sets

We first analyze the set response time of the EQUI∘AGDEQ algorithm when all job sets are batch released. The analysis relies on *local competitiveness argument* [14], which bounds the performance of an online algorithm at any time in terms of the optimal offline scheduler, or rather its two lower bounds presented in Section II-C.

For any job set collection $\mathcal{J}$, we focus on its t-prefix $\mathcal{J}(\overleftarrow{t})$, which according to definition always contains m sets of jobs for any $t > 0$. Recall that the squashed deprived processor allocation for $\mathcal{J}(\overleftarrow{t})$ is given by $\hat{a}_A(\mathcal{J}(\overleftarrow{t})) = \frac{1}{P}\sum_{i=1}^{m} i \cdot a_A(\mathcal{J}_{\gamma(i)}(\overleftarrow{t}))$, where $\gamma(\cdot)$ denotes a permutation of the job sets in $\mathcal{J}(\overleftarrow{t})$ sorted in non-increasing order of deprived processor allocation. At any time t, let $m_t(z)$ denote the number of job sets in $\mathcal{J}(\overleftarrow{t})$ whose deprived processor allocation is at least z under EQUI∘AGDEQ, i.e., $m_t(z) = \sum_{i=1}^{m}[a_A(\mathcal{J}_i(\overleftarrow{t})) \geq z]$. Apparently, $m_t(z)$ is a staircase-like decreasing function of z. From the definition of squashed deprived processor allocation, an alternative expression for $\hat{a}_A(\mathcal{J}(\overleftarrow{t}))$ is given by

$$\hat{a}_A(\mathcal{J}(\overleftarrow{t})) = \frac{1}{P}\int_0^{\infty}\left(\sum_{i=1}^{m_t(z)} i\right) dz, \tag{5}$$

which is a more convenient representation of $\hat{a}_A(\mathcal{J}(\overleftarrow{t}))$ for analyzing batched job sets. Note that the expression presented in Eq. (5) gives a much simpler perspective for squashed deprived processor allocation than the one conceived in [9], hence can be applied to simplify the batched response time analysis therein. For our analysis on set response time, we give both EQUI∘AGDEQ and the optimal scheduler P processors of unit speed. The local performance of EQUI∘AGDEQ is shown in the following lemma.

Lemma 3: Suppose that EQUI∘AGDEQ schedules a collection $\mathcal{J}$ of batched job sets on P processors of unit speed. Then the execution of the job sets satisfies the following

- Running condition: $m_t \leq 2\left(\frac{d\hat{a}_A(\mathcal{J}(t))}{dt} + m_t^B\right)$,

where $\frac{d\hat{a}_A(\mathcal{J}(t))}{dt}$ denotes the rate of change for the squashed deprived processor allocation in an infinitesimal interval during which no job set completes. ■

We can now combine the results of Lemmas 2 and 3 for the set response time of EQUI∘AGDEQ in batched scenario.

Theorem 1: Suppose that EQUI∘AGDEQ schedules a collection $\mathcal{J}$ of m batched job sets on P processors of unit speed. Then its set response time satisfies

$$H_1(\mathcal{J}) \leq \frac{2(1+\rho+\delta-\rho\delta)}{\delta(1-\delta)} H_1^*(\mathcal{J}) + 2mL(\log_\rho P + 1),$$

where $H_1^*(\mathcal{J})$ denotes the set response time of $\mathcal{J}$ under the optimal scheduler on unit-speed processors, δ and ρ denote A-GREEDY's utilization and responsiveness parameters, and L is the quantum length.

Proof: As mentioned in Section II-B, the set response time of job set collection $\mathcal{J}$ scheduled by EQUI ∘ AGDEQ can be expressed as $H_1(\mathcal{J}) = \int_0^{\infty} m_t dt$. Similarly, the total satisfied processor time of $\mathcal{J}$ under EQUI ∘ AGDEQ is given by $t_B(\mathcal{J}) = \int_0^{\infty} m_t^B dt$. Integrating the running condition in Lemma 3, we have $H_1(\mathcal{J}) \leq 2\left(\hat{a}_A(\mathcal{J}) + t_B(\mathcal{J})\right)$. Substituting the bounds of $\hat{a}_A(\mathcal{J})$ and $t_B(\mathcal{J})$ from Lemma 2 into the above inequality, we get $H_1(\mathcal{J}) \leq 2\left(\frac{1+\rho}{\delta}\cdot\hat{w}(\mathcal{J}) + \frac{2}{1-\delta}\cdot l(\mathcal{J}) + mL(\log_\rho P + 1)\right)$. Based on Lemma 1, both squashed work $\hat{w}(\mathcal{J})$ and total span $l(\mathcal{J})$ are lower bounds for the set response time of $\mathcal{J}$. The performance of EQUI∘AGDEQ thus satisfies $H_1(\mathcal{J}) \leq 2\left(\frac{1+\rho}{\delta}\cdot H_1^*(\mathcal{J}) + \frac{2}{1-\delta}\cdot H_1^*(\mathcal{J}) + mL(\log_\rho P + 1)\right) = \frac{2(1+\rho+\delta-\rho\delta)}{\delta(1-\delta)} H_1^*(\mathcal{J}) + 2mL(\log_\rho P + 1)$. ■

C. Analysis for Arbitrarily Released Job Sets

We now analyze the set response time of EQUI∘AGDEQ when the job sets can have arbitrary release time. Note that the squashed work is no longer a lower bound for the set response time in this scenario. Hence, the analysis uses *amortized local competitiveness argument* [14], which bounds the amortized performance of an online algorithm at any time in terms of the optimal offline scheduler through a potential function.

We adopt the potential function used in [19] for the response time analysis of AGDEQ. However, compared to the previous potential function, where only processor allocations of deprived jobs are considered, the one used here needs to incorporate the whole job set collection instead of a single job set. Hence, for each deprived job set at any time, the potential function considers its entire processor allocation, including those of satisfied jobs. In particular, we focus on the t-suffix $\mathcal{J}(\overrightarrow{t})$ of job set collection $\mathcal{J}$, and define $m_t(z)$ to be the number of job sets in $\mathcal{J}(\overrightarrow{t})$ whose deprived processor allocation is at least $\frac{1+\rho}{\delta}\cdot z$ at time t under EQUI∘AGDEQ, i.e., $m_t(z) = \sum_{i=1}^{m}[a_A(\mathcal{J}_i(\overrightarrow{t})) \geq \frac{1+\rho}{\delta}\cdot z]$. Moreover, let $m_t^*(z)$ denote the number of job sets in $\mathcal{J}^*(\overrightarrow{t})$ whose work is at least z under the optimal, i.e., $m_t^*(z) = \sum_{i=1}^{m}[w(\mathcal{J}_i^*(\overrightarrow{t})) \geq z]$. Hence, both $m_t(z)$ and $m_t^*(z)$ are staircase-like decreasing functions of z. We give EQUI∘AGDEQ P processors of speed s, where $s = \frac{2(1+\rho)}{\delta} + \epsilon$ for any $\epsilon > 0$, while the optimal scheduler uses unit-speed processors. The potential function is defined as

$$\Phi(t) = \eta\int_0^{\infty}\left[\left(\sum_{i=1}^{m_t(z)} i\right) - m_t(z)m_t^*(z)\right] dz, \tag{6}$$

where $\eta = \frac{2(1+\rho)}{\delta\epsilon P}$. We now prove the amortized local performance of EQUI∘AGDEQ in the following lemma.

Lemma 4: Suppose that EQUI∘AGDEQ schedules a collection $\mathcal{J}$ of job sets on P processors of speed s, where $s = \frac{2(1+\rho)}{\delta} + \epsilon$ for any $\epsilon > 0$. Then with the potential function defined in Eq. (6), the execution of the job sets satisfies the following

- Boundary condition: $\Phi(0) = 0$ and $\Phi(\infty) = 0$;
- Arrival condition: $\Phi(t)$ does not increase when a new job set arrives;
- Running condition: $m_t + \frac{d\Phi(t)}{dt} \leq \frac{2s}{\epsilon}\left(m_t^* + m_t^B\right)$,

where $\frac{d\Phi(t)}{dt}$ denotes the rate of change for the potential function in an infinitesimal interval during which no job set arrives or completes under either EQUI∘AGDEQ or optimal. ■

The following theorem gives the set response time of EQUI∘AGDEQ for arbitrarily released job sets.

Theorem 2: Suppose that EQUI∘AGDEQ schedules a collection $\mathcal{J}$ of m job sets on P processors of speed s, where $s = \frac{2(1+\rho)}{\delta} + \epsilon$ for any $\epsilon > 0$. Then its set response time satisfies

$$H_s(\mathcal{J}) \leq \left(2 + \frac{4(1+\rho-\rho\delta)}{\delta(1-\delta)\epsilon}\right) H_1^*(\mathcal{J}) + \frac{2msL}{\epsilon}(\log_\rho P + 1),$$

where $H_1^*(\mathcal{J})$ denotes the set response time of $\mathcal{J}$ under the optimal scheduler on unit-speed processors, δ and ρ denote A-GREEDY's utilization and responsiveness parameters, and L is the quantum length.

Proof: As the set response time of EQUI∘AGDEQ is given by $H_s(\mathcal{J}) = \int_0^\infty m_t dt$, and the set response time of the optimal is $H_1^*(\mathcal{J}) = \int_0^\infty m_t^* dt$, integrating the running condition in Lemma 4 and applying the boundary and arrival conditions, we have $H_s(\mathcal{J}) \leq \frac{2s}{\epsilon} \cdot H_1^*(\mathcal{J}) + \frac{2}{\epsilon} \cdot t_B(\mathcal{J})$, where $t_B(\mathcal{J}) = \int_0^\infty s \cdot m_t^B dt$ is the total satisfied processor time for $\mathcal{J}$ under EQUI∘AGDEQ. From Lemma 2, the total satisfied processor time for $\mathcal{J}$ is also given by $t_B(\mathcal{J}) \leq \frac{2}{1-\delta} \cdot l(\mathcal{J}) + msL(\log_\rho P + 1)$. The set response time of $\mathcal{J}$ scheduled by EQUI∘AGDEQ thus satisfies $H_s(\mathcal{J}) \leq \frac{2s}{\epsilon} \cdot H_1^*(\mathcal{J}) + \frac{4}{(1-\delta)\epsilon} \cdot l(\mathcal{J}) + \frac{2msL}{\epsilon}(\log_\rho P + 1)$. Since the total span $l(\mathcal{J})$ is a lower bound for the set response time of $\mathcal{J}$ on unit-speed processors, the theorem is directly implied. ■

D. Discussions

In the preceding two subsections, we have analyzed the set response time of the EQUI∘AGDEQ algorithm for both batched and nonbatched job sets. As Theorem 1 shows, when all job sets are batched, EQUI∘AGDEQ achieves $O(1)$-competitive, since both ρ and δ can be considered as constants. This result improves upon the $O(\frac{\ln n}{\ln \ln n})$-competitiveness of EQUI∘EQUI obtained in [15], where n is the maximum number of jobs in a set. For small values of n, however, the competitive ratio of EQUI∘EQUI can be considered as constant as well, and thus it will perform comparably to EQUI∘AGDEQ, which are confirmed by our simulation results to be shown in Section V. For nonbatched job sets, Theorem 2 shows that EQUI∘AGDEQ achieves $O(1)$-speed $O(1)$-competitive. This result extends the same asymptotic performance of the EQUI∘Y algorithm obtained in [16] from jobs with specific parallelism structure, namely, a sequential phase followed by a fully-parallel phase, to jobs with any parallelism profile.

Note that, for jobs with arbitrary sizes, any non-clairvoyant algorithm has been shown to be $\omega(1)$-competitive with respect to the set response time [15]. Hence, the competitive ratio of EQUI∘AGDEQ obtained here is in the *asymptotic* sense based on the assumption that the jobs under consideration are sufficiently large. Thus, the optimal set response time $H_1^*(\mathcal{J})$ will denominate the additive factors shown in the inequalities of Theorems 1 and 2, which can then be considered as constants. For instance, when job sets are batched and the optimal set response time is much larger than the additive factor $2mL(\log_\rho P + 2)$, which is commonly satisfied by most computation-intensive workloads with realistic quantum size and practical number of processors, the performance of EQUI∘AGDEQ shown in Theorem 1 then becomes $H_1(\mathcal{J}) \leq \left(\frac{2(1+\rho+\delta-\rho\delta)}{\delta(1-\delta)} + o(1)\right) H_1^*(\mathcal{J}) = O(1) \cdot H_1^*(\mathcal{J})$.

While both EQUI∘AGDEQ and EQUI∘EQUI use EQUI at the job-set level to ensure fairness, the performance improvement of EQUI∘AGDEQ for sufficiently large jobs is essentially because the processors are utilized more efficiently by AGDEQ at the job level. On the other hand, EQUI obliviously allocates processors to jobs, which will inevitably incur a large waste of resources when the parallelism of the jobs can vary with time. Hence, both fairness among the job sets and efficiency within each job set play important roles when scheduling for multiple sets of parallel applications.

E. EQUI∘ACDEQ and Its Performance

In this subsection, we introduce an improved adaptive scheduling algorithm EQUI∘ACDEQ, which combines EQUI with the feedback-driven algorithm ACDEQ [20]. As with AGDEQ, the ACDEQ algorithm also uses the DEQ allocator to distribute processors among the jobs in a set. For the desire calculation strategy, it relies on the A-CONTROL scheduler proposed in [21] to mitigate the feedback instability problem of A-GREEDY, which is briefly described in the following.

Suppose that the parallelism of a job stays constantly at a particular value for sufficiently long time. Because of the multiplicative-increase multiplicative-decrease strategy, the processor desires produced by A-GREEDY tend to oscillate around the target parallelism of the job without settling down. To avoid this problem, A-CONTROL provides feedbacks directly based on the job's average parallelism in the previous quantum. Specifically, for each job J_{ij} in a quantum q, let t_q denote the time when the quantum starts. A-CONTROL collects the amount of work completed for the job in the quantum, i.e., $w(J_{ij}, q) = \int_{t_q}^{t_q+L} s\Gamma_{ij}^{k_t}(t)dt$, as well as its span reduced in the quantum, i.e., $l(J_{ij}, q) = \int_{t_q}^{t_q+L} s\Gamma_{ij}^{k_t}(t)/h_{ij}^{k_t} dt$, where k_t represents the phase job J_{ij} is executing at time t, s denotes the processor speed used by A-CONTROL and L is the quantum length. The average parallelism $A(J_{ij}, q)$ of the job in quantum q is then given by $A(J_{ij}, q) = w(J_{ij}, q)/l(J_{ij}, q)$. The processor desire of the job in the next quantum $q+1$ is set directly to the average parallelism in quantum q, i.e., $d(J_{ij}, q+1) = A(J_{ij}, q)$. Apparently, this strategy makes the processor desire more representative of the job's immediate processor requirement, and it effectively eliminates the feedback instability of A-GREEDY [20]. The simulations conducted in [19], [20] showed that ACDEQ indeed outperforms AGDEQ in terms of both stability of the desires and total response time of the jobs. In the next section, we will evaluate EQUI∘ACDEQ

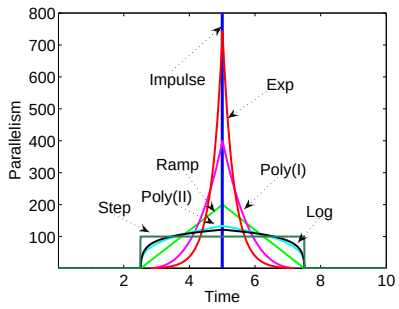

Fig. 1. Seven parallelism variation curves described by Step, Log, Poly(II), Ramp, Poly(I), Exp and Impulse functions.

together with EQUI∘AGDEQ and EQUI∘EQUI in terms of the set response time of multiple job sets.

As for the theoretical analysis, the performance of ACDEQ is bounded in terms of the *transition factor* of the jobs. This factor captures a job's maximum parallelism transition between any two consecutive quanta, and to a certain extent better reflects the degree of difficulty to schedule the job in an online adaptive manner. Due to space constraint, we refer interested readers to [20], [21] for more details about the analysis of ACDEQ.

V. EMPIRICAL EVALUATIONS

In this section, we conduct simulations to compare the performances of three online adaptive schedulers, namely, EQUI∘AGDEQ, EQUI∘ACDEQ, and EQUI∘EQUI.

A. Parallel Workload

The simulations are performed using synthetic *malleable* jobs, which are generated based on Downey's parallel workload model [5]. However, Downey's model only specifies some external characteristics for the jobs such as their arrival time, total work and average parallelism. To add internal structures, each job is divided into a series of segments and each segment is identified by one of the seven parallelism variation curves shown in Figure 1. These variation curves are first proposed in [3] and are specifically designed to capture a range of parallel programming patterns that could represent different sections of an application. (More details on the interpretations of these curves can be found in [3].) Note that, when augmenting the jobs with these parallelism variations, we carefully make sure that the total work and the average parallelism of a job over all segments are not altered from those initially generated by Downey's model.

B. Simulation Setup

We simulate a system with 64 processors, and the aim is to study the performances of the scheduling algorithms when we vary the number of sets, the number of jobs in a set, as well as the jobs' arrival patterns. Following advices from [1], [9], [19], the utilization and responsiveness parameters of A-GREEDY are set to $\delta = 0.8$ and $\rho = 2$, respectively. Based on Downey's model, the arrival rate of the jobs is set proportionally to the number of sets, and hence is tied to the load of the system. We group consecutively released jobs together to form job sets. In our simulation, both the number of sets and the number of jobs in a set are varied from 5 to 100 at an increment of 5 each time. Moreover, to make sure that all jobs within a set are batched, we adjust the release time of each job in a job set to when the first job of the set is released. Because of such adjustment, the gap between the releases of consecutive sets are also correspondingly increased. Hence, it turns out that the system load, i.e., number of active job sets at any time, is still light when the total number of sets reaches 100. To simulate heavy system loads, therefore, we readjust the release time of all jobs to 0, which then corresponds to the batched scenario considered in Section IV-B. In this case, the number of sets is only increased from 5 to 50, and a quick convergence on the performances of the three algorithms can already be observed from the simulation results.

C. Set Response Time

We first focus on the set response time of the three scheduling algorithms, which is shown in Figure 2. First of all, when the number of sets is fixed, the relative set response time is closely related to the number of jobs within each set. In general, EQUI∘AGDEQ and EQUI∘ACDEQ outperform EQUI∘EQUI when there is a moderate to large number of jobs in each set. The reason is that, given the same number of processors, feedback-driven algorithms make use of jobs' execution history to efficiently guide processor allocations, while EQUI is oblivious to jobs' parallelism variations thus results in low system efficiency. However, when each set only contains a small number of jobs, the performance of EQUI becomes better, since all jobs can be easily satisfied by EQUI in this case. The results also show that the relative set response time is significantly impacted by the system loads. In particular, when the load is light, i.e., the sets are nonbatched, EQUI∘AGDEQ and EQUI∘ACDEQ perform much better than EQUI∘EQUI. On the other hand, when all the sets are batch released, representing heavy system load, the performances of EQUI∘AGDEQ and EQUI∘ACDEQ quickly converge to that of EQUI∘EQUI. This is because in this case each set only receives very few processors most of the time, hence frequent processor reallocations within the sets have no apparent benefit. These results demonstrate that feedback-driven schedulers are particularly effective for systems with light loads and a moderate number of jobs in the sets, while EQUI∘EQUI can perform comparably well in other cases. In addition, the performance of EQUI∘ACDEQ is always better than that of EQUI∘AGDEQ, which is due to the more stable desire calculation strategy of the A-CONTROL scheduler.

D. Processor Utilization

Figure 3 shows the relative processor utilizations of the three scheduling algorithms. The results show that the utilizations of the two feedback-driven schedulers are always better than that of EQUI∘EQUI under all system loads. When there are very few jobs in each set, EQUI∘EQUI has particularly bad utilization, since it is blind to the jobs' parallelism variations thus eventually wastes a large amount of resources. Figure 2 shows that EQUI∘EQUI actually achieves better set response time in this case at the cost of poor system utilization. With increases in the system load and the number of jobs, the utilization advantage of the feedback-driven schedulers begins to diminish, since nearly all processors are well utilized by the three schedulers as more jobs are present in the system.

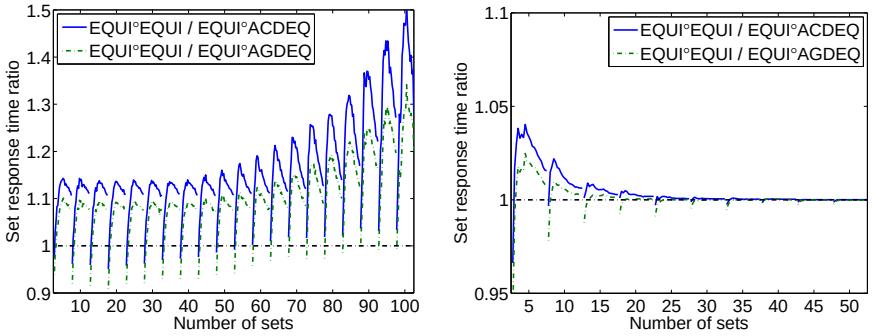

Fig. 2. Set response time of EQUI∘AGDEQ and EQUI∘ACDEQ normalized by that of EQUI∘EQUI when (a) job sets are nonbatched (b) job sets are batched.

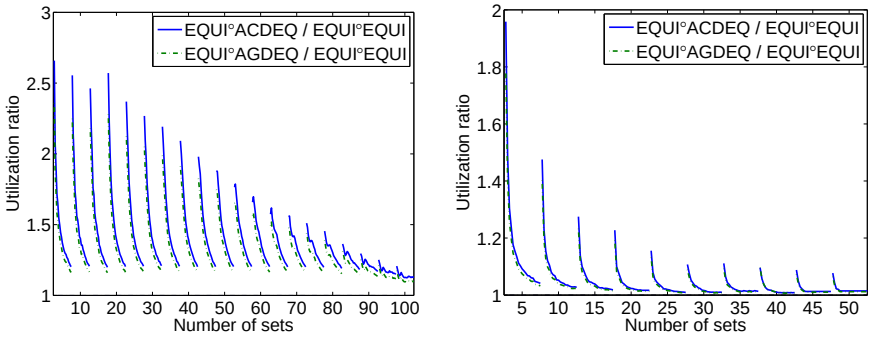

Fig. 3. Processor utilization of EQUI∘AGDEQ and EQUI∘ACDEQ normalized by that of EQUI∘EQUI when (a) job sets are nonbatched (b) job sets are batched.

The results confirm that the feedback-driven algorithms, which take advantage of parallelism correlations of the jobs, indeed achieve better overall system efficiency than EQUI∘EQUI.

VI. Conclusion

In this paper, we have studied online adaptive scheduling to minimize the response time for multiple sets of parallel applications on multiprocessor systems. We have achieved both fairness and efficiency by applying the equi-partitioning algorithm and the feedback-driven algorithms at the job-set level and the job level, respectively. Both theoretical analysis and empirical simulations have confirmed the improved performances of our algorithms compared to an existing scheduler. In our future research, we plan to study hierarchical scheduling, where jobs are organized into more than two levels. While feedback-driven scheduling has been shown to be effective for minimizing makespan under this setting [4], we believe that it will perform well with respect to other related objectives.

References

[1] K. Agrawal, Y. He, W.-J. Hsu, and C. E. Leiserson. Adaptive scheduling with parallelism feedback. In *PPoPP*, New York, USA, 2006.

[2] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[3] Y. Cao, H. Sun, and W.-J. Hsu. Malleable-Lab: A tool for evaluating adaptive online schedulers on malleable jobs. In *PDP*, Pisa, Italy, 2010.

[4] Y. Cao, H. Sun, D. Qian, and W. Wu. Scalable hierarchical scheduling for multiprocessor systems using adaptive feedback-driven policies. In *ISPA*, Taipei, Taiwan, 2010.

[5] A. B. Downey. A parallel workload model and its implications for processor allocation. In *HPDC*, Portland, USA, 1997.

[6] J. Edmonds. Scheduling in the dark. In *STOC*, Atlanta, USA, 1999.

[7] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. In *STOC*, El Paso, USA, 1997.

[8] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems. *IBM Research Report RC19790(87657) 2nd Revision*, 1997.

[9] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, Saint-Malo, France, 2006.

[10] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[11] Y.-K. Kwok, and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.

[12] 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.

[13] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *SODA*, Austin, USA, 1993.

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

[15] J. Robert and N. Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *ESA*, Eilat, Israel, 2007.

[16] J. Robert and N. Schabanel. Pull-based data broadcast with dependencies: be fair to users, not to items. In *SODA*, New Orleans, USA, 2007.

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

[18] R. Sudarsan and C. J. Ribbens ReSHAPE: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment. In *ICPP*, Xi'an, China, 2007.

[19] H. Sun, Y. Cao, and W.-J. Hsu. Competitive two-level adaptive scheduling using resource augmentation. In *JSSPP*, Rome, Italy, 2009.

[20] H. Sun, Y. Cao, and W.-J. Hsu. Efficient Adaptive scheduling of multiprocessors with stable parallelism feedback. In *IEEE Transactions on Parallel and Distributed Systems*, 22(4), 594–607, 2011.

[21] H. Sun and W.-J. Hsu. Adaptive B-Greedy (ABG): A simple yet efficient scheduling algorithm. In *IPDPS*, Miami, USA, 2008.

[22] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *SOSP*, New York, USA, 1989.