New Results on Flow Time with Resource Augmentation

L. Epstein, R. van Stee

Software Engineering (SEN)

SEN-R0028 October 31, 2000

Report SEN-R0028
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

New Results on Flow Time with Resource Augmentation

Leah Epstein*
The Interdisciplinary Center, Herzliya, Israel
Epstein.Leah@idc.ac.il

Rob van Stee†
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Rob.van.Stee@cwi.nl

ABSTRACT

We study the problem of scheduling n jobs that arrive over time. We consider a non-preemptive setting on a single machine. The goal is to minimize the total flow time. We use extra resource competitive analysis: an optimal off-line algorithm which schedules jobs on a single machine is compared to a more powerful on-line algorithm that has l machines. We design an algorithm of competitive ratio $O(\min(\Delta^{1/l}, n^{1/l}))$, where Δ is the maximum ratio between two job sizes, and provide a lower bound which shows that the algorithm is optimal up to a constant factor for any constant l. The algorithm works for a hard version of the problem where the sizes of the smallest and the largest jobs are not known in advance, only Δ is known. This gives a trade-off between the resource augmentation and the competitive ratio.

We also consider scheduling on parallel identical machines. In this case the optimal off-line algorithm has m machines and the on-line algorithm has lm machines. We give a lower bound for this case. Next, we give lower bounds for algorithms using resource augmentation on the speed. Finally, we consider scheduling with hard deadlines.

2000 Mathematics Subject Classification: 68Q25, 68W25
1998 ACM Computing Classification System: F.2.2
Keywords and Phrases: flow time, on-line algorithms, competitive analysis, resource augmentation
Note: Work carried out under theme SEN4 "Evolutionary Systems and Applied Algorithmics".

1. Introduction

Minimizing the total flow time is a well-known and hard problem, which has been studied widely both in on-line and in off-line environments [1, 8, 9]. The flow time $f(J)$ of a job J is defined as its completion time, $C(J)$, minus the time at which it arrived, $r(J)$ (the release time of J). This measure is applicable to systems where the load is proportional to the total number of bits that exist in the system over time (both of running jobs and of waiting jobs). In this paper, we consider on-line algorithms using resource augmentation, and we examine

*Work carried out while this author was at Tel-Aviv University.

†Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002.

the effects on the performance of an algorithm if it has more or faster machines than the off-line algorithm (see [7, 10]).

We consider the following on-line scheduling problem. The algorithm has parallel identical machines, on which it must schedule jobs with different processing requirements that arrive over time. A job J (which arrives at time $r(J)$) with processing requirement $P(J)$ (also called running time or size) that becomes known upon arrival, has to be assigned to one of the machines and run there continuously for $P(J)$ units of time. The objective is to minimize the sum of flow times of all jobs. The total number of jobs is n.

We compare on-line algorithms that are running on lm machines ($l \geq 1$) to an optimal off-line algorithm, denoted by OPT, that is running on m machines but knows all the jobs in advance. Such on-line algorithms are also called l-machine algorithms, since they use l times as much machines as the optimal off-line algorithm. An algorithm that uses the same number of machines as the off-line algorithm, but uses machines which are $s > 1$ times faster, is called a s-speed algorithm.

For a job sequence σ and an on-line algorithm A, we denote the total flow time of σ in the schedule of A on lm machines by $A_{lm}(\sigma)$. We denote the optimal total flow time for σ on m machines by $OPT_m(\sigma)$. The competitive ratio using resource augmentation is defined by

$$r_{m,lm}(A) = \sup_{\sigma} \frac{A_{lm}(\sigma)}{OPT_m(\sigma)},$$

where the supremum is taken over all possible job sequences σ. The goal of an on-line algorithm is to minimize this ratio.

Approximating the flow time is hard even in an off-line environment (see [8, 9]). In an on-line environment it is well known that the best competitive ratio of any algorithm that uses a single machine is n (easily achieved by a greedy algorithm). The problem has been studied introducing resource augmentation by Phillips, Stein, Torng and Wein [10]. They give algorithms with augmentation on the number of machines. These are an $O(\log n)$-machine algorithm (which has a competitive ratio $1+o(1)$) and an $O(\log \Delta)$-machine algorithm (which achieves the competitive ratio 1), where Δ is the maximum ratio between running times of jobs. Both algorithms are valid for every m.

We give an algorithm `Levels` and show $r_{1,l}(\texttt{Levels}) = O(\min(n^{1/l}, \Delta^{1/l}))$, where n is the number of jobs that arrive. This algorithm works for a hard version of this problem where the sizes of the smallest and the largest jobs are not known in advance; only Δ is known in advance. The algorithm in [10] works only if the job size limits are known in advance.

Furthermore, we show that for all on-line algorithms A and number m_1 of off-line machines we have $r_{m_1,l}(A) = \Omega\left(\frac{\min(n^{1/l},\Delta^{1/l})}{(12l)^l}\right)$. This shows that `Levels` is optimal up to a constant factor for any constant l against an adversary on one machine.

In [5], a related problem on a network of links is considered. It immediately follows from our lower bounds, that any constant competitive algorithm has a polylogarithmic number of machines. More precisely, if A has a constant competitive ratio and lm machines, $l \geq \Omega\left(\frac{\sqrt{\log(\min(n,\Delta))}}{m\sqrt{\log\log(\min(n,\Delta))}}\right)$. This result can also be deduced from Theorem 10 in [5]. However, using their proof for the general lower bound would give only an exponent of $\frac{1}{2l}$. Improving the exponent to be the tight exponent $\frac{1}{l}$ is non-trivial. Our results imply that by choosing a given amount of resource augmentation, the competitive ratio is fixed. We adapt the lower

bound for the case where the on-line algorithm has faster machines than the off-line algorithm. This results in a lower bound of $\Omega(n^{1/2m^2})$ on the speed of on-line machines, if $l = 1$.

We also consider the following scheduling problem studied in [3, 10]. Each job J has a deadline $d(J)$. Instead of minimizing the flow time, we require that each job is finished by its deadline, effectively limiting the flow time of job J to $d(J) - r(J)$. The goal is to complete all jobs on time. For this problem, we give lower bounds on the speed and the number of machines required for a non-preemptive on-line algorithm to succeed on any sequence.

Throughout the paper, for a specific schedule ζ for the jobs, we denote the starting time of job J by $S_\zeta(J)$, and its flow time by $f_\zeta(J) = C_\zeta(J) - r(J)$. We omit the subscripts if the schedule is clear from the context.

2. Algorithms with Resource Augmentation

We have the following results for the case where n is not known and the case where OPT has the same number of machines as the on-line algorithm.

Lemma 1 *Any on-line algorithm for minimizing the total flow time on parallel machines has a competitive ratio of $\Omega(n)$ if it does not know n in advance, even if it is compared to an off-line algorithm on one machine.*

Proof We use a number $N >> 1$.

One job of size 1 arrives at time 0. When it is started, N jobs of size $1/N$ arrive with intervals of $1/N$ during the next 1 time. If they are all delayed until time 1, no more jobs arrive and we are done. The optimal flow time on one machine is 3 and the online flow time is $O(N)$.

On the other hand, if one of those jobs is started while the first job is running, N jobs of size $1/N^2$ arrive with intervals of $1/N^2$ during the next $1/N$ time. Depending on the decision by the online algorithm, we continue in this way or stop as soon as it delays N jobs (or reaches the last machine).

When all machines are in use, the online algorithm cannot prevent a flow time of $O(N)$.□

Lemma 2 *$r_{l,l}(A) = \Omega(n/l^2)$ for all algorithms A.*

Proof A single unit job arrives at time 0. Let t be the time at which A starts this job. Let $\mu = \frac{l}{2(n-1)}$. For $j = 0, \ldots, \frac{n-1}{l}$, l jobs of length μ are released at time $t + j\mu$. It is easy to see that the optimal total flow time is $\Theta(l)$ whereas the flow time of A will be $\Omega(n/l)$. Consequently, $r_{l,l}(A) = \Omega(n/l^2)$. □

We define an algorithm `Levels` that knows n. `Levels` uses l priority queues $Q_1, \ldots, Q_l$ (one for each machine) and l variables $D_1 \geq \cdots \geq D_l$. We initialize $Q_i = \emptyset$ and $D_i = 0$. An *event* is either an arrival of a new job or a completion of a job by a machine. Let $\gamma = n^{1/l}$, where n is the number of jobs.

Algorithm `Levels`

- If a few events occur at the same time, the algorithm first deals with all arrivals before it deals with job completions.
- On completion of a job on machine i, if $Q_i \neq \emptyset$, a job of minimum release time among jobs with minimum processing time in Q_i is scheduled immediately on machine i. (The job is dequeued from Q_i.)

- On arrival of a job J, let i be a minimum index of a machine for which $D_i \leq \gamma P(J)$. If there is no such index, take $i = m$. If machine i is idle, J is immediately scheduled on machine i, and otherwise, J is enqueued into Q_i. If $P(J) > D_i, D_i$ is modified by $D_i \leftarrow P(J)$.

We analyze the performance of `Levels` compared to a preemptive OPT on a single machine. Denote the schedule of `Levels` by π. Partition the schedule of each machine into blocks. A block is a maximal sub-sequence of jobs of non-decreasing sizes, that run on one machine consecutively, without any idle time.

- Let N_i be the number of blocks in the schedule of `Levels` on machine i.
- Let $B_{i,k}$ be the k^{th} block on machine i.
- Let $b_{i,k,j}$ be the j^{th} job in block $B_{i,k}$.
- Let $N_{i,k}$ be the number of jobs in $B_{i,k}$.
- Let $P_{i,k}$ be the size of the largest job in blocks $B_{i,1}, \ldots, B_{i,k}$ i.e.

$$\begin{aligned} P_{i,k} &= \max_{1 \leq r \leq k} \max_{1 \leq j \leq N_{i,r}} P(b_{i,r,j}) \\ P_{i,0} &= 0 \quad \text{for all } 1 \leq i \leq l. \end{aligned}$$

- Let $I = \bigcup_{1 \leq i \leq l, 1 \leq k \leq N_i} B_{i,k}$, i.e. I is the set of all jobs.

Similar to the proof in [6], we define a pseudo-schedule ψ on l machines, in which job $b_{i,k,j}$ is scheduled on machine i at time $S_\pi(b_{i,k,j}) - P_{i,k-1}$. Note that ψ is not necessarily a valid schedule, since some jobs might be assigned in parallel, and some jobs may start before their arrival times.

The amount that jobs are shifted backwards increases with time. Therefore, if there is no idle time between jobs in π, there is no idle time between them in ψ either. Note that in ψ, the flow time of a job J can be smaller than $P(J)$, and even negative.

We introduce an extended flow problem. Each job J has two parameters $r(J)$ and $r'(J)$, where $r'(J) \leq r(J)$. $r'(J)$ is the pre-release time of job J. Job J may be assigned starting from time $r'(J)$. The flow time is still defined by the completion time minus the release time, i.e. $f(J) = C(J) - r(J)$. Going from an input σ for the original problem to an input σ' of the extended problem, requires definition of the values of r' for all jobs. Clearly, the optimal total flow time for an input σ' of the extended problem is no larger than the flow time of σ in the original problem.

Let I_i be the set of jobs that run on machine i in π. We define an instance I'_i for the extended problem. I'_i contains the same jobs as I_i. For each $J \in I_i$, $r(J)$ remains the same, Define $r'(J) = \min\{r(J), S_\psi(J)\}$. Clearly $OPT(I) \geq \sum_{i=1}^{l} OPT(I_i) \geq \sum_{i=1}^{l} OPT(I'_i)$, where $OPT(I_i)$ is the preemptive optimal off-line cost for the jobs that `Levels` scheduled on machine i. We consider a preemptive optimal off-line schedule ϕ_i for I'_i on a single machine. In ϕ_i, jobs of equal size are completed in the order of arrival. Ties are broken as in π. The following lemma is similar to [6].

Lemma 3 *For each job $J \in I'_i$, $f_{\phi_i}(J) \geq f_\psi(J)$.*

Proof Since $r_{\phi_i}(J) = r_\psi(J)$ for each job J, we only have to show that in ϕ_i, J does not start earlier than it does in ψ. Assume to the contrary this is not always the case. Let J_1 be the first job in ϕ_i for which $S_{\phi_i}(J_1) < S_\psi(J_1)$. Note that in this case $r'(J_1) < S_\psi(J_1)$ and hence $r(J_1) < S_\psi(J_1)$. Let t be the end of the last idle time before $S_\psi(J_1)$, and let $B_{i,k}$ be the block that contains J_1.

Suppose $P_{i,k-1} \leq P(J_1)$. Then all jobs that run on machine i from time t until time $S_\psi(J_1)$ in ψ are either smaller than $P(J_1)$ or have the same size, but are released earlier. Moreover, these jobs do not arrive earlier than time t, hence in ϕ_i they do not run before time t. They do run before $S_{\phi_i}(J_1)$ because they have higher priority, hence $S_{\phi_i}(J_1) \geq S_\psi(J_1)$, a contradiction.

Suppose $P_{i,k-1} > P(J_1)$. J_1 was available to be run in ψ during the interval $[r(J_1), S_\psi(J_1)]$ since $r(J_1) < S_\psi(J_1)$. In π, all jobs running in the interval $[r(J_1), S_\pi(J_1)]$ are smaller than J_1 (or arrived before, and have the same size), except for the first one, say J_2. Since in ψ, all these jobs are shifted backwards by at least the size of J_2, during $[r(J_1), S_\psi(J_1)]$ only jobs with higher priority than J_1 are run in ψ. J_1 is the first job which starts later in ψ than it does in ϕ_i, so these jobs occupy the machine until time $S_\psi(J_1)$, hence $S_\psi(J_1) \leq S_{\phi_i}(J_1)$. □

Theorem 1 $r_{1,l}(\texttt{Levels}) = O(n^{1/l})$.

Proof Using Lemma 3 we can bound the difference between ψ and π. Since $\texttt{Levels}(b_{i,k,j}) = C_\psi(b_{i,k,j}) + P_{i,k-1} - r(b_{i,k,j})$, we have

$$\begin{aligned}
\texttt{Levels}(I) &= \sum_{1\leq i\leq l} \sum_{1\leq k\leq N_i} \sum_{1\leq j\leq N_{i,k}} (C_\psi(b_{i,k,j}) + P_{i,k-1} - r(b_{i,k,j})) \\
&\leq \sum_{b_{i,k,j}\in I} (C_\psi(b_{i,k,j}) - r(b_{i,k,j})) + \sum_{b_{i,k,j}\in I} P_{i,k-1} \leq OPT(I) + \sum_{b_{i,k,j}\in I} P_{i,k-1}.
\end{aligned}$$

Let P the maximum job size. We show the following properties:

$$P_{i,k-1} \leq \gamma P(b_{i,k,j}) \quad \text{for each job } b_{i,k,j}, 1 \leq i \leq l-1 \tag{2.1}$$

$$P_{l,k-1} \leq \frac{P}{\gamma^{l-1}} \quad \text{for each job } b_{l,k,j} \tag{2.2}$$

Adding both properties together we get

$$\begin{aligned}
\texttt{Levels}(I) &\leq OPT(I) + \sum_{\substack{b_{i,k,j}\in I \\ i\neq l}} P_{i,k-1} + \sum_{b_{l,k,j}\in I} P_{l,k-1} \\
&\leq OPT(I) + \gamma \sum_{b_{i,k,j}\in I} P(b_{i,k,j}) + \sum_{b_{l,k,j}\in I} \frac{P}{\gamma^{l-1}} \\
&\leq OPT(I) + \gamma OPT(I) + n \cdot \frac{OPT(I)}{n^{(l-1)/l}} = (2\gamma + 1) \cdot OPT(I)
\end{aligned}$$

This holds since $OPT(I) \geq P$ and $OPT(I)$ is at least the sum of all job sizes, and since $|I| = n$.

To prove (2.1) we recall that $b_{i,k,j}$ was assigned to machine i because it satisfied $D_i \leq \gamma P(b_{i,k,j})$. If $P_{i,k-1} \leq P(b_{i,k,j})$ we are done. Otherwise the job of size $P_{i,k-1}$ arrived before $b_{i,k,j}$ and hence when $b_{i,k,j}$ arrived, D_i satisfied $D_i \geq P_{i,k-1}$, hence $P_{i,k-1} \leq D_i \leq \gamma P(b_{i,k,j})$.

To prove (2.2) we show by induction that every job J on machine i in `Levels` satisfies $P(J) \leq P/\gamma^{i-1}$. This is trivial for $i = 1$. Assume it is true for some machine $i \geq 1$, then at all times $D_i \leq P/\gamma^{i-1}$ holds. Hence, a job J' that was too small for machine i satisfied $P(J') \leq D_i/\gamma \leq P/\gamma^i$. This completes the proof. □

We now give an example to show that `Levels` does not have a competitive ratio of $O(n^{1/l})$, if it is compared to OPT on l machines. This example also shows that the ratio between the optimal off-line flow on one machine and on l machines can be $\Omega(n)$, suggesting that it is (much) harder to have a good competitive ratio against an off-line algorithm that has more than one machine.

The example consists of n jobs of size 1. Let $k = \frac{n}{l}$. For $j = 0, \ldots, k-1$, l jobs are released at time i. The optimal off-line flow on l machines is clearly n. On the other hand, if only one machine is being used, some calculations give that the flow time is at least $\sum_{1 \leq i \leq n} C_i - \sum_{1 \leq i \leq n} r_i = \frac{n}{2}(n + 2 - k) = n \cdot \Omega(n)$.

Since `Levels` assigns jobs of equal size using only one machine, it has the same performance as OPT on one machine has, and hence a competitive ratio of $\Omega(n)$ when compared to OPT on l machines.

We give a variant of `Levels` with a competitive ratio which depends on Δ, the ratio between the size of the largest job and the size of the smallest job.

Algorithm `Revised Levels`: Run `Levels` with $\gamma = \Delta^{1/l}$.

Theorem 2 $r_{1,l}(\texttt{Revised Levels}) = O(\Delta^{1/l})$.

Proof The proof is very similar to the proof of Theorem 1. The only difference in the proof is that property (2.1) also holds for machine l (this follows from property (2.2) and the definition of Δ), hence the competitive ratio is now $\gamma + 1$. □

Taking $\gamma = \min(n^{1/l}, \Delta^{1/l})$ we can get a competitive ratio of $O(\min(n^{1/l}, \Delta^{1/l}))$.

3. Lower Bounds for Resource Augmentation

Theorem 3 *Let A be an on-line scheduling algorithm to minimize the total flow time on l machines. Then for any $1 \leq m_1 \leq l$ and sequences consisting of $O(n)$ jobs, $r_{m_1,l}(A) = \Omega\left(\frac{n^{1/l}}{(12l)^{l-1}}\right)$.*

We first describe a job sequence σ and then show that it implies the theorem. Let n be an integer. There will be at most $\sum_{i=0}^{l} n^{i/l}$ jobs in σ (note $\sum_{i=0}^{l} n^{i/l} = \Theta(n)$). We build σ recursively, defining the jobs according to the behavior of the on-line algorithm A.

Definition A job j of size α is considered *active*, if the previous active job of size α is completed by A at least α units of time before j is assigned, and j finishes before or when the job that caused its arrival finishes.

The first job in σ has size n and arrives at time 0. We consider it to be an active job. On an assignment of a job j of size α by A, do the following:

- If j is active, and all other machines are running larger jobs (all machines are consequently busy for at least α units of time), n jobs of size 0 arrive immediately. No more

jobs will arrive.

- Otherwise, if j is active, then j causes the arrival of $n^{1/l}$ jobs of size $\frac{1}{3} \cdot \frac{\alpha}{n^{1/l}}$. These jobs arrive starting the time that j is assigned, every $\frac{\alpha}{n^{1/l}}$ units of time, until they all have arrived.
- In all other cases (j is not active), no jobs arrive till the next job that A starts.

Lemma 4 $OPT_1(\sigma) \leq 6n$.

Proof We show that all jobs can be assigned on a single machine, during an interval of length $2n$, so that a job of length α has a flow time of at most 3α. The total flow time then follows.

We show how to assign all jobs of a certain size α so that no active jobs of size α are running at the same time on on-line machines, i. e. the intervals used by A to run active jobs of size α, and the intervals that are used by OPT to run jobs of size α, are disjoint. Smaller jobs are assigned by OPT during the intervals in which A assigned active jobs of size α. Hence, the time slots given by the optimal off-line for different jobs are disjoint.

Finally, we show how to define those time slots. A job j of size α, that arrives at time t, is not followed by other jobs of size α until time $t + 3\alpha$. Since an active on-line job starts at least α units of time after the previous active job of this size (α) is completed, there is a time slot of size at least α during the interval $[t, t + 3\alpha]$ where no active job of size α is running on any of the on-line machines. The optimal off-line algorithm can assign j during that time. This is true also for the first job. Finally, the optimal algorithm can also manage the jobs of size 0 easily by running them immediately when they arrive. Hence, the total time that the optimal off-line machine is not idle is at most $2n$. □

We partition jobs into three types, according to the on-line assignment. A job that arrived during the processing of a job of size α, and has size $\frac{1}{3}\frac{\alpha}{n^{1/l}}$ is either active or *passive* (if it is not active, but completed before the job of size α is completed). Otherwise, the job is called *late*. Let $P(\alpha), T(\alpha)$ and $L(\alpha)$ denote the number of passive, active and late jobs of size α (respectively). Let $N(\alpha) = P(\alpha) + T(\alpha) + L(\alpha)$.

Claim 1 $T(\alpha) \geq \lceil \frac{1}{2l}(P(\alpha) + T(\alpha)) \rceil$

Proof The number of jobs of size α that the on line algorithm can complete during 2α units of time (until a job can be active again) is at most $2n$. □

Now we are ready to prove Theorem 3.

Proof (Of Theorem 3.) According to the definition of the sequence, $N(\alpha) = n^{1/l} \cdot T(3\alpha n^{1/l})$. We distinguish two cases.

Case 1. In all phases $L(\alpha) \leq \frac{1}{2}N(\alpha)$. Hence $T(\alpha) \geq \frac{1}{4l}N(\alpha)$ for all α. This is true for $\alpha = (\frac{1}{3})^{l-1}n^{1/l}$ (the smallest non-zero jobs) and hence there at least $n^{l-1/l} \cdot (\frac{1}{4l})^{l-1} > 0$ such jobs. Therefore the zero jobs arrive and are delayed by at least $(\frac{1}{3})^{l-1} \cdot n^{1/l}$ units of time. Since their flow time is at least $n \cdot n^{1/l} \cdot (\frac{1}{3})^{l-1}$, and the optimal flow time is at most $6n$, the competitive ratio follows.

Case 2. There is a phase where $L(\alpha) > \frac{1}{2}N(\alpha)$. Consider the phase with largest α in which this happens. Since for larger sizes α' we have $L(\alpha') \leq \frac{1}{2}N(\alpha')$, we can bound the number of jobs of size α (for $\alpha = (\frac{1}{3})^i n^{1-i/l}$) by $N(\alpha) \geq n^{i/l}(\frac{1}{4l})^i$. The late jobs are delayed

by at least $\frac{1}{4} \cdot 3\alpha n^{1/l}$ on average. (This is the delay if for each job of size $3\alpha n^{1/l}$ the *last* $\frac{1}{2}n^{1/l}$ jobs of size α that arrive are the ones that are late; in all other cases, the delay is bigger.)

The total flow time is at least

$$\begin{aligned} A_l(\sigma) &\geq L(\alpha) \cdot \frac{1}{4} \cdot 3\alpha n^{1/l} \geq \frac{1}{2} n^{i/l} \left(\frac{1}{4l}\right)^i \frac{1}{4} \left(\frac{1}{3}\right)^{i-1} n^{1-i/l+1/l} = \frac{1}{(4l)^i} \cdot \frac{1}{8} \left(\frac{1}{3}\right)^{i-1} \cdot n^{1+1/l} \\ &= \left(\frac{1}{12l}\right)^i \frac{3}{8} n^{1+1/l} \geq \frac{1}{(12l)^{l-1}} \cdot \frac{3}{8} \cdot n^{1+1/l} = \Omega\left(\frac{n^{1/l}}{(12l)^{l-1}}\right) \cdot \Theta(n) \end{aligned}$$

Since the optimal flow is $\Theta(n)$, the competitive ratio follows. □

Theorem 4 *Let A be an on-line scheduling algorithm to minimize the total flow time on l machines. Then $r_{m_1,l}(A) = \Omega\left(\frac{\Delta^{1/l}}{(12l)^l}\right)$ for any $1 \leq m_1 \leq l$ if the maximum ratio between jobs is Δ.*

Proof We adjust σ by starting with a job of size Δ and fixing $n = \Delta/3^l$. We assume $\Delta \geq 6^l$ so that $n \geq 2^l$ and $n^{1/l} \geq 2$, which is needed for the construction of the sequence.

Starting from here, we build a sequence σ' in exactly the same way as σ, except that we do not let jobs of size 0 arrive. Clearly, $OPT_1(\sigma') \leq 6\Delta$. We can follow the proof of Theorem 3. However, we now know that all the smallest jobs will be late. If they arrive we are in the second case of the proof; but if they do not, then for an earlier α we must have $L(\alpha) > \frac{1}{2}N(\alpha)$. So only Case 2 remains of that proof.

The total flow is at least $\frac{3}{8}\frac{n^{1/l}}{(12l)^l}\Delta = \frac{1}{8}\frac{\Delta^{1/l}}{(12l)^l}\Delta$ (because now $i \leq l$ in stead of $i \leq l-1$), giving the desired competitive ratio. □

A direct consequence of Theorems 3 and 4 is the following bound on the number of machines needed to maintain a constant competitive ratio. This corollary can be also proved using a simple adaptation of Theorem 10 in [5].

Corollary 1 *Any on-line algorithm for minimizing total flow time on m machines that uses resource augmentation and has a constant competitive ratio, is an $\Omega\left(\frac{\sqrt{\log(\min(n,\Delta))}}{m\sqrt{\log\log(\min(n,\Delta))}}\right)$-machine algorithm (on sequences of $\Theta(n)$ jobs).*

Next we consider resource augmentation on the speed as well as on the number of machines. We consider an on-line algorithm which uses machines of speed $s > 1$. The optimal off-line algorithm uses machines of speed 1.

Theorem 5 *Let A be an on-line scheduling algorithm to minimize the total flow time on l machines. Let $s > 1$ be the speed of the on-line machines. Then $r_{m_1,l}(A) = \Omega\left(\frac{n^{1/l}}{s(12ls^2)^{l-1}}\right)$ for any $1 \leq m_1 \leq l$ and sequences consisting of $O(n)$ jobs. Furthermore, $r_{m_1,l}(A) = \Omega\left(\frac{\Delta^{1/l}}{s(12ls^2)^l}\right)$ for any $1 \leq m_1 \leq l$.*

Proof Again, we use a job sequence similar to σ. The jobs of phase i now have size $1/(3s^2n^{1/l})^i$. For the Δ-part of the proof, we fix $n = \Delta/(3s^2)^l$. Similar calculations as in the previous proofs result in the stated lower bounds. □

Corollary 2 *Any on-line algorithm for minimizing total flow time on m machines that uses resource augmentation on the speed and has a constant competitive ratio, is an* $\Omega(n^{1/(2l^2)})$*-speed algorithm (on sequences of* $\Theta(n)$ *jobs) and an* $\Omega(\Delta^{1/(2l^2)})$*-speed algorithm.*

4. Hard Deadlines

We consider the problem of non-preemptive scheduling jobs with hard deadlines. Each arriving job J has a deadline $d(J)$ by which it must be completed. The goal is to produce a schedule, in which all jobs are scheduled such that all of them are completed on time (i.e. by their deadlines). We give a lower bound on the resource augmentation required so that all jobs finish on time. We use a similar lower bounding method to the method we used Section 3. We allow the on-line algorithm resource augmentation in both the number of machines and their speed. We compare an on-line algorithm that schedules on l machines of speed s to an optimal off-line algorithm that uses a single machine of speed 1.

Let Δ denote the ratio between the largest job in the sequence and the smallest job. The lower bound sequence consists of $l+1$ jobs $J_0, \ldots, J_l$ where $P(J_i) = 1/(2s+1)^i$. We define release times and deadlines recursively; $r(J_0) = 0$ and $d(J_0) = 2 + 1/s$. Let π be the on-line schedule, then $r(J_{i+1}) = S_\pi(J_i)$ and $d(J_{i+1}) = C_\pi(J_i)$. Hence J_{i+1} runs in parallel to all jobs $J_0, \ldots, J_i$ in any feasible schedule π.

Lemma 5 *An optimal off-line algorithm on a single machine of speed* 1 *can complete all jobs on time.*

Proof For each $i > 0$, $P(J_i) = 1/(2s+1)^i$, hence $d(J_i) - r(J_i) = P(J_{i-1})/s = \frac{2s+1}{s}P(J_i)$. This holds also for J_0, since $P(J_0) = 1$ and $d(J_0) - r(J_0) = \frac{2s+1}{s}$. All jobs arriving after J_i have release times and deadlines in the interval $[S_\pi(J_i), C_\pi(J_i)]$. The optimal off-line algorithm can schedule J_i outside this time interval, and avoid conflict with future jobs. By induction, previous jobs are scheduled before $r(J_i)$ or after $d(J_i)$, so there is no conflict with them either. If $S_\pi(J_i) - r(J_i) \geq P(J_i)$, schedule J_i at time $r(J_i)$. Otherwise $C_\pi(J_i) = S_\pi(J_i) + P(J_i)/s < r(J_i) + P(J_i)(1+1/s)$, hence J_i is scheduled at time $C_\pi(J_i)$ and completed at $C_\pi(J_i) + P(J_i) < r(J_i) + P(J_i)(2+1/s) = d(J_i)$. □

It is easy to see that the on-line algorithm cannot finish all jobs on time. If the first l jobs finish on time, then all l machines are busy during the time interval $[r(J_l), d(J_l)]$ and it is impossible to start J_l before time $d(J_l)$. Hence we have the following theorem.

Theorem 6 *The on-line algorithm fails, if* $\Delta \geq (2s+1)^l$.

We show some corollaries from the lower bound on Δ. These are necessary conditions for an on-line algorithm to succeed on any sequence. Given machines of constant speed s, the number of machines l must satisfy $l \geq \frac{\log \Delta}{\log(2s+1)}$ i.e. $l = \Omega(\log \Delta)$. On the other hand, for a constant number l of machines, s has to satisfy $2s + 1 \geq \Delta^{1/l}$, i.e. $s = \Omega(\Delta^{1/l})$.

The lower bound on Δ clearly holds also for the case where the optimal off-line algorithm is allowed to use $m_1 > 1$ machines. Consider a k-machine algorithm that always succeeds in building a feasible schedule ($k = l/m_1$), then k satisfies $k = \Omega(\log \Delta/m)$ for constant s. Finally, s satisfies $s = \Omega(\Delta^{1/mk})$ for constant k.

5. Conclusions and Open Problems

We have presented an algorithm for minimizing the flow time on l identical machines with competitive ratio $O(\min(\Delta^{1/l}, n^{1/l}))$ against an optimal off-line algorithm on a single ma-

chine, and we have shown a lower bound of $\Omega\left(\frac{\min(n^{1/l},\Delta^{1/l})}{(12l)^l}\right)$ on the competitive ratio of any algorithm, even against an adversary on one machine. For every constant l, this gives an exact trade-off between the amount of resource augmentation and the number of on-line machines.

An interesting remaining open problem is to find an algorithm which is optimally competitive against an off-line algorithm on a single machine for any l.

Acknowledgements
We thank Kirk Pruhs for helpful discussions.

References

1. B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 198–205, 1999.
2. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. M. L. Dertouzos and A. K.-L. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15:1497–1506, 1989.
4. L. Epstein and R. van Stee. New Results on Flow Time with Resource Augmentation. *Technical Report*, CWI, Amsterdam, in preparation.
5. A. Goel, M. R. Henzinger, S. Plotkin and E. Tardos. Scheduling Data Transfers in a Network and the Set Scheduling Problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999.
6. J.A. Hoogeveen and A.P.A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proc. 5th Int. Conf. Integer Programming and Combinatorial Optimization*, LNCS, pages 404–414. Springer, 1996.
7. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of 36th IEEE Symposium on Foundations of Computer Science*, pages 214–221, 1995.
8. H. Kellerer, T. Tautenhahn, and G.J.Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proc. 28th ACM Symposium on the Theory of Computing*, pages 418–426, 1996.
9. S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proc. 29th ACM Symposium on the Theory of Computing*, pages 110–119, 1997.
10. Cynthia A. Philips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.

11. A.P.A. Vestjens. *On-line Machine Scheduling.* PhD thesis, Technical University Eindhoven, 1997.

12. A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 12th ACM Symposium on Theory of Computing*, 1980.