Minimizing Maximum Flowtime of Jobs with Arbitrary Parallelizability

Kirk Pruhs[1,⋆], Julien Robert[2], and Nicolas Schabanel[3]

[1] Pittsburgh University, USA
[2] Université de Lyon (LIP - ÉNS Lyon), France
[3] CNRS - Université Paris Diderot (LIAFA), France

Abstract. We consider the problem of nonclairvoyantly scheduling jobs, which arrive over time and have varying sizes and degrees of parallelizability, with the objective of minimizing the maximum flow. We give essentially tight bounds on the achievable competitiveness. More specifically we show that the competitive ratio of every deterministic nonclairvoyant algorithm is high, namely $\Omega(\sqrt{n})$ for n jobs. But there is a simple batching algorithm that is $(1+\varepsilon)$-processor $O(\log n)$-competitive. And this simple batching algorithm is optimally competitive as no deterministic nonclairvoyant algorithm can be s-processor $o(\log n)$-competitive for any constant s.

1 Introduction

The founder of chip maker Tilera asserts that a corollary to Moore's law will be that the number of cores/processors will double every 18 months [11]. In this paper we consider one of the many resulting technical challenges that arises in such a future: developing algorithms/policies for scheduling jobs on many processors so as to optimize the resulting quality of service. Such a scheduler will be faced with scheduling jobs with highly varying degrees of parallelizability, that is, when allocated many processors some jobs may be considerably sped up, while on the other extreme, some jobs may not be sped up at all.

We will consider the setting where jobs of varying sizes arrive to the system over time, and must be scheduled online on a collection of identical processors. At each point in time the online scheduler needs to partition the processors among the alive jobs. For each portion of each job, there is an inherient speed-up function that specifies the rate at which the job is processed as a function of the number of processor on which it is run. An operating system scheduler generally needs to be *nonclairvoyant*, that is, the algorithm tipically does not have access to the internal knowledge about jobs, such as the size and the speed-up functions. The standard quality of service measure for a job is its flowtime, which is the length of time between when the job arrives to the system and which it is completed. One then normally obtains a quality of service measure

⋆ Supported in part by NSF grants CNS-0325353, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

for a schedule by taking the ℓ_p norm of the flowtimes for $1 \leqslant p \leqslant \infty$. The ℓ_p norm is the p^{th} root of the sum of the p^{th} powers of the flow times. The ℓ_1 norm is the total, or equivalently average, flowtime, and the ℓ_∞ norm is the maximum flowtime. Intuitively, the higher the value of p, the more importance that is being placed on avoiding starvation of jobs.

Formal definitions for all concepts in the introduction can be found in Section 2.

1.1 Previous Results

The model of speed-up functions that we adopt here was proposed in [4]. [4] showed that the natural algorithm Equi, which shares the processors equally among the jobs, is $O(1)$-competitive for the ℓ_1 norm of flow in the special case that all jobs arrive at the same time. Generalizing to the case of jobs with arbitrary release times, [3] gave a quite involved proof that Equi is $(2+\varepsilon)$-processor $O(\frac{1}{\varepsilon})$-competitive for the ℓ_1 norm of flow. Given that a nonclairvoyant algorithm does not know the speed-up functions, it is not clear what reasonable alternative algorithms there are to Equi, as there is no way for a nonclairvoyant algorithm to avoid the possibility that the jobs that it assigns most processors to may be the least parallelizable. However, [6] showed that, by sharing the processors evenly among most recently arriving constant fraction of the jobs, one obtains an existentially scalable algorithm (see definition p. 6), that is an algorithm that is $O(1)$-competitive with arbitrarily small processor augmentation. [6] also gives a much simpler proof of the competitiveness of Equi proved in [3].

The model was then extended in [16] to include arbitrary precedence constraints among tasks within each job. [16] showed that the introduction of precedence constraints does not affect the minimum processor augmentation required to be competitive for the ℓ_1 norm of flow, even if the resulting competitive ratio depends on the internal dependencies of each job.

The ℓ_p norm of flow, for $1 < p < \infty$ was recently considered in [7]. [7] showed that a simple algorithm that allocates the processors to the most recent alive jobs proportional to the $(p-1)^{\text{st}}$ power of their age is $(2+\varepsilon)$-processor $O(1)$-competitive. Very recently it was shown that this algorithm is existentially scalable [5].

The only previous work on the ℓ_∞ norm of flow was in [15]. [15] showed that if all the jobs are released all together at time 0, Equi is $O(\log n/\log\log n)$-competitive, and there is a matching general lower bound even allowing constant factor processor augmentation.

On single processor nonclairvoyant scheduling, or equivalently for multiprocessor scheduling when all the work is parallel, there has been a fair amount of work done to optimize ℓ_p norms of flow. Let us first consider the ℓ_1 norm. The competitive ratio of every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $\Omega(\log n)$ [12]. There is a randomized algorithm, Randomized Multi-Level Feedback Queues, that is $O(\log n)$-competitive against an oblivious adversary [10,2]. The algorithm Shortest Elapsed Time First, which shares the processors equally among the jobs that have been processed the least to date, is universally scalable [9]. For the ℓ_p norm of flows for $1 < p < \infty$, the competitive ratio of every randomized nonclairvoyant algorithm is $\Omega(n^{(p-1)/3p^2})$, and Shortest Elapsed Time First is universally scalable [1]. The nonclairvoyant algorithm First Come First Served is optimal for maximum flow.

There are many related scheduling problems with other objectives, and/or other assumptions about the machine and job instance. Surveys can be found in [14,13].

1.2 Our Results

So essentially what competitiveness is achievable by a nonclairvoyant algorithm for the ℓ_p norm of flow is known for finite p. In this paper we address the obvious remaining open question: What competitiveness is achievable for the case that $p = \infty$, that is for the objective of maximum flow. We give the following essentially tight results:

– In section 3 we show that the competitive ratio of every deterministic nonclairvoyant algorithm is high, namely $\Omega(\sqrt{n})$.
– In section 4 we show that there is a simple nonclairvoyant batching algorithm OBEQUI that is $(1+\varepsilon)$-processor $O(\log n)$-competitive. In OBEQUI there are always two active batches, the current batch and the next batch. The processors are shared equally among the current batch. Newly arriving jobs are added to the next batch. When all the jobs in the current batch finish, the next batch becomes the current batch.
– In section 5 we show that this simple batching algorithm is optimally competitive as no deterministic nonclairvoyant algorithm can be s-processor $o(\log n)$-competitive for any constant s.
– In section 6 we show that the techniques developed in [16] to handle precedence constraints when the objective is the ℓ_1 norm of flow can be extended to the ℓ_∞ norm of flow. Furthermore, we give here a modular presentation of these reduction-based techniques that will allow easy applications of the concepts in [16] to arbitrary non-clairvoyant setting.

We find it surprising that such a simple batching strategy is optimal, it was far from the first algorithm that we tried to analyze. Given the competitiveness results for nonclairvoyantly scheduling jobs with the objective of the ℓ_p norm of flow on a single and on multiple processors we make the following observations:

– On a single processor the ℓ_∞ norm is the easiest objective for the nonclairvoyant scheduler as First Come First Served produces an optimal schedule, and the best that a nonclairvoyant scheduler can do for the other norms is to be universally scalable. Of course scheduling on multiple processors is harder for a nonclairvoyant scheduler. But on mutiple processors, if jobs can have arbitrary parallelizability, then the ℓ_∞ norm is the hardest objective for the nonclairvoyant scheduler as it is the one objective where it is not possible for the scheduler to be at least existentially scalable. This suggests that perhaps starvation avoidance is a more difficult objective in a multiprocessor setting than in a single processor setting.

– By adding release times, the optimal competitive ratio increases significantly, from $\Theta(\log n/\log\log n)$ to $\Omega(\sqrt{n})$. But adding release dates only raises the competitiveness achievable by a nonclairvoyant algorithm with $(1+\varepsilon)$-processor augmentation from $\Theta(\log n/\log\log n)$ to $\Theta(\log n)$, and a larger constant factor processor augmentation doesn't improve the competitiveness achievable by a nonclairvoyant algorithm.

2 Definitions and Notation

We present here the model in its general form with arbitrary speed-up functions and precedences constraints. It turns out that our result proceeds by reduction to a much simpler setting, including only parallel and sequential speed-up functions and with no precedences constraint, which we will present first. Thus some of these definitions will not be needed before section 6.

The Setting. We consider a sequence of jobs $\{J_1, J_2, \ldots\}$ with release times $\{r_1, r_2, \ldots\}$. Following the terminology of [8,16], each job J_i consists in a set of tasks $\{J_{i,1}, \ldots, J_{i,m_i}\}$ with precedence constraints that the scheduler has to execute over p processors. Each task goes through different phases, where each phase may have a different speed-up function. The scheduler has to decide online the number of processors to allocate to each alive task. The scheduler is *non-clairvoyant*, *i.e.*, discovers the jobs at the time of their arrivals and the tasks at the time they become available; furthermore, it is unaware of the current speed-up of each task (*i.e.*, how they take advantage of more processing power) nor of the amount of work in each task; it is only informed that a task or a job is completed at the time of its completion. As in [3,16,6], we consider that the processors can be divided fractionally: fractional allocation is usually realized through time multiplexing in real systems.

Schedules. A *schedule* $\mathcal{S}_p$ on p processors is a set of piecewise constant functions[4] $\rho_{ij} : t \mapsto \rho_{ij}^t$ where ρ_{ij}^t is the *amount of processors* allotted to the task J_{ij} at time t; (ρ_{ij}^t) are arbitrary non-negative real numbers, such that at any time t: $\sum_{ij} \rho_{ij}^t \leqslant p$.

The jobs. The dependencies are defined as in [16] (extending the definition of [3,4]): each job J_i consists of a directed acyclic graph (*DAG* for short) $(\{J_{i\,1}, \ldots, J_{i\,m_i}\}, \prec)$, where task J_{ij} is released as soon as all tasks J_{ik}, such

[4] Requiring the functions (ρ_{ij}) to be piecewise constant is not restrictive since any finite set of reasonable (*i.e.*, Riemann integrable) functions can be uniformly approximated from below within an arbitrary precision by piecewise constant functions. In particular, all of our results hold if ρ_{ij} are piecewise continuous functions.

that $J_{ik} \prec J_{ij}$, are completed. Job J_i is completed as soon as all its tasks are completed. Each task goes through a sequence of phases $J_{ij}^1, \ldots, J_{ij}^{q_{ij}}$ with different degrees of parallelism. Each phase J_{ij}^k consists in an amount of work w_{ij}^k and a *speed-up function* Γ_{ij}^k. At time t, during its k-th phase, each task J_{ij} progresses at a *rate* $\Gamma_{ij}^k(\rho_{ij}^t)$ which depends on the amount ρ_{ij}^t of processors allotted to J_{ij} by the scheduler, *i.e.*, the amount of work accomplished between t and $t + dt$ in each task J_{ij} during its k-th phase is: $dw = \Gamma_{ij}^k(\rho_{ij}^t)dt$.

Given a schedule $\mathcal{S}_p$ of the jobs $\{J_1, J_2, \ldots\}$. A job or a task is *alive* as soon as it is released and until it is completed. Let c_{ij} denote the *completion time* of task J_{ij}. The *release time* r_{ij} of a task J_{ij} is: $r_{ij} = r_i$ (the release time of Job J_i) if $J_{i,j}$ does not depend on any other task (*i.e.*, if $J_{ik} \not\prec J_{ij}$ for all k); and $r_{ij} = \max\{c_{ik} : J_{ik} \prec J_{ij}\}$, otherwise. Let c_{ij}^k denote the completion time of the k-th phase of task J_{ij}: c_{ij}^k is the first time t' such that $w_{ij}^k = \int_{c_{ij}^{k-1}}^{t'} \Gamma_{ij}^k(\rho_{ij}^t)\,dt$ (with $c_{ij}^0 = r_{ij}$). Each task J_{ij} completes with its last phase, thus: $c_{ij} = c_{ij}^{q_{ij}}$. Job J_i is thus completed at time $c_i = \max_j c_{ij}$. A schedule is *valid* if all jobs eventually complete, *i.e.*, if $c_i < \infty$ for all i.

Cost of a schedule. The *flowtime* F_i of a job J_i is the overall time J_i is alive in the system, *i.e.*, $F_i = c_i - r_i$. The *maximum flowtime* of a schedule $\mathcal{S}_p$ is the maximum of the flowtimes of the jobs: $\text{MaxFlowTime}(\mathcal{S}_p) = \max_i F_i$. Our goal is to design a scheduler that minimizes the maximum flowtime, which corresponds to the largest response time of the system, which is a classic measure of quality of service. We denote by $\text{OPT}_p = \inf\{\text{MaxFlowTime}(\mathcal{S}_p) : \text{valid schedule } \mathcal{S}_p\}$, the optimal cost on p processors.

Speed-up functions. As in [4,16,6], we assume that each speed-up function is *non-decreasing* and *sub-linear* (*i.e.*, such that for all i, j, k, $\rho < \rho' \Rightarrow \frac{\Gamma_{ij}^k(\rho)}{\rho} \geqslant \frac{\Gamma_{ij}^k(\rho')}{\rho'}$). Non-decreasing means that allocating more processors to a job will not slow the processing of that job, and sub-linear means that efficiency decreases as the number of processors increase. As in [4,16,6], two types of speed-up functions will be of particular interest here:

- the *sequential* phases (Seq) where $\Gamma(\rho) = 1$, for all $\rho \geqslant 0$ (the task progresses at a constant rate even if *no* processor is allotted to it, similarly to an idle period); and
- the *parallel* phases (Par) where $\Gamma(\rho) = \rho$, for all $\rho \geqslant 0$.

We say that a job J_i is SeqPar if each of the phases of its tasks is either sequential or parallel. An instance is SeqPar if all of its jobs are SeqPar. For any task J_{ij} of a SeqPar job J_i, we define $\mathsf{Seq}(J_{ij})$ and $\mathsf{Par}(J_{ij})$ as the overall sequential and parallel works in the task respectively:

$$\mathsf{Seq}(J_{ij}) = \sum_{k:\ k\text{th phase of } J_{ij} \text{ is sequential}} w_{ij}^k \quad \text{and} \quad \mathsf{Par}(J_{ij}) = \sum_{k:\ k\text{th phase of } J_{ij} \text{ is fully parallel}} w_{ij}^k.$$

We denote by $\mathsf{Par}(J_i) = \sum_{J_{ij} \in J_i} \mathsf{Par}(J_{ij})$ the total amount of parallel work in the tasks of a SeqPar job J_i. We denote by

$\mathsf{Seq}(J_i) = \max_{J_{ij_1} \prec \cdots \prec J_{ij_k}} \sum_{\ell=1}^{k} \mathsf{Seq}(J_{ij_\ell})$, the maximum amount of sequential work along a chain of tasks in job J_i. We denote by $\mathsf{Par}(J) = \max_i \mathsf{Par}(J_i)$ and $\mathsf{Seq}(J) = \max_j \mathsf{Seq}(J_i)$ for any set of jobs $J = \{J_1, \ldots, J_n\}$.

Lemma 1 (Trivial lower bound, [3,16]) *For any SeqPar instance* $J_1, \ldots, J_n$*, we have:* $\mathrm{OPT}_1(J) \geqslant \max_i(\mathit{Par}(J_i), \mathit{Seq}(J_i))$.

Non-clairvoyant scheduling with precedence constraints. As in [4,16,6], we consider that the scheduler knows nothing about the progress of each tasks and is only informed that a job or a task is completed *at the time of its completion*; in particular, it is not aware of the different phases that each task goes through (neither of the amount of work nor of the speed-up function). Furthermore, tasks are released as soon as they become available without noticing the scheduler of the precedence constraints: if two tasks complete as other tasks are released, the scheduler is unable to guess which spawns which. In particular, as in [16], the order in which the tasks of a given job are released depends heavily on the computed schedule, and the scheduler cannot even reconstruct the DAG *a posteriori* in general. It is only aware at all time of the IDs of the current alive jobs and of their alive tasks.

Competitiveness and resource augmentation. We say that a given scheduler A_p is *c-competitive* if it computes a schedule $A_p(S)$ whose maximum flowtime is at most c times the optimal *clairvoyant* maximum flowtime (that is aware of the characteristics of the phases of each task and of the DAG of each job), *i.e.*, such that $\mathrm{MaxFlowTime}(A_p(J)) \leqslant c \cdot \mathrm{OPT}_p(J)$ for all instances J. A scheduler A_p is *s-processor c-competitive* if it computes a schedule $A_{sp}(J)$ on sp processors whose maximum flowtime is at most c times the optimal maximum flowtime on p processors only, *i.e.*, such that $\mathrm{MaxFlowTime}(A_{sp}(J)) \leqslant c \cdot \mathrm{OPT}_p(J)$ for all instances J [9]. A scheduler A is *universally scalable* if for every $\varepsilon > 0$, there is a constant c_ε such A_p is $(1+\varepsilon)$-processor c_ε-competitive. A family of algorithms $A_{p,\varepsilon}$ is *existentially scalable* if for every $\varepsilon > 0$, there is a constant c_ε such algorithm $A_{p,\varepsilon}$ is $(1+\varepsilon)$-processor c_ε-competitive.

Par$\rightarrow$Seq instances. A special case of SeqPar job will be of particular interest here. A job is said to be Par$\rightarrow$Seq if it consists in one single task consisting of only two phases: *one single parallel phase followed by one final sequential phase.* An instance is said to be Par$\rightarrow$Seq if all its jobs are Par$\rightarrow$Seq. As we will be show in Section 6, proving the competitiveness of our algorithm on Par$\rightarrow$Seq instances (proved in Section 4) will be enough to conclude its competitiveness on instances with arbitrary speed-up functions and precedence constraints.

3 The Lower Bound on the Competitive Ratio

Theorem 2 *There is no deterministic non-clairvoyant* 1*-processor c-competitive algorithm for any* $c < \sqrt{n}/4$ *for maximum flowtime even in the case that each consists of a single SeqPar task.*

Proof. Consider a deterministic non-clairvoyant algorithm A on 1 processor. Let n be the square of an even integer: $n = 4m^2$. The adversary releases 2 jobs J_i and J'_i at each time $t = i \in \{0, 1, \ldots, n/2-1\}$. Each job is composed of a parallel phase followed by a sequential phase of length 1. The amount of parallel work in each job is determined on-the-fly by the adversary according to the schedule computed by A so far (since A is non-clairvoyant, the adversary can set the phases afterwards). The total amount of parallel work within each pair of jobs J_i, J'_i will always be equal to 1. This unit of parallel work is split between J_i and J'_i as follows. Consider each time slot $[t, t+1]$ with $t \in \{i, i+1, \ldots\}$. Both jobs remain in a parallel phase as long as A allots at most $1/\sqrt{n}$ processors to each of them during each time slot $[t, t+1]$ and $t \leqslant i + \sqrt{n}/2 - 1$. Then, either we reach $t = i + \sqrt{n}/2$ and then the adversary sets the amount of parallel work in each job to $\frac{1}{2}$; since none of the jobs is allotted more than $1/\sqrt{n}$ processors on average, none of their parallel phases can be completed at time $i + \sqrt{n}/2$ and the instance is correctly defined. Otherwise, one the of job, say J_i, is allotted more than $1/\sqrt{n}$ processors during time slot $[t, t+1]$. Then, the adversary sets the amount of parallel work in J_i and J'_i to the total amount of processors each received from A between i and t (which sums to some $w < 1$), and gives the remaining parallel work $1 - w$ to J'_i.

The optimum can complete both parallel phases in each pair of jobs during the time unit after their release and thus guarantees a maximum flowtime of 2 for each job.

We claim that $\text{MaxFlowTime}(A) \geqslant \sqrt{n}/2$. Indeed, either there is one pair of jobs J_i, J'_i that were never allotted more than $1/\sqrt{n}$ processors each in each time slot $[t, t+1]$ for $t \in \{i, \ldots, i + \sqrt{n}/2 - 1\}$. Then, both of their parallel phases could not be completed at time $t = i + \sqrt{n}/2 - 1$ and their flowtime is $\geqslant \sqrt{n}/2$. Otherwise, one job in each pair was allotted at least $1/\sqrt{n}$ processors for its final sequential phase. Let $n + T$ be the completion time of the last completed job. $n + T$ has to be at least $n/2 \cdot 1/\sqrt{n} + n$, the total amount of processors wasted on sequential phases plus the total amount, n, of parallel work in the instance. It follows that $\text{MaxFlowTime}(A) \geqslant T \geqslant \sqrt{n}/2 \geqslant \sqrt{n}/4 \cdot \text{OPT}$.

4 Analysis of the Batching Algorithm on Par→Seq instances

We consider here only Par→Seq instances. Section 6 will show that one can reduce the general case to this simpler case. Recall our batching algorithm for job without precedence constraints, named OBEQUI (for Online Batching EQUI): it maintains at all time two batches; at the beginning, one batch contains the first released jobs, and the other one is empty; then repetitively, the jobs contained in the older batch are all scheduled together using the EQUI algorithm (each alive job in the batch receives an equal share of the processors) until all of them completes, while OBEQUI collects all the jobs released in between in the other batch; OBEQUI then switches the batches and restarts. We denote by $\mathcal{B}_k$ the

kth batch of jobs scheduled by OBEQUI. This section is dedicated to proving the following theorem.

Theorem 3 *The simple non-clairvoyant batching algorithm* OBEQUI *is* $(1+\varepsilon)$-*processor* $O(\frac{\log n}{\varepsilon^2})$-*competitive for maximum flowtime on* $\mathsf{Par}\!\rightarrow\!\mathsf{Seq}$ *instances, for all* $\varepsilon > 0$.

First, we give a lower bound on the optimal cost that we will use extensively.

Excess. The *maximum parallel work in excess* of a SeqPar instance $\{J_1, \ldots, J_n\}$ is defined as:

$$\mathsf{Exc}(J) = \min\bigl\{W : \forall(t \leqslant t')\ \textstyle\sum_{r_i \in [t,t']} \mathsf{Par}(J_i) \leqslant t' - t + W\bigr\},$$

i.e. the maximum quantity of parallel work received during any time interval that cannot be scheduled on 1 processor within this interval. By minimality of Exc, consider a time interval $[t, t']$ such that $\sum_{r_i \in [t,t']} \mathsf{Par}(J_i) = t' - t + \mathsf{Exc}(J)$, no schedule can complete the parallel work received during $[t, t']$ before time $t' + \mathsf{Exc}(J)$, it follows that the flowtime of some job released in $[t, t']$ is at least $\mathsf{Exc}(J)$. Together with Lemma 1, we obtain the following lower bound that we will use to analyze our algorithms.

Lemma 4 (Lower bound) *For all* SeqPar *instance* $J_1, \ldots, J_n$,
$$\mathrm{OPT}_1(J) \geqslant \max(\mathsf{Exc}(J), \mathsf{Seq}(J)).$$

We use the following notations: γ_k denotes the completion time of the kth batch, $\mathcal{B}_k$, ($\gamma_0 = -\infty$ by convention); ρ_k denotes the release time of the first job in Batch $\mathcal{B}_k$; n_k denotes the number of jobs in Batch $\mathcal{B}_k$. Note that for all $k \geqslant 2$, $\max(\rho_{k-1}, \gamma_{k-2}) < \rho_k \leqslant \gamma_k$. Note also that the processing of batch $\mathcal{B}_k$ begins exactly at time $\beta_k =_{\text{def}} \max(\rho_k, \gamma_{k-1})$.

Let $T(n, \varepsilon) = \bigl(\frac{8}{\varepsilon} + \frac{32}{\varepsilon^2}\bigr) \log n \cdot \max(\mathsf{Exc}(J), \mathsf{Seq}(J))$.

The main idea of the analysis is to show that if the previous batch lasts at most $T(n, \varepsilon)$, then the next one will be completed in time at most $T(n, \varepsilon)$ as well. The batch $\mathcal{B}_k$ contains all the jobs released between time ρ_k and time $\beta_k = \max(\rho_k, \gamma_{k-1})$. Its processing starts at time β_k and ends at time γ_k. We will show the following:

Lemma 5 *For all* $\tau \geqslant T(n, \varepsilon)$*, if* $\beta_k - \rho_k \leqslant \tau$ *then* $\gamma_k - \beta_k \leqslant \tau$.

Proof. Since all the jobs in $\mathcal{B}_k$ are released in $[\rho_k, \beta_k]$, the total amount of parallel work of this batch is at most $\beta_k - \rho_k + \mathsf{Exc}(J)$ by definition of Exc. Assume that OBEQUI is run on $1 + \varepsilon$ processors with $\varepsilon \leqslant \frac{1}{2}$ (this assumption is not necessary but simplifies the calculations bellow). We now follow the lines of [15]. We partition the processing period of the batch $\mathcal{B}_k$ in two sets: A is the set of all instant $t \in [\beta_k, \gamma_k]$ such that a fraction at least $1 - \frac{\varepsilon}{8}$ of the alive jobs of $\mathcal{B}_k$ are in a parallel phase; $\bar{A}$ is its complementary set, *i.e.* the set of all instant $t \in [\beta_k, \gamma_k]$ such that a fraction more than $\frac{\varepsilon}{8}$ of the active jobs of $\mathcal{B}_k$ are in their final sequential phase. By construction,

$$\gamma_k - \beta_k = \int_A dt + \int_{\bar{A}} dt.$$

At each instant $t \in A$, the amount of parallel work decreases at a rate at least $(1+\varepsilon)(1-\frac{\varepsilon}{8})$. It follows that:

$$\int_A dt \leqslant \frac{\mathsf{Par}(\mathcal{B}_k)}{(1+\varepsilon)(1-\frac{\varepsilon}{8})} \leqslant (\tau + \mathsf{Exc}(J))(1-\tfrac{\varepsilon}{4}),$$

since $\varepsilon \leqslant \frac{1}{2}$. Let $\mathsf{Seq}(\mathcal{B}_k) = \max_{J_i \in \mathcal{B}_k} \mathsf{Seq}(J_i)$. Clearly, $\mathsf{Seq}(\mathcal{B}_k) \leqslant \mathsf{Seq}(J)$. As in [15], we cover $\bar{A}$ with a minimum number q of non-overlapping intervals of length $\mathsf{Seq}(\mathcal{B}_k)$. At the beginning of each of these q intervals, a fraction larger than $\frac{\varepsilon}{8}$ of the alive jobs in $\mathcal{B}_k$ are in their final sequential phase and will then be completed at the end of the interval. It follows that the number of alive jobs decreases by at least a factor $1-\frac{\varepsilon}{8}$ after each of these intervals. Thus, $q \leqslant -\log_{(1-\frac{\varepsilon}{8})} n_k$. Then

$$\int_{\bar{A}} dt \leqslant q \cdot \mathsf{Seq}(\mathcal{B}_k) \leqslant -\frac{\log n_k}{\log(1-\frac{\varepsilon}{8})}\mathsf{Seq}(J) \leqslant \frac{8}{\varepsilon} \cdot \log n \cdot \mathsf{Seq}(J).$$

It follows that:

$$\gamma_k - \beta_k \leqslant \left(1-\tfrac{\varepsilon}{4}\right)(\tau + \mathsf{Exc}(J)) + \tfrac{8}{\varepsilon} \cdot \log n \cdot \mathsf{Seq}(J).$$

We are now left with proving that:

$$\left(1-\tfrac{\varepsilon}{4}\right)(\tau + \max(\mathsf{Exc}(J), \mathsf{Seq}(J))) + \tfrac{8}{\varepsilon} \cdot \log n \cdot \max(\mathsf{Exc}(J), \mathsf{Seq}(J)) \leqslant \tau$$

whenever $\tau \geqslant T(n,\varepsilon) = \left(\frac{8}{\varepsilon} + \frac{32}{\varepsilon^2}\right)\log n \cdot \max(\mathsf{Exc}(J), \mathsf{Seq}(J))$. This holds since by subtracting τ from both sides of this inequation, we get:

$$\begin{aligned}
&-\tfrac{\varepsilon}{4}\tau + \left(1 - \tfrac{\varepsilon}{4} + \tfrac{8\log n}{\varepsilon}\right)\max(\mathsf{Exc}(J), \mathsf{Seq}(J)) \\
&\leqslant \left(-\tfrac{\varepsilon}{4}\left(\tfrac{8}{\varepsilon} + \tfrac{32}{\varepsilon^2}\right)\log n + 1 - \tfrac{\varepsilon}{4} + \tfrac{8}{\varepsilon}\log n\right)\max(\mathsf{Exc}(J), \mathsf{Seq}(J)) \\
&\leqslant 0. \qquad \square
\end{aligned}$$

By immediate induction, the flowtime of every job is at most $2T(n,\varepsilon)$: $T(n,\varepsilon)$ for waiting to be scheduled, plus $T(n,\varepsilon)$ for its batch to be completed. Thus, Theorem 3 follows by the lower bound for OPT given in Lemma 4.

5 The General Lower Bound

This section will be devoted to proving the following theorem.

Theorem 6 *For all $\varepsilon > 0$, there is no deterministic non-clairvoyant $(1+\varepsilon)$-processor c-competitive algorithm for $c < \frac{1}{2} \cdot \log n$. This holds even if instances are restricted such that each job consists of a single $\mathsf{Par}\!\rightarrow\!\mathsf{Seq}$ task.*

Consider a deterministic non-clairvoyant algorithm A on $1+\varepsilon$ processors. Let $n = b \cdot m$ be the product of two integers such that: $m \sim n^{1-1/\sqrt{\log n}}$ and $b \sim n^{1/\sqrt{\log n}} = e^{\sqrt{\log n}}$. Let $F = \log n$. The adversary releases m jobs $J_1^i, \ldots, J_m^i$ at each integer time $t = i \in \{0, 1, \ldots, b-1\}$; the set $J_1^i, \ldots, J_m^i$ is referred as the ith batch with $0 \leqslant i < b$. Each job is composed of a parallel phase followed by a sequential phase of length 1. The adversary will ensure that the total amount of parallel work in each batch is at most 1, so that the optimum can schedule all the parallel work between t and $t+1$ on one processor and thus complete every job within 2 time units, i.e. $\mathrm{OPT}_1 \leqslant 2$.

The adversary sets the parallel work of the jobs in each batch as follows. Let j_t^i denote the number of alive jobs in the i-th batch at time $i+t$ ($j_0^i = m$). At each time $i+t$, with $t \in \{1, \ldots, F\}$, and as long as $j_{t-1}^i > 0$, the adversary sorts the j_{t-1}^i surviving jobs of the batch by non-decreasing average number of processors allotted by A during $[i+t-1, i+t]$. The adversary selects the maximum k such that the amount of processors allotted by A to the k first jobs in that order is at most $1/F$. The adversary sets these k first jobs in a parallel phase during $[i+t-1, i+t]$ and sets the $j_t^i - k$ others in their final sequential phase (they are thus completed at time t). Note that after that $j_t^i = k$. All the surviving jobs (if any) are forced to enter their final sequential phase at time $t = i + F$.

Note that at most $1/F$ parallel work is injected into the jobs of the batch in each time slot; since the lifetime of each batch is at most F, each batch contains at most one unit of parallel work in total, which ensures that for this instance $\mathrm{OPT}_1 \leqslant 2$ as claimed earlier. We will now prove that $\mathrm{MaxFlowTime}(A) = \Omega(\log n)$.

Let s_t^i denote the total amount of processors allotted by A during $[i+t, i+t+1]$ to the surviving jobs of the i-th batch.

Lemma 7 *For all i and $t < F$ such that $j_t^i > 0$, we have: $j_{t+1}^i = j_t^i$ if $s_t^i \leqslant \frac{1}{F}$; and $j_{t+1}^i > \frac{j_t^i}{F \cdot s_t^i} - 1$, otherwise.*

Proof. During $[i+t, i+t+1]$, if $s_t^i \leqslant \frac{1}{F}$, all the jobs are set in a parallel phase and are still alive at time $i+t+1$, thus $j_{t+1}^i = j_t^i$. Otherwise, each alive job in the i-th batch is allotted on average s_t^i / j_t^i processors. It follows, by the maximality of k, that $(k+1)s_t^i / j_t^i > \frac{1}{F}$ and thus $j_{t+1}^i = k > \frac{j_t^i}{F \cdot s_t^i} - 1$.

Simple algebraic manipulation yields the following corollary:

Corollary 8 *For all i and $t < F$ such that $j_t^i \geqslant (1+\varepsilon)F^3$ and $s_t^i > \frac{1}{F}$, we have:*

$$j_{t+1}^i > \left(1 - \frac{1}{F^2}\right) \cdot \frac{j_t^i}{F \cdot s_t^i}.$$

Let $\Delta^i = \{t : 0 \leqslant t < F \text{ and } j_t^i \geqslant (1+\varepsilon)F^3 \text{ and } s_t^i > \frac{1}{F}\}$ denote the set of the time slots in the lifetime of the ith batch, where it receives from A at least $\frac{1}{F}$ processors, and where the number of alive jobs is at least $(1+\varepsilon)F^3$, so that the lower bound of the corollary above applies. Let $T^i = \#\Delta^i$ denote the size of Δ^i and $t_+^i = \max \Delta^i$. Note that the maximum flowtime of the jobs in the i-th batch is at least T^i.

Note that as long as $j_t^i \geqslant (1+\varepsilon)F^3$, we have $j_{t+1}^i < j_t^i$ only for $t \in \Delta^i$, and $j_{t+1}^i = j_t^i$ otherwise. If the lifetime of the ith batch is F, then the maximum flow time of A is F and we are done. Let us now assume that the flowtime of all batches is less than F. It follows that for all i, $j_t^i < (1+\varepsilon)F^3$ for some $t < F$, in particular: $\Delta^i \neq \varnothing$, $t_+^i \geqslant T^i > 0$, and $j_{1+t_+^i}^i < (1+\varepsilon)F^3$ (indeed, since $j_{t+1}^i < j_t^i$ only for $t \in \Delta^i$ as long as $j_t^i \geqslant (1+\varepsilon)F^3$, this threshold is crossed exactly between t_+^i and $1+t_+^i$).

Let $\overline{s}^i = \frac{1}{T^i}\sum_{t\in\Delta^i} s_t^i$ denote the average amount of processors allotted to the batch during the time slots in Δ^i. Note that $\overline{s}^i > \frac{1}{F}$ by construction. By Corollary 8,

$$(1+\varepsilon)F^3 > j_{1+t_+^i}^i > j_0^i \prod_{t\in\Delta^i} \frac{1-1/F^2}{F\cdot s_t^i} \geqslant \frac{m}{\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right)^{T^i}},$$

by log-concavity of the product. Now, by taking the log of both ends of the inequality above,

$$T^i > \frac{\log m - \log((1+\varepsilon)F^3)}{\log\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right)}, \tag{1}$$

since $\overline{s}^i > \frac{1}{F}$ and thus $\log\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right) > 0$.

To get the $\Omega(\log n)$ lower bound on some T^i, it suffices now to show that some batch gets a small enough average amount of processors $\overline{s}^i$ compared to $1/F$.

Lemma 9 *For large enough n, there exists a batch i such that $\overline{s}^i \leqslant \dfrac{e}{F}$*

Proof. Consider some constant K to be chosen later. We proceed by contradiction and assume that for all batches, $\overline{s}^i > K/F$. Since the lifetime of every of the b batches is at most F by construction, the total amount of processors used by Algorithm A is at most $(b+F)(1+\varepsilon)$. But, each batch i receives $\overline{s}^i$ processors on average during T^i time. It follows that:

$$(1+\varepsilon)(b+F) \geqslant \sum_{i=0}^{b-1} T^i \cdot \overline{s}^i \geqslant \sum_{i=0}^{b-1} \frac{\overline{s}^i}{\log\left(\frac{F\cdot\overline{s}^i}{1-1/F^2}\right)} \cdot (\log m - \log((1+\varepsilon)F^3)), \quad \text{by (1)}.$$

But $s \mapsto s/\log(a\cdot s)$ is increasing for $s \geqslant e/a$. Assume $K \geqslant e\cdot(1-1/F^2)$. Now $\overline{s}^i > K/F$ for all i by hypothesis, so:

$$\begin{aligned}(1+\varepsilon)\cdot n^{1/\sqrt{\log n}} \sim (1+\varepsilon)(b+F) &\geqslant b\cdot\frac{K/F}{\log(F\cdot(K/F)\,/\,(1-1/F^2))}\cdot(\log m - \log((1+\varepsilon)F^3)) \\ &= b\cdot\frac{K}{\log(K/(1-1/F^2))}\cdot\frac{(1-\frac{1}{\sqrt{\log n}})\log n - O(\log\log n)}{\frac{1}{1+\varepsilon}\log n} \\ &\sim (1+\varepsilon)\cdot\frac{K}{\log K}\cdot n^{1/\sqrt{\log n}}\end{aligned}$$

We obtain thus a contradiction for large enough n when K is chosen so that $K \geqslant e$ and $\frac{K}{\log K} > 1$, which is true for $K = e$ for all $\varepsilon > 0$. □

To conclude, consider now a batch i such that $\overline{s}^i \leqslant \frac{e}{F}$. By (1),

$$\text{MaxFlowTime}(A) \geqslant T^i \geqslant \frac{(1-\frac{1}{\sqrt{\log n}})\log n - O(\log\log n)}{\log(e/(1-1/F^2))} \geqslant (1-o(1))\cdot\log n\cdot\frac{\text{OPT}_1}{2}.$$

6 Reduction from the general setting to Par $\to$ Seq instances *(omitted due to space constraints)*

Using reductions from [3,15,16], we are able to show that Theorem 3 extends to the general setting, as follows:[5]

Theorem 10 *For all $\varepsilon > 0$, for all instance $J_1, \ldots, J_n$ with arbitrary speed-up functions and precedence constraints, there exists a simple algorithm* OBEQUI $\circ$ EQUI *that is $(1+\varepsilon)$-processor $\frac{(\kappa(J)+1)c_\varepsilon}{2} \cdot \log n$-competitive, where $\kappa(J)$ denotes the maximum number of independent tasks in a job of the instance, and $c_\varepsilon \leqslant 16(\frac{1}{\varepsilon} + \frac{4}{\varepsilon^2})$ for $\varepsilon \leqslant \frac{1}{2}$.*

References

1. Bansal, N., Pruhs, K.: Server scheduling in the ℓ_p norm: a rising tide lifts all boat. In: STOC. pp. 242–250 (2003)
2. Becchetti, L., Leonardi, S.: Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J. ACM 51(4), 517–539 (2004)
3. Edmonds, J.: Scheduling in the dark. TCS 235(1), 109–141 (2000)
4. Edmonds, J., Chinn, D.D., Brecht, T., Deng, X.: Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. J. of Scheduling 6(3), 231–250 (2003)
5. Edmonds, J., Im, S., Moseley, B.: Online scalable scheduling for the lk-norms of flow time without conservation of work, personal communication
6. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. In: SODA. pp. 685–692 (2009)
7. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling jobs with varying parallelizability to reduce variance. In: SPAA (2010)
8. He, Y., Hsu, W.J., Leiserson, C.E.: Provably efficient online non-clairvoyant adaptive scheduling. In: IPDPS. pp. 1–10 (2007)
9. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47, 214–221 (2000)
10. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. J. ACM 50(4), 551–567 (2003)
11. Merritt, R.: CPU designers debate multi-core future. EETimes (2008)
12. Motwani, R., Phillips, S., Torng, E.: Nonclairvoyant scheduling. TCS 130(1), 17–47 (1994)
13. Pruhs, K.: Competitive online scheduling for server systems. SIGMETRICS Perf. Eval. Rev. 34(4), 52–58 (2007)
14. Pruhs, K., Sgall, J., Torng, E.: Online Scheduling, chap. 15 of *Handbook of scheduling: Algorithms, models and performance analysis* (J. Y-T. Leung ed.). Chapman & Hall/CRC (2004)
15. Robert, J., Schabanel, N.: Non-clairvoyant batch set scheduling: Fairness is fair enough. In: ESA. vol. LNCS 4698, pp. 742–753 (2007)
16. Robert, J., Schabanel, N.: Non-clairvoyant scheduling with precedence constraints. In: SODA. pp. 491–500 (2008)

[5] Section 6 can be downloaded in the full version of the present article from the web pages of the authors and will be available in the upcoming journal version of this article.