Scheduling in the Dark (Improved Result)

Jeff Edmonds*
York University, Canada, jeff@cs.yorku.ca

June 5, 2007

Abstract

We considered non-clairvoyant multiprocessor scheduling of jobs with arbitrary arrival times and changing execution characteristics. The problem has been studied extensively when either the jobs all arrive at time zero, or when all the jobs are fully parallelizable, or when the scheduler has considerable knowledge about the jobs. This paper considers for the first time this problem without any of these three restrictions yet when our algorithm is given more resources than the adversary. We provide new upper and lower bound techniques applicable in this more difficult scenario. The results are of both theoretical and practical interest.

In our model, a job can arrive at any arbitrary time and its execution characteristics can change through the life of the job from being anywhere from fully parallelizable to completely sequential. We assume that the scheduler has no knowledge about the jobs except for knowing when a job arrives and knowing when it completes. (This is why we say that the scheduler is completely in the dark.) Given all this, we prove that the scheduler algorithm Equi-partition, though simple, performs within a constant factor as well as the optimal scheduler as long as it is given at least twice as many processors. More over, we prove that if none of the jobs are "strictly" fully parallelizable, then Equi-partition performs competitively with no extra processors.

*Author is supported by NSERC Canada.

We also consider other variations: faster processors; fewer preemptions; and a wider range of execution characteristics.

This improved version improves the journal version [7] in two ways. The first way allows the optimal scheduler to complete the fully parallelizable work independently from the sequential work. This extra freedom is needed in [9, 11, ?]. The second improvement is a tightening of the competitive ratio from $\frac{2s}{(s-2)}$ to $1 + \mathcal{O}(\frac{\sqrt{s}}{s-2})$.

Key Words: scheduling, competitive ratio, equal-partition, response

1 Introduction

Suppose you are setting up a multi-processor system. One issue is how to schedule the incoming jobs. This paper compares the following two strategies. The first strategy uses lots of time of mathematicians, programmers, and cpu to guesstimate the types of jobs that are currently in your system and that are likely to arrive in the future and then attempts to approximate this NP-complete scheduling problem. The second strategy buys $2 + \epsilon$ times as many processors and then schedules using the simple Equi-partition algorithm. Given the inexpensive price of processors, the second strategy may well be cheaper. This paper proves that mean response times of the jobs under the second strategy is at most a constant factor times that of the second.

Given a fixed number of processors, the *scheduling task* is to continually allocate and reallocate processors to jobs as they arrive and complete. For example, *Equi-partition* always partitions the processors evenly among all jobs that are still alive and *Balance* gives all the processors to the most newly arrived job.

The goal is to minimize the average *response time*, i.e., the time during which a job has arrived but has not yet completed. Such a scheduling algorithm is said to be *on-line* if it lacks knowledge of which jobs will arrive in the future. It is said to be *non-clairvoyant* if it also lacks all knowledge about the jobs that are currently in the system, except for knowing when a job arrives and knowing when it completes. It is said to be *competitive* if, despite this lack of knowledge, it always performs only a constant times worse than the optimal all knowing scheduler.

The problem has been studied extensively but always within a model that is restricted in at least one of the following three ways: either the jobs all arrive at time zero, or all the jobs are fully parallelizable, or the scheduler has considerable knowledge about the jobs. This paper considers for the first time this problem without any of these three restrictions. Without these restrictions, few of the previous proof techniques are relevant. New upper and lower bound techniques are described here.

In our model, a job can arrive at any arbitrary time. A job can have an arbitrary number of phases. The execution characteristics of each phase of each job can be anywhere from being fully parallelizable to being completely sequential. A *speedup function* for each phase of each job defines the rate $\Gamma(\beta)$ at which work is completed in the phase as a function of the number of processors β allocated to it. For example, a fully parallelizable phase has $\Gamma(\beta) = \beta$ and a sequential phase has $\Gamma(\beta) = 1$. For the main result, we require only that each speedup function is sublinear and nondecreasing. Finally, we assume that the scheduler has no knowledge about the jobs except for knowing when a job arrives and knowing when it completes. This is why we say that the scheduler is completely in the dark.

It would seem that not knowing which jobs are fully parallelizable and which are sequential would make competitive scheduling impossible. The worst case set of jobs for a given non-clairvoyant scheduler will be such that any job allocated many processors will happen to be sequential. These processors are effectively wasted. The optimal schedule, knowing which jobs are sequential, allocates only the required number of processors to these jobs. In addition to these sequential jobs, a stream of fully parallelizable jobs arrive. The optimal schedule is able to complete each such job before the next arrives. Because the non-clairvoyant scheduler continues to wasted resources on the sequential jobs, it is unable to complete this stream of work and falls further and further behind.

We prove improved lower bounds showing that being competitive in this situation is impossible. However, an old Chinese saying says that two (blind) shoe makers are better then one politician. Analogously, we prove that the scheduler Equi-partition, though simple, performs within a constant factor as well as the optimal scheduler as long as it is given at least twice as many processors. The extra processors are enough to compensate for the lack of knowledge.

Classically, when one says that an algorithm is within a constant factor "as good as" the optimal algorithm, they assume that both algorithms are given the same resources. We feel that it is also valid to say this even when the algorithm is give a constant factor more resources than the optimal adversary. This idea was first introduced by [13].

A result that is perhaps even more surprising is that if none of the jobs are "strictly" fully parallelizable, then Equi-partition performs competitively with no extra processors. We also consider schedules that reallocate processors (preempt) only when the number of jobs in the system goes up or down by a factor of two (in some sense $\log n$ times). We conclude by giving competitive scheduling algorithms when the jobs have superlinear speedup or even more general speedup functions.

As noted, our results require techniques that are completely new. For example, the previous results prove that their algorithm is competitive by proving that at every point in time, the number of jobs alive under their algorithm is within a constant fraction of that under the optimal schedule. This, however, is simply not true with our less restricted model. There are job sets such that for a period of time the ratio between the numbers of alive jobs under the two schedules is unbounded. We use a potential function to prove that this can only happen for a relatively short period of time.

1.1 Competitive Ratio

In non-clairvoyant scheduling some relevant information, e.g. when jobs will arrive in the future, is not available to the scheduling algorithm S. The standard way to measure the adverse effect of this lack of knowledge is the competitive ratio $\mathrm{Min}_{S\in\mathcal{S}}\mathrm{Max}_{J\in\mathcal{J}} F\left(S(J)\right)/F\left(OPT(J)\right)$, where $\mathcal{S}$ denotes the class of non-clairvoyant schedulers being considered, $\mathcal{J}$ denotes the class of job sets to be scheduled, $F\left(S(J)\right)$ denotes the cost of the schedule $S(J)$ produced by the online algorithm S on job set J and $OPT(J)$ denotes an optimal (unrestricted) schedule for the job set. The standard way to interpret the competitive ratio is as the payoff to a game played between an online algorithm and an all powerful malevolent adversary that selects the job set J to be scheduled. At times this is similar to playing poker with someone who besides being the best player can also select all the cards, while you cannot even look at your own hand.

Competitive analysis has been criticized because it often yields ratios that are unrealistically high and thus fails to identify the class of online algorithms that work well. The scheduling problems that we consider are good examples of this phenomenon in that their competitive ratios are unbounded while there are simple non-clairvoyant schedulers that perform reasonably well in practice. Research in this area generally tries to give an advantage back to the non-clairvoyant scheduler in order to achieve a more realistic ratio. This advantage either restricts the class $\mathcal{J}$ of job sets from which the adversary can choose, increases the knowledge of the non-clairvoyant scheduler, or increases the power of the scheduler in some way. By covering many of these issues, we bring many of these paths of research together.

1.2 Previous Results

The problem has been studied extensively when either the jobs all arrive at time zero, or when all the jobs are fully parallelizable, or when the scheduler has considerable knowledge about the jobs.

Most scheduling results depend heavily on the scheduler having complete knowledge of the amount of work and the execution characteristics of the jobs as they arrive [25, 29, 30]. Hence, to various degrees of success, compilers and run-time systems attempt to give hints to the scheduler about this information. To avoid this we consider only non-clairvoyant schedulers.

Consider now the results on batch jobs, i.e., all jobs arrive at time zero. Motwani *et al.* [22] prove that on fully parallelizable jobs, the scheduler Equipartition algorithm [28], which partitions the processors evenly between the unfinished jobs and preempts only when jobs complete, has mean response time within two of optimal. They also prove that no non-clairvoyant scheduler has a better competitive ratio.

Turek *et al.* [29] consider jobs with a single phase of an arbitrary nondecreasing and sublinear speedup function. Without using preemptions, they achieve a competitive ratio of two. However, the algorithm requires complete knowledge of the jobs' workload and speedup functions and a perhaps excessive computation time of $O(n(n^2 + p))$.

In contrast, Deng, Gu, Brecht, and Lu [5] do not let the scheduler know the amount of work per job, but still assume that it knows the speedup func-

tion. They consider jobs that are fully parallelizable up to some number of processors, $\llcorner$. They show that DEQ, an algorithm similar to Equi-partition, achieves the same competitive ratio of two when the jobs have a single phase, and of four when they are allowed to have multiple phases.

Finally, Edmonds, Chinn, Brecht, and Deng [10] considers many classes of speedup functions and assumes that the scheduler has no knowledge of the jobs. For example, they show that the simple Equi-partition algorithm achieves a competitive ratio of $2 + \sqrt{3}$ where jobs have multiple phases of different nondecreasing sublinear speedup functions. Within the same model, they also prove a lower bound of $e \approx 2.71$ for any non-clairvoyant scheduler.

We now consider the results about scheduling jobs that are fully parallelizable and have arbitrary arrival times. The optimal schedule is easy to compute. Simply allocate all the processors to the jobs with least remaining work. This, however, requires the scheduler to know the amount of work per job. Kalyanasundaram and Pruhs [12] give a non-clairvoyant randomized scheduling algorithm with competitive ratio of $\widetilde{\Theta}(\log n \log\log n)$. The best deterministic and non-clairvoyant schedulers seem to be *Equi-partition* and *Balance*. Equi-partition, or $EQUI$ for short, partitions its processors evenly among the jobs that are still alive. Balance, BAL, allocates all of its processors to the job that has been allocated processors for the shortest length of time. Lacking knowledge about the jobs being scheduled, these schedulers do not perform well. They have competitive ratios of $\Omega(n/\log n)$ and of $\Omega(n)$ respectively [19, 22]. Lower bounds for general non-clairvoyant schedulers have been more difficult to obtain, but Motwani *et al.* [22] have a very simple proof that no deterministic non-clairvoyant scheduler can achieve a competitive ratio better than $\Omega(n^{1/3})$ and randomized $\Omega(\log n)$. One way of avoiding this bound is by requiring that the work of the largest and the smallest of the fully parallelizable jobs have a ratio of at most k. Then the competitive ratio is $\Theta(k)$ [22]. Even if this ratio is unbounded, Kalyanasundaram and Pruhs [13] achieve a competitive ratio of $\frac{s}{s-1} = 1 + \frac{1}{\epsilon}$ by giving their Balance scheduler processors of speed $s = 1 + \epsilon$. Recently, [1], achieve a competitive ratio of $\frac{2}{s}$ for $s \geq 2$, proving that the scheduler with lots of resources is competitive. (We have a simple proof achieving $\frac{4}{s}$.)

1.3 Summary of the Results

Like Edmonds *et al.*[10], we consider jobs that have an arbitrary number of phases, each with an arbitrary sublinear-nondecreasing speedup function. Like Kalyanasundaram and Pruhs, we consider arbitrary arrival times and give the non-clairvoyant scheduler s times as many resources.

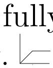
With the help of both sequential and fully parallelizable jobs, we achieve a $\Omega(\sqrt{n})$ lower bound on the competitive ratio for randomized non-clairvoyant schedulers. This is in marked contrast to the deterministic $\Omega(n^{1/3})$ and randomized $\widetilde{\Theta}(\log n)$ bounds [12, 22] which only use fully parallelizable jobs. With speed $s = 1+\epsilon$ processors, our randomized lower bound is $\Omega(\frac{1}{\epsilon})$, where no previous bound was known.

The scheduler Balance, when only given fully parallelizable jobs, has a deterministic ratio of $\frac{s}{s-1} = 1+\frac{1}{\epsilon}$ with speed $s = 1+\epsilon$ and a randomized ratio of $\mathcal{O}(\log n \log\log n)$ [12, 13]. However, we show that if the jobs are even slightly not fully parallelizable, eg., $\Gamma(\beta) = \beta^{1-\alpha}$, then both their deterministic and their randomized versions of Balance can have an arbitrarily bad competitive ratio, $\Omega\left(s^{-1/\alpha}n\right)$, even when given arbitrarily fast processors. The reason is that Balance allocates all of its processors to the newly released job. This job, however, may not be able to efficiently utilize this many processors. It seems then that the only non-clairvoyant scheduler that will perform well without knowing the speedup functions of the jobs is Equi-partition.

Yet on the other hand, Kalyanasundaram and Pruhs [13] modify the set of jobs constructed by Motwani *et al.* so that Equi-partition with $s = 1+\epsilon$ times as many resources has a competitive ratio of $\Omega(n^{1-\epsilon})$. We prove that if you increase the resources by a factor of $s > 2$, then Equi-partition becomes competitive. More specifically, when the scheduler is given p processors of speed $s = 2+\epsilon$, the competitive ratio is between $\frac{2}{3}(1+\frac{1}{\epsilon})$ and $2+\frac{4}{\epsilon}$. With lots of extra speed, $s \geq 4$, the schedule is still competitive with a ratio between $\frac{2}{s}$ and $\frac{16}{s}$. Alternatively, when the scheduler is given sp processors of speed 1, the competitive ratio is between $1+\frac{1}{\epsilon}$ and $2+\frac{4}{\epsilon}$ for $s = 2+\epsilon$. However, with lots of extra processors $s >> 1$, a lower bound of $1+\frac{1}{s}$ proves that the scheduler is no longer competitive. (Note having more instead of faster processors is a weaker model. More processors help when the jobs are fully parallelizable but not when they are sequential.) Though the upper and lower bounds do not match perfectly as to the constant, they are tight with

respect to the fact that Equi-partition requires more than twice the resources to achieve a constant competitive ratio.

We go on to prove the surprising result that if all the jobs are *strictly sublinear*, i.e., are not fully parallel, then Equi-partition performs competitively with no extra processors. The intuition is that if Equi-partition falls behind, then it has more uncompleted jobs in the system and hence allocates fewer processors to each job and hence each job utilizes the processors that it is given more efficiently. We refer to this as Equi-partition *automatically self adjust* the number of processors wasted on jobs that cannot fully utilize them. More specifically, we show that if all the speedup functions are no more fully parallelizable than $\Gamma(\beta) = \beta^{1-\alpha}$ than the competitive ratio is at most $2^{\frac{1}{\alpha}}$. For intuition, suppose the adversary allocates $\frac{p}{n}$ processors to each of n jobs and Equi-partition falls behind enough so that it has $2^{\frac{1}{\alpha}} n$ uncompleted jobs. Then it allocates $p/(2^{\frac{1}{\alpha}} n)$ processors to each completing work at an overall rate of $(2^{\frac{1}{\alpha}} n)\Gamma(p/(2^{\frac{1}{\alpha}} n)) = 2 \cdot n\Gamma(p/n)$. This is a factor of 2 more than that by the adversary. Hence, as in the previous result, Equi-partition has twice the speed and so performs competitively. In this situation, the Motwani like lower bound for Equi-partition seems to have a competitive ratio of $1.48^{\frac{1}{\alpha}}$.

There are other classes of speedup functions that are definitely not fully parallelizable, because $\Gamma(p) << p$ yet that do not fit our definition of "strictly" sublinear. Such jobs seem to occur in practice and are used in many simulations [2]. It is interesting, that the Motwani like lower bound for these is the same as that for fully parallelizable jobs. The key property is that these speedup functions are fully parallelizable when the job is allocated a small number of processors, eg.  or $\Gamma(\beta) = (\widehat{\beta}\beta + 1)/(\widehat{\beta} + \beta)$.

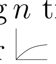
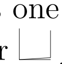
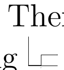
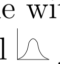
Edmonds *et al.*[10] go on to prove a large table of results. We prove analogous results, except with arbitrary job arrival times. There is a competitive non-clairvoyant scheduler with $(8 + \epsilon)p$ processors that only preempts when the number of jobs in the system goes up or down by a factor of two (in some sense $\log n$ times). There is one with $(4 + \epsilon)p$ processors that includes both sublinear  and superlinear  jobs. There is also one with $\mathcal{O}(p \log p)$ processors that includes both nondecreasing  and gradual  jobs.

Figure 1 summarizes the above mentioned results.

The paper is laid out as follows. Section 2 formally defines the model. Section 3 provides some intuition. Section 4 provides an upper bound for

$EQUI_s$ and Section 5 for $EQUI^s$. Section 6 considers strictly sublinear jobs. Section 7 proves lower bounds. Section 8 reduces the number of preemptions. Section 9 considers sublinear and superlinear jobs and Section 10 considers jobs that are only restricted to being nondecreasing or gradual. The paper ends with some open problems.

2 The Model: Schedulers and Speedup Functions

We consider a set of n jobs that are to be executed on p processors, (n can tend to infinity). A set of jobs J is defined to be $\{J_1, \ldots, J_n\}$ where *job* J_i has a *release/arrival time* r_i and a sequence of phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i}\rangle$. Each phase is an ordered pair $\langle w_i^q, \Gamma_i^q\rangle$, where w_i^q is a nonnegative real number that denotes the amount of *work* and Γ_i^q is a function, called the *speedup function*, that maps a nonnegative real number to a nonnegative real number. $\Gamma_i^q(\beta)$ represents the rate at which work is executed for phase q of job i when given β processors.

A scheduler S allocates the p processors for each point in time to the jobs in the given jobs set J in a way such that all the work completes. More formally, a *schedule* $\mathcal{S}$ *for a given job set* J *with* n *jobs on* p *processors* is a function from $\{1, \ldots, n\} \times [0, \infty)$ to $[0, p]$, where $\mathcal{S}(i, t)$ is the number of processors allocated to job J_i at time t. (We allow a job to be allocated a non-integral number of processors.) Requiring that for all t, $\sum_{i=1}^{n} \mathcal{S}(i, t) \leq p$ ensures that at most p processors are allocated at any given time. Requiring that for all i, there exist $r_i = c_i^0 < c_i^1 < \ldots < c_i^{q_i}$ such that for all $1 \leq q \leq q_i$, $\int_{c_i^{q-1}}^{c_i^q} \Gamma_i^q(\mathcal{S}(i, t))\ dt = w_i^q$ ensures that before a phase of a job begins, the job must have been released and all of the previous phases of the job must have completed. If $c_i^0, c_i^1, \ldots, c_i^{q_i}$ are the smallest such values that satisfy this condition, then the completion time of phase q of job J_i under S is c_i^q. The *completion* time of a job J_i, denoted c_i, is the completion time of the last phase of the job. A job is said to be *alive* at time t, if it has been released, but has not completed, i.e., $r_i \leq t \leq c_i$. The *response time* of job J_i, $c_i - r_i$, is the length of the time interval during which the job is *alive*. We refer to an algorithm $S(J)$ for producing schedules as a *scheduler*. The goal of the scheduler is to minimize the average response time, $\frac{1}{n}\sum_{i \in J}(c_i -$

r_i), of the jobs it must schedule. This goal is equivalent to minimizing the *flow time of J under scheduler S*, denoted $F(S(J))$, which is $\sum_{i \in J}(c_i - r_i)$. An alternative formalization is to integrate over time the number of jobs n_t alive at time t, $F(S(J)) = \sum_{i \in J} \int_0^\infty (J_i \text{ is alive a time } t)\delta t = \int_0^\infty n_t \delta t$. Recall, the competitive ratio of a scheduler S on a class of job sets $\mathcal{J}$ is $\text{Max}_{J \in \mathcal{J}} F(S(J))/F(OPT(J))$.

2.1 Non-Clairvoyant Schedulers, $EQUI$ and BAL

The class of schedulers, $\mathcal{J}$, considered in this paper are the *non-clairvoyant* schedulers, meaning that they have no knowledge of the jobs. They do not know if or when jobs will be released in the future. They do not know the work w_i^q or the speedup functions Γ_i^q of the jobs currently alive. They are not able to detect when a particular phase of a job completes. At time t, they only know the release r_i times of the jobs that have already been released, the completion times c_i of the jobs that have already completed, and the number of jobs n_i currently alive in the system.

The two examples of *non-clairvoyant* schedulers that are often used in practice are *Equi-partition* and *Balance*. (Though no one implements Balance directly, Unix uses a multi-level feedback (MLF) queue algorithm which in a way approximates Balance). We define $EQUI$ to be the scheduler that allocates an equal number of processors to each job that is currently alive. That is, for all i and t, if job J_i is alive at time t, then $\mathcal{EQUI}(i,t) = p/n_t$, where n_t is the number of jobs that are alive at time t. The schedule BAL is defined in [13] to be the schedule that allocates all of its processors to the job that has been allocated processors for the shortest length of time.

For these schedulers to be competitive it is necessary to give them more power. Define $EQUI^s$ to be the same as $EQUI$ except that each of its p processors execute work s times faster than a normal processor, i.e., if a job phase has speedup function Γ then its work is completed at a rate of $s \times \Gamma(\beta)$ when allocated β processors. This is equivalent to giving jobs with work w_i/s instead of w_i. Giving jobs arriving at time sr_i instead of at time r_i is also the same, except that that flow time is a factor of s larger. Define $EQUI_s$ to be the same as $EQUI$ except that it has sp processors, i.e., $\mathcal{EQUI}_s(i,t) = sp/n_t$. Note that for sublinear speedup curves, $EQUI^s$ is at least as powerful $EQUI_s$, because $EQUI_s$ works s times faster on fully

parallelizable jobs and no faster on sequential jobs, while $EQUI^s$ work s times faster on sequential jobs as well. BAL^s and BAL_s are defined similarly.

2.2 Classes of Speedup Functions

We now describe some specific speedup functions and some classes of speedup functions.

A phase of a job is said to be *fully parallelizable* if its speedup function is $\Gamma(\beta) = \beta$. (See Fig 2:a.) A phase of a job is said to be *sequential* if its speedup function is $\Gamma(\beta) = 1$, for all $\beta \geq 0$. (See Fig 2:b). Note that such phases are good for OPT because it can achieve a work rate of one while allocating zero processors to it. If a non-clairvoyant scheduler, not knowing that the phase is sequential, allocates more processors to it, the rate of work is not any better. It would be more reasonable to require at least one processor to be allocated to the job to achieve a rate of one, but this definition both simplifies the proof and makes the result stronger.

A speedup function Γ is *nondecreasing* iff $\Gamma(\beta_1) \leq \Gamma(\beta_2)$ whenever $\beta_1 \leq \beta_2$. A job phase with a nondecreasing speedup function executes no slower if it is allocated more processors. (See Fig 2:a-e.) This is a reasonable assumption if in practice a job can determine whether additional processors will speed up its execution and if not can refuse to use some of those allocated to it.

A speedup function Γ is *sublinear* iff $\Gamma(\beta_1)/\beta_1 \geq \Gamma(\beta_2)/\beta_2$ whenever $\beta_1 \leq \beta_2$. (See Fig 2:a-c.) A measure of how efficient a job utilizes its processors is $\Gamma(\beta)/\beta$, which is the work completed by the job per time unit per processor. A sublinear speedup function is one whose efficiency does not increase with more processors. This is a reasonable assumption if in practice β_1 processors can simulate the execution of β_2 processors in a factor of at most β_2/β_1 more time.

A speedup function Γ is *strictly-sublinear* by α iff $\forall \beta_1 \leq \beta_2$, $\Gamma(\beta_2)/\Gamma(\beta_1) \leq (\beta_2/\beta_1)^{1-\alpha}$. (See Fig 2:b-c.) An example of such a speedup function is $\Gamma(\beta) = \beta^{1-\alpha}$. We refer to this as the *almost fully parallelizable* speedup function. The class also includes everything in between this and the sequential speedup function.

In contrast, a speedup function Γ is *superlinear* iff $\Gamma(\beta_1)/\beta_1 \leq \Gamma(\beta_2)/\beta_2$

whenever $\beta_1 \leq \beta_2$. Such speedup functions occur in programs that are highly parallelizable and have a strong time-space trade off. (See Fig 2:d.) Fig 2:e is an example of a speedup function that is neither sublinear nor superlinear, but is non-decreasing. A speedup function is said to be *gradual* [10] iff for every number of processors β_1 and there is a value $a \in [1..2]$ such that for all $\beta_2 \in [a\beta_1/2, a\beta_1]$, $\Gamma(\beta_2) \geq \frac{1}{2}\Gamma(\beta_1)$. A small point is that we also required $\Gamma(\beta)$ to be nondecreasing for $\beta \leq$. (See Fig 2:a-f and specifically f.)

3 Intuition: EQUI Self Adjusts

Before giving the formal proofs, we first provide some intuition into the powers and limitations of non-clairvoyant schedulers. The intuition behind Equi-Partition is that because the jobs cannot be distinguished wrt work or speedup functions, allocate the processors evenly between them. The intuition behind Balance is as follows. With only fully parallelizable jobs, the optimal schedule is to allocate all the processors to the job with the least remaining work. Balance tries to mimic this by allocating all the processors to the job that has completed the least amount of work. Note if every job was half done, then this would be the correct measure. BAL also has the feature that if two fully parallelizable jobs are ever in the system at the same, then independent of their arrival times, the one with the least amount of work completes first.

As said, Balance performs very poorly when the jobs are not fully parallelizable because it allocates all of its processors to a single job that may not be able to efficiently utilize this many processors. Then, given the volume of work that arrives, it is unable to complete any of the jobs, even when it has processors of arbitrarily large speed s. OPT, on the other hand, knowing both the speedup function of the jobs and the rate at which work arrives, knows just the right number of processors to allocate to each job in order to achieve just the right level of efficiency to complete all the work at the rate it arrives. $EQUI$, without knowing any of this information, does a surprisingly good job at automatically *self adjusting* to this optimal number of processors. Initially, $EQUI$, like Balance, may allocate more processors to some jobs then the jobs can utilize and hence the schedule falls behind OPT. As the number of alive jobs increases, $EQUI$ allocates few processors to each job and hence computes each more efficiently. If $EQUI$ has more resources

than OPT and it is sufficiently utilizing these resources, then $EQUI$ will then catch up with OPT.

The job sets that are the most difficult for $EQUI$ seem to be modifications of that given by Motwani [13, 22]. Each job set consists of a *stream* of n fully parallelizable jobs and a few *extra* jobs. The stream is defined by partitioning time into intervals (r_i, r_{i+1}) of length $t_i = r_{i+1} - r_i$. The i^{th} stream job J_i has release time r_i and work $w_i = t_i p$. For now, the extra jobs are sequential. The number that are alive at time t is ℓ_t. Think of ℓ_t as being some constant bigger than s.

The schedule OPT ignores the extra jobs and completes the stream job J_i during the time interval (r_i, r_{i+1}) by allocating all p processors to it. The extra sequential jobs complete with zero processors. Hence, at time t, there are only $\ell_t + 1$ jobs alive.

It is hard to believe that any non-clairvoyant scheduler, even with sp processors, can perform well here. It does not know which of the jobs is fully parallelizable. Hence it wastes most of the processors on sequential jobs, allocating only $sp/(\ell_t+1) < p$ processors to the fully parallelizable job. With only this many processors, this job falls further and further behind and then other fully parallelizable jobs arrive. These too fall behind. $EQUI_s$, however, is able to automatically "self adjust" the number of processors wasted on the sequential jobs so that it performs competitively. It may take a while for the system under $EQUI_s$ to reach a "steady state", but when it does, m_t, which denotes the number of fully parallelizable jobs alive at time t, converges to $\widetilde{m}_t = \ell_t/(s-1)$. At this time, $EQUI_s$ has $\ell_t + \widetilde{m}_t$ jobs alive and OPT has $\ell_t + 1$. Hence, the competitive ratio is $F(EQUI_s(J)/F(OPT(J)) = (\ell_t + \ell_t/(s-1))/(\ell_t + 1) \leq \frac{s}{s-1}$, which is $1 + \frac{1}{\epsilon}$ for $s = 1 + \epsilon$.

We will show that a key resource for getting and keeping m_t high is the work, W_t, completed by OPT but not by $EQUI_s$ by time t. The fully parallelizable work is released and completed by OPT at a rate of p. $EQUI_s$ allocates $\frac{sp}{\ell_t+m_t}$ of its sp processors to each of the $\ell_t + m_t$ jobs alive and hence it completes fully parallelizable work at a rate of $\frac{spm_t}{\ell_t+m_t}$. The sequential work is released and completed at the same rate by each scheduler. Hence, overall, W_t changes at a rate of $p - \frac{spm_t}{\ell_t+m_t}$. We say that the system reaches a *steady state* when amount of work in W_t remains constant, giving $\widetilde{m}_t = \ell_t/(s-1)$.

If $m_t < \widetilde{m}_t$ than fully parallelizable work is being released faster then it is being completed. Hence, $EQUI_s$ falls further behind and m_t increases. On

the other hand, if $m_t > \widetilde{m}_t$ than fully parallelizable work is being completed faster then it is being released. Hence, $EQUI_s$ catches up. Eventually m_t must decrease. Otherwise, W_t will decrease to zero, at which time m_t is zero. The conclusion is that $EQUI_s$ tends towards a steady state with $m_t = \widetilde{m}_t = \ell_t/(s-1)$.

A second key factor for getting and keeping m_t large is the amount of work remaining per job. We have seen that the total work W_t completed by OPT and not by $EQUI_s$ is limited and is getting smaller. Hence, to have the number of jobs m_t get larger, the work per job must get much much smaller. This is what happens in the lower bound Theorem 7. In contrast, suppose that all the fully parallelizable jobs are the same size, i.e., $t_i = 1$. In this case, W_t is linearly proportional to m_t. Both experimental and theoretical analysis show that m_t quickly converges to $\ell/(s-1)$ without exceeding it. We conjecture that if the t_i were chosen randomly or chosen in a worst case way and then randomly ordered, then the expected competitive ratio would be $\frac{s}{s-1} = 1 + \frac{1}{\epsilon}$ for $s = 1 + \epsilon$. This would match the lower bound in Theorem 5.

4 Upper Bound for $EQUI_s$

This section proves the upper bound for $EQUI_s$.

Theorem 1. *Consider any set of jobs J with arbitrary arrival times, each job has an arbitrary number of phases, each phase has an arbitrary sublinear-nondecreasing speedup function. Suppose OPT has p processors and $EQUI_s$ has $s * p$ processors. Then $F(EQUI_s(J))/F(OPT_1(J)) \leq F(EQUI_s(J))/F(\widehat{OPT}_1(J)) \leq 1 + \mathcal{O}(\frac{\sqrt{s}}{s-2})$.*

This theorem improves the journal version [7] of the corresponding theorem in two ways. The first way allows the optimal scheduler to complete the fully parallelizable work independently from the sequential work. This extra freedom is needed in [9, 11, **?**].

More formally, $\widehat{OPT}_1(J)$ is defined as follows. It is the optimal scheduling algorithm for the jobs J where the scheduler is allowed some additional scheduling options. (It follows that $\widehat{OPT}_1(J) \leq OPT_1(J)$.) Every time this more powerful scheduler executes a job, it is allowed to skip any amount of

any sequential phase that it likes. However, the measure of how well it does is $\widehat{OPT}_1(J) = \widehat{OPT}_1^{par}(J) + \widehat{OPT}_1^{seq}(J)$, where $\widehat{OPT}_1^{par}(J)$ is the average flow time $Avg_{i,j}[c^O_{i,j} - a_{i,j}]$ of the jobs after these sequential phases have been cut out and $\widehat{OPT}_1^{seq}(J)$ is the average over all jobs of the total amount of sequential work cut out during the execution of the job that services the request.

The second improvement is a tightening of the competitive ratio from $\frac{2s}{(s-2)}$ to $1 + \mathcal{O}(\frac{\sqrt{s}}{s-2})$. When speed $s = 2 + \epsilon$ for small ϵ, both results give $\mathcal{O}(\frac{1}{\epsilon})$. On the other hand, if speed s is large, then the improvement is from $2 + \mathcal{O}(\frac{1}{s})$ to $1 + \mathcal{O}(\frac{1}{\sqrt{s}})$. This new result is needed in [9, 11]. It is likely that the competitive ratio should be $1 + \mathcal{O}(\frac{1}{s})$, but as of yet that is unattainable.

Proof Sketch: The proof follows the intuition given in Section 3. In the first step, Lemma 1 proves that the worst case job sets are those that are *stream lined with respect to OPT*. As in Section 3, these job sets consists of a stream of fully parallelizable jobs and ℓ_t extra sequential jobs. However, this stream is a little more general in that a job may have both sequential and fully parallelizable phases. The key property is that OPT never has more than one fully parallelizable phase to execute. (See Fig:3.)

In the second step, Lemma 2 proves that the competitive ratio when considering only such job sets is at least $2s/(s-2)$. As in Section 3, it is sufficient to bound the average value of m_t as a function of ℓ_t, where m_t is the number of fully parallelizable stream jobs that are alive within $EQUI_s$ at time t and ℓ_t is the number of sequential jobs. A steady state value was argued to be $\widetilde{m}_t = \ell_t/(s-1)$. Lemmas 3 and 4 together prove that $\int_0^\infty (m_t - \mathcal{O}(\frac{\sqrt{s}}{s-2})\ell_t)\delta t \leq 0$. This is proven by again considering the work W_t that has been completed under OPT, but not under $EQUI_s$ by time t. We define $\widehat{F}_t$ to be a carefully designed function of this work and prove that the potential function $\int_0^t (m_{t'} - \widehat{m}_{t'})\delta t' + \widehat{F}_t$ is non-increasing with t. To do this, we note that $W_{t+\delta t} - W_t$ consists of the work completed by OPT during the time interval $[t, t+\delta t]$ minus the work completed by $EQUI_s$ during this same interval. During this time, OPT completes lots of work on one fully parallelizable job and $EQUI_s$ completes a little work on all m_t fully parallelizable jobs. Even though the sequential phases may get executed at different time under the two schedules, accounting for them is not difficult, because independent of when they are executed, their execution rate is fixed. ■

We now give the more formal proof.

Proof of Theorem 1: Consider a job set J as described in the Theorem. Lemma 1 converts this into job set J' that is *stream lined with respect to* OPT such that $F(EQUI_s(J')) \geq F(EQUI_s(J))$ and $F(OPT(J')) \leq F(OPT(J))$. Lemma 2 then proves that the competitive ratio when considering only such job sets is at least $2s/(s-2)$. From these two facts the result follows. $F(EQUI_s(J))/F(OPT(J)) \leq F(EQUI_s(J'))/F(OPT(J')) \leq 2s/(s-2)$. ■

The literature [10, 29] presents two bounds on the flow time under OPT. These are known as the squashed area bound and the height bound, respectively. This paper uses these bounds implicitly. However, in order to simplify the proof of the main lemma, the ways in which these bounds are used are separated out and combined with the proof of these bounds. The result is the following lemma.

We will say that a job set J is *stream lined with respect to* OPT iff 1) every phase of every job is either sequential or fully parallelizable and 2) OPT is able to execute every job at its maximum possible speed, i.e., allocate zero processors to every sequential and p processors to every fully parallelizable phase, without any job ever waiting for processors. The reason is that the fully parallelizable phases of the jobs fit together so that at most one at a time is alive. (See Fig 3.)

Lemma 1. *Consider any non-clairvoyant scheduler S_s with sp processors. For every job set J with sublinear-nondecreasing speedup functions, there is a job set J' that is stream lined with respect to OPT, such that $F(S_s(J')) \geq F(S_s(J))$ and $F(OPT(J')) \leq F(OPT(J))$. Moreover, when executing J', S_s is never ahead of OPT on any job.*

Proof of Lemma 1: Let S_s and J be as stated. We change the job set J into a job set J' a little bit at a time, in the order that OPT completes the work. For each point in time T, we define a job set J^T, where the work completed before time T under OPT has been changed to that in J' and the work completed after T is still that in J.

The inductive step is to construct job set $J^{T+\delta T}$ from J^T, where δT is an infinitesimally short interval of time. Consider in turn each job J_i that is alive under OPT during the interval $(T, T+\delta T)$ and the interval of work completed under OPT on it during this time. Let $(\tau_i, \tau_i + \delta\tau_i)$ be the interval

during which S_s completes this same work. Because these intervals of time and work are infinitesimally short we can assume WLOG that neither phase changes nor processor reallocations occur under OPT or S while completing this work. Let Γ_i denote the speedup function of this interval of work. Let β_i^{OPT} denote the number of processors allocated under OPT to it and $\beta_i^{S_s}$ the number under S_s.

There are two cases. First suppose $\beta_i^{OPT} \leq \beta_i^{S_s}$. Because the speedup function Γ_i is non-decreasing, we know S_s is completing this work at least as quickly as OPT, i.e., $\Gamma_i(\beta_i^{OPT}) \leq \Gamma_i(\beta_i^{S_s})$, and hence requires no more time, i.e., $\delta\tau_i \leq \delta T$.

We modify J^T to $J^{T+\delta T}$ in this case by replacing this interval of work in job J_i with a sequential phase with work δT. This change will not increase the flow time under OPT, because OPT can complete the same interval of work during the same interval of time, while allocating the phase zero processors. The β_i^{OPT} processors it had used for this work can instead be used on another job.

In contrast, this change cannot decrease the flow time under S_s. First note that because S_s is non-clairvoyant, it cannot differentiate between the job sets J^T and $J^{T+\delta T}$. Hence, it still allocates $\beta_i^{S_s}$ processors to complete this work, even though they are effectively wasted. Now note that S_s had completed the original work in time $\delta\tau_i$ but now it requires time δT. As seen, $\delta\tau_i \leq \delta T$. Finally, note that with this change scheduler S_s does not get ahead of OPT on this interval of work.

In the second case, suppose $\beta_i^{OPT} \geq \beta_i^{S_s}$. Because the same work is completed in time δT at a rate of $\Gamma_i(\beta_i^{OPT})$ and in time $\delta\tau_i$ at a rate of $\Gamma_i(\beta_i^{S_s})$, it follows that $\delta\tau_i/\delta T = \Gamma_i(\beta_i^{OPT})/\Gamma_i(\beta_i^{S_s})$. Because the speedup function is sublinear, $\Gamma_i(\beta_i^{OPT})/\Gamma_i(\beta_i^{S_s}) \leq \beta_i^{OPT}/\beta_i^{S_s}$. Hence, $\delta\tau_i \leq (\beta_i^{OPT}/\beta_i^{S_s})\delta T$.

WLOG, assume that the jobs alive under OPT during the interval $(T, T+\delta T)$ are $J_1, \ldots, J_{N_T}$. We modify J^T to $J^{T+\delta T}$ in this case by replacing this interval of work in job J_i with a sequential phase with work $\sum_{i'<i}(\beta_{i'}^{OPT}/p)\delta T$, followed by a fully parallelizable phase with work $\beta_{i'}^{OPT}\delta T$, followed by a sequential phase with work $\sum_{i'>i}(\beta_{i'}^{OPT}/p)\delta T$. First we need to check that this change does not increase the flow time under OPT. If OPT allocates all p processors to the fully parallelizable phase, then it will complete this phase in time $(\beta_i^{OPT}/p)\delta T$. Hence, OPT completes the three phases in time $\sum_{i'}(\beta_{i'}^{OPT}/p)\delta T = \delta T$. Also note that these fully parallelizable phases fit

together so that at most one at a time is requiring processors.

Now we need to check that this change does not decrease the flow time under S_s. As before, because S_s is non-clairvoyant, it still allocates $\beta_i^{S_s}$ processors to complete this work on job J_i. Hence, it completes the fully parallelizable phase with work $\beta_{i'}^{OPT}\delta T$ in time $(\beta_i^{OPT}/\beta_i^{S_s})\delta T \geq \delta\tau_i$. In addition, the scheduler S_s must complete the sequential phases. Finally, note that scheduler S_s does not get ahead of OPT on this interval of work. ■

Lemma 2. *For any job set J that has only fully parallelizable or sequential phases and that is stream lined with respect to OPT, $F(EQUI_s(J)) \leq 2s/(s-2) \cdot F(OPT(J))$.*

Proof of Lemma 2: Let J be the worst case set of stream lined jobs. Let n_t denote number of jobs alive under $EQUI_s$ at time t. Let m_t denote the number of these that are within a fully parallelizable phase at this time and let ℓ_t denote the same except for sequential phases. Let N_t, M_t, L_t denote the same numbers except under OPT. Using these, the competitive ratio can be expressed as

$$\frac{F\left(EQUI_s(J)\right)}{F\left(OPT(J)\right)} = \frac{\int_0^\infty n_t \delta t}{\int_0^\infty N_t \delta t} = \frac{\int_0^\infty \left(\ell_t + m_t\right) \delta t}{\int_0^\infty \left(L_t + M_t\right) \delta t}$$

The schedules $EQUI_s$ and OPT execute the sequential phases of jobs at the same rate because these phases (as defined for this paper) complete at a rate of one even if no processors are allocated to them. From this we know that both $\int_0^\infty \ell_t \delta t$ and $\int_0^\infty L_t \delta t$ are simply the sum of the work of all sequential phases of all jobs. Because these integrals are the same, L_t can be replaced with ℓ_t in the above bound for the competitive ratio. One should note, however, that it is possible that the number of sequential phases being executed at a given time under the two schedules might be very different, i.e., $\ell_t \neq L_t$. This occurs if a sequential phase is delayed under $EQUI_s$ because the same job has a fully parallelizable phase preceding the sequential phase.

The intuition in Section 3 indicated that the *steady state* the number of fully parallelizable under $EQUI$ is $\ell_t/(s-1)$. Below Lemmas 3 and 3 proves

that on average m_t is not more than this. More formally, Lemma 3 proves that $\int_0^\infty (m_t - \widehat{m}_t)\delta t \leq 0$, where $\widehat{m}_t = \frac{2\ell_t+1}{s-2} + \frac{s}{s-2}\frac{\ell_t}{1+\frac{\ell_t}{m_t}}$. Lemma 4 proves that from this it follows that $\int_0^\infty (m_t - \widehat{\widehat{m}}_t)\delta t \leq 0$, where $\widehat{\widehat{m}}_t = \mathcal{O}(\frac{\sqrt{s}}{s-2})\ell_t)$. Finally, from this the result follows easily.

$$\begin{aligned}\frac{F\left(EQUI_s(J)\right)}{F\left(OPT(J)\right)} &= \frac{\int_0^\infty \left(\ell_t + \widehat{m}_t\right)\delta t}{\int_0^\infty \left(L_t + M_t\right)\delta t} + \frac{\int_0^\infty \left(m_t - \widehat{m}_t\right)\delta t}{\int_0^\infty \left(L_t + M_t\right)\delta t} \\ &\leq \frac{\int_0^\infty \left(\ell_t + \mathcal{O}(\frac{\sqrt{s}}{s-2})\ell_t\right)\delta t}{\int_0^\infty \left(\ell_t + M_t\right)\delta t} + 0 \leq \max_t \frac{\mathcal{O}(1+\frac{\sqrt{s}}{s-2})\ell_t}{(\ell_t + m m_t)} \leq \mathcal{O}(1 + \frac{\sqrt{s}}{s-2})\end{aligned}$$

Recall that M_t is either zero or one and one maximizes the above expression, giving $\max_t \frac{2s\ell_t+1}{(s-2)(\ell_t+1)} \leq \frac{2s}{(s-2)}$. ■

The remaining step is to prove that on average m_t is $\mathcal{O}(\ell_t)$.

Lemma 3. *$\int_0^\infty (m_t - \widehat{m}_t)\delta t \leq 0$, where $\widehat{m}_t = \frac{2\ell_t+1}{s-2} + \frac{s}{s-2}\frac{\ell_t}{1+\frac{\ell_t}{m_t}}$.*

Proof of Lemma 3: The proof defines a potential function $F_T + \widehat{F}_T$. The first part F_T is defined to be $\int_0^T (m_t - \widehat{m}_t)\delta t$. The second part $\widehat{F}_T$ is a function of the amount of work that has not been completed by $EQUI_s$ by time T, but that has been competed by OPT. (Recall, that by Lemma 1, there is no work that has been completed by $EQUI_s$ and not by OPT.) If there is lots of such work than $\int_T^\infty (m_t - \widehat{m}_t)\delta t$ can be large. Intuitively $\widehat{F}_T$ acts as an upper bound for how large $\int_T^\infty (m_t - \widehat{m}_t)\delta t$ can be, given the fact that there is all this work that has not been completed by $EQUI_s$, but that has been competed by OPT. It would follow that $F_T + \widehat{F}_T$ is an upper bound on $\int_0^T (m_t - \widehat{m}_t)\delta t + \int_T^\infty (m_t - \widehat{m}_t)\delta t = \int_0^\infty (m_t - \widehat{m}_t)\delta t$. Our goal is to prove that this is at most zero.

The formal steps are different. The main step is to prove that the function $F_T + \widehat{F}_T$ is non-increasing with time T. Because it is not differentiable at the points in time T where a job phase begins or completes under $EQUI_s$, at these points we prove that the function is continuous. For other points in time, we prove that its derivative is at most zero. Next, we note that $F_0 + \widehat{F}_0$ is zero, F_0 zero by definition and $\widehat{F}_0$ because initially $EQUI_s$ does not have any uncompleted work. Because $F_T + \widehat{F}_T$ is non-increasing $F_\infty + \widehat{F}_\infty$ is at

most zero. Again $\widehat{F}_\infty$ is zero because in the end $EQUI_s$ does not have any uncompleted work. We can conclude that $F_\infty = \int_0^\infty (m_t - \widehat{m}_t)\delta t \leq 0$.

The work completed by OPT by time T and not by $EQUI_s$ will be characterized by the following definitions. (See Figure 3.) For $t \geq T$, define m_t^T to be number of fully parallelizable phases executing under $EQUI_s$ at the time instance t for which OPT has completed this same instance of work by time T. (Note OPT does not need to have completed the entire phase.) Similarly, define ℓ_t^T to be the number of sequential phases and $n_t^T = \ell_t^T + m_t^T$ to be the number of jobs with this property. The potential function on this work is defined to be $\widehat{F}_T = \int_T^\infty \frac{s}{s-2} \frac{(m_t^T)^2}{n_t} \delta t$. Recall, $F_T = \int_0^T (m_t - \widehat{m}_t)\delta t$.

Claim 1. *The potential function $F_T + \widehat{F}_T$ is continuous even at the points in time T where a job phase begins or terminates.*

Proof of Claim 1: The influences on F_T and $\widehat{F}_T$ caused by moving the boundary of the integrations $\int_0^T$ $\int_T^\infty$ are continuous because the integrands are finite. For a fixed value of t, the numbers of jobs m_t^T and ℓ_t^T may change by an integer amount when changing T by an infinitesimal amount δT. However, within the time interval $[T, T + \delta T]$, OPT is able only to complete an infinitesimal amount of work. This same work gets completed under $EQUI_s$ during an infinitesimal interval of time $[\tau, \tau + \delta\tau]$. Hence for fixed values of t, m_t^T and ℓ_t^T change in this way only for values of t within such infinitesimally small intervals of time. Hence, the integration over t only changes an infinitesimal amount. ■

Consider some fixed point in time T at which no job phase begins or completes under $EQUI_s$. What remains to be done is to prove that $[(F_{T+\delta T} + \widehat{F}_{T+\delta T}) - (F_T + \widehat{F}_T)]/\delta T \leq 0$. Let δT be small enough so that under both schedules no job phase begins or completes within either the interval $(T, T + \delta T)$ or the interval $(\tau, \tau + \delta\tau)$. Here $(\tau, \tau + \delta\tau)$ is a time interval during which $EQUI_s$ completes the same interval of fully parallelizable work that OPT completes during the interval $(T, T + \delta T)$. The next step is to prove the following relations between the defined counts.

Claim 2.

1. *$\ell_T^T = \ell_T$, $m_T^T = m_T$, and $n_T^T = n_T$.*
2. *For $t \geq T$, $n_t^T \leq n_T$.*

3. $\ell_t^{T+\delta T} \geq \ell_t^T$.

4. *If* $M_T = 0$*, then for all* $t \geq T$, $m_t^{T+\delta T} = m_t^T$*. If* $M_T = 1$*, then there is some interval of time* $(\tau, \tau + \delta\tau)$ *of length* $\delta\tau = \frac{n_\tau}{s}\delta T$ *such that for* $t \notin (\tau, \tau + \delta\tau)$, $m_t^{T+\delta T} = m_t^T$ *and for* $t \in (\tau, \tau + \delta\tau)$, $m_t^{T+\delta T} = m_t^T + 1$.

Proof of Claim 2: By Lemma 1, $EQUI_s$ is never ahead of OPT on any job. Hence if $EQUI_s$ is executing a job at time T, then OPT has completed it at time T or earlier. It follows that $\ell_T = \ell_T$, $m_T^T = m_T$, and $n_T^T = n_T$.

For the second point, consider some time $t \geq T$ and some job J_i included in the count n_t^T. By definition, J_i had a bit of its work completed under OPT before or at time T. Hence, J_i must have arrived at time T or earlier. By the definition of $EQUI_s$, we know that this schedule executes J_i continuously from its arrival time $\leq T$ until it completes the job. By the definition of n_t^T, $EQUI_s$ has not completed J_i by time $t \geq T$. It follows that $EQUI_s$ works on J_i at time T. In conclusion, J_i is included in the count n_T and hence $n_t^T \leq n_T$. (Note, however, because job J_i may switch between fully parallelizable and sequential phases, $m_t^T \leq m_T$ and $\ell_t^T \leq \ell_T$ are not necessarily true.)

The third point, $\ell_t^{T+\delta T} \geq \ell_t^T$, is true simply because the condition to be included in the first count has been relaxed from that of the second.

In order to prove the final point, recall that by the definition of the set of jobs being stream lined with respect to OPT, OPT works on either $M_T = 0$ or $M_T = 1$ fully parallelizable phase at time T. If $M_T = 0$, then the fully parallelizable work completed by OPT before time $T + \delta T$ is the same as that before time T. Hence, the requirements for a job to be included in the counts $m_t^{T+\delta T}$ and m_t^T are the same and so $m_t^{T+\delta T} = m_t^T$.

If $M_T = 1$, let J_i be the job whose fully parallelizable phase is worked on by OPT during the interval $(T, T + \delta T)$. Let $(\tau, \tau + \delta\tau)$ be the time interval during which $EQUI_s$ completes the same interval of work. Because δT is sufficiently small, no phase of any job begins or terminates under $EQUI_s$ during the interval $(\tau, \tau + \delta\tau)$. Hence, the number of jobs executing under $EQUI_s$ is the fixed number n_τ and the interval of fully parallelizable work is completed under $EQUI_s$ at a rate of $\Gamma(\frac{sp}{n_\tau}) = \frac{s}{n_\tau}p$. OPT completes this work with all p processors at a rate of $\Gamma(p) = p$. It follows that $\delta\tau = \frac{n_\tau}{s}\delta T$.

For $t \notin (\tau, \tau + \delta\tau)$, the fully parallelizable work completed under $EQUI_s$ at time t and by OPT before time $T + \delta T$ is the same as that by OPT before

time T. Hence, as before, $m_t^{T+\delta T} = m_t^T$. For $t \in (\tau, \tau + \delta\tau)$, there is one extra job that was completed by OPT before time $T + \delta T$ then that before time T. Therefore, $m_t^{T+\delta T} = m_t^T + 1$. ■

We are now ready to take the derivative of $F_T + \widehat{F}_T$.

$$\begin{aligned} F_T &= \int_0^T (m_t - \widehat{m}_t)\delta t \\ \left[F_{T+\delta T} - F_T\right]/\delta T &= m_T - \widehat{m}_T \end{aligned} \tag{1}$$

and

$$\begin{aligned} \widehat{F}_T &= \int_T^\infty \frac{s}{s-2}\frac{(m_t^T)^2}{n_t}\delta t \\ &\left[\widehat{F}_{T+\delta T} - \widehat{F}_T\right]/\delta T \\ &= \left[\int_{T+\delta T}^\infty \frac{s}{s-2}\frac{(m_t^{T+\delta T})^2}{n_t}\delta t - \int_T^\infty \frac{s}{s-2}\frac{(m_t^T)^2}{n_t}\delta t\right]/\delta T \\ &= \left[\int_{T+\delta T}^\infty \frac{s}{s-2}\frac{(m_t^{T+\delta T})^2}{n_t} - \frac{s}{s-2}\frac{(m_t^T)^2}{n_t}\delta t\right]/\delta T \qquad (2) \\ &- \left[\int_T^{T+\delta T} \frac{s}{s-2}\frac{(m_t^T)^2}{n_t}\delta t\right]/\delta T \qquad (3) \end{aligned}$$

We separately bound these lines, L_2 and L_3.

If $M_T = 0$, then L_2 is zero because for all $t \geq T$, $m_t^{T+\delta T} = m_t^T$ by Claim 2.4. If $M_T = 1$, then the integrand in L_2 is zero for the same reason except for $t \in (\tau, \tau + \delta\tau)$. For t within this range of length $\delta\tau = \frac{n_\tau}{s}\delta T$, we have $m_t^{T+\delta T} = m_t^T + 1$. This gives

$$\begin{aligned} L_2 &= \left[\frac{s}{s-2}\frac{(m_t^T+1)^2}{n_t} - \frac{s}{s-2}\frac{(m_t^T)^2}{n_t}\right] \times \left[\frac{n_\tau}{s}\right] \\ &= \frac{2m_\tau^T + 1}{s-2} \leq \frac{2n_T + 1}{s-2} = \frac{2m_T + 2\ell_T + 1}{s-2} \end{aligned}$$

The last inequality is because $m_\tau^T \leq n_\tau^T \leq n_T$ by Claim 2.2.[1]

L_3 is $\frac{s}{s-2}\frac{(m_T^T)^2}{n_T}$. By Claim 2.1, $\ell_T^T = \ell_T$ and $m_T^T = m_T$. By definition, $m_T + \ell_T = n_T$. Therefore,

$$L_3 = \frac{s}{s-2}\frac{(m_T^T)^2}{n_T} = \frac{s}{s-2}\frac{(m_T)^2}{m_T+\ell_T} = \frac{s}{s-2}\frac{m_T}{1+\frac{\ell_T}{m_T}} = \frac{s}{s-2}\frac{m_T(1+\frac{\ell_T}{m_T})-\ell_T}{1+\frac{\ell_T}{m_T}} = \frac{s}{s-2}\left(m_T - \frac{\ell_T}{1+\frac{\ell_T}{m_T}}\right.$$

Combining these three bounds gives

$$\begin{aligned}
&\left[\left(F_{T+\delta T} + \widehat{F}_{T+\delta T}\right) - \left(F_T + \widehat{F}_T\right)\right]/\delta T \\
&= L_1 + L_2 - L_3 \\
&\leq [m_T - \widehat{m}_T] + \left[\frac{2m_T + 2\ell_T + 1}{s-2}\right] - \frac{s}{s-2}\left[m_T - \frac{\ell_T}{1+\frac{\ell_T}{m_T}}\right] \\
&= \frac{2\ell_T+1}{s-2} + \frac{s}{s-2}\frac{\ell_T}{1+\frac{\ell_T}{m_T}} - \widehat{m}_T
\end{aligned}$$

Which, by the definition of $\widehat{m}_T$, is at most zero. ■

Lemma 4. *If* $\int_0^\infty (m_t - \widehat{m}_t)\delta t \leq 0$, *where* $\widehat{m}_t = \frac{2\ell_t+1}{s-2} + \frac{s}{s-2}\frac{\ell_t}{1+\frac{\ell_t}{m_t}}$, *then* $\int_0^\infty (m_t - \widehat{\widehat{m}}_t)\delta t \leq 0$, *where* $\widehat{\widehat{m}}_t = \mathcal{O}(\frac{\sqrt{s}}{s-2})\ell_t)$.

Proof of Lemma 4: Write the conclusions of the lemma as $\frac{\int_0^\infty m_t \delta t}{\int_0^\infty \ell_t \delta t} = \mathcal{O}(\frac{\sqrt{s}}{s-2})$.

Setting $b_t = \frac{m_t}{\ell_t}$ transforms the lemma as follows. If $\int_0^\infty (b_t - \frac{2+\frac{1}{\ell_t}}{s-2} + \frac{s}{s-2}\frac{1}{1+\frac{1}{b_t}})\ell_t \delta t \leq 0$, then $\frac{\int_0^\infty b_t \ell_t \delta t}{\int_0^\infty \ell_t \delta t} = \mathcal{O}(\frac{\sqrt{s}}{s-2})$.

[1] This ℓ_T is introduced because jobs that are currently sequential may have later fully parallel phases. Unfortunately this term increases the competitive ratio from $1 + \mathcal{O}(\frac{1}{s-2})$ to $1 + \mathcal{O}(\frac{\sqrt{s}}{s-2})$. I have not decided whether or not I think it is necessary.

Setting $\delta q = \ell_t \delta t$ and $q = \int_0^t \ell_{t'} \delta t'$ transforms the lemma as follows. If $\int_0^1 (b_q - \frac{2+\frac{1}{\ell_q}}{s-2} + \frac{s}{s-2}\frac{1}{1+\frac{1}{b_q}})\delta q \leq 0$, then $\frac{\int_0^1 b_q \delta q}{\int_0^1 1\delta 1} = \int_0^1 b_q \delta q = \mathcal{O}(\frac{\sqrt{s}}{s-2})$.

Suppose that b_q was a constant b. Solving this equation as required gives that $b = \mathcal{O}(\frac{\sqrt{s}}{s-2})$. What remains is to prove that the worst case is when b_q is a constant. We prove that using the contra positive. Under the restriction that $\int_0^1 b_q \delta q = b$, the amount $\int_0^1 (b_q - \frac{2+\frac{1}{\ell_q}}{s-2} + \frac{s}{s-2}\frac{1}{1+\frac{1}{b_q}})\delta q$ is minimized when b_q is a constant.

A standard trick to use here is as follows. If F is a function with positive second derivative, then $\sum_q F(b_q)$ is minimized subject to $\sum_q b_q = b$ all the b_q have the same value. Help from maple will tell us that indeed, the second derivative of the required F is positive. ■

5 Upper Bound for $EQUI^s$

Consider now the scheduler $EQUI^s$ which has p processors of speed s, in contrast to $EQUI_s$ which has sp processors of speed 1. Given jobs with sublinear speedup curves, $EQUI^s$ is at least as powerful as $EQUI_s$. Restricted to fully parallelizable jobs, the models are equivalent. This is not the case with strictly sublinear jobs. For example, sequential jobs execute s times faster under $EQUI^s$ than under $EQUI_s$.

On the other hand, the following example demonstrates that having the sequential jobs execute faster does not necessarily help. Suppose job J_i is released at time zero, has a sequential phase of work i and a fully parallelizable phase of work p. Under OPT, the flow time is $\sum i + 1$. Note $\sum i$ of this flow is sequential work. However, if OPT was able to complete the sequential work in zero time, then the flow time would still be $\sum i + 1$. Now, however, $\sum i$ of this flow is idle time.

When $s = 2 + \epsilon < 4$, our upper bound for $EQUI^s$ and $EQUI_s$ are the same, $2 + \frac{4}{\epsilon}$. However, for large s, $EQUI^s$ is competitive with a ratio of at most $\frac{16}{s}$, while $EQUI_s$ is not, having a ratio of at least $1 + \frac{1}{s}$.

Theorem 2. *Consider any set of jobs J with arbitrary arrival times, each job has an arbitrary number of phases, each phase has an arbitrary sublinear-nondecreasing speedup function. Suppose the processors un-*

der $EQUI^s$ execute s times faster than under OPT. Then for $s \geq 4$, $F(EQUI^s(J))/F(OPT(J)) \leq \frac{16}{s}$.

Proof of Theorem 2: The first step is to prove that with s speed, the flow time decreases by at least a factor of s or more generally, $F(EQUI^{ab}(J)) \leq \frac{1}{a}F(EQUI^b(J))$. Let $J_{r/a}$ be the same set of jobs except job J_i is released at time r_i/a instead of at time r_i. Clearly, $F(EQUI^{ab}(J_{r/a})) = \frac{1}{a}F(EQUI^b(J))$, because the entire schedule is simply scaled. Then note that $F(EQUI^s(J)) \leq F(EQUI^s(J_{r/a}))$, because the jobs are completing less for resources. Letting $a = s/4$ and $b = 4$, the theorem follows. $F(EQUI^s(J)) \leq F(EQUI^s(J_{4r/s})) = \frac{4}{s}F(EQUI^4(J)) \leq \frac{4}{s}F(EQUI_4(J)) \leq \frac{4}{s}\frac{2\times 4}{4-2}F(OPT(J))$. ■

The same technique is able to prove that BAL_s is $\frac{4}{s}$ competitive for large values of s. A tighter bound of $\frac{2}{s}$ has recently been proven [1].

Theorem 3. *With only fully parallelizable jobs, for $s \geq 2$, $F(BAL_s(J))/F(OPT(J)) \leq \frac{4}{s}$.*

Proof of Theorem 3: $F(BAL_s(J)) \leq \frac{2}{s}F(BAL_{1+1}(J)) \leq \frac{2}{s}(1 + \frac{1}{1})F(OPT(J))$. ■

6 Strictly Sub-Linear Speedup Functions

Recall a speedup function Γ is *strictly-sublinear* by α iff $\forall \beta_1 \leq \beta_2$, $\Gamma(\beta_2)/\Gamma(\beta_1) \leq (\beta_2/\beta_1)^{1-\alpha}$. This includes the *almost fully parallelizable* speedup function $\Gamma(\beta) = \beta^{1-\alpha}$, ⌞, the sequential speedup function ⌞, and everything in between. It is quite surprising that $EQUI$ with no extra resources is competitive for such jobs.

Theorem 4. *Consider any set of jobs J with arbitrary arrival times, each job has an arbitrary number of phases, and each phase has an arbitrary strictly-sublinear by α, nondecreasing speedup function. Suppose OPT and $EQUI$ have the same number of processors, i.e., $s = 1$. Then $F(EQUI(J))/F(OPT(J)) \leq 2^{1/\alpha}$.*

The proof is similar to that for Theorem 1. First Lemma 5 proves that the worst case job sets are those with only sequential and almost fully parallelizable phases. Then Lemma 6 bounds the competitive ratio for such job sets. The main difference is that here we do not require the job set to be stream lined wrt OPT. OPT may decide to work on many almost fully parallelizable phases simultaneously. In fact, as seen in the example in Theorem 6, it can be necessary for OPT to do this.

Lemma 5. *Consider any non-clairvoyant scheduler S_s with sp processors. For every job set J as stated, there is a job set J' with only sequential and almost fully parallelizable phases such that $F(S_s(J')) \geq F(S_s(J))$ and $F(OPT(J')) \leq F(OPT(J))$. Moreover, S_s is never ahead on any job.*

Proof of Lemma 5: The proof is similar to that in Lemma 1. Let Γ_i denote the speedup function of the interval of work completed under OPT in J_i during the time interval $(T, T + \delta T)$. Let $(\tau_i, \tau_i + \delta\tau_i)$ be the interval during which S_s completes the same work. Let β_i^{OPT} denote the number of processors allocated under OPT to it and $\beta_i^{S_s}$ the number under S_s.

The case that changes is when $\beta_i^{OPT} \geq \beta_i^{S_s}$. As before, $\delta\tau_i/\delta T = \Gamma_i(\beta_i^{OPT})/\Gamma_i(\beta_i^{S_s})$. Now, however, because the speedup function is sublinear, $\Gamma_i(\beta_i^{OPT})/\Gamma_i(\beta_i^{S_s}) \leq (\beta_i^{OPT}/\beta_i^{S_s})^{1-\alpha}$. Hence, $\delta\tau_i \leq (\beta_i^{OPT}/\beta_i^{S_s})^{1-\alpha}\delta T$.

We modify J^T to $J^{T+\delta T}$ in this case by replacing this interval of work in job J_i with an almost fully parallelizable phase with work $(\beta_i^{OPT})^{1-\alpha}\delta T$. First we need to check that this change does not increase the flow time under OPT. OPT can still allocate β_i^{OPT} processors to the phase. Because it is almost fully parallelizable it completes at a rate of $\Gamma(\beta_i^{OPT}) = (\beta_i^{OPT})^{1-\alpha}$ completing in time δT as before.

Now we need to check that this change does not decrease the flow time under S_s. As before, because S_s is non-clairvoyant, it still allocates $\beta_i^{S_s}$ processors to complete this work on job J_i. Hence, it completes the almost fully parallelizable work at a rate of $\Gamma(\beta_i^{S_s}) = (\beta_i^{S_s})^{1-\alpha}$ completing in time $(\beta_i^{OPT}/\beta_i^{S_s})^{1-\alpha}\delta T \geq \delta\tau_i$. Therefore, S_s requires at least as much time to complete this interval as it did with the original work. Finally, note that as before scheduler S_s does not get ahead of OPT on this interval of work. ■

Lemma 6. *For any job set J that has almost fully parallelizable or sequential phases, $F(EQUI(J)) \leq 2^{1/\alpha} \cdot F(OPT(J))$.*

Proof of Lemma 6: The proof is very similar to that for Lemma 2. The result follows quickly after proving that $\int_0^\infty (m_t - \widehat{m}_t)\delta t \leq 0$ for some $\widehat{m}_t$. As in Lemma 3, this is proved by proving that a certain potential function $F_T + \widehat{F}_T$ is non-increasing. We start by defining $F_T + \widehat{F}_T$ and taking its derivative.

$$F_T + \widehat{F}_T = \int_0^T (m_t - \widehat{m}_t)\delta t + \int_T^\infty f_t\left(m_t^T, \ell_t^T\right)\delta t,$$

where $f_t(m,\ell) = \dfrac{(m-\ell)(m+\ell)^{1-\alpha}}{\alpha\,(n_t)^{1-\alpha}}$.

$$\frac{\left(F_{T+\delta T} + \widehat{F}_{T+\delta T}\right) - \left(F_T + \widehat{F}_T\right)}{\delta T}$$

$$= \quad m_T - \widehat{m}_T \tag{4}$$

$$+ \quad \left[\int_{T+\delta T}^\infty f_t\left(m_t^{T+\delta T}, \ell_t^{T+\delta T}\right) - f_t\left(m_t^{T+\delta T}, \ell_t^T\right)\delta t\right]/\delta T \tag{5}$$

$$+ \quad \left[\int_{T+\delta T}^\infty f_t\left(m_t^{T+\delta T}, \ell_t^T\right) - f_t\left(m_t^T, \ell_t^T\right)\delta t\right]/\delta T \tag{6}$$

$$- \quad f_T\left(m_T^T, \ell_T^T\right) \tag{7}$$

We separately bound each of these three lines and then set $\widehat{m}_T$ to be $m_T + L_5 + L_6 - L_7$ to make this derivative zero. As before, the line L_5 is at most zero, because $\frac{\delta f_t(m,\ell)}{\delta \ell} \leq 0$ and because $\ell_t^T \leq \ell_\tau^{T+\delta T}$. The line L_7 is also similar to that before

$$f_T\left(m_T^T, \ell_T^T\right) = \frac{(m_T^T - \ell_T^T)(m_T^T + \ell_T^T)^{1-\alpha}}{\alpha\,(n_T)^{1-\alpha}} = \frac{m_T - \ell_T}{\alpha}$$

because by Claim 2.1, $l_T^T = \ell_T$, $m_T^T = m_T$, and $m_T + \ell_T = n_T$.

Bounding L_6 will harder than it was in the proof of Lemma 3 because is M_T no longer restricted to zero or one. Suppose wlog that $J_1, \ldots, J_{M_T}$ are the jobs with active $\Gamma(\beta) = \beta^{(1-\alpha)}$ phases under OPT during the time interval $(T, T+\delta T)$ and that during this time OPT allocates $a_i p$ processors to J_i, where $\sum_{i\in[1..M_T]} a_i = 1$.

Recall that m_t^T is defined to be number of fully parallelizable phases executing under $EQUI_s$ at the time instance t for which OPT has completed this same instance of work by time T. Define $m_t^{T,i}$ in the same way, but include as well in the count jobs $J_{i'}$ if $i' < i$ and this same instance of work is completed during the interval $(T, T + \delta T)$. We will need the following properties of $m_t^{T,i}$.

Claim 3.

1. $m_t^{T,1} = m_t^T$ *and* $m_t^{T,M_T+1} = m_t^{T+\delta T}$.
2. *For each* $i \in [1..M_T]$, *there is some interval of time* $(\tau_i, \tau_i + \delta\tau_i)$ *of length* $\delta\tau_i = (a_i n_{\tau_i})^{1-\alpha}\delta T$ *such that for* $t \notin (\tau_i, \tau_i + \delta\tau_i)$, $m_t^{T,i+1} = m_t^{T,i}$ *and for* $t \in (\tau_i, \tau_i + \delta\tau_i)$, $m_t^{T,i+1} = m_t^{T,i} + 1$.
3. $m_t^{T,i+1} \leq m_t^T + M_T$.

Proof of Claim 3: The first point follows directly from the definitions.

The proof of the second point is similar to that for Claim 2.4. The only difference between $m_t^{T,i+1}$ and $m_t^{T,i}$ is whether it includes job J_i because of work OPT completes during the interval $(T, T + \delta T)$. Let $(\tau_i, \tau_i + \delta\tau_i)$ be the time interval during which $EQUI_s$ completes the same interval of work on J_i. Then in a way similar to that in Claim 2.4, we can prove that for $t \notin (\tau_i, \tau_i + \delta\tau_i)$, $m_t^{T,i+1} = m_t^{T,i}$ and for $t \in (\tau_i, \tau_i + \delta\tau_i)$, $m_t^{T,i+1} = m_t^{T,i} + 1$. OPT completes this work at a rate of $\Gamma(a_i p) = (a_i p)^{1-\alpha}$ and $EQUI$ completes it at a rate of $\Gamma(\frac{p}{n_{\tau_i}}) = (\frac{p}{n_{\tau_i}})^{1-\alpha}$. It follows that the length of the interval is $\delta\tau_i = (a_i n_{\tau_i})^{1-\alpha}\delta T$.

The final point follows from the second and from the fact that OPT is working on only M_T almost fully parallelizable jobs during the time interval $(T, T + \delta T)$. ■

Claim 3.1 allows us to telescope line L_6. Then Claim 3.2 allows us to remove the integrations.

$$\begin{aligned} L_6 &= \left[\sum_{i \in [1..M_T]} \int_{T+\delta T}^{\infty} f_t\left(m_t^{T,i+1}, \ell_t^T\right) - f_t\left(m_t^{T,i}, \ell_t^T\right) \delta t \right] / \delta T \\ &= \sum_{i \in [1..M_T]} \left[f_{\tau_i}\left(m_{\tau_i}^{T,i+1}, \ell_{\tau_i}^T\right) - f_{\tau_i}\left(m_{\tau_i}^{T,i+1} - 1, \ell_{\tau_i}^T\right) \right] \times (a_i n_{\tau_i})^{1-\alpha} \end{aligned}$$

By convexity, $f_{\tau_i}(m,\ell) - f_{\tau_i}(m-1,\ell) \leq \frac{\delta f_{\tau_i}}{\delta m}(m,\ell)$. Differentiating $f_{\tau_i}(m,\ell) = \frac{(m-\ell)(m+\ell)^{1-\alpha}}{\alpha\left(n_{\tau_i}\right)^{1-\alpha}}$ gives

$$\begin{aligned} L_6 &\leq \sum_{i\in[1..M_T]} \frac{(2-\alpha)m_{\tau_i}^{T,i+1} + \alpha\ell_{\tau_i}^T}{\alpha\left(m_{\tau_i}^{T,i+1} + \ell_{\tau_i}^T\right)^{\alpha}} \times (a_i)^{1-\alpha} \\ &\leq \sum_{i\in[1..M_T]} \frac{(2-\alpha)}{\alpha}\left(m_{\tau_i}^{T,i+1} + \ell_{\tau_i}^T\right)^{1-\alpha} \times (a_i)^{1-\alpha} \\ &\leq \frac{(2-\alpha)}{\alpha}\left(n_T + M_T\right)^{1-\alpha} \times \sum_{i\in[1..M_T]} (a_i)^{1-\alpha} \end{aligned}$$

The last inequality uses Claim 3.2 that $m_{\tau_i}^{T,i+1} \leq m_{\tau_i}^T + M_T$ and Claim 2.2 that $m_{\tau_i}^T + \ell_{\tau_i}^T = n_{\tau_i}^T \leq n_T$. Finally, note that by convexity arguments it is clear that $\sum_{i\in[1..M_T]}(a_i)^{1-\alpha}$ under the restriction that $\sum_{i\in[1..M_T]} a_i = 1$ is maximized by having all the $a_i = 1/M_T$. This gives $M_T \times (1/M_T)^{1-\alpha} = (M_T)^{\alpha}$ and

$$L_6 \leq \frac{(2-\alpha)}{\alpha}\left(n_T + M_T\right)^{1-\alpha}\left(M_T\right)^{\alpha}$$

A bound on each of the three lines, $L_{??}$, L_2, and L_3 has been found. As said, we set $\widehat{m}_T$ to be $m_T + L_5 + L_6 - L_7$ to make the derivative of $F_T + \widehat{F}_T$ zero. From this we get that $\int_0^\infty (m_t - \widehat{m}_t)\delta t \leq 0$. From here, we proceed as done in the proof of Lemma 2.

$$\begin{aligned} \frac{F\left(EQUI_s(J)\right)}{F\left(OPT(J)\right)} &\leq \frac{\int_0^\infty \left(\ell_t + \widehat{m}_t\right)\delta t}{\int_0^\infty \left(L_t + M_t\right)\delta t} + 0 \leq \max_T \frac{\ell_T + \widehat{m}_T}{\ell_T + M_T} \\ &\leq \max_T \frac{\ell_T + [m_T] + [0] + \left[\frac{(2-\alpha)}{\alpha}\left(n_T + M_T\right)^{1-\alpha}\left(M_T\right)^{\alpha}\right] - \left[\frac{m_T - \ell_T}{\alpha}\right]}{\ell_T + M_T} \\ &= \max_T \frac{(2-\alpha)\left(n_T + M_T\right)^{1-\alpha}\left(M_T\right)^{\alpha} - (1-\alpha)n_T + 2\ell_T}{\alpha\left(\ell_T + M_T\right)} \end{aligned}$$

The last equality collected terms using $\ell_T + m_T = n_T$. Differentiating wrt n_T gives that this is maximized with $n_T = (2-\alpha)^{1/\alpha} M_T - M_T$. Plugging

this in and simplifying gives

$$= \max_T \frac{\frac{2}{\alpha}\ell_T + \left((2-\alpha)^{1/\alpha} + \frac{1-\alpha}{\alpha}\right) M_T}{\ell_T + M_T}$$

This is maximized either when ℓ_T goes to infinity and $M_T = 0$ or when $\ell_T = 0$ and M_T goes to infinity. Hence, the competitive ratio is at most

$$= \max\left(\frac{2}{\alpha}, (2-\alpha)^{1/\alpha} + \frac{1-\alpha}{\alpha}\right)$$

For $\alpha \in (0..1]$, this is at most $2^{1/\alpha}$. ■

7 Lower Bounds

Here, we present lower bounds both for non-clairvoyant schedulers in general and also for the specific schedulers BAL_s $EQUI_s$.

With the help of both sequential and fully parallelizable jobs, we achieve a $\Omega(\sqrt{n})$ lower bound on the competitive ratio for randomized non-clairvoyant schedulers. This is in marked contrast to the deterministic $\Omega(n^{1/3})$ and randomized $\widetilde{\Theta}(\log n)$ bounds [12, 22] which only use fully parallelizable jobs. With speed $s = 1+\epsilon$ processors, our randomized lower bound is $\Omega(\frac{1}{\epsilon})$, where no previous bound was known.

Theorem 5. *The competitive ratio of any randomized non-clairvoyant scheduler is $\Omega(\sqrt{n})$ if the jobs are allowed to be either fully parallelizable or sequential. If the scheduler is given speed $1+\epsilon$ processors then the ratio is at least $\Omega(\min(\frac{1}{\epsilon}, \sqrt{n}))$.*

As a warm up, try to find the flaw in the following proof that, as conjectured, the completive ratio is $\Omega(n)$.

Flawed $\Omega(n)$ Proof: Consider a stream of jobs where every time unit, one fully parallelizable job arrives with work p and one sequential job with work 1. OPT allocates all p processors to the fully parallelizable job, completing it in the alloted 1 time unit. The sequential job, requiring no processors,

completes in the alloted 1 time unit as well. The flow time is then n. A non-clairvoyant scheduler, not knowing which job is which will waste half the resources on the sequential job. Hence, it completes only half of the fully parallelizable work that has arrived. In the best case, this means that it has completed at most half of these jobs. This gives a flow time of $\Omega(n^2)$ for a competitive ratio of $\Omega(n)$. ■

There are two scheduling strategies that beat this bound. In the first, the scheduler simply sits idle for the first time unit. Then at time i, the $i-1^{st}$ sequential job would have completed on its own, so the scheduler will know which job is the $i-1^{st}$ fully parallelizable job and complete it. This gives a ratio of 1.5. $EQUI$, by *automatically self adjusting* as described in the intuition section, also does surprisingly well. We will leave it as an exercise that its ratio is $\mathcal{O}(\sqrt{n})$. (We find it interesting that this matches our lower bound.) Both BAL and its randomized version given in [12] have ratio $\Omega(n)$ for this job set.

The proof of our lower bound is much the same as that in Motwani *et al.* [22].

Proof of Theorem 5: Let $\epsilon \geq 0$. We use Yao's technique [31] and prove a lower bound on the competitive ratio of a deterministic algorithm on a job set chosen randomly from the following probability distribution.

The jobs are released in two phases. In the first phase, at time zero, $k = \Omega(\sqrt{n})$ jobs are released. Each such job is independently with probability $\rho = \frac{1}{8(1+\epsilon)}$ chosen to be a fully parallel job with $2p$ work. Otherwise, it is a sequential job with work $\frac{k}{2}$. Let F be the set of fully parallelizable jobs.

OPT completes the fully parallelizable jobs one at a time, each with all p processors. The expected number is $\frac{k}{8(1+\epsilon)}$ and Chernoff bounds give that with extremely high probability no more than $\frac{k}{4(1+\epsilon)}$ arrive. OPT can complete each in 2 time units, so can complete them all by time $t_1 = \frac{k}{2(1+\epsilon)}$. The sequential jobs complete on their own without any processors.

Fix some non-clairvoyant scheduler $S_{1+\epsilon}$ with speed $1+\epsilon$ and allow it to run for the same t_1 time units. Let x_i be the amount of processor time that it allocates to job J_i during this time. In general, the scheduler $S_{1+\epsilon}$ may be such that these amounts depend on the which jobs arrive or complete during this time. However, in this case, we claim that the values x_i are well defined and independent of the job set randomly chosen. Being non-clairvoyant, the

scheduler does not know which jobs are fully parallelizable and which are sequential until the time at which the job completes. If the job happens to be fully parallelizable, then it completes with resources $x_i = \frac{2p}{1+\epsilon}$. The sequential jobs do not require any processors, hence there is no point in ever allocating more resource than this to a job, i.e., $x_i \leq \frac{2p}{1+\epsilon}$. Because of this, no processor time needs to be reallocated to other jobs because some fully parallelizable job completes. The sequential jobs, even with speed $1+\epsilon$, do not complete during the t_1 time. Hence, resources are not reallocated when they complete either.

The total processor time during this time is $\sum_{i\in[1..k]} x_i = pt_1 = \frac{pk}{2(1+\epsilon)}$ and the average per job is $Avg_{i\in[1..k]}x_i = \frac{p}{2(1+\epsilon)}$. Let $X = \{J_i \mid x_i \leq \frac{p}{(1+\epsilon)}\}$ be those jobs who do not receive enough processor time to complete half their work if they happened to be fully parallelizable. Note that $|X| \geq \frac{k}{2}$. We assume that the jobs not in X and the sequential jobs complete by time t_1. However, the jobs in $F \cap X$ have p units of work remaining at this time. The expected number of such jobs is $\rho|X| \geq \frac{1}{8(1+\epsilon)}\frac{k}{2}$. Chernoff bounds give that with extremely high probability $|F \cap X| \geq \frac{k}{32(1+\epsilon)}$.

This completes the first phase. The flow time for the phase is $\Theta(k^2)$ under both schedulers. Under OPT, all the first round work has been completed and under $S_{1+\epsilon}$, $\Omega(k)$ jobs have at least p work remaining.

The second phase consist of a stream of ℓ fully parallelizable jobs each with work p. They arrive every time unit starting at time t_1 and ending at time $t_2 = t_1 + \ell$, where $\ell = \min(\frac{k}{32(1+\epsilon)\epsilon}, k^2)$. OPT is able to complete each as it arrives for a flow time of ℓ.

Suppose $\epsilon \geq \Omega(\frac{1}{k})$, so that $\ell = \frac{k}{32(1+\epsilon)\epsilon}$. In time ℓ, $S_{1+\epsilon}$ with speed $1+\epsilon$ is able to complete the $\frac{k}{32(1+\epsilon)} + \frac{k}{32(1+\epsilon)\epsilon} = \frac{k}{32\epsilon}$ jobs, i.e. those remaining from the first phase and the new ℓ jobs. The number alive is $\Omega(k)$ at time t_1 and decreases linearly to zero at time $t_1 + \ell$. Therefore, the flow time for this phase is $\Omega(k\ell)$. On the other hand, if $\epsilon \leq \mathcal{O}(\frac{1}{k})$, so that $\ell < \frac{k}{32(1+\epsilon)\epsilon}$, then $S_{1+\epsilon}$ will still have uncompleted jobs at time $t_1 + \ell$. Hence, the flow time is still $\Omega(k\ell)$.

The total number of jobs is $n = k + \ell = \mathcal{O}(k^2)$. To conclude the competitive ratio is $Flow(S_{1+\epsilon}(J))/Flow(OPT(J)) = \frac{\Omega(k^2)+\Omega(k\ell)}{\mathcal{O}(k^2)+\mathcal{O}(\ell)} \geq \Omega(\frac{\ell}{k}) = \Omega(\min(\frac{1}{\epsilon}, \sqrt{n}))$. ■

Now we present a number of examples of jobs sets that act as lower bounds on the competitive ratio under the specific scheduler, BAL_s or $EQUI_s$. It is interesting, that on the job sets on which BAL_s performs poorly, $EQUI_s$ behaves like OPT and similarly, on the job sets on which $EQUI_s$ performs poorly, BAL_s behaves like OPT. This is one of reasons that it is a difficult and open problem whether the same lower bounds apply generally to non-clairvoyant schedulers.

Though $BAL_{1+\epsilon}$ performs competitively on fully parallelizable jobs, it performs very poorly when given sublinear jobs. For example, when a sequential job arrives, it allocates all the processors to it and all are wasted.

Theorem 6. *There is a job set that contains only jobs that are almost fully parallelizable, i.e.,* $\Gamma(\beta) = \beta^{1-\alpha}$ *for which* BAL^s *has a competitive ratio of* $\Omega(s^{-1/\alpha}n)$.

Proof of Theorem 6: Job J_i, in the job set, is released at time i, has speedup function $\Gamma(\beta) = \beta^{1-\alpha}$, and work $(1+\epsilon)sp^{1-\alpha}$ for some small $\epsilon > 0$. During the time interval $[i, i+1]$, BAL^s completes work on job J_i at a rate of $s \times \Gamma(p) = sp^{1-\alpha}$, not quite completing the job. When the job J_{i+1} is released, all the processors are reallocated to it, hence J_i is never completed. It follows that the response time of job J_i is at least $n - i$ for a flow time of $n^2/2$. In contrast, OPT allocates p/M processors to each job, where $M = [(1+\epsilon)s]^{1/\alpha}$. Each job, completing at a rate of $\Gamma(p/M) = (p/M)^{1-\alpha}$ $= (p/[(1+\epsilon)s]^{1/\alpha})^{1-\alpha} = [(1+\epsilon)sp^{1-\alpha}]/[(1+\epsilon)s]^{1/\alpha}$, requires $M = [(1+\epsilon)s]^{1/\alpha}$ time units to complete. Hence, at any point in time, there are at most M jobs alive and so OPT is able to allocate p/M processors to each. The flow time under OPT is Mn giving a competitive ratio of $n/(2[(1+\epsilon)s]^{1/\alpha})$. ■

It is interesting to note that $EQUI$ with no additional resources, i.e., $s = 1$, is able to achieve a competitive ratio of 1 for this job set. Without knowing either the speedup function of the jobs or the rate at which work arrives, is able to automatically discover this optimal number of processors by *self adjusting*.

Motwani [13, 22] give a jobs set that is difficult for $EQUI$. We modify these to job sets that are difficult for $EQUI_s$ and for $EQUI^s$.

Theorem 7. *Even if restricted to fully parallelizable jobs, there is a job set for which* $EQUI_s$ *and* $EQUI^s$ *have competitive ratios of* $\Omega(n/\log n)$ *with*

$s = 1$, $\Omega\left(n^{1-\epsilon}\right)$ *with* $s = 1+\epsilon$, $\frac{2}{3}(1+\frac{1}{\epsilon})$ *with* $s = 2+\epsilon$, *and* $\frac{s}{s}$ *with* $s > 2$. *Considering sequential and fully parallelizable jobs, there is a jobs set for which* $EQUI_s$ *has a competitive ratio of* $\frac{(s-1)}{(s-2)} = 1+\frac{1}{\epsilon}$, *where* $s = 2+\epsilon$. *Considering only jobs that are strictly sublinear by* α *and non-decreasing, eg.* $\Gamma(\beta) = \beta^{1-\alpha}$, *there is a jobs set for which* $EQUI$ *with no extra resources seems to have a competitive ratio of* $1.48^{\frac{1}{\alpha}}$. *This last result does not apply to speedup curves that are strictly sublinear in the sense that* $\Gamma(p) << p$, *but are linear when the jobs are allocated small numbers of processors, eg.* $\llcorner$ *or* $\Gamma(\beta) = (\widehat{\beta}\beta+1)/(\widehat{\beta}+\beta)$.

Proof of Theorem 7: For the first results, the job set consists of a *stream* of n fully parallelizable jobs and $\ell = \ell_t$ extra fully parallelizable. The i^{th} stream job J_i has release time $r_i = \sum_{i'=0}^{i-1} t_i$ and work $w_i = t_i p$, where $t_0 = 1$ and $t_i = t_{i-1} - \frac{s}{\ell+i}t_{i-1}$. The ℓ extra jobs are the same as the first stream job J_0, having released at time r_0 and work $w_0 = 1 \times p$. OPT ignores the extra jobs and uses all p processors to complete the stream in place, with a flow time of $\sum_{i=0}^{n-1}(\ell+1)t_i$. After time t_n, OPT must complete the ℓ extra jobs with an additional flow time of $\frac{\ell^2}{2}$. See Figure 4.

In contrast $EQUI^s$ executes all the jobs, completing none. By induction, we can see that at time r_i there are $\ell+i+1$ jobs alive, each with w_i work remaining. It is true for $i = 0$, so assume that at time r_{i-1} there are $\ell+i$ jobs alive with w_{i-1} work remaining. For the next t_{i-1} time steps, $EQUI^s$ allocates $p/(\ell+i)$ speed s processors to each job, leaving each with $t_{i-1}p - \frac{sp}{\ell+i}t_{i-1} = w_i$ work remaining. At time r_i, job J_i is released, giving $\ell+i+1$ jobs alive each with w_i work remaining. The flow time for this part is $\sum_{i=0}^{n-1}(\ell+i+1)t_i$. After time r_n, $EQUI^s$ must complete the remaining work. This requires an additional flow time of $t_n(\ell+n)^2/s$.

For sufficiently large $\ell >> s$, we can solve $t_i = \prod_{i'=1}^{i}\left(1-\frac{s}{\ell+i'}\right) \approx e^{\sum_{i'=1}^{i} -\frac{s}{\ell+i'}} \approx (\frac{\ell}{\ell+i})^s$. (This is also obtained by telescoping $\frac{\ell+i-s}{\ell+i}\frac{\ell+i-1-s}{\ell+i-1}\ldots$). This gives

$$\frac{F(EQUI^s(J))}{F(OPT(J))} = \frac{\left(\sum_{i=0}^{n-1}(\ell+i+1)\left(\frac{\ell}{\ell+i}\right)^s\right) + \Theta\left(n^{2-s}\right)}{\left(\sum_{i=0}^{n-1}(\ell+1)\left(\frac{\ell}{\ell+i}\right)^s\right) + \frac{\ell^2}{2}} \geq \frac{\ell^s\left[(\ell+i)^{-s+2}/(-s+2)\right]_0^n + \Theta\left(n^{2-s}\right)}{(\ell+1)\ell^s\left[(\ell+i)^{-s+1}/(-s+1)\right]_0^n + \frac{\ell^2}{2}}.$$

For $s = 1$, we get $\frac{\Theta(n)}{\Theta(\log n)}$. For $s = 1+\epsilon$, we get $\frac{\Theta\left(n^{-s+2}\right)}{\Theta(1)} = \Theta\left(n^{1-\epsilon}\right)$. For $s = 2+\epsilon$ and n tending to infinity, we get $\frac{\ell^2/(s-2)}{\ell^2/(s-1)+\ell^2/2} = \frac{2(s-1)}{(s-2)(s+1)} > \frac{2}{s}$.

Now consider the scheduler $EQUI_s$, when sequential jobs are allowed. Recall, that extra processors do not speed up sequential jobs. The job set is the same except that the ℓ extra job are sequential. They are released at time 0 and have work $\sum_{i'=0}^{n-1} t_i$ so that, as before, they are alive during the entire interval $[r_0, r_n]$. The only change to the schedules is that under OPT these extra jobs complete on their own and hence do not need to be completed at in the end at a cost of $\frac{\ell^2}{2}$. This changes the competitive ratio to $\frac{\ell^2/(s-2)}{\ell^2/(s-1)} = \frac{s-1}{s-2} = 1 + \frac{1}{\epsilon}$.

When considering $EQUI^s$ and sequential jobs, the job set is the same as above except that the extra sequential jobs have s times as much work. The schedule under $EQUI^s$ will be the same as that before, because now it can complete the sequential work s times faster. The flow time of OPT changes from $\left(\sum_{i=0}^{n-1}(\ell+1)t_i\right)$ to $\left(\sum_{i=0}^{n-1}(s\ell+1)t_i\right)$, which for large ℓ is different by a factor of s. Hence, the competitive ratio is $\frac{s-1}{s(s-2)}$. It is interesting that this lower bound is lower than that with only fully parallelizable jobs.

Now consider only jobs that are strictly sublinear by α and non-decreasing, eg. $\Gamma(\beta) = \beta^{1-\alpha}$. The job set is the same as that initially, except $\ell = 1$, $t_0 = 1$, and $t_i = t_{i-1} - \frac{1}{(\ell+i)^{1-\alpha}} t_{i-1}$. We were only able to integrate the resulting functions using Maple (and complaints from the tech staff about space usage) when considering specific values of α. Therefore, we computed the competitive ratio for $\alpha = \frac{1}{k}$ for $k = 2, 3, \ldots, 64$. Half of these Maple failed to integrate. The other half gave a competitive ratio tending quickly to $1.48^{\frac{1}{\alpha}}$.

Finally consider only the speedup function $\Gamma(\beta) = \beta$ for $\beta \leq \widehat{\beta}$ and $\Gamma(\beta) = \widehat{\beta}$ for $\beta \geq \widehat{\beta}$, $\llcorner$. The jobs set consists of $\frac{p}{\widehat{\beta}}$ identical copies of that for full parallelizable jobs, except each job has $w_i = t_i\widehat{\beta}$ instead of $w_i = t_i p$ work. The schedules under OPT and under $EQUI_s$ are the same as that before, except the number of processors allocated is the fraction $\frac{\widehat{\beta}}{p}$ of what it was before. Both flow times increase by the fraction $\frac{p}{\widehat{\beta}}$ so the competitive ratio remains the same. The same trick works for $\Gamma(\beta) = (\widehat{\beta}\beta + 1)/(\widehat{\beta} + \beta)$, except some additional large constant times as many copies are needed. ■

8 Restricting the Number of Preemptions

Preemptive scheduling allows the number of processors allocated to a job to be changed after the job starts its execution. This helps adapt to the uncertain and changing nature of jobs and workloads. Unfortunately, preemption may incur large overheads if it is applied frequently. To account for the cost preemptions, Edmonds *et al.*[10] prove that when the jobs all arrive at time zero, the number of preemptions can be decreased from n to $\log n$ with only a factor two increase in the competitive ratio. We prove a similar statement for jobs with arbitrary arrivals. If we double the resources to Equi-partition again, then we can modify the scheduler so that it preempts infrequently (in some since a logarithmic number of times) while staying competitive.

Theorem 8. *Let $c > 1$ be some constant (eg $c = \sqrt{2}$). There is a non-clairvoyant scheduler $EQUI'_{c^2 s}$ that has $c^2 sp$ processors and a competitive ratio of $\frac{2s}{s-2}$ and only preempts when the number of jobs in the system goes up or down by a factor of c.*

Proof of Theorem 8: Suppose that the last time a preemption occurred, there were $n_t = c^k$ jobs alive. Jobs can arrive and can complete. However, the scheduler does not preempt again until the number of alive jobs drops under $n_t/c = c^{k-1}$ or increases above $cn_t = c^{k+1}$. See Figure 5:A. During this time, each job is allocated $\frac{sp}{c^{k-1}}$ processors. Note no more than $c^2 sp$ processors are needed and each job always has as many processors as it would under $EQUI_s$. Hence, the competitive ratio is at least as good. ■

9 Nondecreasing Sublinear or Superlinear Speedup Functions

The previous assumption that all speedup functions are sublinear is not true when the jobs are both highly parallel and have a strong time-space trade off. In such a situation, the speedup function may be superlinear, $\sqcup\!\!\sqcup$. Edmonds *et al.*[10] prove that, when the jobs all arrive at time zero, allowing each phases of each jobs to be either sublinear or superlinear only increases the competitive ratio by a factor of two, even though the non-clairvoyant sched-

uler does not know which phases are which. The number of preemptions does, however, become infinite. We prove an analogous result.

Theorem 9. *There is a non-clairvoyant algorithm $HEQUI_{2s}$ that has $2sp$ processors and a competitive ratio of $\frac{2s}{s-2}$ for every job set J in which each phase of each job is either nondecreasing sublinear or superlinear.*

Proof of Theorem 9: The scheduler $HEQUI_{2s}$ (short for "Hybrid Equipartition") performs $EQUI_s$ with sp of the processor, allocating $\frac{sp}{n_t}$ processor to each of the n_t alive jobs and performs Round-Robin with the other sp processors allocating p processors to each job for $\frac{s}{n_t}$ fraction of the time. See Figure 5:B. The proof follows easily from Theorem 1 by converting each job set J in which each phase of each job is either nondecreasing sublinear or superlinear into a job set J' with only nondecreasing sublinear phases where $F(EQUI_s(J')) \geq F(HEQUI_{2s}(J))$ and $F(OPT(J')) \leq F(OPT(J))$.

The nondecreasing sublinear phases of J are not changed. Because $HEQUI_{2s}$ has sp processors executing $EQUI_s$, it completes the jobs at least as well as $EQUI_s$. Consider a slice of superlinear work completed under OPT during the time interval $[T, T + \delta T]$ with β^{OPT} processors and under $HEQUI_{2s}$ during $[\tau, \tau + \delta\tau]$ when there are n_τ jobs alive. Change this to a fully parallelizable phase with work $\beta^{OPT}\delta T$. Note OPT still completes this work with no change in the schedule.

Because OPT completes the original work, the amount of this work must be $w = \delta T\Gamma(\beta^{OPT})$. Because $HEQUI_{2s}$ allocates p processors to the phase for $\frac{s}{n_\tau}\delta\tau$ time, we know that $w \geq \frac{s}{n_\tau}\delta\tau\Gamma(p)$. Because the phase is superlinear, $\Gamma(\beta^{OPT})/\beta^{OPT} \leq \Gamma(p)/p$. This gives $\delta\tau \leq \frac{n_\tau}{sp}\beta^{OPT}\delta T$, which is the length of time for $EQUI_s$ to complete the new fully parallelizable phase with work $\beta^{OPT}\delta T$. ■

10 Nondecreasing Speedup Functions and Gradual Speedup Functions

Edmonds *et al.*[10] also considers the class of speedup functions whose only restriction is that they are *nondecreasing*. Such a function might be sublinear for some of its range and superlinear for other parts of its range, $\llcorner\!\sqsubset$. They also

consider the very general class of *gradual* speedup functions, $\curlywedge$. For both of these classes, they give a scheduler under which batch jobs have competitive ratio of $\Theta(\log p)$. [2] We prove an analogous result.

Theorem 10. *There is a non-clairvoyant algorithm $HEQUI'_{4s\log p}$ that has $4sp\log p$ processors and a competitive ratio of $\frac{2s}{s-2}$ for every set of gradual (or nondecreasing) jobs.*

Proof of Theorem 10: We now consider jobs that are only restricted to being nondecreasing or gradual. Such jobs may execute efficiently only when allocated a specific number of processors. A non-clairvoyant scheduler, not knowing this number, must executes each job with a large range of processor allocations. Separately for each $k \in [0..\log p]$, the scheduler $HEQUI'_{4s\log p}$ allocates 2^k processors to each of the n_t alive jobs for $\frac{4}{2^k}\frac{sp}{n_t}$ fraction of the time. The proof follows easily from Theorem 1 by converting each gradual job set J into a job set J' with only fully parallelizable jobs, where $F(EQUI_s(J')) \geq F(HEQUI'_{4s\log p}(J))$ and $F(OPT(J')) \leq F(OPT(J))$.

Consider a slice of gradual work completed under OPT during the time interval $[T, T+\delta T]$ with β^{OPT} processors and under $HEQUI'_{4s\log p}$ during $[\tau, \tau+\delta\tau]$ when there are n_τ jobs alive. Change this to a fully parallelizable phase with work $\beta^{OPT}\delta T$. Note OPT still completes this work with no change in the schedule.

Because OPT completes the original work, the amount of this work must be $w = \delta T\Gamma(\beta^{OPT})$. Because the phase is gradual, there is a $k \in [0..\log p]$ such that $2^k \leq 2\beta^{OPT}$ and $\Gamma(2^k) \geq \frac{1}{2}\Gamma(\beta^{OPT})$. Because $HEQUI'_{4s\log p}$ allocates 2^k processors to the phase for $\frac{4}{2^k}\frac{sp}{n_t}\delta\tau$ time, we know that $w \geq \frac{4}{2^k}\frac{sp}{n_t}\delta\tau\Gamma(2^k)$. This gives $\delta\tau \leq \frac{n_\tau}{sp}\beta^{OPT}\delta T$, which is the length of time for $EQUI_s$ to complete the new fully parallelizable phase with work $\beta^{OPT}\delta T$. ■

[2] Their nondecreasing result is stated as $\mathcal{O}(\log n)$. However, the batch model assumes that the number of jobs n is at most the number of processors p because all these jobs arrive and are executed at once. Because our jobs arrive at arbitrary times, n is assumed to be much bigger then p.

11 Open Problems

The performance of Equi-partition has been studied extensively using simulation, experimental, and queuing theoretical approaches. Our research constitutes a theoretical confirmation of these efforts.

The main open problem is to close the gaps between the lower bounds on the competitive ratio known for general non-clairvoyant schedulers and those known for the specific schedulers Equi-partition and Balance. This gap is given under various models in the following table.

	$s = 1$	$s = 1 + \epsilon$
◿	$\Omega(n^{1/3})$ vs $\Omega(\frac{n}{\log n})$	$\Omega(1)$ vs $\Omega(\frac{1}{\epsilon})$
⌙, ◺, or ◿	$\Omega(n^{1/2})$ vs $\Omega(\frac{n}{\log n})$	$\Omega(\frac{1}{\epsilon})$ vs $\Omega(n^{1-\epsilon})$

Giving the scheduler randomness helps the fully parallelizable case a great deal, lowering the completive ratio down to $\widetilde{\Theta}(\log n \log\log n)$. However, it is unknown whether randomness helps in the case where there may also be sequential jobs. A separate question is whether having the jobs be chosen randomly helps. A stronger adversarial model would allow the adversary to choose the jobs and then choose the arrival times subject to them arriving in a randomly chosen order.

Finally, some of the constants in this paper could be improved.

Thanks: I would like to thank Patrick Dymond, Faith Fich, Xiaotie Deng, and Toni Pitassi for their continued support with scheduling.

References

[1] P. Berman and C. Coulston. Speed is more powerful than clairvoyance SWAT 98.

[2] T. Brecht and K. Guha. Using parallel program characteristics in dynamic multiprocessor allocation policies. *Performance Evaluation*, 27 & 28:519–539, Oct. 1996.

[3] S. H. Chiang, R. K. Mansharamani, and M. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor

scheduling policies. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 33–44, 1994.

[4] X. Deng and P. Dymond. On multiprocessor system scheduling. In *Seventh ACM Symposium on Parallel Architectures and Algorithms*, June 1996.

[5] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *Seventh Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 159–167, Atlanta, Georgia, January 1996.

[6] X. Deng and E. Koutsoupias. Competitive implementation of parallel programs. In *Fourth Annual ACM–SIAM Symposium on Discrete Algorithms,*, pages 455–461, 1993.

[7] J. Edmonds. Scheduling in the Dark Improved results: manuscript 2001. *Blum's Special Issue of the Journal of Theoretic Computer Science,* 1999.

[8] *Proc.* 31^{st} *Ann. ACM Symp. on Theory of Computing*, pp. 179-188, 1999.

[9] J. Edmonds, On the Competitiveness of AIMD-TCP within a General Network Submitted to *Journal Theoretical Computer Science Lecture Notes in Computer Science*, Volume 2976/2004. *LATIN, Latin American Theoretical Informatics,* pp. 577-588, 2004.

[10] J. Edmonds, D. Chinn, T. Brecht, X. Deng. Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics. In 29^{th} *Ann. ACM Symp. on Theory of Computing*, pp. 120-129, 1997 and submitted to the *SIAM Journal on Computing.*

[11] J. Edmonds, S. Datta, and P. Dymond, TCP is Competitive Against a Limited Adversary *SPAA, ACM Symp. of Parallelism in Algorithms and Achitectures*, pp. 174-183, 2003.

[12] B. Kalyanasundaram and K. Pruhs. Minimizing flow time nonclairvoyantly In *Proceedings of the 38th Symposium on Foundations of Computer Science*, October 1997.

[13] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 214–221, October 1995.

[14] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[15] M. Kumar. Measuring parallelism in computation-intensive scientific/engineering applications. *IEEE Transactions on Computers*, 37(9):1088–1098, September 1988.

[16] S. Leonardi and D. Raz, Approximating total flow time on parallel machines, In *ACM Symposium on Theory of Computing*, 1997.

[17] S. Leutenegger and M. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–236, Boulder, Colorado, May 1990.

[18] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing*, pages 322–333, 1988.

[19] Matsumoto. Competitive Analysis of the Round Robin Algorithm 3rd International Symposium on Algorithms and Computation, 71-77, 1992.

[20] 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, May 1993.

[21] C. McCann and J. Zahorjan. Scheduling memory constrained jobs on distributed memory parallel computers. In *Proceedings of International Joint Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 95 and Performance 95*, pages 208–219, 1995.

[22] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theoretical Computer Science* (Special Issue on Dynamic and On-Line Algorithms), 130 (1994), pp. 17–47. Preliminary Version: Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 422–431.

[23] T. Nguyen, R. Vaswani, and J. Zahorjan. Maximizing speedup through self-tuning of processor allocation. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 463–468, Waikiki, HI, Apr. 1996.

[24] C. Phillips, C. Stein, E. Torng, J. Wein. Optimal Time-Critical Scheduling Via Resource Augmentation In 29^{th} *Ann. ACM Symp. on Theory of Computing*, pp. 140-149, 1997 and submitted to the *SIAM Journal on Computing.*

[25] U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, and P. Yu. Smart SMART bounds for weighted response time scheduling. To appear in *SIAM Journal on Computing.*

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

[27] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[28] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 159–166, 1989.

[29] 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 *6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, June 1994.

[30] J. Turek, U. Schwiegelshohn, J. Wolf, and P. Yu. Scheduling parallel tasks to minimize average response time. In *Proceedings of the 5th SIAM Symposium on Discrete Algorithms*, pages 112–121, 1994.

[31] A. Yao, Probablistic Computations: Towards a Unified Measure of Complexity. In *Proc. of 18th IEEE Symp. on Foundations of Computer Schience* (1977) 222-227.

[32] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–225, Boulder, Colorado, May 1990.

	$s = 1$	$s = 1 + \epsilon$	$s = 2 + \epsilon$	$s = 4 + 2\epsilon$	$s = \mathcal{O}(\log p)$
Batch ⌞, ⌞, or ◿	$[2.71, 3.74]$				
Det. Non-clair ◿	$\Omega(n^{\frac{1}{3}})$	–			
Rand. Non-clair ◿	$\widetilde{\Theta}(\log n)$	–			
Rand. Non-clair ⌞ or ◿	$\Omega(n^{\frac{1}{2}})$	$\Omega(\frac{1}{\epsilon})$			
BAL_s ◿	$\Omega(n)$	$1 + \frac{1}{\epsilon}$	$\frac{2}{s}$		
BAL_s ⌞	$\Omega(s^{-1/\alpha} n)$				
$EQUI_s$ ⌞, ⌞, or ◿	$\Omega(\frac{n}{\log n})$	$\Omega(n^{1-\epsilon})$	$[1 + \frac{1}{\epsilon}, 2 + \frac{4}{\epsilon}]$	≥ 1	
$EQUI^s$ ⌞, ⌞, or ◿	$\Omega(\frac{n}{\log n})$	$\Omega(n^{1-\epsilon})$	$[\frac{2}{3}(1 + \frac{1}{\epsilon}), 2 + \frac{4}{\epsilon}]$	$[\frac{2}{s}, \frac{16}{s}]$	
$EQUI$ ⌞ or ⌞	$[1.48^{1/\alpha}, 2^{1/\alpha}]$				
$EQUI'_s$ Few Preempts			$\Omega(n^{1-\epsilon})$	$\Theta(1)$	
$HEQUI_s$ ⌞ or ⌞			$\Omega(n^{1-\epsilon})$	$\Theta(1)$	
$HEQUI'_s$ ⌞ or ⌒	$\Omega(n)$				$\Theta(1)$

Figure 1: Each row represents a specific scheduler and a class $\mathcal{J}$ of job sets. Here $EQUI_s$ denotes the Equi-partition scheduler with s times as many processors and $EQUI^s$ the one with processors that are s times as fast. The graphs give examples of speedup functions from the class of those considered. The columns are for different extra resources ratios s. Each entry gives the corresponding ratio between the given scheduler and the optimal.

a: Fully Par | b: Sequential | c: Typical NonDec Sub-Lin | d: Worst Case SupLin | e: Worst Case NonDec | f: Typical Gradual

Figure 2: Examples of speedup functions.

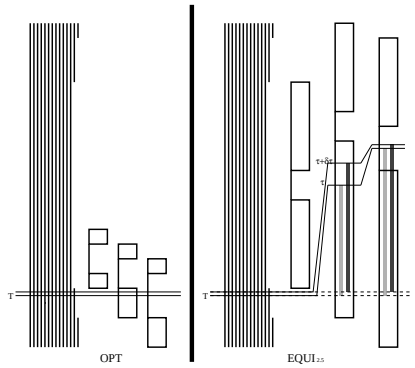


Figure 3: On the left is the OPT schedule and on the right the $EQUI_{2.5}$ schedule for some job set J. Each sequential phase is indicated with a line (closely spaced). Each fully parallelizable phase is indicated by a box. To ease calculations, J is rigged so that the number of jobs alive under $EQUI_{2.5}$ is always $n_t = 15$. Hence, $EQUI_{2.5}$ allocates $\frac{2.5p}{15}$ processors to each job. Given this, the fully parallelizable phases require six times as long to complete as under OPT. The sequential phases, which dominate the flow, require the same time under the two schedulers. The solid horizontal lines on both the left and on the right indicate how much work has been completed under OPT by time T and by time $T + \delta T$. The dotted lines on the right indicate the same for $EQUI_{2.5}$. The work W_T not completed by $EQUI_{2.5}$, but completed by OPT by time T is indicated by a light shaded bar. The dark shaded bar indicates the same for time $T + \delta T$. The bottom difference between these indicate the interval of work completed under $EQUI_{2.5}$ during the interval $[T, T + \delta T]$. The top difference indicates that completed under OPT during the interval $[T, T + \delta T]$ and under $EQUI_{2.5}$ during the interval $[\tau, \tau + \delta\tau]$.

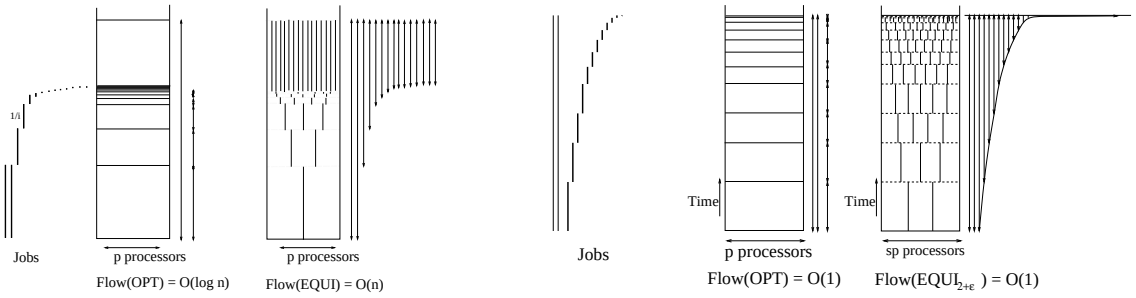


Figure 4: The left most figure represents the Motwani job set with $s = 1$ and fully parallelizable jobs. The beginning of each line gives the arrival time of the job and the end give the completion time when the job is allocated all p processors. The second and third figures represent the schedule of these jobs under OPT and under $EQUI$. The area of each region represents the processor-time block allocated to each job. Each arrow indicates when a job arrives and completes. The remaining figures are the same except $s = 2 + \epsilon$ and two of the jobs are sequential. These Motwani examples are designed so that as long as jobs are being release, none of the fully parallelizable jobs complete. The larger the number of fully parallelizable alive jobs m_t gets, the smaller the number of processors each job is allocated, the longer each job requires to complete, the larger m_t gets. This feed back continues. More specifically, the work of the arriving job is set to be the same as the work remaining in each of the other jobs under $EQUI$. Hence, all the jobs always have the same remaining work.

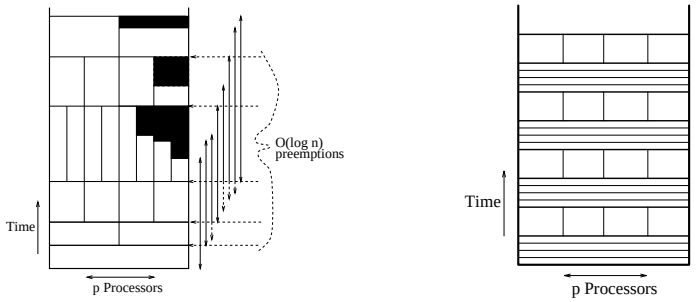


Figure 5: A: $EQUI$ with "$\log n$" preemptions. B: $HEQUI_{2s}$ simultaneously running $EQUI_s$ and $Round - Robin_s$.