Flow and Stretch Metrics for Scheduling Continuous Job Streams

Michael A. Bender* Soumen Chakrabarti† S. Muthukrishnan‡

1 Introduction

Many servers, such as web and database servers, receive a continual stream of requests. These requests may require an amount of processing time that varies over several orders of magnitude. The servers should schedule these requests to provide the "best" and "fairest" possible service to users. However, it is difficult to quantify this objective.

In this paper, we isolate and study the problem of scheduling a continuous stream of requests of varying sizes. More precisely, assume a request or *job* j has *arrival time* a_j and requires *processing time* t_j. We aim to schedule these jobs with a guaranteed *quality of service* (QoS). QoS is interpreted in many ways (e.g., throughput, avoiding jitter or delay, etc.). Here we adopt the widely-accepted requirement that the schedule be responsive to *each* job and avoid starvation of *any* job [1].

In this paper, we propose two novel scheduling metrics, namely, *max-stretch* and *max-flow*. These metrics gauge the responsiveness of the scheduler to *each* job. Surprisingly, despite the extensive research on scheduling algorithms in the past few decades, these metrics seem to be new. We initiate the study of optimizing these metrics under varying circumstances (offline/online, preemptive/nonpreemptive, etc.). We prove positive and negative theoretical results. We also report experiments with several scheduling heuristics on real data sets (HTTP server log requests at U.C. Berkeley). We present our observations on the overall "fairness" of various metrics and scheduling strategies. In what follows, we describe our metrics and results in detail.

Well-Known Performance Metrics. Let C_j denote the *completion time* of job j under some schedule. The two classical optimization metrics in Scheduling Theory are *makespan* (that is, $\max_j C_j$) and *average completion time* (that is, $(1/n)\sum_j C_j$). Both these measures can be approximated well under various scenarios. (For example, see the classical results in [11] for makespan and the recent developments [12] for average completion time.) However, neither is a suitable measure when jobs arrive in continuous streams.[1]

For jobs arriving continuously, a relevant parameter is the time a job j spends in the system, namely, the *flow time* defined as $F_j = C_j - a_j$ for job j. For over two decades [2], the *sum-flow*, $\sum_j F_j$, has been considered the suitable metric to optimize for continuous job arrivals. The *Shortest Remaining Processing Time* (SRPT) heuristic produces a schedule with optimum sum-flow [15] when there is a single processor. Unfortunately, this heuristic has the well-known drawback that it leads to starvation. That is, some jobs may be delayed to an unbounded extent. Inducing starvation is an inherent property of the sum-flow metric. In particular, there exists inputs where *any* optimal schedule for sum-flow forces the starvation of some job (see Lemma 2.1). This property is undesirable for continuous job streams.

Our Metrics. From the discussion above, it is natural to conclude that in order to avoid starvation, one should bound the flow time of *each* job. This motivates our first metric *max-flow*, $\max_j F_j$.

Our second metric, *max-stretch*, is motivated by the database literature [3, 19]. In this area, it is a well-accepted premise that the important parameter in scheduling continuous arrivals is by what factor a query (request/job) is slowed down *relative to the time it takes on an unloaded system.*[2] Formally, the *stretch* (also called *slowdown*) of a job j in a schedule is defined as $\frac{F_j}{t_j}$. As in max-flow, in order to avoid starvation, we bound the stretch of each job rather than the sum of the stretches of all jobs. This motivates our second metric, *max-stretch*, defined as $\max_j \frac{F_j}{t_j}$.

Both our metrics seem natural and complement ex-

*Information Sciences Center, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ 07974 and Aiken Computation Laboratory, Harvard University, 33 Oxford Street, Cambridge, MA 02138. bender@das.harvard.edu.

†IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. soumen@almaden.ibm.com.

‡Information Sciences Center, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ 07974. muthu@research.bell-labs.com.

[1] The makespan is a suitable measure for batched jobs, and average completion time is useful in some compiler optimization settings [6].

[2] Typical HTTP server responses vary over three orders of magnitude. It is unreasonable to get the same average response time (of say 10 minutes) for a 300 byte GIF file and also a 6 MBytes tarfile. On the other hand, guaranteeing a response time 200% of their request time may seem reasonable. This motivates the definition of the stretch metric.

isting metrics in the scheduling algorithms literature.[3] We study the problem of producing schedules that minimize our two metrics. We consider different settings: offline (the job arrival times are specified ahead of time), online (jobs are released only at their arrival time and the algorithm must decide only based on already released jobs), nonpreemptive (jobs must be processed without interruption), and preemptive (jobs may be suspended and resumed latter with no cost). In what follows, we state our theoretical and experimental results separately.

Theoretical Results. We assume that there are n jobs to be scheduled and unless otherwise specified, we consider a single processor.

1. Nonpreemptive max-flow. We consider the nonpreemptive max-flow minimization problem on $p \geq 1$ processors. This problem is trivially NP-Hard for $p \geq 2$. We show that the *First In First Out* (FIFO) strategy is an online algorithm having a competitive ratio of $3 - 2/p$ and that is tight.[4]

   When $p = 1$, FIFO is optimal. This observation is of some interest because it was previously unknown what metric the popular FIFO strategy optimizes. It helps validate our claim that max-flow is a natural metric.

2. Nonpreemptive max-stretch. We consider the nonpreemptive max-stretch minimization problem on a single processor. We prove that no polynomial-time algorithm can approximate the minimum max-stretch to within a factor $O(n^{1-\epsilon})$, for any constant $\epsilon > 0$ unless $P = NP$.

   Thus, preemption is necessary to achieve efficient algorithms that guarantee jobs do not starve. Although the sum-flow metric is similarly hard to approximate [16], sum-flow and max-stretch differ in following crucial way: Although approximations for nonpreemptive sum-flow may be obtained by massaging an optimal/approximate preemptive sum-flow schedule, this technique is unlikely to work for max-stretch. This is because there can be a large gap between the optimum max-stretch of the preemptive and nonpreemptive schedules (see Lemma 2.2).

3. Preemptive offline max-stretch. With preemption, we prove that there is a polynomial-time offline algorithm that, for any fixed ϵ, outputs a schedule having max-flow at most $1 + \epsilon$ times the optimum (minimum) max-flow. We prove a similar result for the max-stretch problem. They are used to obtain online preemptive algorithms for max-stretch described below.

4. Preemptive online max-stretch. Finally we consider online arrivals of jobs and the problem of minimizing the max-stretch with preemption. Let Δ be the ratio of the largest to the smallest job processing time. We prove these lower bounds. Any online algorithm must have a competitive ratio of $\Omega(\Delta^{1/3})$ even if there are only three job sizes. If there are two job sizes, any algorithm must have a competitive ratio approaching $\sqrt{2}$ as Δ increases. Our first negative result shows that optimizing the max-stretch online is a hard problem even on a single processor. (In contrast, recall that on a single processor SRPT optimizes sum-flow and FIFO optimizes max-flow.)

   We also present an on-line algorithm that has a competitive ratio of $O(\Delta^{1/2})$. If there are just two job sizes then the problem becomes easier. We present an on-line algorithm that has competitive ratio of $(1+\sqrt{5})\,/\,2$ when servicing jobs of two sizes.

   All algorithms work by guessing a suitable target max-stretch. We show that if this guess overestimates the optimal max-stretch currently, this *provably* helps. See Section 5 for details.

Experimental Results. We ran experiments using a real data set and compared the three metrics: sum-flow, max-flow, and max-stretch. Our goal was to understand their performance on parameters we did not incorporate into our metric, such as the standard deviation of the flow and stretch for various jobs, uniformity in penalizing different jobs etc. We hoped to determine the "fairness" of the metrics.

Our experiments are described in detail in Section 6. We collected HTTP server logs from U.C. Berkeley and undertook a detailed simulation study by programming an event-driven simulator together with the various schedulers. We compared FIFO (for max-flow), SRPT (for sum-flow) and a heuristic we call DEDF (for max-stretch). Since our online algorithm for max-stretch is somewhat expensive computationally, we used our intuition from there to derive the heuristic DEDF (details in Section 6).

Our observations are stated in Section 6 in more detail. We conclude that the stretch metric is the most promising of the three for fair scheduling of continuous job streams. Max-flow gives poor performance for

[3] In [22], for weight w_j of job j, the authors consider minimizing $\sum_j w_j F_j$ under a nonstandard model. Clearly, setting $w_j = 1/t_j$ reduces this expression to the sum of the stretches (denoted *sum-stretch*) metric. The sum-stretch metric has the same drawback that the sum-flow metric has, namely, it may induce starvation. Our interest, in contrast, is in max-stretch.

[4] An online algorithm has a competitive ratio of σ if for each input instance the objective value of the schedule is at most σ times the optimal objective values.

job mixes with many short jobs, and is mostly unsuitable. SRPT aggressively favors short jobs, leading to unreasonable starvation of long jobs. In a job mix with relatively flat job-time distribution, SRPT is therefore undesirable, In contrast, our DEDF heuristic for max-stretch gives the best performance in terms of smoothing out the distribution of flow and stretch across requests.

Our experimental work should not be construed as a complete solution for scheduling web servers; in practice there are additional considerations such as diverse and fluctuating network bandwidth available to the requesting clients. For our experiments we elided these details and assumed that a reasonably stable estimate of available bandwidth is known to the server. Nevertheless, the HTTP logs provided an appropriate and convenient real-life data set, and our preliminary results already appear sufficiently interesting to program and evaluate the heuristic on a heavily loaded real HTTP server. Database server scheduling is another promising area to apply our heuristics based on the new metric since the issue of fairness across diverse job mixes (e.g., transactions vs. decision support; see [3] is relevant. There, too, the entire systems solution involves other considerations such as buffer management, I/O, locking, etc. We are currently running further experiments and evaluating how to integrate the scheduling heuristics into the existing systems in the most effective manner.

Map. In Section 2 we present a collection of simple observations about our metrics. In Section 3, we present results for online max-flow. In Section 4, we present results for offline max-flow and max-stretch with and without preemption. In Section 5, we present results for online max-stretch with preemption. In Section 6, we describe our experiments and our observations.

2 Some Observations

Recall that there is only a single processor. Although SRPT minimizes the sum-flow, it can be very unfair to some jobs in that they get arbitrarily delayed. This is a property of the sum-flow metric itself as we show below:

LEMMA 2.1. *There exist inputs such that any schedule that minimizes the sum-flow must have unbounded max-flow and max-stretch.*

For some metrics, approximate preemptive schedules may be obtained by taking an optimal/approximate nonpreemptive schedule and massaging it. This for example is true for the sum-flow metric. One cannot hope to do so for max-stretch as we show:

LEMMA 2.2. *There exist inputs such that the optimal preemptive max-stretch and the optimal nonpreemptive max-stretch differ by a factor $\Omega(n)$.*

Processor sharing is a scheduling policy where time units are divided arbitrarily finely between jobs and all jobs currently in the system get equal share of the machine (i.e., if there are k jobs, each gets $1/k$ units of the processing done every time unit). This strategy turns out to be unfair as well.

LEMMA 2.3. *Processor sharing has $\Omega(n)$ competitive ratio for max-stretch.*

3 Nonpreemptive Max-Flow Approximation

In this section, we consider the max-flow metric: simple analysis shows that the First-In-First-Out (FIFO) scheduling strategy (that is, scheduling jobs in the order they arrive) is fairly good.

THEOREM 3.1. *FIFO is a $3 - 2/P$-competitive on-line nonpreemptive algorithm for max-flow, where P is the number of processors; furthermore, this bound is tight for FIFO.*

The following corollary is of interest: set $P = 1$ and it follows that FIFO strategy is optimal for minimizing the maxflow on a single processor. It is easy to see that no online nonpreemptive algorithm can have a competitive ratio at most $3/2$. The lower bound is easy by an (adaptive) adversary argument. Set $P = 2$. At time zero, two jobs of length 1 are released. If the algorithm decides to put them on the same processor, the adversary stops. Otherwise, the adversary submits a job of length two at time ϵ for some $\epsilon > 0$.

4 Offline Max-flow and Max-stretch

In [16], it was shown that no polynomial time algorithm can approximate the sum-flow to a factor $O(n^{1/2-\epsilon})$ for any $\epsilon > 0$ in the offline nonpreemptive case unless $P = NP$ (they also gave a scheduling algorithm with performance guarantee of $O(n^{1/2})$ for sum-flow). We show that the max-stretch metric is much harder to approximate. To show this, we use the fact that the following problem is NP-complete in the strong sense [8].

3-Partition: Given set A of $3m$ integers, a positive integer bound B, such that $\frac{B}{4} < a < \frac{B}{2}$ for all $a \in A$ and $\sum_{a \in A} a = mB$, can A be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that $\sum_{a \in A_i} a = B$ for all $i = 1, \ldots, m$?

THEOREM 4.1. *No polynomial time offline algorithm can approximate the nonpreemptive max-stretch problem to within a factor of $\Omega(n^{1-\epsilon})$ for arbitrarily small $\epsilon > 0$, unless $P = NP$.*

Next we consider offline preemptive case. We present an approximation for fairly general metrics. Suppose that we want to minimize *maxwt-flow*, that is, *maximum weighted flow* $D = \max_j w_j(C_j - a_j)$ for general w_j's. For max-flow, all $w_j = 1$; for max-stretch, $w_j = 1/t_j$.

THEOREM 4.2. *There exists an algorithm that finds a preemptive offline schedule with max-stretch at most $1+\epsilon$ times the optimal max-stretch in time polynomial in n and $1/\epsilon$, for any fixed $\epsilon > 0$.*

Observe that the framework enables one to attach priorities to the jobs.

5 On-line Max-stretch

In this section, we explore *on-line preemptive* algorithms for max-stretch. We consider the case of a single processor. In view of the results from Section 4, it is clear there is no hope for the on-line case without preemption. We will show that, even in the preemptive case, the global future information used in the above off-line algorithm is essential.

To develop intuition, we briefly consider the offline case. We can determine the feasibility of scheduling a set of tasks with a given stretch S (or any other *maxwt-flow* metric). This is because each job must be completed by a specific time (determined by the job's arrival time and processing time) to have a max-stretch of S. Therefore, we proceed by setting a *deadline* for each job and attempt to schedule the jobs. We schedule using the *Earliest Deadline First (EDF)* strategy, where we always execute the job in the system having the earliest deadline. The EDF strategy minimizes the maximum lateness of the jobs and hence finds a feasible schedule if one exists. Consequently, in the offline setting, we can perform a binary search for the optimal value of the max-stretch.

In the online setting, considered in the rest of this section, the algorithms must begin scheduling before knowing the optimal value of the max-stretch. The on-line algorithms maintain an *estimate* of the max-stretch and adjust the estimate as new jobs arrive. We call this type of algorithm an **adjustable-deadline EDF** algorithm. We focus on a specific algorithm that we call **stretch-so-far EDF** (see Figure 1). This algorithm bases its estimate on the optimal stretch of the jobs that have arrived so far.

We provide some definitions. Let S^* denote the optimal max-stretch of all jobs, and let $S^*_{\leq j}$ denote the optimal max-stretch of jobs $1 \ldots j$. If job i has an earlier deadline than job j, then we say that i has *higher priority* than j or that i is a *predecessor* of j. For a given schedule, we say that job i is *in the system* at time t, if it has already arrived, but is not yet completed.

$(\mathcal{E}_1, \ldots, \mathcal{E}_n)$ are estimates of the max stretch.

adjustable-deadline EDF: For all j, during the time interval $(a_j, a_{j+1}]$ (between the arrival time of the j-th and the $(j+1)$-st job):

1. Assign (or reassign) job i the deadline (priority) $D_i = a_i + \mathcal{E}_j \cdot t_i$.
2. At all times execute the job in the system having the earliest deadline.

Optimal algorithm: Estimates $\mathcal{E}_1 = \ldots = \mathcal{E}_n = S^*$.

Stretch-so-far EDF: Estimates $\mathcal{E}_j := \alpha \cdot S^*_{\leq j}$, $j = 1, \ldots, n$, where α is a parameter of the algorithm that can be tuned.

Figure 1: The family of **adjustable-deadline EDF** algorithms. These algorithms use an estimate of the optimal max-stretch to assign deadlines to each job. If the stretch estimate changes, the deadlines change. When the stretch estimate is, in fact, the optimal max-stretch then the algorithm is optimal. In this section, we study the **stretch-so-far EDF** scheduler, where the stretch estimate is α times the optimal max-stretch of the jobs that have arrived so far.

We present some observations useful for understanding possible algorithms. The on-line scheduler has the continual dilemma of choosing a good estimate of the optimal max-stretch. More specifically, the algorithm repeatedly faces the following difficult situation. The algorithm must schedule an incomplete small job and an incomplete large job, where the large job arrives before the small job. If the estimate of the max-stretch is small enough, the large job may have the earlier deadline and thus the higher priority. If the estimate of the max-stretch is larger, the small job now has the earlier deadline and the higher priority. As Lemma 5.4 in this section demonstrates, it may be difficult and critical to make the right choice.

Notice that if the estimate of the max-stretch is in fact S^*, then the algorithm is optimal. We compare the on-line algorithms in the section with this particular optimal algorithm (see Figure 1). Notice that regardless of the stretch estimates, jobs of the same size are executed FIFO.

In our presentation here, we assume $S^* \leq j$ can be computed for any j – that we can only compute this to $1+\epsilon$ approximation will not affect our analysis significantly. We will also assume that Δ is known to the online algorithm – this assumption can be easily removed, but we omit those details here.

5.1 Two Job Sizes. In Section 5.1, we analyze how the algorithm **stretch-so-far EDF** performs in the setting where there are two sizes of jobs. We assume that small jobs have unit length and large jobs have length Δ. We begin with the following lemma about **adjustable-deadline EDF** algorithms. The lemma shows that larger stretch estimates give higher priority to small jobs, thus delaying small jobs less and large jobs correspondingly more.

LEMMA 5.1. *Consider two executions of* **adjustable-deadline EDF**, *one having stretch estimates* $\bar{\mathcal{E}} = (\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n)$ *and the other having stretch estimates* $\bar{\mathcal{F}} = (\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n)$. *Suppose that for all* i, $\mathcal{E}_i \leq \mathcal{F}_i$. *Then,*

(1) At all times, both schedules have executed the same amount of work. (2) At all times, the $\bar{\mathcal{F}}$*-schedule has executed at least as much work on* small *jobs as the* $\bar{\mathcal{E}}$*-schedule (and consequently the* $\bar{\mathcal{E}}$*-schedule has executed at least as much work on* large *jobs as the* $\bar{\mathcal{F}}$*-schedule).*

Proof. (1) Immediate because the algorithm is greedy and executes work if some exists in the system.

(2) Proof by induction. We first define some terms. For the $\bar{\mathcal{E}}$ schedule at time a_γ, let $\mathcal{S}_{\bar{\mathcal{E}}}(a_\gamma)$ and $\mathcal{L}_{\bar{\mathcal{E}}}(a_\gamma)$ be the total amount of work in the system on small and large jobs respectively. For the $\bar{\mathcal{F}}$ schedule, let $\mathcal{S}_{\bar{\mathcal{F}}}(a_\gamma)$ and $\mathcal{L}_{\bar{\mathcal{F}}}(a_\gamma)$ be defined analogously.

Now assume inductively that at time a_γ,

$$\mathcal{S}_{\bar{\mathcal{F}}}(a_\gamma) \geq \mathcal{S}_{\bar{\mathcal{E}}}(a_\gamma) \text{ and } \mathcal{L}_{\bar{\mathcal{F}}}(a_\gamma) \leq \mathcal{L}_{\bar{\mathcal{E}}}(a_\gamma).$$

We prove that at time $a_{\gamma+1}$,

$$\mathcal{S}_{\bar{\mathcal{F}}}(a_{\gamma+1}) \geq \mathcal{S}_{\bar{\mathcal{E}}}(a_{\gamma+1}) \text{ and } \mathcal{L}_{\bar{\mathcal{F}}}(a_{\gamma+1}) \leq \mathcal{L}_{\bar{\mathcal{E}}}(a_{\gamma+1}).$$

Consider executing the rest of the jobs in the $\bar{\mathcal{E}}$ and $\bar{\mathcal{F}}$ schedules according to stretch estimates $\mathcal{E}_\gamma$ and $\mathcal{F}_\gamma$. Focus on an unexecuted small job i in the $\bar{\mathcal{F}}$-schedule. (By the inductive hypothesis and because small jobs are serviced FIFO, i is unexecuted in the $\bar{\mathcal{E}}$-schedule.) Job i is worked on in the $\bar{\mathcal{F}}$-schedule precisely when all of the higher priority jobs (small or large) are completed. Let the amount work of these jobs be denoted $y_{\bar{\mathcal{F}}}(i)$. Let $y_{\bar{\mathcal{E}}}(i)$ be defined analogously for the $\bar{\mathcal{E}}$-schedule. (Notice that no subsequently-arriving job can have a higher priority than i. Thus, in the $\bar{\mathcal{F}}$-schedule, x more work on job i will be completed at time

$$a_\gamma + y_{\bar{\mathcal{F}}}(i) + x\,.$$

Similarly for the $\bar{\mathcal{E}}$-schedule.

Now we show that for each job i,

$$y_{\bar{\mathcal{F}}}(i) \leq y_{\bar{\mathcal{E}}}(i)\,.$$

With the goal of proving the previous inequality, we describe how priorities change when the stretch estimate is adjusted. If a job ℓ's deadline is before job i's deadline according to stretch estimate $\mathcal{E}_\gamma$ then we write $\ell \prec_{\mathcal{E}_\gamma} i$. For *any size* job ℓ and *small* job i, $\ell \prec_{\mathcal{F}_\gamma} i \Rightarrow \ell \prec_{\mathcal{E}_\gamma} i$. For *any size* job ℓ and *large* job j, $\ell \prec_{\mathcal{E}_\gamma} j \Rightarrow \ell \prec_{\mathcal{F}_\gamma} j$.

In other words, among all jobs, completed or not, the predecessors of small job i in the $\bar{\mathcal{F}}$-schedule form a subset of the predecessors of i in the $\bar{\mathcal{E}}$-schedule. Furthermore, all of the completed small jobs are predecessors of i. Finally, at time a_γ, the $\bar{\mathcal{F}}$-schedule has completed at least as many small jobs as the $\bar{\mathcal{E}}$-schedule. From these three preceding observations, we deduce that $y_{\bar{\mathcal{F}}}(i) \leq y_{\bar{\mathcal{E}}}(i)$, for each small job i.

Thus, x units of work are accomplished on small job i in the $\bar{\mathcal{F}}$-schedule before in the $\bar{\mathcal{E}}$-schedule for all small jobs i and all values x. Therefore, at all times after a_γ, more work is accomplished on short jobs in the $\bar{\mathcal{F}}$-schedule than in the $\bar{\mathcal{E}}$-schedule. Since this claim is true at the specific time $a_{\gamma+1}$, the inductive hypothesis is proved.

We now analyze **stretch-so-far EDF**.

THEOREM 5.1. *If there are jobs of two lengths, then the algorithm* **stretch-so-far EDF** *with* $\alpha = 1$ *is 2-competitive for max-stretch.*

Proof. We prove the following two statements.

(1) A *large* job never completes later in **optimal** than in **stretch-so-far EDF**.

(2) A *small* job i completes at most $S^*_{\leq i}$ time units later in **stretch-so-far EDF** that in **optimal**.

From these two statements and because for all i, $S^* \geq S^*_{\leq i}$, the competitive ratio for max-stretch is at most 2.

We first prove statement (1). It is feasible to schedule all of the jobs using the stretch estimate S^*. The algorithm **stretch-so-far EDF** gives large jobs at least as high a priority as the stretch estimate S^*. Statement (1) follows from Lemma 5.1.

We now prove statement (2). Suppose that short job i completes x units later in **stretch-so-far EDF** than in **optimal**. This happens because at least x units of work are accomplished on *large* jobs not predecessors of i in **optimal**, *instead* of working on (already arrived) predecessors of i or on i itself.

Call this set of delayed or *displaced* small jobs $\mathcal{D}$. Consider the completion time τ, when the last *displacing* (large) job ℓ is completed. All of the displaced jobs in $\mathcal{D}$

(a) arrive before time τ;

(b) have an earlier deadline than job ℓ's deadline in **optimal**;

(c) (at least) until time τ, have a later deadline than ℓ in **stretch-so-far EDF**.

It follows from Lemma 5.1 that when a large job completes, it completes before its deadline. In particular, job ℓ's deadline is after time τ. Therefore all of the jobs in $\mathcal{D}$

(d) have later deadlines than time τ in **stretch-so-far EDF**.

(e) Notice that until time a_i, the optimal max-stretch so far is at most $S^*_{\leq i}$.

From conditions (a), (d), and (e), it is straightforward to show that $x \leq 2 \cdot S^*_{\leq i}$. The x units of small jobs must arrive after time $\tau - S^*_{\leq i}$ to satisfy conditions (d) and (e). The x units of small jobs must not have deadlines after $\tau + S^*_{\leq i}$ to satisfy conditions (a) and (e). Since there is feasible schedule having stretch estimate $S^*_{\leq i}$, $x \leq 2 \cdot S^*_{\leq i}$.

To improve the bound, we note that (at least) until time τ, there are also x units of work from displacing jobs that, by condition (c), also have deadlines before $\tau + S^*_{\leq i}$ in **stretch-so-far EDF**. Since there is feasible schedule having stretch estimate $S^*_{\leq i}$ that completes both the displacing jobs and displaced jobs by their deadlines, we obtain $x \leq S^*_{\leq i}$.

THEOREM 5.2. *If there are jobs of two lengths, then the algorithm* **stretch-so-far EDF** *with* $\alpha \geq 1$, *is* $\max(\alpha, (1+\alpha)/\alpha)$*-competitive for max-stretch.*

Proof. We first show that no *large* job has stretch greater than $\alpha \cdot S^*_{\leq i}$. This follows because it is feasible to schedule all of the jobs with stretch estimate $\alpha \cdot S^*$. Scheduling according to **stretch-so-far EDF** only gives higher priority to large jobs, and the claim follows from Lemma 5.1.

We now consider *small* jobs. Running the algorithm with $\alpha > 1$, we give higher priority to small jobs than when $\alpha = 1$. Therefore, by Lemma 5.1, small jobs are never later in the $\alpha > 1$ schedule than in the $\alpha = 1$ schedule. There are two cases.

(1) At time a_i, $\alpha \cdot S^*_{\leq i} < S^*$. Then from the proof of Theorem 5.1 and the previous statement, job i finishes at most $S^*_{\leq i}$ units after **optimal**'s deadline, having stretch $S^* + S^*_{\leq i}$. Since the optimal stretch is S^*, the competitive ratio is $(\alpha+1)/\alpha$.

(2) At time a_i, $\alpha \cdot S^*_{\leq i} \geq S^*$. Suppose that small job i completes x units later in **stretch-so-far EDF** than in **optimal**. This delay occurs only because at least x units of work is accomplished on large jobs not predecessors of i in **optimal** *instead* of working on (already arrived) predecessors of i. If the stretch estimate is greater than the max-stretch, then any predecessor of short job i according to the stretch estimate is also a predecessor according to the max-stretch. Thus, this delay can only occur when the stretch estimate is less than the optimal max-stretch, that is $\alpha \cdot S^*_{\leq i} < S^*$. At this time, we know from (1) that $x \leq S^*/\alpha$. Therefore small i can have stretch at most $S^* + S^*/\alpha$, yielding a competitive ratio of $(\alpha+1)/\alpha$.

COROLLARY 5.1. *For jobs of two lengths, choosing* $\alpha = (1+\sqrt{5})/2$ *yields a* $(1+\sqrt{5})/2$*-competitive algorithm for max-stretch.*

We now provide a lower bound on the competitive ratio when there are jobs of two sizes; proof is omitted here.

THEOREM 5.3. *When there are two job sizes there is no c-competitive algorithm for max-stretch, where c approaches* $\sqrt{2}$ *(from below) as* Δ *tends to* ∞.

5.2 Multiple Job Sizes. In Section 5.2, we analyze how the algorithm **stretch-so-far EDF** performs in the setting where there are three or more sizes of jobs. As before, let the largest job have length Δ and the smallest have unit length. With three job sizes, the problem becomes harder. We obtain the following lower bound that depends on Δ.

THEOREM 5.4. *For three lengths of jobs there is no* $\Delta^{\frac{1}{3}}/2$*-competitive on-line algorithm for max-stretch.*

Proof. To obtain a contradiction, assume that some $\Delta^{\frac{1}{3}}/2$-competitive algorithm exists. The adversary sends the following series of jobs.

> PHASE 1: Jobs 1 and 2 both have length Δ and appear at time 0, i.e., $(a_1, t_1) = (0, \Delta)$ and $(a_2, t_2) = (0, \Delta)$.
>
> PHASE 2: Jobs 3 through $\lceil (\Delta^{\frac{4}{3}} - 2\Delta)/k \rceil$ have length k and appear at intervals of k from time $2 \cdot \Delta - k$ until, but not including, time $\Delta^{\frac{4}{3}}$, i.e., they are $(2\Delta - k, k), (2\Delta, k), \ldots, (\lceil (\Delta^{\frac{4}{3}} - 2\Delta)/k \rceil \cdot k + 2\Delta - k, k)$.

The optimal schedule for these jobs executes the jobs in the order of their arrival times. It has stretch 2, because job 2 is deferred by Δ units of time and the jobs in phase 2 are deferred by k units of time.

Since the optimal schedule has stretch 2 and the algorithm is $\Delta^{\frac{1}{3}}/2$-competitive, jobs 1 and 2 must complete by time $\Delta^{\frac{4}{3}}$. (If not, the adversary sends no more jobs and we reach a contradiction.) Thus, one of the length k jobs cannot complete before time $\Delta^{\frac{4}{3}} + k \leq \lceil (\Delta^{\frac{4}{3}} - 2\Delta)\,/\,k \rceil \cdot k + 2\Delta + k < \Delta^{\frac{4}{3}} + 2k$. Now,

> PHASE 3: from k units after the last length-k job arrives (e.g., time $\lceil (\Delta^{\frac{4}{3}} - 2\Delta)\,/\,k \rceil \cdot k + 2\Delta$) until time $2\Delta^{\frac{4}{3}} - k - 1$, the adversary sends a string of unit-length jobs, one every time unit.

Notice that during the interval $[0, 2\Delta^{\frac{4}{3}}]$ there are $2\Delta^{\frac{4}{3}} - \epsilon$, $0 \leq \epsilon < 1$, units of work that enter the system and that because the processor is never idle, all the work is completed by time $2\Delta^{\frac{4}{3}} - \epsilon$. The best stretch that the algorithm can obtain is $\min\left(k+1, (\Delta^{\frac{4}{3}} - \epsilon)/k\right)$. This bound follows from two cases. If all length k jobs complete before time $2\Delta^{\frac{4}{3}} - \epsilon$, then some unit-length job must complete last at time $2\Delta^{\frac{4}{3}} - \epsilon$ and therefore must have stretch at least $k+1$. Otherwise, some length k job, completing last, has stretch at least $(\Delta^{\frac{4}{3}} - \epsilon)/k$.

However, for a proper choice of k, the optimal max-stretch is less than $2\Delta^{\frac{1}{3}}$. The optimal schedule delays finishing job 2 until time $2\Delta^{\frac{4}{3}} - \epsilon$ and executes all length k and unit-length jobs as soon as they arrive. In particular, if $k = \Delta^{\frac{2}{3}} - 1$ then the competitive ratio is greater than $\Delta^{\frac{1}{3}}/\,2$. Therefore we reach a contradiction.

THEOREM 5.5. *For multiple job lengths and $\alpha = 1$,* **stretch-so-far EDF** *is Δ-competitive for max-stretch. This result is tight to within a constant factor.*

We provide an example showing that the preceding analysis is tight to within a constant factor; details of this example are omitted here.

THEOREM 5.6. *For multiple job lengths,* **stretch-so-far EDF** *is $\max(\alpha, 1 + \Delta\,/\,\alpha)$-competitive for max-stretch.*

Proof of Theorems 5.5 and 5.6. We first analyze the performance of **stretch-so-far EDF** with $\alpha = 1$. Suppose that job i completes during the interval $(a_j, a_{j+1}]$. We prove that job i completes at most $\Delta \cdot S^*_{\leq j}$ units after its deadline (i.e., i's deadline during the interval $(a_j, a_{j+1}]$).

Suppose i completes x units after its deadline. This happens because at least x units of work are accomplished on jobs *not* predecessors of i *according to stretch estimate $S^*_{\leq j}$ instead* of working on (already arrived) predecessors of i or on i itself. Denote the set of jobs *not* predecessors of i by $\mathcal{D}$. These *displacing* jobs have these characteristics:

(a) The jobs must all be larger than job i. This follows because given a larger stretch estimate, these jobs have later deadlines than i.

(b) The jobs must arrive before time a_i. This follows from (a) and because in **stretch-so-far EDF**, they have earlier deadlines than i.

(c) The jobs must have deadlines after time a_i according to stretch estimate $S^*_{\leq j}$. This follows because these jobs have deadlines after i's deadline according to stretch estimate $S^*_{\leq j}$.

(d) Notice that the optimal max-stretch so far is at most $S^*_{\leq j}$.

From these previous conditions it is straightforward to show that $x \leq 2 \cdot S^*_{\leq j} \cdot \Delta$. The x units must arrive after $a_i - S^*_{\leq j} \cdot \Delta$ by conditions (c) and (d). The x units must complete by $a_i + S^*_{\leq j} \cdot \Delta$ by conditions (b) and (d). By condition (d), $x \leq 2 \cdot S^*_{\leq j} \cdot \Delta$. To improve the bound, we note that there are also x units of *displaced* work that, arriving before time a_i must be completed by time $a_i + S^*_{\leq j} \cdot \Delta$. Thus, again by condition (d), we obtain $x \leq S^*_{\leq j} \cdot \Delta$.

The case of $\alpha > 1$ is similar to that with only two jobs; we omit the details. □

COROLLARY 5.2. *For multiple job lengths, setting $\alpha = O(\sqrt{\Delta})$ yields a $O(\sqrt{\Delta})$-competitive algorithm for max-stretch.*

6 Experiments

Our broad goal is to understand "fairness" in scheduling continuous job streams in practical settings. In the treatment above, we considered a few notions of fairness that could be analyzed, namely, the max flow and stretch metrics, in the worst-case competitive ratio sense. In practice, two additional issues arise. First, the max norm may be too sensitive to outliers; perhaps the *distribution* of flow or stretch is important as well. (We do not have a handle on the analysis of sum of stretch, for instance.) Second, interesting insights may be added to our worst-case analysis by experimenting with real-life datasets.

6.1 The setup. We used HTTP server logs[5] to generate job streams. For our purposes, a log is a series of records, each recording the arrival time of the HTTP request, and the length in bytes of the response. Some statistics about the workload follow. We considered $n = 2000$ jobs arriving over 3580 seconds. The smallest file requested was 49 bytes; the largest was about 760 kB. The total transfer was about 29.8 MB. There is no record in the logs about the actual time taken to transfer each file (job). We built event-driven simulators to ingest these logs and run various scheduling heuristics, making the following simplifying assumptions.

[5] At the server `http://HTTP.CS.Berkeley.EDU`.

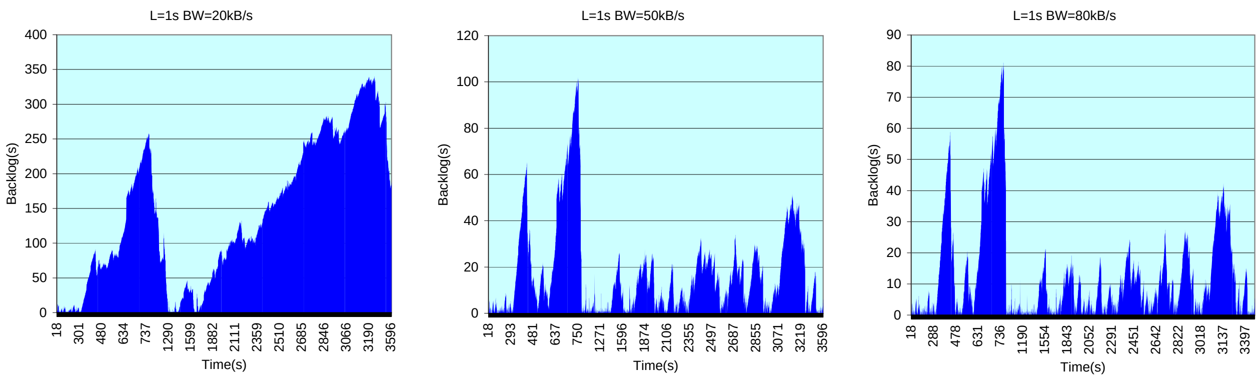

Figure 2: Backlog simulation.

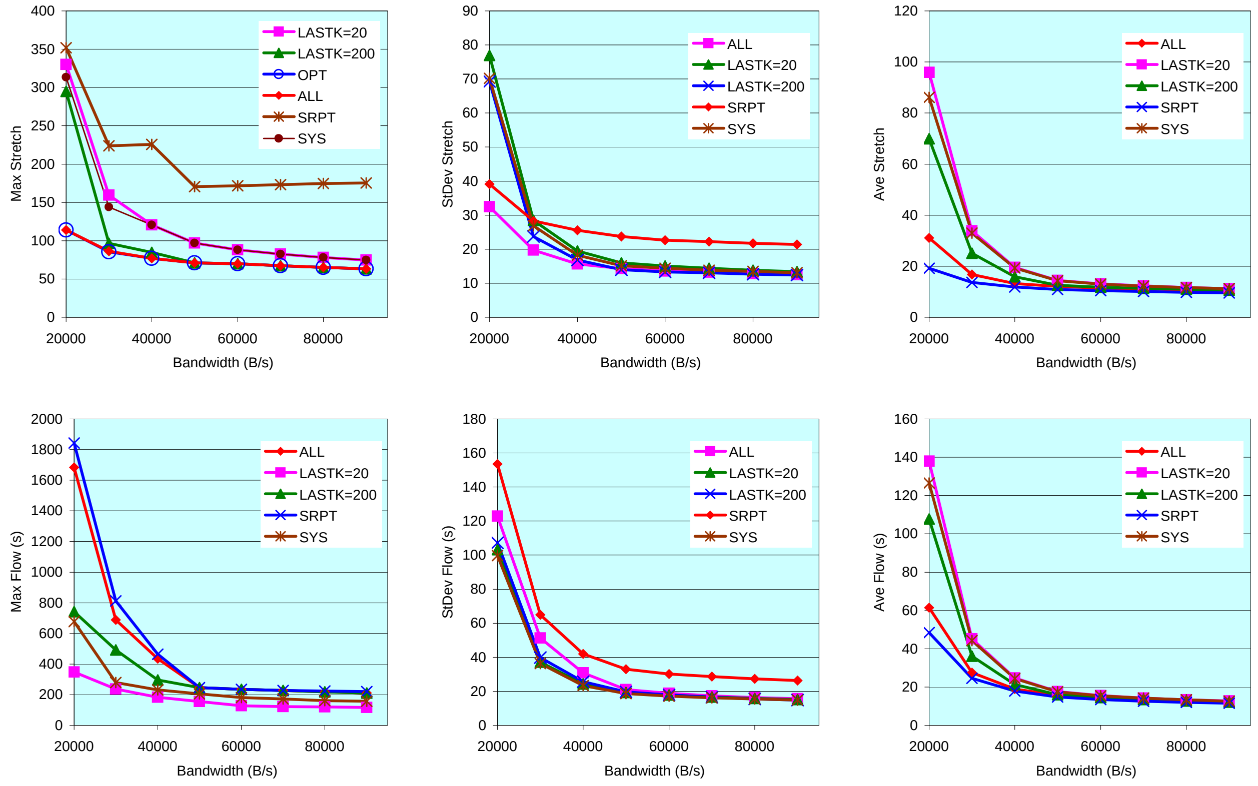

Figure 3: Comparison of different strategies.

- The only resource we model is a set of network ports with certain limited bandwidth. It is reasonable to ignore CPU and disk bandwidth issues given their typical values and those of wide-area network links. The total time for a response is assumed to follow the usual "startup plus time-per-byte" model.

- We keep the simulation simple by assuming that the bandwidth is relatively unchanged during a transfer and that it can be estimated at job arrival time. This means that the requested web page size in bytes (together with recent measurements on available throughput) directly gives the jobs running time.

- We assume that each log entry in the log file is a job independent of others; in reality, the log is a merge of *trails* from many users, where a user can be represented as a precedence chain of jobs, with some distribution of "think time" between job submissions. We are currently extending the simulations to study such interactive systems.

- The queries are independent of the scheduler's actions. In particular, there is no system or user *timeout*. Thus, we assume that the client does not click the "stop" button midway; i.e., the entire length of each document is actually transferred to the client.

We experiment with 2000 client requests arriving over an hour at the U.C. Berkeley EECS HTTP web server. The server logs give us file transfer lengths ℓ_j for each access (job) j. We convert this into a "job time" t_j by assuming latency L and bandwidth B parameters and computing $t_j = L + \ell_j/B$. We fix L to one second, and vary B between 20 and 90 kB/s. These values were chosen on the basis of a few dozen HTTP accesses over North America and a least-square fit of the transfer times. To make sure our parameters are reasonable, we plot in Figure 2 the "backlog" work in the system at all times for some values of B. When a job j arrives, the backlog jumps up by the job time t_j. At other times, the backlog, if positive, decreases with slope -1. If the backlog function returns to zero frequently, the workload is reasonable for the online stretch scenario we are dealing with. At 20 kB/s, the system is mildly overloaded. At larger bandwidth (smaller job times), the system is not overloaded, as hinted by the frequent return to zero in the second and third charts.

6.2 The max-stretch heuristic. The online algorithm and its analysis motivated the following heuristic.

1. At all times, we maintain a "goal" stretch value D_0.
2. Given D_0 and a waiting list J of jobs, we assign starting deadlines d_j to each job:

$$d_j = \alpha D_0 t_j + a_j,$$

where $\alpha \geq 1$ is the factor discussed in the analysis of the online algorithm.

3. We pick the job with the smallest d_j to run next.
4. After every job j is completed, we update D_0.

The heuristic is tuned by choosing D_0, α and different strategies for updating D_0. We call this entire heuristic as DEDF (Dynamic EDF).

D_0 could be set to the offline optimal max-stretch of all arrived jobs, found using earliest deadlines or network flows. In practice, it may be too expensive to solve the offline problem for the entire history at every arrival. Besides, this estimate could be overly pessimistic; a momentary surge of arrivals a long time ago will induce the scheduler to compute a large D_0 ever after. In practice, the scheduler may be designed to have a short-term memory of the stretches of the last few jobs completed. In some of our experiments, we use the largest stretch of the last k jobs that completed. Initially D_0 can be set to one. Also, α can be tuned to give a family of online heuristics. The worst-case analysis shows that increasing α beyond one decreases the max-stretch.

6.3 Results. We present a small selection of our results here. See Figure 3 for the comparison of our DEDF heuristics with SRPT (shortest remaining processing time) and offline optimal OPT. The variations of DEDF are ALL, which uses an offline optimal over all jobs that have arrived so far to compute the max stretch estimate; SYS, which uses for the estimate only those jobs that are currently running or waiting in the system; and LASTK, which uses the last k jobs that arrived for the estimate. There are $n = 2000$ arrivals; we use $k = 20$ and $k = 200$. All diagrams have bandwidth between 20 and 90 kB/s. The first diagram plots max stretch. SRPT is the worst; it is clearly not optimizing max stretch. ALL is essentially indistinguishable from OPT at all points shown, and between 2.4 and 3 times smaller than SRPT. The others lie in between. The second diagram shows standard deviation of stretch. Here, too, SRPT is unfair compared to ALL. The others are worse than SRPT at low bandwidth but get better quickly with increasing bandwidth. The improvement in fairness is at a mild cost of overall mean stretch as shown in the third chart; there too ALL is very close to SRPT. The LASTK variants have some of the lowest max flow in the fourth chart; this is because as k gets small, LASTK starts approximating FIFO, and

FIFO is optimal for max flow. The fifth chart shows that the max stretch algorithms are superior to SRPT even in standard deviation of flow. SRPT provably wins for average flow, as is also supported by the last chart, but ALL is not far off; at all but the smallest bandwidths, the others are close as well. ALL is most consistently good, but it is expensive to compute. Therefore, in practice one can use one of the simpler variants based on which is closest to ALL under the desired metric. This is not the same in all cases.

7 References

[1] M. J. Bach. *The Design of the UNIX Operating System.* Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1986.

[2] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

[3] K. P. Brown, M. Mehta, M. J. Carey, and M. Livny. Towards automated performance tuning for complex workloads. Technical Report TR 1247, Computer Sciences Department, University of Wisconsin, Madison, WI, Sept. 1994.

[4] M. J. Carey, M. Livny, and H. Lu. Dynamic task allocation in a distributed database system. *CS TR.*

[5] S. Chakrabarti, C. Phillips, A. Schulz, D. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 1996.

[6] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. R. Rau, and M. Schlansker. Profile-driven instruction level parallel scheduling with application to super blocks. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, pages 58–67, Paris, France, Dec. 2–4, 1996. IEEE Computer Society TC-MICRO and ACM SIGMICRO.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. ACM SIGCOMM '89*, pages 1–12, Austin, TX, Sept. 1989.

[8] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[9] S. J. Golestani. A seli-clocked fair queueing scheme for broadband applications. In *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies on Networking for Global Communciation. Volume 2*, pages 636–646, Los Alamitos, CA, USA, June 1994. IEEE Computer Society Press.

[10] P. Goyal, X. Guo, and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. Technical Report CS-TR-96-12, The University of Texas at Austin, Department of Computer Sciences.

[11] R. L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.

[12] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and online algorithms. In *Symposium on Discrete Algorithms (SODA).* ACM-SIAM, 1996.

[13] L. A. Hall. *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*, chapter Approximation Algorithms for Scheduling. PWS Publishing Company, Boston, MA, 1996.

[14] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: offline and online algorithms. *Math of OR*, 1997. To appear.

[15] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In *CRC Handbook of Computer Science*, 1997. To appear.

[16] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single maching. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 418–426, Philadelphia, Pennsylvania, 22–24 May 1996.

[17] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. pages 445–522, 1993.

[18] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 110–119, El Paso, Texas, 4–6 May 1997.

[19] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. In R. Agrawal, S. Baker, and D. Bell, editors, *Very large data bases, VLDB '93: proceedings of the 19th International Conference on Very Large Data Bases, August 24–27, 1993, Dublin, Ireland*, pages 354–367, Palo Alto, Calif., USA, 1993. Morgan Kaufmann Publishers.

[20] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. Technical Report TR 1151, Computer Science Department, University of Wisconsin-Madison, June 1993.

[21] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. *Lecture Notes in Computer Science*, 955:86, 1995.

[22] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 May 1997.

[23] M. Pinedo. *On-Line Algorithms, Lecture Notes in Computer Science.* Prentice Hall, 1995.

[24] J. Sgall. On-line algorithms, lecture notes in computer science. In A. Fiat and G. Woeginger, editors, *On-Line Scheduling – A survey*, Berlin, 1997. Springer Verlag. To appear.

[25] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, Feb. 1992.

[26] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. *SIGCOMM 90*, pages 19–29, 1990.