Improved Results for Scheduling Batched Parallel Jobs by Using a Generalized Analysis Framework

Yuxiong He, Hongyang Sun, and Wen-Jing Hsu

School of Computer Engineering, Nanyang Technological University, Singapore
{yxhe, sunh0007, hsu}@ntu.edu.sg

Abstract: We present two improved results for scheduling batched parallel jobs on multiprocessors with mean response time as the performance metric. These results are obtained by using a generalized analysis framework where the response time of the jobs is expressed in two contributing factors that directly impact a scheduler's competitive ratio. Specifically, we show that the scheduler IGDEQ is 3-competitive against the optimal while AGDEQ is 5.24-competitive. These results improve the known competitive ratios of 4 and 10, obtained by Deng et al. and by He et al., respectively. For the common case where no fractional allotments are allowed, we show that slightly larger competitive ratios can be obtained by augmenting the schedulers with the round-robin strategy.

Keywords: Multiprocessor Scheduling, Batched parallel Jobs, Mean response time

1 Introduction

One major issue in scheduling parallel job on multiprocessor systems is how to efficiently share the multiprocessor resource among a number of competing jobs while ensuring each job a required level of quality of services (see e.g. [19, 8, 7, 18, 23, 33, 15, 10, 21, 16, 28, 12, 32, 24, 25, 38, 35, 30, 40, 37, 27, 31, 17, 5]). Mean response time is an important measure in multiuser environments where we desire as many users as possible to get fast responses. In this paper, we focus on the mean response time for a set of batched parallel jobs to be scheduled on multiprocessors using competitive analysis [9]. Specifically, we consider scheduling a set $\mathcal{J} = \{J_1, J_2, \ldots, J_{|\mathcal{J}|}\}$ of parallel jobs on P identical processors, where all jobs in $\mathcal{J}$ are released at time 0. Moreover, we allow each parallel job to have time-varying execution characteristics with changing degrees of parallelism. The objective is to bound the mean response time of a scheduling algorithm in terms of its competitive ratio against the optimal scheduler.

Many existing results in the literature [31, 17, 15, 14, 21] address the problem of scheduling batched parallel jobs on multiprocessors under various job models. For fully parallelizable jobs, whose executions can be sped up linearly with increased processor allocations, Motwani et al. [31] showed that RR (Round-Robin) algorithm is 2-competitive in terms of the mean response time. Deng et al. [15, 14] showed that DEQ (Dynamic Equi-partitioning) algorithm is 4-competitive for the mean response time when jobs are modeled by either DAGs or time-varying parallelism profiles, where a parallelism profile of a job characterizes how many processors the job can effectively use at any time of its execution. Edmonds et al. [17] showed that the mean response time obtained by EQUI (Equi-Partitioning) algorithm is $(2 + \sqrt{3})$-competitive for jobs modeled by multiple phases of arbitrarily non-decreasing and sub-linear speedup functions, where a job with sub-linear speedup functions is one whose execution efficiency does not increase with more processors and a job with non-decreasing speedup functions executes no slower if it is allocated more processors. Finally, He et al. [21] showed that a two-level AGDEQ (Adaptive Greedy Dynamic Equi-partitioning) algorithm is $O(1)$-competitive for the mean response time of jobs modeled by dynamically unfolding directed acyclic graph (or DAG for short), which specifies the precedence constraints for the tasks of a job to be executed.

Although different job models are used in the literature, it will be very useful to apply a common framework to analyze the mean response time of different algorithms. In this paper, we present such a framework and illustrate its application on the three online algorithms mentioned above, namely, EQUI, DEQ, and AGDEQ. The main idea of the framework is to express the response time of the jobs in terms of two contributing factors that directly affect a scheduler's competitive ratio, and to relate them to the lower bounds of the jobs' mean response time. The competitive ratio of the scheduler is then obtained through minimizing the constant coefficients of the lower bounds. In this paper, we focus on batched parallel jobs modeled by DAGs. However,

the same analysis can be applied to other job models as well, such as parallelism profiles and multiple phases of speedup functions, etc.

We choose to present the algorithms using a common two-level adaptive scheduling architecture [19, 1, 21], where the scheduling of jobs is explicitly divided into two separate levels. At the system level, an OS allocator decides the processor allocations for the jobs, and at the user level a task scheduler schedules each job with the allotted processors. Moreover, in order to utilize processors more efficiently, the task schedulers also provide feedbacks to the OS allocator in terms of each job's processor desire, based on which the OS allocator reallocates the processors periodically. We will show that many scheduling algorithms can be expressed in this architecture, and that it also provides a unified platform, on which our analysis framework can be conveniently presented. In the two-level context, we rename EQUI and DEQ algorithms to XEQUI and IGDEQ, respectively, for reasons to be explained in detail in Section 6.

Using our analysis framework, we improve the competitive ratios of IGDEQ and AGDEQ to 3 and 5.24 from their previously known results of 4 and 10 proved by Deng et al. [14, 15] and He et al. [21], respectively. We also show that the same best known ratio of 3.73 can be obtained for EQUI using the analysis framework as previously proved by Edmonds et al. [17]. Moreover, we present a round-robin strategy that augments the two-level adaptive algorithms in heavily-loaded systems, where the number of jobs is more than the number of processors and no fractional processor allotments are allowed. We prove that slightly larger ratios can be obtained by the three algorithms considered in this stricter model and that the same results can be carried over to the corresponding round-robin augmented scheduler, provided that certain conditions are satisfied. The framework presented in this paper naturally extends the analysis used previously in [14, 15, 17, 21], and we believe that it can potentially be applied to analyze other scheduling algorithms as well.

The remainder of this paper is organized as follows. Section 2 gives a formal description of the job model, the scheduling model, and the objective function. Section 3 describes the lower bounds for the mean response time of a batched set of parallel jobs. Section 4 presents the two-level scheduling architecture, followed by the description of the analysis framework in Section 5. Section 6 applies the analysis framework to XEQUI, IGDEQ and AGDEQ algorithms and shows their respective competitive ratios. Section 7 presents the round-robin augmented scheduling paradigm and gives the revised ratios. Finally, Section 8 concludes the paper.

2 Models and Objective Function

We model the execution of a parallel job J_i as a dynamically unfolding directed acyclic graph (DAG) such that $J_i = (V_i, E_i)$, where V_i and E_i represent the set of vertices and the set of edges for J_i, respectively. Each vertex $v \in V_i$ represents a unit-time task, and each edge $e \in E_i$ represents a dependency between two tasks. The ***work*** $T_1(J_i)$ of job J_i corresponds to the total number of vertices in the DAG, i.e., $T_1(J_i) = |V_i|$. The ***span*** $T_\infty(J_i)$ of job J_i corresponds to the number of vertices on the longest dependency chain in the DAG. A task is said to be ***ready*** for execution if all of its parents have been executed. The release time r_i of job J_i is the time when the job becomes first available for processing. For a batched job set $\mathcal{J}$, we assume that all jobs are released at time 0, i.e., $r_i = 0$ for all $J_i \in \mathcal{J}$.

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

Our objective is to minimize the mean response time for the job set, which is defined as follows.

Definition 1 The ***response time*** of a job J_i under a schedule χ is its completion time $T_\chi(J_i)$. The ***total response time*** of a job set $\mathcal{J}$ under a schedule χ is given by $R_\chi(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} T_\chi(J_i)$ and the ***mean response time*** is $\overline{R}_\chi(\mathcal{J}) = R_\chi(\mathcal{J}) / |\mathcal{J}|$.

We evaluate the mean response time of an online scheduling algorithm using competitive analysis, in which the online algorithm is compared to the optimal scheduler in terms of its competitive ratio. Specifically, let $\chi(A)$ denote the schedule produced by an online algorithm A for a job set $\mathcal{J}$. Then algorithm A is said to be ***c-competitive*** if there exists a constant b such that $\overline{R}_{\chi(A)}(\mathcal{J}) \le c \cdot \overline{R}^*(\mathcal{J}) + b$ holds for any job set $\mathcal{J}$, where $\overline{R}^*(\mathcal{J})$ denotes the mean response time of job set $\mathcal{J}$ scheduled by an optimal scheduler.

3 Mean Response Time Lower Bounds

This section describes two lower bounds for the mean response time of batched parallel jobs. We first present the following definitions, which have been shown to be the fundamental properties related to the mean response time of parallel jobs in batched scenario [38, 39, 15, 17, 21].

Definition 2 Given a finite list $\mathcal{A} = \langle \alpha_i \rangle$ of n nonnegative numbers, let $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ denote a permutation on $\{1, 2, \ldots, n\}$ that satisfies $\alpha_{f(1)} \leq \alpha_{f(2)} \leq \cdots \leq \alpha_{f(n)}$. The ***squashed sum*** of $\mathcal{A}$ is defined as

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

It is not hard to see that the above permutation f on list $\mathcal{A}$ gives the minimum value for the squashed sum formulation described by Equation (2). Thus, an alternative definition for the squashed sum of $\mathcal{A}$ is given by

$$\text{sq-sum}(\mathcal{A}) = \min_{g \in \Upsilon} \sum_{i=1}^{n} (n - i + 1)\alpha_{g(i)},$$

where $\Upsilon = \{g | g : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}\}$ denotes the set of all permutations on $\{1, 2, \ldots, n\}$.

Definition 3 The ***squashed work area*** of a job set $\mathcal{J}$ on a set of P processors is defined as

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

where $T_1\left(J_i\right)$ is the work of job $J_i \in \mathcal{J}$.

Definition 4 The ***aggregate span*** of a job set $\mathcal{J}$ is defined as

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

where $T_\infty\left(J_i\right)$ is the span of job $J_i \in \mathcal{J}$.

The research in [38, 39, 15, 21] has established the following two lower bounds for the mean response time of any batched parallel job set $\mathcal{J}$:

$$\overline{R}^*(\mathcal{J}) \geq \text{swa}\left(\mathcal{J}\right) / \left|\mathcal{J}\right|, \quad (1)$$

$$\overline{R}^*(\mathcal{J}) \geq T_\infty\left(\mathcal{J}\right) / \left|\mathcal{J}\right|. \quad (2)$$

Hence, both the aggregate span $T_\infty\left(\mathcal{J}\right)$ and the squashed work area $\text{swa}\left(\mathcal{J}\right)$ are lower bounds for the total response time $R^*(\mathcal{J})$ of job set $\mathcal{J}$.

4 Two-level Adaptive Scheduling Architecture: XY Algorithm

In this section, we describe an adaptive scheduling architecture [1, 21], where the scheduling of parallel jobs is divided into two distinct levels. At the system level, an *OS allocator* decides the processor allocations for all jobs in the system, and a *task scheduler* for each job at the user level schedules the tasks of the job with the allotted processors. When a specific task scheduler X and a specific OS allocator Y are used, we call the resulting two-level adaptive scheduling algorithm XY.

Typically for two-level adaptive scheduling, the execution of the jobs are carried out in scheduling quanta, and the processors are only reallocated by the OS allocator after a quantum expires. Throughout this paper, in order to focus on the general framework and to simplify its analysis, we assume that the length of the scheduling quantum is chosen to be the execution time of a unit-size task, that is, the processors are reallocated after each time step. As has been shown in [1, 21], we can also apply the same analysis to the more general case, where the length of the scheduling quantum can take an arbitrary number of time steps. Moreover, in order for the OS allocator to allocate processors more efficiently, the task scheduler at the user level can provide feedback to the OS allocator at the end of each time step indicating the job's processor desire for the next step. Depending on the task scheduler X, the processor desire can be based on the exact number of ready tasks (called *instantaneous parallelism*) of the job in the next step, or an estimate of the job's instantaneous parallelism in case that the exact parallelism is not obtainable, or any other reasonable figure that reflects the number of processors the

job can effectively utilize. In addition, depending on the OS allocator Y, the processor desires provided by the task scheduler may be used differently (or may not be used at all) when deciding the processor allotments to the jobs.

We define the following notations. At time step t, let $d(J_i, t)$ and $a(J_i, t)$ denote the processor desire and the processor allotment for job J_i, respectively. We assume that no matter what task scheduler X is used, it schedules the ready tasks of the job in a greedy manner. That is, at any time t with $a(J_i, t)$ processors, if job J_i has more than $a(J_i, t)$ ready tasks available, then X schedules any $a(J_i, t)$ of the ready tasks. Otherwise, X schedules all ready tasks. Hence, the only difference between different task schedulers is in their strategies for calculating the processor desires, and the only difference between different OS allocators is in their strategies for calculating processor allocations. In [22], the interaction between the task scheduler and the OS allocator is referred to as the processor *request-allocation protocol*. At each time step t in this protocol, we say that a job J_i is *satisfied* if its processor allotment is at least its processor desire, i.e., $a(J_i, t) \geq d(J_i, t)$. Otherwise, the job is *deprived* if $a(J_i, t) < d(J_i, t)$.

5 Generalized Analysis Framework

In this section, we present the generalized analysis framework for the mean response time of any two-level adaptive scheduling algorithm on batched parallel jobs. We first introduce a few preliminary concepts and notations.

Preliminaries

We need the notion of ***t-suffix*** to simplify our presentation. At a time step t, where $0 \leq t \leq T(\mathcal{J})$, t-suffix denoted as $\overrightarrow{t} = \{t, t+1, \ldots, T(\mathcal{J})\}$ represents the set of time steps from time t to the completion time $T(\mathcal{J})$ of job set $\mathcal{J}$ scheduled by a two-level algorithm XY. We will be interested in the t-suffix of the jobs, namely, the portions of jobs that remain after an arbitrary time step t. Formally, define the t-suffix of a job $J_i \in \mathcal{J}$ to be the portion of the job induced by those vertices in $V(J_i)$ that execute on or after time t, i.e., $J_i\left(\overrightarrow{t}\right) = \left(V\left(J_i\left(\overrightarrow{t}\right)\right), E\left(J_i\left(\overrightarrow{t}\right)\right)\right)$, where $V\left(J_i\left(\overrightarrow{t}\right)\right) = \{v \in V(J_i) : \tau(v) \geq t\}$ and $E\left(J_i\left(\overrightarrow{t}\right)\right) = \left\{(u, v) \in E(J_i) : u, v \in V\left(J_i\left(\overrightarrow{t}\right)\right)\right\}$. The t-suffix of job set $\mathcal{J}$ is $\mathcal{J}\left(\overrightarrow{t}\right) = \left\{J_i\left(\overrightarrow{t}\right) : J_i \in \mathcal{J} \text{ and } V\left(J_i\left(\overrightarrow{t}\right)\right) \neq \emptyset\right\}$.

We also need to define the following concepts for the jobs. At any time t, a job J_i scheduled by the XY algorithm can be characterized by a certain property, which usually indicates the relationship between the processor allotment and the parallelism of the job at time t, or the execution status of the job, etc. Our analysis relies on identifying two properties A and B for the active jobs at any time,[1] where a job is said to be *active* at time t if it is not yet completed by t. Let $\mathcal{J}(t)$ denote the set of all active jobs at time t. Then the set of active jobs that satisfy property A is denoted as $\mathcal{J}_A(t)$ and the set of active jobs that satisfy property B is denoted as $\mathcal{J}_B(t)$. The two sets $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ need not be disjoint, but they usually cover the whole set of jobs $\mathcal{J}(t)$, i.e., $\mathcal{J}_A(t) \bigcup \mathcal{J}_B(t) = \mathcal{J}(t)$. Otherwise, it can be difficult to conduct the analysis as will be seen from our examples in the later part of this section.

Let $a_A(J_i) = \sum_{t=1}^{T(\mathcal{J})} a(J_i, t) \cdot [J_i(t) \in \mathcal{J}_A(t)]$ denote the total processor allotment of job J_i when the job satisfies property A, where $[x]$ is 1 if proposition x is true and 0 otherwise. Let $s_B(J_i) = \sum_{t=1}^{T(\mathcal{J})} [J_i(t) \in \mathcal{J}_B(t)]$ denote the total number of time steps of job J_i when the job satisfies property B.

Definition 5 The ***squashed allotment area*** *with property* A for a job set $\mathcal{J}$ scheduled by XY algorithm is defined as

$$\text{saa}_A(\mathcal{J}) = \frac{1}{P} \text{sq-sum}(\langle a_A(J_i) \rangle).$$

Definition 6 The ***total steps*** *with property* B for a job set $\mathcal{J}$ scheduled by XY algorithm is defined as

$$\text{ts}_B(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} s_B(J_i).$$

[1] Two such properties may be whether a job is "satisfied" or "deprived" in its processor desire, as described in the preceding section. As will be seen shortly, the purpose of choosing the two properties is that we want to bound the jobs' response time through their use of the processors under these two scenarios separately. Identifying suitable properties, holds the key to fruitful analysis, and, as such, they do require some insights about the schedulers concerned. With the few examples demonstrated in Section 6, however, the ideas should become clearer.

Analysis Framework

The idea behind our analysis framework is to bound the total response time of the job set $\mathcal{J}$ scheduled by XY algorithm using the two lower bounds given in Section 3, namely the squashed work area and the aggregated span. In many cases, however, it can be difficult to directly associate the total response time with the two lower bounds, while it is more convenient to associate the total response time with the squashed allotment area and the total steps. Therefore, we choose to quantify the relationship among the total response time, the squashed allotment area with property A and the total steps with property B in the first step of our analysis. In the second step, we bound the squashed allotment area with property A in terms of the squashed work area, and bound the total steps with property B in terms of the aggregated span. Finally, by combining the results from these two steps, we are able to obtain the total response time of the two-level algorithm XY in terms of the two lower bounds of the job set — the squashed work area and the aggregate span. Since the optimal scheduler can do no better than these lower bounds, we hence obtain the competitive ratio of algorithm XY with respect to the total response time or equivalently the mean response time of the job set. Specifically, upon choosing the two properties A and B, our detailed analysis develops as follows:

Step (1): Bound the total response time of the XY algorithm on any set of batched parallel jobs $\mathcal{J}$ in terms of the squashed allotment area with property A and the total steps with property B. That is, find constant coefficients c_1 and c_2 such that the following inequality is satisfied:

$$R_{XY}(\mathcal{J}) \leq c_1 \cdot \mathrm{saa}_A(\mathcal{J}) + c_2 \cdot \mathrm{ts}_B(\mathcal{J}). \tag{3}$$

Step (2): Bound the squashed allotment area with property A in terms of the squashed work area and bound the total steps with property B in terms of the aggregate span for the job set $\mathcal{J}$ scheduled by the XY algorithm. That is, find constant coefficients c_3,c_4 and c_5 such that the following inequalities are satisfied:

$$\begin{aligned} \mathrm{saa}_A(\mathcal{J}) &\leq c_3 \cdot \mathrm{swa}(\mathcal{J}), & (4)\\ \mathrm{ts}_B(\mathcal{J}) &\leq c_4 \cdot T_\infty(\mathcal{J}) + c_5 \cdot n. & (5) \end{aligned}$$

Now, combining the results from step (1) and step (2) above, we can show that the total response time of the job set scheduled by the XY algorithm satisfies

$$R_{XY}(\mathcal{J}) \leq C_1 \cdot \mathrm{swa}(\mathcal{J}) + C_2 \cdot T_\infty(\mathcal{J}) + C_3 \cdot n,$$

where $C_1 = c_1 \cdot c_3$, $C_2 = c_2 \cdot c_4$ and $C_3 = c_2 \cdot c_5$. Thus, given the mean response time lower bounds of the job set in Inequalities (1) and (2), we have $\overline{R}_{XY}(\mathcal{J}) \leq (C_1 + C_2) \cdot \overline{R}^*(\mathcal{J}) + C_3$, which indicates that the XY algorithm is $(C_1 + C_2)$-competitive.

In the remaining of this section, we will elaborate Step (1) and Step (2) of the above framework. In Section 6, we will apply this analysis framework to prove the competitive ratios for three scheduling algorithms.

Elaboration of step (1)

Step (1) is to relate the total response time to the squashed allotment area with property A and the total steps with property B for job set $\mathcal{J}$ so that Inequality (3) holds. To prove Inequality (3), we can use induction on the t-suffix of the job set $\mathcal{J}\left(\overrightarrow{t}\right)$ scheduled by the XY algorithm with values of c_1 and c_2 to be determined.

Base case: Let $T(\mathcal{J})$ denote the completion time of the job set $\mathcal{J}$, and let $t = T(\mathcal{J}) + 1$. In this case, we have $\mathcal{J}\left(\overrightarrow{t}\right) = \emptyset$. It follows that $R_{XY}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) = 0$, $\mathrm{saa}_A\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) = 0$, and $\mathrm{ts}_B\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) = 0$. Thus, Inequality (3) holds trivially regardless of the values of c_1 and c_2.

Inductive step: Suppose that Inequality (3) holds at time step $t+1$, that is, the inductive hypothesis is

$$R_{XY}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) \leq c_1 \cdot \mathrm{saa}_A\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) + c_2 \cdot \mathrm{ts}_B\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right). \tag{6}$$

We will show that Inequality (3) also holds at time step t, that is,

$$R_{XY}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) \leq c_1 \cdot \mathrm{saa}_A\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) + c_2 \cdot \mathrm{ts}_B\left(\mathcal{J}\left(\overrightarrow{t}\right)\right). \tag{7}$$

Let $\Delta\,\mathrm{r}_{XY}$ denote the change of the total response time, $\Delta\,\mathrm{saa}_A$ the squashed allotment area, and $\Delta\,\mathrm{ts}_B$ the total steps from time t to $t+1$, i.e.,

$$\begin{aligned} \Delta\,\mathrm{r}_{XY} &= R_{XY}\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - R_{XY}\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right),\\ \Delta\,\mathrm{saa}_A &= \mathrm{saa}_A\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - \mathrm{saa}_A\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right),\\ \Delta\,\mathrm{ts}_B &= \mathrm{ts}_B\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - \mathrm{ts}_B\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right). \end{aligned}$$

Thus, we need only prove the following inequality in order to show that Inequality (7) holds,

$$\Delta \mathrm{r}_{XY} \leq c_1 \cdot \Delta \mathrm{saa}_A + c_2 \cdot \Delta \mathrm{ts}_B \,. \tag{8}$$

Since each job in $\mathcal{J}(t)$ contributes 1 unit of time to the total response time from time t to $t+1$, we have

$$\Delta \mathrm{r}_{XY} = |\mathcal{J}(t)| \,, \tag{9}$$

and since each job in $\mathcal{J}_B(t)$ reduces its total steps with property B by 1 from time t to $t+1$, we have

$$\Delta \mathrm{ts}_B = |\mathcal{J}_B(t)| \,. \tag{10}$$

The change of squashed allotment area $\Delta \mathrm{saa}_A$ with property A is not so straightforward and may depend on the specific OS allocator Y and the choice of property A. Now, substituting Equations (9) and (10) into Inequality (8), we need only prove the following inequality

$$|\mathcal{J}(t)| \leq c_1 \cdot \Delta \mathrm{saa}_A + c_2 \cdot |\mathcal{J}_B(t)| \,, \tag{11}$$

which can be satisfied by *choosing appropriate values for* c_1 *and* c_2 upon obtaining the lower bound on $\Delta \mathrm{saa}_A$. Inequality (11) may be difficult to satisfy if job sets $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ do not cover the whole job set $\mathcal{J}(t)$ at time t. Thus, the properties A and B are usually chosen such that $\mathcal{J}_A(t) \bigcup \mathcal{J}_B(t) = \mathcal{J}(t)$, which indicates $|\mathcal{J}_A(t)| + |\mathcal{J}_B(t)| \geq |\mathcal{J}(t)|$.

Elaboration of step (2)

Step (2) is to relate the squashed allotment area with property A to the squashed work area so that Inequality (4) holds, and to relate the total steps with property B to the aggregated span so that Inequality (5) holds. To establish Inequalities (4) and (5), we can analyze, for each job J_i scheduled by the task scheduler X, relationship between the total allotment of the job with property A and the work of the job, as well as relationship between the total number of steps of the job with property B and the span of the job. That is, we can establish the following relations:

$$a_A(J_i) \leq c_3 \cdot T_1(J_i), \tag{12}$$

$$s_B(J_i) \leq c_4 \cdot T_\infty(J_i) + c_5, \tag{13}$$

where the *values of* c_3*,* c_4 *and* c_5 *depend on the specific task scheduler X used.*

First, we present a lemma that will be useful to the proof of Inequality (4).

Lemma 1 *Let* $\langle \alpha_i \rangle$ *and* $\langle \beta_i \rangle$ *be two lists of nonnegative numbers with* n *elements each. Suppose that* $\alpha_i \leq \beta_i$ *for* $i = 1, 2, \ldots, n$*, then we have*

$$\text{sq-sum}(\langle \alpha_i \rangle) \leq \text{ sq-sum}(\langle \beta_i \rangle).$$

Proof. Let $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ be a permutation satisfying $\alpha_{f(1)} \leq \alpha_{f(2)} \leq \ldots \leq \alpha_{f(n)}$, and let $g : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ be a permutation satisfying $\beta_{g(1)} \leq \beta_{g(2)} \leq \ldots \leq \beta_{g(n)}$. We will show that $\alpha_{f(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, n$. Then, directly from the definition of squashed sum, the claim holds.

We prove $\alpha_{f(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, n$ by contradiction. Suppose that there exists $j \in \{1, 2, \ldots, n\}$ such that $\alpha_{f(j)} > \beta_{g(j)}$. Then, there must be at least j numbers less than $\alpha_{f(j)}$ in $\langle \beta_i \rangle$, namely $\beta_{g(1)}, \beta_{g(2)}, \ldots, \beta_{g(j)}$. Since $\alpha_i \leq \beta_i$ for $i = 1, 2, \ldots, n$, we have $\alpha_{g(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, j$. Thus, there are at least j numbers less than $\alpha_{f(i)}$ in $\langle \alpha_i \rangle$, namely, $\alpha_{g(1)}, \alpha_{g(2)}, \ldots, \alpha_{g(j)}$. However, since $\alpha_{f(j)}$ is the jth smallest number in $\langle \alpha_i \rangle$, we obtain a contradiction that there are at most $j - 1$ numbers less than $\alpha_{f(j)}$ in $\langle \alpha_i \rangle$, thereby establishing $\alpha_{f(i)} \leq \beta_{g(i)}$ for $i = 1, 2, \ldots, n$. □

By applying Lemma 1 and Inequality (12), and according to Definitions 3 and 5, we obtain

$$\begin{aligned} \mathrm{saa}_A(\mathcal{J}) &= \frac{1}{P} \text{sq-sum}(\langle a(J_i) \rangle) \\ &\leq \frac{1}{P} \text{sq-sum}(\langle c_3 \cdot T_1(J_i) \rangle) \\ &= c_3 \cdot \frac{1}{P} \text{sq-sum}(\langle T_1(J_i) \rangle) \\ &= c_3 \cdot \mathrm{swa}(\mathcal{J}) \,. \end{aligned}$$

To prove Inequality (5), we can simply sum up the number of steps that satisfy property B given in Inequality (13) over all jobs, that is,

$$\begin{aligned} \mathrm{ts}_B(\mathcal{J}) &= \sum_{J_i \in \mathcal{J}} s_B(J_i) \\ &\leq \sum_{J_i \in \mathcal{J}} (c_4 \cdot T_\infty(J_i) + c_5) \\ &\leq c_4 \cdot \sum_{J_i \in \mathcal{J}} T_\infty(J_i) + c_5 \cdot n \\ &= c_4 \cdot T_\infty(\mathcal{J}) + c_5 \cdot n. \end{aligned}$$

Remarks. The analysis presented in this section suggests that in order to conclude on the mean response time performance of a two-level algorithm XY on batched parallel jobs, we need only choose two appropriate properties A and B for the jobs according to the scheduling algorithm analyzed, and then search for a lower bound on the change of the squashed allotment area $\Delta \operatorname{saa}_A$ with property A in a time step, as well as for the values of c_3 and c_4 that satisfy Inequalities (12) and (13) with property A and B, respectively.

6 Applications of the Analysis Framework

In this section, we will describe two OS allocators and two task schedulers, which when combined together can form three different two-level adaptive scheduling algorithms. We apply the generalized analysis framework given in the preceding section to the three algorithms and show their mean response time performances on batched parallel jobs.

Equi-Partitioning (EQUI) OS allocator

Equi-Partitioning (EQUI) [37, 17] OS allocator divides the total number of processors equally among all active jobs in the system at each time step. Hence, each job receives $P/|\mathcal{J}(t)|$ processors at each time t regardless of the task scheduler's processor desire. Reallocation of processors only occurs when a job finishes execution and leaves the system. In this section, as with [15, 17, 21], we assume that the processor allotments can be non-integral. The fractional allotment to a job can be interpreted as time-sharing a processor with other jobs. In the next section, we will present strategies for dealing with strictly integral allotments.

Dynamic Equi-Partitioning (DEQ) OS allocator

Dynamic Equi-Partitioning (DEQ) [30, 14, 15] OS allocator shares the total number of processors among all active jobs in the system similarly as EQUI, except that DEQ never allots more processors to a job than the job's desire. Let $\mathcal{J}(t)$ denote the set of active jobs at time t. Based on the processor desires from all jobs in $\mathcal{J}(t)$, DEQ allocates the processors as follows:

```
DEQ (t, J(t), P)
 1  if J(t) = ∅    return
 2  S ← {J_i ∈ J(t) : d(J_i, t) ≤ P/|J(t)|}
 3  if S = ∅
 4      for each J_i ∈ J(t)
 5              a(J_i, t) ← P/|J(t)|
 6      return
 7  else
 8      for each J_i ∈ S
 9              a(J_i, t) ← d(J_i, t)
10      DEQ (t, J(t) − S, P − Σ_{J_i ∈ S} a(J_i, t))
```

As can be seen in the above 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 2 and line 9). After that, the equal share of processors will be recalculated excluding the jobs already satisfied and the processors already allocated. The remaining processors will then be allocated to the remaining jobs by recursively calling the main procedure (line 10) until all jobs' processor desires are more than the equal share, in which case each remaining job will be deprived and

get the current equal share of processors (lines 3-6). According to this algorithm, if there are deprived jobs at a time step t, then all P processors must have been allocated at t. In addition, all deprived jobs have the same number of allotted processors, which is greater than the number of processors allotted to any satisfied job, and is greater than the initial equal share $P/|\mathcal{J}(t)|$.

Instantaneous Greedy (IG) task scheduler

Instantaneous Greedy (IG) task scheduler simply uses the number of ready tasks, or the instantaneous parallelism of a job at any time step t as the processor desire for the job. Again, we assume that the instantaneous parallelism (or the number of ready tasks) of a job may be non-integral, resulted from executing the job with fractional allotments in previous time steps.

Adaptive Greedy (AG) task scheduler

Adaptive Greedy (AG) task scheduler [1] estimates a job's processor desire for a time step based on the execution characteristics of the job in the previous step, namely whether the processor desire has been satisfied and whether the allotted processors have been efficiently utilized. Specifically, let $u(J_i, t)$ denote the work completed by AG on time step t. Then AG calculates the processor desire $d(J_i, t)$ for any step $t > 1$ with a multiplicative-increase multiplicative-decrease (MIMD) strategy as follows:

```
AG (t)
1  if u(J_i, t-1) < a(J_i, t-1)
2      then d(J_i, t) = d(J_i, t-1)/2    ▷ inefficient
3  elseif a(J_i, t-1) ≥ d(J_i, t-1)
4      then d(J_i, t) = 2d(J_i, t-1) ▷ efficient and satisfied
5  else d(J_i, t) = d(J_i, t-1) ▷ efficient and deprived
```

The processor desire for the very first step is set to the total number of processors P, i.e., $d(J_i, 1) = P$. Unlike IG task scheduler, AG is not utilizing the instantaneous parallelism of the job and hence is *non-clairvoyant*, which is desirable if information about the job's instantaneous parallelism is difficult to obtain. The rationale of the MIMD strategy of AG is as follows. If the allotted processors in the previous step were not utilized efficiently, then the parallelism of the job may not be as high. Therefore the processor desire is reduced by a factor of 2 for the current step (line 1 and line 2). If the allotted processors were utilized efficiently and the processor desire was satisfied, then the parallelism of the job could be even higher. Thus, to execute the job more quickly, the processor desire is increased by a factor of 2 for the current step (line 3 and line 4). Lastly, if the allotted processors were utilized efficiently but the desire was 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 current step (line 5). In [1], a job is referred to as *accounted* at a time step t if the job is both deprived in its processor desire as well as efficient in its processor utilization at t. Otherwise, the job is *deductible*.

Note that we modify the original AG algorithm proposed in [1] in two ways. Firstly, we set the multiplicative factor to 2 while in [1] it can take any value greater than 1. Secondly, we set the initial processor desire to P while in [1] it is set to 1. The modification is because the multiplicative factor and the initial desire only affect the processor waste of the job on deductable time steps [1], which we show in this paper is independent of mean response time for the job set. In addition, as will be seen in the later part of the section, the second modification also tightens the deductible time of the job by a logarithmic factor in terms of the total number of processors in comparison to the bound given in [1].

Now, combining the two OS allocators and the two task schedulers presented so far, we get three different two-level scheduling algorithms, namely XEQUI, IGDEQ, AGDEQ. Since EQUI OS allocator does not utilize the processor desires from the task scheduler, no matter what task scheduler X is coupled with EQUI, the same two-level algorithm will result as far as the jobs' processor allotments are concerned. Hence, we use "XEQUI" to denote the two-level algorithm that couples EQUI with any task scheduler. From the analysis to be given shortly, however, we can see that the competitive ratio of XEQUI is minimized when EQUI is "coupled" with IG task scheduler for analytical purposes. Table 1 summarizes the constant coefficients as well as the competitive ratios for the three two-level algorithms. Their detailed analysis is presented in the following subsections.

Table 1: The constant coefficients and the competitive ratios for XEQUI, IGDEQ and AGDEQ.

	c_1	c_2	c_3	c_4	$C_1 + C_2$
XEQUI	2.15	1.58	1	1	3.73
IGDEQ	2	1	1	1	3
AGDEQ	2.34	1.45	1	2	5.24

XEQUI two-level scheduler

XEQUI couples the OS allocator EQUI with the task scheduler IG. The choice of IG as the task scheduler gives the best competitive ratio for XEQUI. We choose jobs with property A to be the "deprived" jobs and jobs with property B to be the "satisfied" jobs for XEQUI algorithm.

According to Step (1) given in Section 5, we need to derive the change of the squashed allotment area $\Delta \operatorname{saa}_A$ for jobs that are deprived from time t to $t+1$. The following lemma shows a property of the squashed sum drawing analogy from the XEQUI algorithm.

Lemma 2 *Let $\langle \alpha_i \rangle$ and $\langle \beta_i \rangle$ be two lists of nonnegative numbers with n elements each, and let $h \geq 0$ be any number. Suppose that $\beta_i = \alpha_i + s_i$, where s_i is either h or 0 for $i = 1, 2, \ldots, n$. Let l denote the number of instances of s_i that have value h. We have*

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

Proof. Let $\Upsilon = \{g | g : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}\}$ denote the set of all permutations on $\{1, 2, \ldots, n\}$. Let $f \in \Upsilon$ denote a permutation that satisfies $\beta_{f(1)} \leq \beta_{f(2)} \leq \cdots \leq \beta_{f(n)}$. Then, according to Definition 2, we have

$$\begin{aligned}
\text{sq-sum}\left(\langle \beta_i \rangle\right) &= \sum_{i=1}^{n} (n-i+1)\beta_{f(i)} \\
&= \sum_{i=1}^{n} (n-i+1)(\alpha_{f(i)} + s_{f(i)}) \\
&= \sum_{i=1}^{n} (n-i+1)\alpha_{f(i)} + \sum_{i=1}^{n} (n-i+1)s_{f(i)} \\
&\geq \min_{g \in \Upsilon} \sum_{i=1}^{n} (n-i+1)\alpha_{g(i)} + \min_{k \in \Upsilon} \sum_{i=1}^{n} (n-i+1)s_{k(i)} \\
&= \text{sq-sum}\left(\langle \alpha_i \rangle\right) + \text{sq-sum}\left(\langle s_i \rangle\right) .
\end{aligned}$$

The squashed sum of $\langle s_i \rangle$, according to definition, is simply given by $\text{sq-sum}\left(\langle s_i \rangle\right) = \sum_{i=n-l+1}^{n} (n-i+1) \cdot h = \sum_{i=1}^{l} i \cdot h = hl(l+1)/2$. □

For the analysis of XEQUI, we now use Lemma 2 to complete Step (1). Let α_i and β_i represent the total deprived allotment for a job J_i from time $t+1$ and t onwards, respectively, i.e., $\alpha_i = \sum_{t'=t+1}^{T(\mathcal{J})} a(J_i, t') \cdot [a(J_i, t') < d(J_i, t')]$ and $\beta_i = \sum_{t'=t}^{T(\mathcal{J})} a(J_i, t') \cdot [a(J_i, t') < d(J_i, t')]$. Let $s_i = P / |\mathcal{J}(t)|$ denote the deprived allotment for job J_i on time t. By Definition 5 and Lemma 2, we have

$$\begin{aligned}
\Delta \operatorname{saa}_A &= \operatorname{saa}_A\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - \operatorname{saa}_A\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) \\
&= \frac{1}{P}\left(\text{sq-sum}(\langle \beta_i \rangle) - \text{sq-sum}(\langle \alpha_i \rangle)\right) \\
&\geq \frac{|\mathcal{J}_A(t)|\left(|\mathcal{J}_A(t)| + 1\right)}{2\,|\mathcal{J}(t)|} .
\end{aligned} \tag{14}$$

Substituting Inequality (14) into Inequality (11), we need to choose values for c_1 and c_2 to satisfy

$$|\mathcal{J}(t)| \leq c_1 \cdot \frac{|\mathcal{J}_A(t)|\left(|\mathcal{J}_A(t)| + 1\right)}{2\,|\mathcal{J}(t)|} + c_2 \cdot |\mathcal{J}_B(t)| . \tag{15}$$

Given $|\mathcal{J}(t)| = |\mathcal{J}_A(t)| + |\mathcal{J}_B(t)|$, we can simplify Inequality (15) to

$$(c_1 - 2)\,|\mathcal{J}_A(t)|^2 + (2c_2 - 4)\,|\mathcal{J}_A(t)|\,|\mathcal{J}_B(t)| + (2c_2 - 2)\,|\mathcal{J}_B(t)|^2 \geq 0 . \tag{16}$$

One set of sufficient conditions that satisfy Inequality (16) is given by

$$c_1 - 2 \geq 0 \quad \Longrightarrow \quad c_1 \geq 2 \ , \tag{17}$$

$$2c_2 - 2 \geq 0 \quad \Longrightarrow \quad c_2 \geq 1 \ , \tag{18}$$

$$4(c_1 - 2)(2c_2 - 2) \geq (2c_2 - 4)^2 \quad \Longrightarrow \quad 2c_1c_2 - 2c_1 \geq c_2^2 \ . \tag{19}$$

For task scheduler IG, since all processors allotted to deprived jobs will be used efficiently on the work of the jobs, consequently, we have $a_A(J_i) \leq T_1(J_i)$, which according to the elaboration of Step (2) gives $\mathrm{saa}_A(\mathcal{J}) \leq \mathrm{swa}(\mathcal{J})$.

On a satisfied time step for a job scheduled by task scheduler IG, the number of allotted processors is at least the instantaneous parallelism of the job. Therefore, all ready tasks of the job on that step will be executed, which will result in the span of the job being reduced by 1. Thus, the total number of satisfied steps for the job is no more than the total span of the job, i.e., $s_B(J_i) \leq T_\infty(J_i)$, which gives $\mathrm{ts}_B(\mathcal{J}) \leq T_\infty(\mathcal{J})$.

Now, minimizing $c_1 + c_2$ with the constraints given in Inequalities (17), (18) and (19) yields $c_1 = 1 + 2/\sqrt{3} \approx 2.15$ and $c_2 = 1 + 1/\sqrt{3} \approx 1.58$. Therefore, the competitive ratio of XEQUI is $c_1 + c_2 = 2 + \sqrt{3} \approx 3.73$.

IGDEQ two-level scheduler

IGDEQ couples the OS allocator DEQ with the task scheduler IG. We choose jobs with property A to be "all" jobs in the system and jobs with property B to be the "satisfied" jobs.

We need to derive the change of the squashed allotment area $\Delta\,\mathrm{saa}_A$ for all jobs from time t to $t+1$. The following lemma abstracts the processor allocations for the DEQ algorithm and is helpful for our derivation.

Lemma 3 *Let $\langle \alpha_i \rangle$ and $\langle \beta_i \rangle$ be two lists of nonnegative numbers with n elements each, and let $h \geq 0$ be any number. Suppose that $\beta_i = \alpha_i + s_i$, where $0 \leq s_i \leq h$ for $i = 1, 2, \ldots, n$, and $\sum_{i=1}^{n} s_i = P$. Let l denote the number of instances of s_i that have value h. If $l > 0$, then we have*

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

Proof. From the proof of Lemma 2, we can see that Inequality (14) still holds here. Therefore, we need only show that $\text{sq-sum}(\langle s_i \rangle) \geq P(l+1)/2$.

To simplify the notations, rename the elements of list $\langle s_i \rangle$ such that $s_1 \leq s_2 \leq \cdots \leq s_{n-l} < s_{n-l+1} = s_{n-l+2} = \cdots = s_n = h$. Since $\sum_{i=1}^{n} s_i = P$, we have $h = (P - \sum_{i=1}^{n-l} s_i)/l$. The squashed sum of $\langle s_i \rangle$ is thus given by

$$\begin{aligned}
\text{sq-sum}(\langle s_i \rangle) &= \sum_{i=1}^{n} (n-i+1)s_i \\
&= \sum_{i=1}^{n-l} (n-i+1)s_i + \sum_{i=n-l+1}^{n} (n-i+1)s_i \\
&= \sum_{i=1}^{n-l} (n-i+1)s_i + \frac{(P - \sum_{i=1}^{n-l} s_i)}{l} \sum_{i=1}^{l} i \\
&= \sum_{i=1}^{n-l} (n-i+1)s_i + \frac{(P - \sum_{i=1}^{n-l} s_i)}{l} \cdot \frac{l(l+1)}{2} \\
&= \frac{\sum_{i=1}^{n-l} (2n-2i+2)s_i}{2} + \frac{(P - \sum_{i=1}^{n-l} s_i)(l+1)}{2} \\
&= \frac{P(l+1)}{2} + \frac{\sum_{i=1}^{n-l} (2n-2i-l+1)s_i}{2} \\
&\geq \frac{P(l+1)}{2}
\end{aligned}$$

The last inequality holds because $s_i \geq 0$ and $2n - 2i - l + 1 > 0$ for $i = 1, 2, \ldots, n-l$. □

Let α_i and β_i denote the total allotment for job J_i from time $t+1$ and t onwards, respectively, i.e., $\alpha_i = \sum_{t'=t+1}^{T(\mathcal{J})} a(J_i, t')$ and $\beta_i = \sum_{t'=t}^{T(\mathcal{J})} a(J_i, t')$. Let s_i denote the processor allotment for job J_i on time t. We assume that there is at least one deprived job on time t; otherwise Inequality (11) will hold trivially as long as

$c_2 \geq 1$. Thus, we have $\sum_{i=1}^{n} s_i = P$. Let $\mathcal{J}_D$ denote the set of deprived jobs at time t. By Definition 5 and Lemma 3, we have

$$\begin{aligned} \Delta \operatorname{saa}_A &= \operatorname{saa}_A\left(\mathcal{J}\left(\overrightarrow{t}\right)\right) - \operatorname{saa}_A\left(\mathcal{J}\left(\overrightarrow{t+1}\right)\right) \\ &= \frac{1}{P}\left(\text{sq-sum}(\langle\beta_i\rangle) - \text{sq-sum}(\langle\alpha_i\rangle)\right) \\ &\geq \frac{|\mathcal{J}_D(t)|+1}{2}. \end{aligned} \tag{20}$$

Substituting Inequality (20) into Inequality (11), we need to choose values for c_1 and c_2 to satisfy

$$|\mathcal{J}(t)| \leq c_1 \cdot \frac{|\mathcal{J}_D(t)|+1}{2} + c_2 \cdot |\mathcal{J}_B(t)|\,. \tag{21}$$

Given $|\mathcal{J}(t)| = |\mathcal{J}_D(t)| + |\mathcal{J}_B(t)|$, the choices of $c_1 \geq 2$ and $c_2 \geq 1$ can obviously satisfy Inequality (21).

Since DEQ never allots more processors than a job desires, all allotted processors will be efficiently utilized by the task scheduler IG. Hence, we also have $c_3 = 1$ and $c_4 = 1$ in this case. Therefore, taking the minimum values of c_1 and c_2, the competitive ratio of IGDEQ is given by $c_1 + c_2 = 3$.

AGDEQ two-level scheduler

AGDEQ couples the OS allocator DEQ with the task scheduler AG. We choose jobs with property A to be the "accounted" jobs and jobs with property B to be the "deductible" jobs. Recall that a job is accounted if it is both deprived and efficient, and a job is deductible if it is either satisfied or inefficient.

Since an accounted job on time t is also deprived, according to the DEQ algorithm, all accounted jobs get the same allotment, which is greater than the equal share of $P/|\mathcal{J}(t)|$. The scenario described by Lemma 2 is also applicable to the accounted jobs here, and therefore the change of the squashed allotment area $\Delta \operatorname{saa}_A$ for the accounted jobs from time t to $t+1$ satisfies Inequality (14) as well. Thus, the values of c_1 and c_2 are also constrained by Inequalities (17), (18) and (19).

In addition, an accounted job J_i on a time step is also efficient. Thus, all processors allotted to the accounted job are efficiently utilized. Hence, again we have $c_3 = 1$. We now bound the value of c_4. By setting the initial desire to 1, the authors in [1] have established the following bound on the total number of deductible steps for a job J_i scheduled by AG: $s_B(J_i) \leq 2T_\infty(J_i) + \lg P + 1$. This bound is obtained by considering the total number of inefficient steps and the total number of efficient and satisfied steps of the job, separately. The former is at most the span of the job, which is reduced by 1 on each inefficient step. The latter is related to the former by exploring the correspondence between the set of inefficient steps and the set of efficient and satisfied steps induced by the multiplicative-increase multiplicative-decrease strategy [3]. Adopting the same analysis, we can show that, by setting the initial desire to P, the total number of deductible steps turns out to satisfy $s_B(J_i) \leq 2T_\infty(J_i) + 1$. Hence, we have $c_4 = 2$.

Minimizing $c_1 + 2c_2$ with the constraints given in Inequalities (17), (18) and (19) yields $c_1 = 1 + 3/\sqrt{5} \approx 2.34$ and $c_2 = 1 + 1/\sqrt{5} \approx 1.45$. Therefore, the competitive ratio of AGDEQ is $c_1 + 2c_2 = 3 + \sqrt{5} \approx 5.24$.

Remarks. The competitive ratios of IGDEQ and AGDEQ shown in Table 1 improve upon the original ratios of the respective algorithms given in [14] and [21]. The improvements come from choosing more appropriate properties A and B for each algorithm, and our analysis framework given in Section 5 provides sufficient flexibility in this regard. Note that both our results shown in Table 1 and the results of [14, 17, 21] assume that fractional processor allotments are allowed. In the next section, we will present strategies that handle strictly integer allotments.

7 Round Robin Augmented Scheduling: RR-XY Algorithm

In the previous sections, we assumed that the OS allocator can allot fractional number of processors to a job. In this section, we provide strategies to restrict the allotments to strictly integer values. In particular, we present a round robin (RR) strategy that augments a two-level adaptive scheduling algorithm XY when the number of jobs in the system is more than the number of processors. We call the resulting algorithm RR-XY. We will describe the round robin strategy in detail and analyze the mean response time of RR-XY on batched parallel jobs.

First of all, when the number of jobs in the system is not more than the number of processors at time t, i.e., $|\mathcal{J}(t)| \leq P$, using EQUI as an example, we can allot $\lceil P/|\mathcal{J}(t)| \rceil$ processors to $(P \bmod |\mathcal{J}(t)|)$ arbitrarily selected jobs, and allot $\lfloor P/|\mathcal{J}(t)| \rfloor$ processors to the rest on that time step. The same strategy can be applied

to DEQ on deprived jobs, as the processor desires from both IG and AG are strictly integral as well. With this simple strategy, we observe that the changes of the squashed sum as given in Inequalities (2) and (3) will be reduced by at most a factor of 2, since in the worst case, the integral allotment to a job is at least half of its original allotment. We can then recalculate the constant coefficients and the competitive ratios for the three algorithms, and their revised values are given in Table 2.

Table 2: The constant coefficients and the competitive ratios for XEQUI, IGDEQ and AGDEQ when the processor allotments are restricted to integer values.

	c_1	c_2	c_3	c_4	$C_1 + C_2$
XEQUI	4.12	1.71	1	1	5.83
IGDEQ	4	1	1	1	5
AGDEQ	4.3	1.58	1	2	7.46

On the other hand, when the number of jobs is more than the number of processors at time t, i.e., $|\mathcal{J}(t)| > P$, we use RR-XY algorithm to schedule the jobs. RR-XY works by maintaining a queue of all jobs in the system. At the beginning of each time step, it pops P jobs from the front of the queue, and allots one processor to each of them. At the end of the step, RR-XY pushes the P jobs back to the end of the queue if they are not completed. In this paper, since we assume that all jobs in the job set $\mathcal{J}$ arrive at time step 0, the number of uncompleted jobs decreases monotonically. When the number of uncompleted jobs drops down to P or below, RR-XY switches back to the XY algorithm.

In order to analyze the performance of RR-XY, we divide the batched job set $\mathcal{J}$ into two subsets: *RR set* and *XY set*. The RR set, denoted as $\mathcal{J}_{RR}$, includes all jobs in $\mathcal{J}$ that are entirely scheduled by RR throughout their execution. The XY set, denoted as $\mathcal{J}_{XY}$, includes all jobs in $\mathcal{J}$ that are scheduled by RR at the beginning, and by XY eventually. There exists a unique time step t called the *final RR step* such that t is the last step when jobs are scheduled by RR, while from step $t+1$ onwards, XY takes over. Hence, there must be more than P uncompleted jobs at the beginning of t, and no more than P uncompleted jobs immediately after t. For convenience, Let $n = |\mathcal{J}|$, $n_1 = |\mathcal{J}_{RR}|$ and $n_2 = |\mathcal{J}_{XY}|$. Therefore, we have $n = n_1 + n_2$. Since RR always gives each job an equal share of computation time, the work done for two uncomplete jobs in $\mathcal{J}_{RR}$ at any time differs by at most one. Hence, a job in $\mathcal{J}_{RR}$ with strictly less work will be completed no later than a job in $\mathcal{J}_{RR}$ with more work. Rename the jobs according to their work such that $T_1(J_1) \le T_1(J_2) \le \cdots \le T_1(J_n)$, breaking the ties according to the completion time of the jobs. We have $\mathcal{J}_{RR} = \{J_1, J_2, \ldots, J_{n_1}\}$ and $\mathcal{J}_{XY} = \{J_{n_1+1}, J_{n_1+2}, \ldots, J_n\}$.

Now, suppose that a job $J_i \in \mathcal{J}_{RR}$ is completed at time $t = T(J_i)$, then any other job $J_j \in \mathcal{J}$ and $j < i$ should have also completed by t. For any other job $J_k \in \mathcal{J}$ and $k > i$, no more than P of them should have completed one more unit of work than the total work of J_i, and all the rest should have completed no more work than the total work of J_i. Therefore, the total amount of work completed in $\mathcal{J}$ by time t is no more than $(n-i)\,T_1(J_i) + \sum_{j=1}^{i} T_1(J_j) + P$. Since the P processors are always busy at all time under the schedule of RR, the completion time $T(J_i)$ of job J_i is thus given by

$$T(J_i) < \frac{1}{P}\left((n-i)T_1(J_i) + \sum_{j=1}^{i} T_1(J_j)\right) + 1. \tag{22}$$

We are now ready to show the performance of RR-XY algorithm. We prove in the following theorem that the competitive ratio of XY algorithm can be carried over to RR-XY algorithm, provided that certain conditions are satisfied.

Theorem 4 *Suppose that the total response time for any batched job set $\mathcal{J}$ scheduled by XY algorithm on P processors, where $|\mathcal{J}| \le P$, satisfies*

$$R_{XY}(\mathcal{J}) \le C_1 \cdot \mathrm{swa}(\mathcal{J}) + C_2 \cdot T_\infty(\mathcal{J}) + C_3 \cdot n. \tag{23}$$

Then RR-XY algorithm is $(C_1 + C_2)$-competitive with respect to the mean response time of any batched job set, provided that C_1 and C_3 are constants and $C_1 \ge 2$.

Proof. Our analysis is divided into the following three steps: (1) we calculate the total response time $R(\mathcal{J}_{RR})$ of the RR set. (2) we derive the total response time $R(\mathcal{J}_{XY})$ of the XY set. (3) we sum them up and obtain the main result.

(1) Calculate $R(\mathcal{J}_{RR})$: Since for any job $J_i \in \mathcal{J}_{RR}$, its response time is given by Inequality (22), the total response time of all jobs in $\mathcal{J}_{RR}$ is then

$$\begin{aligned} R(\mathcal{J}_{RR}) &< \frac{1}{P}\left(\sum_{i=1}^{n_1}(n-i)T_1(J_i) + \sum_{i=1}^{n_1}\sum_{j=1}^{i}T_1(J_j)\right) + n_1 \\ &= \frac{1}{P}\left(\sum_{i=1}^{n_1}(n-i)T_1(J_i) + \sum_{i=1}^{n_1}(n_1-i+1)T_1(J_i)\right) + n_1 \\ &\leq \frac{2}{P}\sum_{i=1}^{n_1}(n-i+1)T_1(J_i) - \frac{n_2}{P}\sum_{i=1}^{n_1}T_1(J_i) + n_1. \end{aligned} \tag{24}$$

(2) Calculate $R(\mathcal{J}_{XY})$: The n_2 jobs in $\mathcal{J}_{XY}$ are scheduled by RR until the time step $t = T(J_{n_1})$ at which job J_{n_1} completes, and then scheduled by XY afterwards. The total response time of $\mathcal{J}_{XY}$ is then

$$R(\mathcal{J}_{XY}) = R\left(\mathcal{J}_{XY}\left(\overrightarrow{t+1}\right)\right) + n_2 \cdot T(J_{n_1}). \tag{25}$$

For each job $J_i \in \mathcal{J}_{XY}$, the work completed by time t is at least $T_1\left(J_i\left(\overleftarrow{t}\right)\right) \geq T_1(J_{n_1}) - 1$, and the remaining work after t is then given by $T_1\left(J_i\left(\overrightarrow{t+1}\right)\right) \leq T_1(J_i) - T_1(J_{n_1}) + 1$. The squashed work area $\mathcal{J}_{XY}\left(\overrightarrow{t+1}\right)$, according to Definition 2, Equation (3) and Lemma 1, satisfies

$$\begin{aligned} \mathrm{swa}\left(\mathcal{J}_{XY}\left(\overrightarrow{t+1}\right)\right) &\leq \frac{1}{P}\sum_{i=1}^{n_2}(n_2-i+1)\left(T_1(J_i) - T_1(J_{n_1}) + 1\right) \\ &= \frac{1}{P}\sum_{i=n_1+1}^{n}(n-i+1)\left(T_1(J_i) - T_1(J_{n_1}) + 1\right) \\ &= \frac{1}{P}\sum_{i=n_1+1}^{n}(n-i+1)T_1(J_i) - \frac{n_2(n_2+1)}{2P}T_1(J_{n_1}) + \frac{n_2(n_2+1)}{2P} \\ &\leq \frac{1}{P}\sum_{i=n_1+1}^{n}(n-i+1)T_1(J_i) - \frac{n_2(n_2+1)}{2P}T_1(J_{n_1}) + P. \end{aligned} \tag{26}$$

The last step of Inequality (26) is because $n_2 \leq P$ from the description of the RR-XY algorithm. Now, with condition $C_1 \geq 2$, and substituting Inequalities (22), (23) and (26) into Equation (25), we have

$$\begin{aligned} R(\mathcal{J}_{XY}) &= R\left(\mathcal{J}_{XY}\left(\overrightarrow{t+1}\right)\right) + n_2 \cdot T(J_{n_1}) \\ &\leq C_1 \cdot \mathrm{swa}\left(\mathcal{J}_{XY}\left(\overrightarrow{t+1}\right)\right) + C_2 \cdot T_\infty\left(\mathcal{J}_{XY}\left(\overrightarrow{t+1}\right)\right) + C_3 \cdot n_2 + n_2 \cdot T(J_{n_1}) \\ &\leq \frac{C_1}{P}\sum_{i=n_1+1}^{n}(n-i+1)T_1(J_i) - \frac{C_1}{2}\cdot\frac{n_2(n_2+1)}{P}T_1(J_{n_1}) + C_1 \cdot P \\ &\quad + C_2 \cdot T_\infty(\mathcal{J}) + C_3 \cdot n_2 + \frac{n_2}{P}\left(n_2 T_1(J_{n_1}) + \sum_{i=1}^{n_1}T_1(J_i)\right) + n_2 \\ &\leq \frac{C_1}{P}\sum_{i=n_1+1}^{n}(n-i+1)T_1(J_i) + C_2 \cdot T_\infty(\mathcal{J}) + \frac{n_2}{P}\sum_{i=1}^{n_1}T_1(J_i) \\ &\quad + C_1 \cdot P + C_3 \cdot n_2 + n_2. \end{aligned} \tag{27}$$

(3) Calculate $R(\mathcal{J})$: Summing together $R(\mathcal{J}_{RR})$ in Inequality (24) and $R(\mathcal{J}_{XY})$ in Inequality (27), the total response time of $\mathcal{J}$ is

$$\begin{aligned} R(\mathcal{J}) &= R(\mathcal{J}_{RR}) + R(\mathcal{J}_{XY}) \\ &\leq \frac{C_1}{P}\sum_{i=1}^{n}(n-i+1)T_1(J_i) + C_2 \cdot T_\infty(\mathcal{J}) + C_1 \cdot P + (C_3+1)\cdot n \\ &\leq C_1 \cdot \mathrm{swa}(\mathcal{J}) + C_2 \cdot T_\infty(\mathcal{J}) + (C_1 + C_3 + 1)\cdot n. \end{aligned}$$

The mean response time of job set $\mathcal{J}$ is then $\overline{R}(\mathcal{J}) \leq (C_1 + C_2) \cdot \overline{R}^*(\mathcal{J}) + C_1 + C_3 + 1$, which indicates that RR-XY is $(C_1 + C_2)$-competitive when C_1 and C_3 are constants. □

Based on Theorem 4, we can conclude that the competitive ratios summarized in Table 2 for the three algorithms we considered also apply to the corresponding algorithms with round-robin augmentation.

8 Discussion and Conclusion

We have presented a generalized framework for analyzing the mean response time of online scheduling algorithms on batched parallel jobs. We have demonstrated the use of this framework on three two-level adaptive scheduling algorithms, namely XEQUI, IGDEQ, and AGDEQ, and we have improved the competitive ratios of IGDEQ and AGDEQ from their previous results.

In fact, the three algorithms studied in this paper represent schedulers with different degrees of non-clairvoyance, and therefore can be used in different system environments. The simplest algorithm XEQUI, being completely non-clairvoyant, is probably most suitable in highly dynamic environments, where no information about the jobs is known. In other circumstances, where the jobs' current parallelism is available, the clairvoyant algorithm IGDEQ can achieve better efficiency with its more dynamic processor allocation policy. In yet other systems, where the jobs' current parallelism is not known, but their past execution characteristics can be measured and they do not change frequently, AGDEQ that is only aware of the history of the jobs, may yield desirable performances in this case. As we have demonstrated in this paper, the clairvoyant algorithm IGDEQ indeed gives the best competitive ratio among the three algorithm. It may be a bit surprising, however, that AGDEQ algorithm, which utilizes the jobs' past parallelism, has a larger competitive ratio than the non-clairvoyant algorithm XEQUI. The reason is because we do not assume that the future parallelism of the jobs is correlated to their past, and AGDEQ that attempts to explore this correlation may end up having a poorer processor distribution among the jobs in the worst case and hence a larger competitive ratio. Nevertheless, the empirical results in [36, 2, 22] have shown that adaptive schedulers that utilize jobs' historical parallelism actually have much better performance in practice and that a variant of AGDEQ in fact outperforms EQUI on different workload conditions [2]. Incidentally, several empirical results [37, 30, 27, 29, 40, 13, 19] have also studied various two-level algorithms and their performances on different platforms for the scheduling of parallel jobs.

Finally, for scheduling non-batched sequential or parallel jobs, some researchers [11, 20, 26, 4, 6, 16, 34] have presented various algorithms and analyzed their performances. In particular, using resource augmentation analysis [26], which allows an online algorithm to have access to more processors than the optimal, Edmonds [16] has shown that EQUI achieves $O(1)$-speed $O(1)$-competitiveness with respect to the mean response time on parallel jobs modeled by multiple phases of arbitrary non-decreasing and sub-linear speedup functions. Using the same model, Robert et al. [34] have generalized the results to jobs with precedence constraints. It will be of particular interest for the parallel scheduling community to see if such analysis can be extended to two-level adaptive scheduling algorithms, such as IGDEQ and AGDEQ, to bound their mean response time performances in this more general setting.

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. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, New York, NY, USA, 2003.

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

[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] G. E. Blelloch and J. Greiner. A provable time and space efficient implementation of NESL. In *ICFP*, pages 213–225, Philadelphia, PA, USA, 1996.

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

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

[10] T. Brecht, X. Deng, and N. Gu. Competitive dynamic multiprocessor allocation for parallel applications. In *IPDPS*, pages 448 – 455, San Antonio, TX, USA, 1995.

[11] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *SODA*, pages 609–618, Philadelphia, PA, USA, 1997.

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

[13] J. Corbalán and J. Labarta. Improving processor allocation through run-time measured efficiency. In *IPDPS*, pages 74–79, San Francisco, CA, USA, 2001.

[14] X. Deng and P. Dymond. On multiprocessor system scheduling. In *SPAA*, pages 82–88, Padua, Italy, 1996.

[15] 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.

[16] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, Atlanta, GA, USA, 1999.

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

[18] Z. Fang, P. Tang, P.-C. Yew, and C.-Q. Zhu. Dynamic processor self-scheduling for general parallel nested loops. *IEEE Transactions on Computers*, 39(7):919–929, 1990.

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

[20] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line algorithms. In *SODA*, pages 142–151, Philadelphia, PA, USA, 1996.

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

[22] 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.

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

[24] K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. In *SODA*, pages 490–498, Philadelphia, PA, USA, 1999.

[25] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA*, pages 86–95, New York, NY, USA, 2005.

[26] B. Kalyanasundaram and K. R. Pruhs. Minimizing flow time nonclairvoyantly. *Journal of the ACM*, 50(4):551–567, 2003.

[27] S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *SIGMETRICS*, pages 226–236, Boulder, CO, USA, 1990.

[28] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *SODA*, pages 167–176, Philadelphia, PA, USA, 1994.

[29] S. Majumdar, D. L. Eager, and R. B. Bunt. Scheduling in multiprogrammed parallel systems. In *SIGMETRICS*, pages 104–113, Santa Fe, NM, USA, 1988.

[30] 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.

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

[32] G. Mounie, C. Rapine, and D. Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *SPAA*, pages 23–32, New York, NY, USA, 1999.

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

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

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

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

[37] 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.

[38] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response time. In *SPAA*, pages 200–209, Cape May, NJ, USA, 1994.

[39] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In *SODA*, pages 112–121, Philadelphia, PA, USA, 1994.

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