Minimizing Flow Time Nonclairvoyantly

Bala Kalyanasundaram *
Computer Science Department
University of Pittsburgh
kalyan@cs.pitt.edu

Kirk R. Pruhs *
Computer Science Department
University of Pittsburgh
kirk@cs.pitt.edu

Abstract

We consider the problem of scheduling a collection of dynamically arriving jobs with unknown execution times so as to minimize the average flow time. This is the classic CPU scheduling problem faced by time-sharing operating systems. In the standard 3-field scheduling notation this is the nonclairvoyant version of $1 \mid pmtn, r_j \mid \sum(C_j - r_j)$. It is easy to see that every algorithm that doesn't unnecessarily idle the processor is at worst n-competitive, where n is the number of jobs. Yet there is no known nonclairvoyant algorithm, deterministic or randomized, with a competitive ratio provably $O(n^{1-\epsilon})$. In this paper with give a randomized nonclairvoyant algorithm, RMLF, that has competitive ratio $O(\log n \log\log n)$ against an adaptive adversary. RMLF is a slight variation of the multi-level feedback (MLF) algorithm used by the Unix operating system, further justifying the adoption of this algorithm. It is known that every randomized nonclairvoyant algorithm is $\Omega(\log n)$-competitive, and that every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$-competitive.

1 Introduction

We consider the problem of scheduling a collection of dynamically arriving jobs with unknown execution times so as to minimize the average flow time. This is the classic CPU scheduling problem faced by time-sharing operating systems (see for example [5]). In the standard 3-field scheduling notation [10] this is the nonclairvoyant version of $1 \mid pmtn, r_j \mid \sum(C_j - r_j)$. It is easy to see that every algorithm that doesn't unnecessarily idle the processor is at worst n-competitive, where n is the number of jobs. Yet there is no known nonclairvoyant algorithm, deterministic or randomized, with a competitive ratio provably $O(n^{1-\epsilon})$, where n is the number of jobs, and ϵ is any constant.

*Supported in part by NSF Grant CCR-9734927.

In this paper with give a randomized nonclairvoyant algorithm, RMLF, that has a competitive ratio of $O(\log n \log\log n)$. Motwani, Phillips, and Torng [14] showed that every randomized nonclairvoyant scheduling algorithm is $\Omega(\log n)$-competitive, and that every deterministic nonclairvoyant scheduling algorithm is $\Omega(n^{1/3})$-competitive.

In the deterministic multi-level feedback algorithm (MLF) each job J_j gets 2^{i+1} units of processing time while in queue Q_i before being promoted to next highest queue, Q_{i+1}, and the processor is always running the job in the front of the lowest nonempty queue. Our algorithm RMLF is identical to MLF except that the processing time that J_j receives before being promoted to Q_{i+1} is 2^{i+1} minus an exponentially distributed random variable. The MLF scheduling algorithm seems to have been initially implemented in the Compatible Time-Sharing System (CTSS) [12], one of the first time-sharing systems, and is used by the Unix operating system [1]. Our results provide further justification for the adoption of MLF as the scheduling strategy.

Another perhaps interesting aspect of these results is that we allow the adversary to be adaptive. This is as far as we know the first use of an adaptive adversary in the context of an online scheduling problem, instead of the usual assumption of an oblivious adversary. An *oblivious adversary* must completely specify the input before the online algorithm begins execution. In the context of this problem, an *adaptive adversary* may decide what jobs to release at time t based on the random events internal to the online algorithm up to time t. However, the adaptive adversary must decide the length of all the jobs at their release times. Adopting an adaptive adversary has the advantage of capturing situations where past events may affect when jobs are released. For example, if a job J_j is spawned from another job J_h at some point in the code of J_j, then the release time of J_h obviously depends on how J_j is scheduled. If we allow the adversary to modify the length of jobs based on the random events internal to the online algorithm, then the deterministic lower bounds still hold. Background on online algorithms and different types of adversaries can be found in [3].

1.1 Definitions and Preliminaries

In the scheduling problem $1 \mid pmtn, r_j \mid \sum(C_j - r_j)$, the machine environment consists of a single unit speed processor. Each job J_j, $1 \leq j \leq n$, has a *release time* r_j and a positive *length*, or *execution time*, x_j. In this paper, we assume that there is a lower bound on the length of each job that is known a priori by the online scheduler. It turns out to be convenient to scale our time units so that the length of the shortest job is 2. Job J_j can not be run before time r_j. Preemption is allowed, that is, a scheduler may instantaneously suspend the execution of a job, and later restart the execution of that job from the point of suspension. The *completion time* c_j of a job J_j is the earliest time at which J_j has been allocated enough time to finish execution. The *response/flow time* for J_j is $F_j = c_j - r_j$. The optimality criteria is the *average response/flow time* $\frac{1}{n}\sum_{j=1}^{n} F_j = \frac{1}{n}\sum_{j=1}^{n}(C_j - r_j)$, or equivalently the total response/flow time $\sum_{j=1}^{n} F_j$. It is well known that the optimal scheduling algorithm is Shortest Remaining Processing Time (SRPT), which always runs the job with the least amount of remaining processing time [4].

An *online* scheduler does not become aware of J_j until time r_j. The online scheduler does not have a priori knowledge of the number of jobs. A *clairvoyant* online scheduler learns x_j at time r_j. In contrast, a *nonclairvoyant* scheduler does not learn x_j at time r_j, and more generally, can not deduce x_j until it has run J_j to completion. So note that an online clairvoyant scheduler could run SRPT, but a nonclairvoyant scheduler does not have sufficient information to run SRPT.

In this paper we assume that time is continuous, not discrete.

In our setting, the competitive ratio of a randomized algorithm A is the supremum over all inputs I of n jobs of the ratio $E[A(I)]/Opt(I)$. Here $A(I)$ denotes the total flow time for the schedule produced by the online algorithm A on input I, and the expectation is over random events internal to the algorithm. $Opt(I)$ denotes the optimal total flow time. 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 specifies the input I, and schedules I optimally.

Throughout the paper we use time dependent variables. If ν is a time dependent variable, we use $\nu(t)$ to denote the value of ν at time t. If t is understood, or is irrelevant, we will simply refer to ν instead of $\nu(t)$.

1.2 Related Results

For the problem $1 \mid pmtn, r_j \mid \sum F_j$, Motwani, Phillips and Torng [14] showed that every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$-competitive, and that every randomized nonclairvoyant algorithm is $\Omega(\log n)$-competitive. The randomized lower bound in [14] holds even if the online algorithm knows the distribution on the jobs lengths. Thus, one might argue that our results show that knowing the distribution of the job lengths does not buy the online scheduler much. A variant of the randomized lower bound construction in [14] shows that every randomized nonclairvoyant algorithm is $\Omega(P)$-competitive, where P is the ratio of the length of the longest job to the length of the shortest job. Note that the number of jobs used this lower bound construction is exponential in P. Run-to-completion is P-competitive [14]. The competitive ratio for the Round Robin algorithm is $\Theta(n/\log n)$, the lower bound can be found in [13, 14], and the upper bound can be found in [7]. Kalyanasundaram and Pruhs [9] show that the online algorithmBALANCE (also called the Shortest Elapsed Time algorithm [5]) which always runs the job that has been run the least, is constant competitive if the online scheduler is equipped with a faster processor than the adversary, and this bound was improved in [2]. Average case analyses of various algorithms, including MLF can be found in [5].

For the problem of $1 \mid pmtn \mid \sum F_j$ (all release times are 0), Motwani, Phillips and Torng [14] showed that Round Robin has a competitive ratio approaching 2 as n grows, and this is optimal among nonclairvoyant algorithms even allowing randomization.

For the problem $Pm \mid pmtn, r_j \mid \sum F_j$ (multiple identical processors), Leonardi and Raz [11] showed that SRPT is $O(\log \min(\frac{n}{m}, P))$-competitive, and that every randomized online clairvoyant algorithm is $\Omega(\max(\log \frac{n}{m}, \log P))$-competitive. Phillips, Stein, Torng, and Wein [15] show that SRPT is 1-competitive if equipped with processors twice as fast as the adversary's processors.

Background on online scheduling can be found in [16].

2 Algorithm Description

Definition 1

a. Let n denote the number of jobs.

b. Let $w_j(t)$ be the amount of time that RMLF has run J_j before time t.

c. Let $y_j(t) = x_j - w_j(t)$ be the remaining time that J_j needs to be processed at time t.

d. Let $\alpha(i, j)$ be the number of jobs that have been released before the time that J_j was promoted to Q_i.

e. Let L denote the length of the longest job.

f. Let $\ell = 1 + \lg L$.

For notational simplicity, we assume that n and L are integer powers of 2 (if not one could just round them up to the next integer power of 2). We are now ready to give the RMLF algorithm.

Algorithm RMLF: At all times the collection of released, but uncompleted, jobs are partitioned into queues, $Q_0, Q_1, \ldots, Q_\ell$. We say that Q_i is lower than Q_j for $i < j$. Each job $J_j \in Q_i$, is classified as either red or blue, and was assigned target $T_{i,j} \in [2^i, 2^{i+1}]$ when it entered Q_i. RMLF maintains the invariant that it is always running the job at the front of the lowest nonempty queue.

Let $\beta_{i,j}$, $0 \le i \le \ell$ and $1 \le j \le n$, be an exponentially distributed random variable with probability distribution function $\text{Prob}[\beta_{i,j} \le x] = 1 - \exp(\frac{-\tau x \ln \alpha(i,j)}{2^i})$, where $\tau = 12$ (we leave τ as a variable to better see the effect of τ in later arguments).

When a job J_h is released at time r_h, RMLF takes the following actions:

- Job J_h is enqueued on Q_0.
- Job J_h is classified as blue.
- The target $T_{0,h}$ is set to $\max(1, 2 - \beta_{0,h})$.
- If, just prior to r_h, it was the case that Q_0 was empty, and that RMLF was running a job J_j, RMLF then takes the following actions:
  - Job J_j is preempted. Note that J_j remains at the front of its queue.
  - Job J_j is classified as red.
  - RMLF begins running J_h.

If at some time t, a job $J_j \in Q_{i-1}$ is being run when $w_j(t)$ becomes equal to $T_{i-1,j}$ then RMLF takes the following actions:

- Job J_j is dequeued from Q_{i-1}.
- Job J_j is enqueued on Q_i.
- Job J_j is classified as blue.
- The target $T_{i,j}$ is set to $\max(2^i, 2^{i+1} - \beta_{i,j})$.

Whenever a job is completed, it is removed from its queue. Recall that RMLF maintains the invariant that it is always running the job at the front of the lowest nonempty queue.

Note that the red-blue classification will be used only in the analysis. A job J_i is classified as red at a particular point of time t if J_i was preempted between when its last target was set and time t. Note that each queue has at most one red job, and if a queue contains a red job then that red job is at the front of the queue. So for a job J_j that is blue at time t, it is the case that $w_j(t)$ is equal to J_j's previous target. This equality does not hold for red jobs. This will require us to account for red and blue jobs differently. The following observation will later prove useful.

Observation 1 Consider two jobs J_i and J_j that have been released on or before time t, but not completed by RMLF at time t. If at time t it is the case that J_i is in a queue that is higher than the queue that J_j is in, then J_i was released no later than J_j. If at time t, J_i and J_j are on the same queue Q_k, and J_i is closer to the front of Q_k than J_j is, then J_i was released no later than J_j.

3 Algorithm Analysis

Definition 2

a. Let $U_A(t)$ be the set of jobs that are released by time t, but not completed by the scheduler A by time t.

b. Let ADV be the adversarial scheduler.

The crux of the analysis is to show that, with high probability, it is the case that, at almost all times t (the late times defined below) it is the case that

$$|U_{\mathrm{RMLF}}(t)| = O(\log n \log\log n)\ |U_{\mathrm{ADV}}(t)|$$

That is, the number of jobs that RMLF has not finished is at most $O(\log n \log\log n)$ times the number of jobs that the adversary has not finished. The following well-known fact (see for example [5]) shows why it is sufficient to compare the number of jobs that the adversary and RMLF have uncompleted at various times.

Fact 3 The total flow time for a deterministic algorithm A is $\int_0^\infty |U_A(t)|dt$, and by linearity of expectation, the total flow time for a randomized algorithm A is $\int_0^\infty E[|U_A(t)|]dt$.

We now give some intuition. If the adversary completes a job $J_i \in U_{\mathrm{RMLF}}(t)$ by time t, then the adversary needs to run J_i at some time s when RMLF is running some other job J_j. We think of this as the adversary borrowing time from J_j to give to J_i. This borrowing can also be transitive. For example, the adversary can borrow time from a job J_k to give to a job J_j, and can then borrow time from J_j to give to J_i. So the net result of this borrowing is that the adversary is trying to borrow time from jobs in $U_{\mathrm{ADV}}(t)$ to complete jobs in $U_{\mathrm{RMLF}}(t)$. Because of the random way in which the targets were set, we will argue that, in some amortized sense, for most jobs $J_i \in U_{\mathrm{RMLF}}(t)$ it is the case that the remaining processing time, $y_i(t)$, is not too small relative to the total processing time $x_i(t)$. Since RMLF always runs jobs at the front of the lowest nonempty queue, the adversary can not borrow time from jobs in the higher queues to give to jobs in the lower queues. Thus to complete most jobs $J_i \in U_{\mathrm{RMLF}}(t)$ the adversary must borrow a reasonable fraction of x_i from jobs $J_j \in U_{\mathrm{ADV}}(t)$ such that $x_j = O(x_i)$. Hence, the cardinality of $U_{\mathrm{ADV}}(t)$ can not be too small. This concept of borrowing was also used in [9].

Definition 4

a. The queues $Q_\ell, Q_{\ell-1}, \ldots Q_{\ell+1-2\lg n}$ are called *big*. The other queues are called *small*.

b. A time t is *early* if RMLF is running a job J_j in a small queue. Otherwise t is *late*.

c. A job J_j is *short for Q_i* if $2^{i-1} \le x_j \le 2^{i-1} + 2^{i-2}$. A job J_j is *long for Q_i* if $x_j > 2^{i-1} + 2^{i-2}$.

d. A job J_j is *unlucky* if it is short for some Q_i, and is promoted to Q_i at some time.

e. Let $BQ_i(t)$ be the queue formed from $Q_i(t)$ by retaining only the jobs in $Q_i(t)$ that are long for Q_i, and that are blue at time t.

f. We say that a queue Q_i is *bad* at some time t if there is a contiguous subqueue C of $BQ_i(t)$ with

$$\sum_{J_j \in C} y_j(t) \le \frac{|C|\; 2^{i-1}}{8\tau \ln n}\ \exp\left(\frac{-12 \ln n}{|C|}\right)$$

We now give some more intuition. It is not hard to see that the time spent running a job at the front of a small queue is so small that its contribution to the total flow time is negligible. Recall that we want to argue that at most times it is the case that the remaining processing times of the jobs in U_{RMLF} is not too small. It is quite possible that some jobs are promoted to a queue with very little remaining processing time. We would like to ignore those unlucky jobs with a hope that there are too many of them. After having eliminated the unlucky jobs, we want to make sure that the sum of remaining processing time of every collection jobs in a queue is reasonably large. A bad queue is one that has accumulated jobs whose aggregate remaining processing times are too small. We will then show that the probability that any big queue is ever bad is at most $1/n^2$.

3.1 High Probability Arguments

Here we argue that certain events that are bad for RMLF happen with low probability.

3.1.1 Unlucky Jobs

Here we argue that with high probability it is the case that RMLF doesn't have too many unlucky jobs.

Lemma 5 For some sufficiently large constant λ, the probability that the number of unlucky jobs exceeds $\lambda\sqrt{\lg n}$ is at most $1/n^2$.

Proof: Let J_j be a job that is short for Q_i. Let $x_j = 2^{i-1} + \gamma$, where $\gamma \leq 2^{i-2}$. Recall that $T_{i-1,j} = \max(2^{i-1}, 2^i - \beta_{i-1})$ and if J_j is unlucky for Q_{i-1} then it must be the case that $T_{i-1,j} < x_j$. Consider the setting of the threshold $T_{i-1,j}$ as a Bernoulli trial with outcomes J_j is unlucky for Q_{i-1} or J_j is lucky for Q_{i-1}. The probability that the kth trial results in an unlucky job is at most $\exp(\frac{-\tau \ln k}{2})$, $1 \leq k \leq n$. Let $v_1, \ldots, v_n$ be Bernoulli trials, with the probability of v_k being unlucky being $\exp(\frac{-\tau \ln k}{2})$. Hence, the probability that there are more than γ unlucky jobs is at most the probability that there are more than γ unlucky v_k. The number S of unlucky v_k is binomially distributed. The expected number $E[S]$ of unlucky v_k is $\sum_{k=1}^{n} 1/k^{\tau/2}$, the variance $\sigma^2[S]$ is $\sum_{k=1}^{n} [(1 - 1/k^{\tau/2})/k^{\tau/2}]$ [6]. Both $E[S]$ and $\sigma^2[S]$ are $O(1)$ since $\tau > 2$. The following Chernoff tail bound for a binomially distributed random variable S, with mean μ and variance σ^2, is well known (see [6]):

$$\text{Prob}[S - \mu \geq x] \leq e^{-x^2/4\sigma^2}$$

Thus in our case, by picking a sufficiently large λ (say $\lambda = 2\sigma$), and by letting $x = \lambda\sqrt{\lg n}$, one can make this tail bound less than $1/n^2$. ■

3.1.2 Bad Queues

One strategy for an adversary is to force the online algorithm to have many unfinished jobs with little remaining processing time. In this subsection, we will argue that RMLF defeats such an adversarial strategy with high probability. More precisely, we will argue that the probability that RMLF ever has a bad big queue is at most $1/n^2$. Since there are at most $2 \lg n$ big queues, it is sufficient to show that the probability that a particular big queue Q_i is ever bad at most $1/(2n^2 \lg n)$.

At any time t, note that $y_j \geq 2$ for each blue job $J_j \in Q_0$ since blue jobs in Q_0 have never been run by RMLF. Also, recall that we have made an assumption that 2 is the lower bound on the length of any job. Hence, Q_0 is never bad. So we now fix our attention a particular big queue Q_{i+1}.

Consider an arbitrary collection $\mathcal{J}$ of jobs. We want to show that the conditional probability that Q_{i+1} is ever bad, given that $\mathcal{J}$ is exactly the collection of jobs that were in BQ_{i+1} at some time, is at most $1/(2n^2 \lg n)$. That is, we want to show that

$$\text{Prob}[Q_{i+1} \text{ is ever bad } | \mathcal{J} = \cup_{t'} BQ_{i+1}(t')] \leq 1/(2n^2 \lg n) \tag{1}$$

Observe that $\sum_{\mathcal{J}} \text{Prob}[\mathcal{J} = \cup_{t'} BQ_{i+1}(t')] = 1$. Therefore, we can conclude that the probability that Q_{i+1} is ever bad is at most $1/(2n^2 \lg n)$.

We turn our attention to proving equation 1. So for the rest of subsection 3.1.2 we assume that $\mathcal{J} = \cup_{t'} BQ_{i+1}(t')$, and all of our probabilities are conditioned on this. We number the jobs in $\mathcal{J}$ so that if $J_j \in \mathcal{J}$ was promoted to Q_{i+1} before $J_h \in \mathcal{J}$ then $j < h$. Note that by observation 1 this is equivalent to ordering the jobs in $\mathcal{J}$ by increasing order of release times.

We now think of $\mathcal{J}$ as being an ordered list. It is clear that at any time t, the jobs in $BQ_{i+1}(t)$ are a contiguous collection of jobs from the ordered list $\mathcal{J}$; That is, if $J_h \in BQ_{i+1}(t)$, $J_j \in BQ_{i+1}(t)$, and $h \leq k \leq j$, then $J_k \in BQ_{i+1}(t)$. There are at most $\binom{n}{2} \leq n^2$ contiguous sublists of $\mathcal{J}$, and hence, there are at most $\binom{n}{2}$ choices for the subqueue C of BQ_{i+1} that forces Q_{i+1} to be bad. So we now fix a particular contiguous sublist C of $\mathcal{J}$. In order to prove equation 1 it is sufficient to show that the probability that C ever forces Q_{i+1} to be bad, given that all the jobs in C are promoted to Q_{i+1}, is at most $1/(2n^4 \lg n)$. Thus for the rest of subsection 3.1.2 we assume that all the jobs in C are promoted to Q_{i+1}, and all of our probabilities are conditioned on this.

Assume that we now renumber the jobs so that the jobs in C are $J_1 \ldots J_k$, still ordered by increasing order of release times. Let t be a time such that C forces Q_{i+1} to be bad at that time. Since C must be a subqueue of $BQ_{i+1}(t)$, each job in C must be blue and thus must not have been run since its arrival to the queue Q_{i+1}. Therefore, for each job $J_j \in C$

$$y_j(t) = x_j(t) - T_{i,j} = x_j(t) - \max(2^i, 2^{i+1} - \beta_{i,j}) \qquad (2)$$

In lemma 8 we will bound $\text{Prob}[\sum_{j=1}^{k} y_j(t) \leq \sigma]$ by relating this probability to the probability $\text{Prob}[\sum_{j=1}^{k} \beta_{i,j} \leq \sigma]$. Recall that $\beta_{i,j}$ is an exponentially distributed random variable.

The following basic facts about exponential distributions can be found in [8]. The expected value of an exponentially distributed random variable β, with probability distribution function $\text{Prob}[\beta \leq x] = 1 - e^{-x/b}$, is b [8]. Recall that $\beta_{i,j}$ has probability distribution function $\text{Prob}[\beta_{i,j} \leq x] = 1 - \exp(\frac{-\tau x ln \alpha(i,j)}{2^i})$. Hence, in our context this yields that $E[\beta_{i,j}] = 2^i/(\tau \ln \alpha(i,j))$. Exponential distributions are memoryless in that

$$\text{Prob}[\beta_{i,j} \leq \sigma + \nu \mid \beta_{i,j} \geq \nu] = \text{Prob}[\beta_{i,j} \leq \sigma]$$

We show in lemma 6 that with low probability it is the case that $\sum_{j=1}^{k} \beta_{i,j}$ is too small. We then show in lemma 7 that we can bound the probability that a particular $y_j(t)$ is small by the probability that a $\beta_{i,j}$ is small.

Lemma 6

$$\text{Prob}\left[\sum_{j=1}^{k} \beta_{i,j} \leq 4\tau \ln n \ \exp(\frac{-6 \ln n}{k})\right] \leq 1/n^6$$

Proof: For notational convenience, let $c = 4\tau \ln n \ \exp(\frac{6 \ln n}{k})$. So restating, we want to show that $\text{Prob}[\sum_{j=1}^{k} \beta_{i,j} \leq k2^i/c] \leq 1/n^6$. The sum S of k random variables, each with probability distribution function $1 - e^{-x/b}$, has the Erlang form of the gamma distribution with probability distribution function $\text{Prob}[S \leq x] = 1 - e^{-x/b}(\sum_{h=0}^{k-1} \frac{(x/b)^h}{h!})$ [8]. In our case, $S = \sum_{j=1}^{k} \beta_{i,j}$, and $b = \frac{2^i}{\tau \ln n}$. Notice that the sum $\sum_{h=0}^{k-1} \frac{(x/b)^h}{h!}$ is the first k terms of the Taylor series for $e^{x/b}$. Hence, we get

$$\text{Prob}[S \leq x] \quad \leq \quad 1 - e^{-x/b}(e^{x/b} - \sum_{h=k}^{\infty} \frac{(x/b)^h}{h!})$$

$$\leq e^{-x/b} \sum_{h=k}^{\infty} \frac{(x/b)^h}{h!}$$
$$\leq \sum_{h=k}^{\infty} \frac{(x/b)^h}{h!}$$

Now letting $x = 2^i k/c$ we get

$$\text{Prob}[S \leq 2^i k/c] \leq \sum_{h=k}^{\infty} \frac{\left(\frac{k\tau \ln n}{c}\right)^h}{h!} \leq 2\frac{\left(\frac{k\tau \ln n}{c}\right)^k}{k!}$$

The last inequality follows since $ch \geq 4k\tau \ln n$ for $h \geq k$. Using Stirling's approximation $k! \geq \sqrt{2\pi k}(\frac{k}{e})^k$ [6], we get

$$\text{Prob}[S \leq 2^i k/c] \leq \frac{\left(\frac{e\tau \ln n}{c}\right)^k}{\sqrt{k}}$$

Hence, to show $\text{Prob}[S \leq 2^i k/c] \leq 1/n^6$ it is sufficient to show $(\frac{c}{e\tau \ln n})^k \geq n^6$. Solving for c we get that this is equivalent to $c \geq e\tau \ln n \ \exp(\frac{6 \ln n}{k})$, which holds by the definition of c. ∎

Lemma 7 **For all $J_j \in C$, and for all $\gamma < 2^{i-1}$,** $\text{Prob}[y_j(t) \leq \gamma] \leq \text{Prob}[\beta_{i,j} \leq \gamma]$.

Proof: Consider a job $J_j \in C$. Observe that $x_j \geq 2^i + 2^{i-1}$ since J_j is long for Q_i. We break into two cases.

In the first case, assume that $2^i + 2^{i-1} \leq x_j \leq 2^{i+1}$. By using equation 2, and the fact that $\gamma < 2^{i-1}$ we get that

$$\begin{aligned} \text{Prob}[y_j(t) \leq \gamma] &= \text{Prob}[x_j - (2^{i+1} - \beta_{i,j}) \leq \gamma] \\ &= \text{Prob}[\beta_{i,j} \leq \gamma + 2^{i+1} - x_j] \end{aligned}$$

Recall that these probabilities are conditioned on the fact that all jobs in C are promoted to Q_{i+1}. Hence $x_j \geq T_{i,j}$, or equivalently, $\beta_{i,j} \geq 2^{i+1} - x_j$. Therefore,

$$\text{Prob}[\beta_{i,j} \leq \gamma + 2^{i+1} - x_j \mid \beta_{i,j} \geq 2^{i+1} - x_j] = \text{Prob}[\beta_{i,j} \leq \gamma]$$

by the memorylessness exponential distributions.

In the second case assume that $2^{i+1} \leq x_j$. First notice that $\text{Prob}[y_j(t) \leq \gamma] \leq \text{Prob}[y_j(t) \leq \gamma + (x_j - 2^{i+1})]$. Recall that $y_j(t) = x_j - T_{i,j} = (x_j - 2^{i+1}) + (2^{i+1} - T_{i,j})$ by definition of $T_{i,j}$. Therefore, $y_j(t) \leq \gamma + (x_j - 2^{i+1})$ implies that $2^{i+1} = T_{i,j} \leq \gamma$, which in turn implies that $\beta_{i,j} \leq \gamma$ since $\gamma < 2^{i-1}$. The result then follows since

$$\text{Prob}[y_j(t) \leq \gamma + (x_j - 2^{i+1})] = \text{Prob}[\beta_{i,j} \leq \gamma]$$

∎

Lemma 8 Let $\sigma = \frac{|C|\ 2^{i-1}}{4\tau \ln n}\ \exp(\frac{-6\ln n}{|C|})$. Then

$$\mathrm{Prob}[\sum_{J_j \in C} y_j(t) \leq \sigma] \leq \mathrm{Prob}[\sum_{j=1}^{|C|/2} \beta_{i,j} \leq \sigma]$$

Proof: Observe that for all $C' \subseteq C$ it is the case that

$$\mathrm{Prob}[\sum_{J_j \in C} y_j(t) \leq \sigma] \leq \mathrm{Prob}[\sum_{J_j \in C'} y_j(t) \leq \sigma]$$

We now define a random variable D to be the set of all jobs J_j in C such that $y_j(t) < 2^{i-1}$. Now observe that

$$\mathrm{Prob}[\sum_{J_j \in C} y_j(t) \leq \sigma] \leq \mathrm{Prob}[\sum_{J_j \in D} y_j(t) \leq \sigma]$$

We now argue that if $|D| \leq |C|/2$ then $\sum_{J_j \in C} y_j(t) > \sigma$. If $|D| \leq |C|/2$ then there are at least $|C|/2$ jobs in C with remaining processing time at least 2^{i-1}. Hence,

$$\sum_{J_j \in C} y_j(t) \geq \frac{|C|2^{i-1}}{2} > \sigma$$

So for the rest of the proof we assume that $|D| > |C|/2$, and condition all probabilities on this. Then

$$\mathrm{Prob}[\sum_{J_j \in C} y_j(t) \leq \sigma] \leq \mathrm{Prob}[\sum_{J_j \in D} y_j(t) \leq \sigma]$$

Now applying lemma 7, we get

$$\mathrm{Prob}[\sum_{J_j \in D} y_j(t) \leq \sigma] \leq \mathrm{Prob}[\sum_{J_j \in D} \beta_{i,j} \leq \sigma] \leq \mathrm{Prob}[\sum_{j=1}^{|C|/2} \beta_{i,j} \leq \sigma]$$

■

From lemma 6 and lemma 8 we can conclude that the probability that C ever forces Q_{i+1} to be be bad, given that all the jobs in C are promoted to Q_{i+1}, is at most $1/(2n^4 \lg n)$. Thus we are finally able to conclude with the main result of subsection 3.1.2.

Lemma 9 The probability that RMLF ever has a bad big queue is at most $1/n^2$.

3.2 Deterministic Analysis

We now fix any deterministic computation DMLF of RMLF during which RMLF never encounters a bad big queue, or more than $\lambda\sqrt{\lg n}$ unlucky jobs. We first note that we can restrict out attention to late times, and to counting blue jobs.

Lemma 10 $\int_{t \text{ is early}} |U_{\mathrm{DMLF}}(t)| dt = O(\text{optimal total flow time})$.

Proof: The total time that DMLF runs any particular job while it is in a small queue is $O(L/n^2)$. Recall that L is the length of the longest job. Therefore the total length of early times is $O(L/n)$. Since there are at most n uncompleted jobs at any time, $\int_{t \text{ is early}} |U_{\mathrm{DMLF}}(t)| dt = O(L)$. The optimal total flow time is clearly at least L. ■

Lemma 11 For all late t, the number of red jobs in $U_{DMFL}(t)$ is at most $2 \lg n$.

Proof: Observe that each queue can have at most one red job at any time. The lemma follows since at most $2 \lg n$ queues are not empty at any late time t. ■

We are now ready to prove that

$$|U_{\mathrm{DMLF}}(t)| = O(\log n \log \log n)\ |U_{\mathrm{ADV}}(t)|$$

for all late times t.

Definition 12 Let t be some arbitrary late time satisfying $|U_{\mathrm{DMLF}}(t)| > 0$.

For the rest of this section we will be concerned with the state of DMLF and the adversary at time t, unless we explicitly state otherwise.

3.2.1 Partitioning Uncompleted Jobs

In this section we partition the blue long jobs in $U_{\mathrm{DMLF}}(t)$ into classes P^h, $1 \le h \le f$, in such a way that the aggregate remaining processing time in each P^h is lower bounded by the time that any job in P^h has been processed by DMLF at time t. This will eventually let us conclude in lemma 18 that $|U_{\mathrm{ADV}}(t)| \ge f/2$.

Definition 13 Let t be the time under consideration.

a. Let BL be the set of jobs $J_j \in U_{\mathrm{DMLF}}(t)$ such that J_j is blue at time t, and such that if $J_j \in Q_h(t)$ then J_j is long for Q_h.

b. Let $\mathcal{P} = \{P^1, \ldots, P^f\}$ be defined inductively as follows: P^i is the collection of $\lceil 26\tau \lg n \ln \ln n \rceil$ jobs in $BL - \cup_{j=1}^{i-1} P^i$ (if enough such jobs exist) with earliest release times. Note that the jobs in BL, with latest release times, may not be contained in an element of $\mathcal{P}$ if there are insufficient jobs to form P^{f+1}.

c. Let $Q_{s(h)}, Q_{s(h)+1}, \ldots, Q_{d(h)}$ be the queues with nonempty intersection with P^h.

d. Let $PQ_{i,h} = P^h \cap Q_i$ be the jobs in P^h and Q_i.

e. Let $Y_{i,h} = \sum_{J_j \in PQ_{i,h}} y_j$ be the remaining processing time of those jobs in P^h and Q_i.

f. Let $n_{i,h} = |PQ_{i,h}|$ be the number of jobs in $PQ_{i,h}$.

Note that by observation 1 and the way that we have constructed $\mathcal{P}$, it is the case that $s(h) \geq d(h+1)$, $1 \leq h \leq f-1$, that is, no job in P^h is in any lower queue than the queue of a job in P^{h+1}. We now argue in lemma 14 that the remaining processing time in each P^h is reasonably large.

Lemma 14 For sufficiently large n, and for all $P^h \in \mathcal{P}$, $\sum_{J_j \in P^h} y_j > 2^{s(h)+1}$.

Proof: For notational simplicity, we fix a $P = P^h$, and drop reference to h in our notation. So, for short, $n_i = n_{i,h}, Y_i = Y_{i,h}, s = s(h)$ and $d = d(h)$. Note that if any $n_i \geq 24\tau \ln n$ then the result follows immediately since no queue is ever bad. To see this, one need only plug $|C| = n_i \geq 24\tau \ln n$ into the definition of a bad queue. Now to reach a contradiction assume that each $n_i \leq 24\tau \lg n$, and that $\sum_{J_j \in P} y_j = \sum_{i=s}^{d} Y_i \leq 2^{s+1}$. Note that since no queue is ever bad, it must be the case that each

$$Y_i \geq 2^{i-1} \frac{n_i}{8\tau \ln n \exp(\frac{\tau \ln n}{n_i})}$$

Consider the following optimization problem:

$$\boxed{\begin{array}{c} \min \sum_{i=s}^{d} 2^{i-1} \frac{n_i}{8\tau \ln n \exp(\frac{\tau \ln n}{n_i})} \\[2ex] \text{subject to } \sum_{i=s}^{d} n_i = 26\tau \lg n \ln \ln n \end{array}}$$

Observe that $\frac{n_i}{8\tau \ln n \exp(\frac{\tau \ln n}{n_i})}$ increases with increasing n_i. Therefore, it is easy to verify that in an optimal solution it must be the case that $n_i \geq n_{i+1}$, $s(h) = s \leq i < d(h) = d$, otherwise one can find a better solution by slightly increasing n_i and slightly decreasing n_{i+1}. Suppose, $n_i > 0$ in the optimal solution for some i satisfying $s < i \leq d$. It must be the case that

$$\frac{n_i 2^{i-1}}{\exp(\frac{\tau \ln n}{n_i})} \leq \frac{(n_s + n_i) 2^{s-1}}{\exp(\frac{\tau \ln n}{n_s + n_i})}$$

Otherwise, one could further minimize the objective function by setting n_s to $n_s + n_i$ and n_i to zero. Since $n_i \leq 24\tau \ln n$ for all $s \leq i \leq d$,

$$\frac{2^{i-1}}{\exp(\frac{\tau \ln n}{n_i})} \leq \frac{n_s + n_i}{n_i} \frac{2^{s-1}}{\exp(\frac{\tau \ln n}{n_s + n_i})}$$

$$
\begin{aligned}
&\leq 48\tau \ln n \frac{2^{s-1}}{\exp(\frac{1}{48})} \\
&\leq 48\tau 2^{s-1} \ln n
\end{aligned}
$$

After simplification this yields

$$\frac{\tau \ln n}{n_i} \geq (i-s)\ln 2 - \ln(48\tau \ln n)$$

Note that for $i \geq s + \lg(96\tau \ln n)$ we get that

$$(i-s)\ln 2 - \ln(48\tau \ln n) \geq \ln 2$$

and

$$n_i \leq \frac{\tau \ln n}{(i-s)\ln 2 - \ln(48\tau \ln n})$$

Now we split the sum of the n_i into two parts:

$$\sum_{i=s}^{d} n_i \leq \sum_{i<s+\lg(96\tau \ln n)} n_i + \sum_{i \geq s+\lg(96\tau \ln n)} n_i$$

Since each $n_i \leq 24\tau \ln n$ the first sum can be bounded as follows

$$
\begin{aligned}
\sum_{i<s+\lg(96\tau \ln n)} n_i &\leq \ln(96\tau \ln n) \cdot 24\tau \ln n \\
&= 24\tau \ln n \ln \ln n + 24\tau \ln(96\tau) \ln n \\
&= 24\tau \ln n \ln \ln n + O(\log n)
\end{aligned}
$$

Using our bound on n_i for large i, the second sum is can be bounded as follows

$$
\begin{aligned}
&\sum_{i \geq s+\lg(96\tau \ln n)} n_i \\
\leq &\sum_{i \geq s+\lg(96\tau \ln n)} \frac{\tau \ln n}{(i-s)\ln 2 - \ln(48\tau \ln n)} \\
\leq &(\tau \ln n) \sum_{i=1}^{2\lg n} \frac{1}{i \ln 2} \\
= &(\tau \lg n) \sum_{i=1}^{2\lg n} \frac{1}{i} \\
= &(\tau \lg n)(\ln(2\lg n) + O(1))
\end{aligned}
$$

The first inequality is valid due to the bound on n_i for large i. The second is valid by our choice of where to split the sum and a translation of the summation index. The last inequality is uses the fact that $\sum_{i=1}^{b} \frac{1}{i} = \ln b + O(1)$ [6]. Hence for sufficiently large n we get

$$\sum_{i=s}^{d} n_i \le 24\tau \ln n \ln\ln n + \tau \lg n \ln(2\lg n) + O(\log n)$$
$$< 25\tau \lg n \ln\ln n + O(\log n)$$
$$< 26\tau \lg n \ln\ln n$$

This is a contradiction to the definition of P^h. ■

3.2.2 Borrowing Time

Recall that t is some arbitrary but fixed late time at which we are attempting to lower bound $|U_{\mathrm{ADV}}(t)|$. We will assume from here on out that the adversary does not run any portion of any job $J_i \in U_{\mathrm{ADV}}(t)$ before time t, since running J_i before time t does not aid the adversary in minimizing $|U_{\mathrm{ADV}}(t)|$ unless J_i is completed by time t by the adversary. In order to show that $|U_{\mathrm{ADV}}(t)| \ge f/2$, we will demonstrate that the adversary is quite restricted in the manner that it can re-apportion time to complete jobs in $U_{\mathrm{DMLF}}(t)$. This will require several definitions. Recall that f is the cardinality of our partition $\mathcal{P}$ of the jobs in BL.

Definition 15

a. The *borrow network* contains one vertex for each job. Let $f_{h,j}$ be the amount of time, before time t, that the adversary was running J_j when DMLF was running J_h. Then there is a directed edge from J_h to job J_j with flow $f_{h,j}$ in the borrow network. Think of this flow as signifying that the adversary is borrowing $f_{h,j}$ units of time from J_h to give to J_j. (See [6] for background on network flow.)

b. In the borrow network, the *sources* are the jobs in U_{ADV}, and the *sinks* are the jobs in $U_{\mathrm{DMLF}} - U_{\mathrm{ADV}}$. Note that flow is conserved at non-sink and non-source vertices, which are the jobs that are completed by time t by both the adversary and DMLF.

c. For a source J_i, and a sink J_j, let $F_{i,j}$ be the net flow from J_i to J_j in the borrow network (note that the definition of $F_{i,j}$ may not be unique, and any arbitrary flow that meets the source and sink demands is fine.) We say that the adversary borrowed $F_{i,j}$ units of time from J_i to give to J_j.

We now want to define a quantity called the last queue value of a job. We want this quantity to be defined even for jobs that have already left the system. In order to do this we need to introduce a new scheduling algorithm IMLF that is almost identical to DMLF, except that in IMLF the jobs remain in the queues even after they are completed by IMLF. If some job J_j is completed at some time, IMLF promotes J_j to the next queue at that time. Also, if a previously completed job J_j is at the front of the lowest nonempty queue, then J_j is instantaneously promoted to the next queue. Note that the schedules produced by DMLF and IMLF are identical, just the internal states of the algorithms are different.

Definition 16 Let t' be some time not after t, and consider the algorithm IMLF. For a job $J_j \in Q_i(t')$ we define the *last queue value* of J_j at time t' to be $lq_j(t') = i$ if there is some time t'' such that

- queues $Q_1, \ldots, Q_{i-1}$ are empty at time t'',
- $t'' \leq t'$, and
- t'' is after J_j was promoted to Q_i.

Otherwise $lq_j(t') = i - 1$.

Observation 2 If $lq_j(t') = i$ then J_j was never run by DMLF while on queue Q_{i+1}, and hence $w_j(t')$ is at most 2^{i+1}.

We now are ready to show how DMLF restricts the manner in which borrowing can occur.

Lemma 17 If there is a directed path from J_h to J_j in the borrow network then $lq_j(t) \geq lq_h(t)$.

Proof: Consider the execution of IMLF. It is sufficient to show that if there is a directed edge from J_h to J_j in the borrow network then $lq_j(t) \geq lq_h(t)$. If we can show this, then the lemma follows by induction. Consider a time t' that IMLF was running J_h. If this was because J_h was on a lower queue than J_j then J_h will never be promoted to a queue before J_j. So assume that J_h and J_j were on the same queue, say Q_i, at time t'. Since J_h is being run at time t', applying the promotion rule for the completed jobs we can conclude that the lower queues are empty at time t'. As a result, $lq_h(t') = lq_j(t') = i$. While J_h will always be promoted to a particular queue before J_j, lq_h will not be incremented until the lower queues are empty, and hence until J_j is also promoted to the same queue as J_h. ■

We are now finally ready to lower bound $|U_{\mathrm{ADV}}(t)|$.

Lemma 18 $|U_{\mathrm{ADV}}(t)| \geq \max(1, \frac{f}{2})$.

Proof: The fact that $|U_{\mathrm{ADV}}(t)| \geq 1$ if $|U_{\mathrm{DMLF}}| > 0$ follows since DMLF doesn't unnecessarily idle the processor. So assume from here on that $f \geq 2$. We grant the adversary the power to borrow time from any job J_i to complete any job J_j provided $lq_j(t) \geq lq_i(t)$. Note that by lemma 17, this is being strictly generous to the adversary. Recall from definition 13 that $\mathcal{P}$ is a collection of disjoint sets of jobs. Let $P^{\sigma(1)}, P^{\sigma(2)}, \ldots, P^{\sigma(k)}$, be the sets in $\mathcal{P}$ with empty intersection with $U_{\mathrm{ADV}}(t)$. Also assume that we number the sets in $\mathcal{P}$ in such a way that $\sigma(i) < \sigma(j)$ whenever $i < j$. By renumbering, let $U_{\mathrm{ADV}} = \{J_i \mid 1 \leq i \leq |U_{\mathrm{ADV}}|\}$, where the numbering is by increasing order of lq values, that is, $lq_a(t) \leq lq_b(t)$ for $a \leq b$.

We now wish to show by induction on i, $1 \leq i \leq k$, that U_{ADV} contains at least i jobs, and for all h, $1 \leq h \leq i$, and $lq_h \leq s(\sigma(h))$, that is, the hth smallest lq value is not greater than the index of the lowest queue in $P^{\sigma(h)}$.

Consider the base case of $i = 1$. The set U_{ADV} is not empty because DMLF never unnecessarily idles the processor. If $lq_1 > s(\sigma(1))$ then the lq value of every job in U_{ADV} also greater than $s(\sigma(1))$ by the definition of J_1. By the definition of $P^{\sigma(1)}$, DMLF has at least one uncompleted job J_m in $Q_{s(\sigma(1))} \cap P^{\sigma(1)}$ at time t. By the definition of lq value, $lq_m(t) \leq s(\sigma(1))$. But then by lemma 17 it would not be possible for the adversary to borrow any time to complete J_m by time t. This is a contradiction to the assumption that the adversary has completed all the jobs in $P^{\sigma(1)}$.

Now consider the case of a general i, $1 < i \leq k$. Assume to reach a contradiction that $lq_i > s(\sigma(i))$, or that $|U_{\mathrm{ADV}}| = i - 1$. By lemma 14, partition $P^{\sigma(h)}$, $1 \leq h \leq i-1$ needs to borrow at least $2^{s(\sigma(h))+1}$ units of time. By the definition of $P^{\sigma(i)}$, DMLF has at least one uncompleted job J_ℓ in $Q_{s(\sigma(i))} \cap P^{\sigma(i)}$ at time t. That is, $y_\ell > 0$. Hence, to complete the jobs in $\mathcal{S} = \{J_\ell\} + \cup_{h=1}^{i-1} P^{\sigma(h)}$ the adversary needs to borrow at least $S = y_\ell + \sum_{h=1}^{i-1} 2^{s(\sigma(h))+1}$ units of time.

We now claim that the adversary can borrow the S units of time, required to complete the jobs in $\mathcal{S}$, only from the jobs $J_1, \ldots, J_{i-1}$. If $|U_{\mathrm{ADV}}| = i - 1$ then this is obvious. Now consider the case that $|U_{\mathrm{ADV}}| > i - 1$. By the definition of $\mathcal{P}$, every job in $\mathcal{S}$ is in a queue with index no bigger than $s(\sigma(i))$. By the definition of lq, the lq value of every job in $\mathcal{S}$ is at most $s(\sigma(i))$.

To reach our contradiction to the assumption that $lq_i > s(i)$, or that $|U_{\mathrm{ADV}}| = i - 1$ we now examine the amount of time that the adversary can borrow from the jobs $J_1, \ldots, J_{i-1}$. For each job J_h, $1 \leq h \leq i-1$, we have that $lq_h \leq s(\sigma(h))$ by induction. By observation 2, $w_h \leq 2^{s(\sigma(h))+1}$. Hence, the total amount of time that can be borrowed from jobs $J_1, \ldots, J_{i-1}$ is at most $\sum_{h=1}^{i-1} 2^{s(\sigma(h))+1}$, which falls short of S by y_ℓ units of time.

We have thus established by induction that $|U_{\mathrm{ADV}}(t)| \geq k$. By the definition of k, there are $f-k$ elements of $\mathcal{P}$ with nonempty intersection with U_{ADV}, and hence $|U_{\mathrm{ADV}}(t)| \geq f-k$. Therefore, $|U_{\mathrm{ADV}}(t)| \geq \max(f-k, k) \geq \frac{f}{2}$. ■

3.3 Putting the Pieces Together

Our desired bound on the competitive ratio can now be directly derived.

Lemma 19 For all late times t,

$$|U_{\mathrm{DMLF}}(t)| = O(\log n \log\log n)\ |U_{\mathrm{ADV}}(t)|$$

Proof: By lemma 5 and lemma 11 the number of jobs in $U_{\mathrm{DMLF}}(t)$, but not in BL, is $O(\log n)$. By lemma 18 $|BL| \leq O(\log n \log\log n)|U_{\mathrm{ADV}}|$, and $|U_{\mathrm{ADV}}(t)| \geq 1$. Hence, we can conclude that $|U_{\mathrm{DMLF}}(t)| = O(\log n \log\log n)\ |U_{\mathrm{ADV}}(t)|$, thus finishing the proof of lemma 19. ■

Theorem 20 RMLF is $O(\log n \log\log n)$-competitive against an adaptive adversary.

Proof: By fact 3 we can immediately conclude from lemma 19 that DMLF is $O(\log n \log\log n)$-competitive. Since RMLF does not unnecessarily idle the processors, it is at worst n-competitive. Hence, given lemma 11 and lemma 5, the competitive ratio of RMLF is at most $O(\log n \log\log n) + n\frac{2}{n^2} = O(\log n \log\log n)$. ■

Acknowledgments

Part of this research was conducted by the second author was visiting the Combinatorial Optimization Group in the Department of Mathematics at the Eindhoven University of Technology. In particular, the second author would like to thank Arjen Vestjens and Gerhard Woeginger for many helpful conversations and suggestions.

References

[1] M. Bach, *The Design of the Unix Operating System*, Prentice-Hall, 1986.

[2] P. Berman and C. Coulston, "Speed is more powerful than clairvoyance", Scandinavian Workshop on Algorithms and Theory, 255 -263, 1998.

[3] A. Borodin, and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[4] P. Brucker, *Scheduling Algorithms*, Springer-Verlag, 1995.

[5] E. Coffman, and P. Denning, *Operating Systems Theory*, Prentice-Hall, 1973.

[6] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[7] J. Edmonds, "Scheduling in the dark", ACM Symposium on Theory of Computing, 179 – 188, 1999.

[8] M. Evan, N. Hastings, B. Peacock, *Statistical Distributions*, John Wiley, 1993.

[9] B. Kalyanasundaram, and K. Pruhs, "Speed is as powerful as clairvoyance", *IEEE Symposium on Foundations of Computation*, 214 – 221, 1995.

[10] E. Lawler, J.K. Lenstra, A. Rinnooy Kan, and D. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity", in S. Graves, A. Rinnooy Kan, and P. Zipkin (eds.), *Logistics of Production and Inventory*, Handbooks in OR & MS 4, Elsevier Science, Chapter 9, 445–522, 1993.

[11] S. Leonardi, and D. Raz, "Approximating total flow time on parallel machines", *ACM Symposium on Theory of Computing*, 110 – 119, 1997.

[12] S. Madnick, and J. Donovan, *Operating Systems*, McGraw-Hill, 1974.

[13] T. Matsumoto, "Competitive analysis of the round robin algorithm', *International Symposium on Algorithms and Computation*, 71 – 77, 1992.

[14] R. Motwani, S. Phillips, and E. Torng, "Non-clairvoyant scheduling", *Theoretical Computer Science*, **130**, 17 – 47, 1994.

[15] C. Phillips, C. Stein, E. Torng, and J. Wein "Optimal time-critical scheduling via resource augmentation", *ACM Symposium on Theory of Computing*, 140 – 149, 1997.

[16] J. Sgall, "On-line scheduling - a survey", *On-Line Algorithms: The State of the Art*, eds. A. Fiat and G. Woeginger, Lecture Notes in Computer Science, Springer-Verlag.