Scalably Scheduling Processes with Arbitrary Speedup Curves

JEFF EDMONDS, York University
KIRK PRUHS, University of Pittsburgh

We give a scalable ($(1+\epsilon)$-speed $O(1)$-competitive) nonclairvoyant algorithm for scheduling jobs with sublinear nondecreasing speedup curves on multiple processors with the objective of average response time.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Multiprocessor, scheduling

ACM Reference Format:
Edmonds, J. and Pruhs, K. 2012. Scalably scheduling processes with arbitrary speedup curves. ACM Trans. Algor. 8, 3, Article 28 (July 2012), 10 pages.
DOI = 10.1145/2229163.2229172 http://doi.acm.org/10.1145/2229163.2229172

1. INTRODUCTION

Computer chip designers are agreed upon the fact that chips with hundreds to thousands of processors will dominate the market in the next decade. The founder of chip maker Tilera asserts that a corollary to Moore's law will be that the number of cores/processors will double every 18 months [Merritt 2008]. Intel's director of microprocessor technology asserts that while processors will get increasingly simple, software will need to evolve more quickly than in the past to catch up [Merritt 2008]. In fact, it is generally agreed that developing software to harness the power of multiple processors is going to be a much more difficult technical challenge than the development of the hardware. In this article, we consider one such software technical challenge: developing operating system algorithms/policies for scheduling processes with varying degrees of parallelism on a multiprocessor.

We will consider the setting where n processes/jobs arrive to the system of m processors over time. Job J_i arrives at time r_i, and has a work requirement w_i. At each point of time, a scheduling algorithm specifies which job is run on each processor at that time. An operating system scheduling algorithm generally needs to be *nonclairvoyant*, that is, the algorithm does not require internal knowledge about jobs, say, for example, the jobs' work requirement, since such information is generally not available to the operating systems. Job J_i completes after its w_i units of work have been processed. If a job J_i completes at time C_i, then its response time is $C_i - r_i$. In this article we will consider the schedule quality-of-service metric *total response time*, which for a schedule

Authors' addresses: J. Edmonds, Department of Computer Science, York University, 4700 Keele St., Toronto, Ont., Canada M3J 1P3; K. Pruhs (corresponding author), Department of Computer Science, University of Pittsburgh, 210 South Bouquet St., Sennott Square Building, Room 6415, Pittsburgh, PA 15260; email: kirk@cs.pitt.edu.

DOI 10.1145/2229163.2229172 http://doi.acm.org/10.1145/2229163.2229172

S is defined to be $F(S) = \sum_{i=1}^{n}(C_i - r_i)$. For a fixed number of jobs, total response time is essentially equivalent to average response time. Average response time is by far the mostly commonly used schedule quality-of-service metric. Before starting our discussion of multiprocessor scheduling, let us first review resource augmentation analysis and single processor scheduling.

Resource augmentation analysis compares an online scheduling algorithm against an offline optimal scheduler with less powerful resources. An online scheduling algorithm A is s-speed c-competitive if

$$\max_I \frac{F(A_s(I))}{F(\mathrm{Opt}_1(I))} \leq c,$$

where $A_s(I)$ is the schedule produced by algorithm A with speed s processors on input I, and $\mathrm{Opt}_1(I)$ is an optimal schedule for unit speed processors on input I, and $F(S)$ is the total response time for schedule S [Kalyanasundaram and Pruhs 2000; Phillips et al. 2002]. An online scheduling algorithm A is s-processor c-competitive if

$$\max_I \frac{F(A_s(I))}{F(\mathrm{Opt}_1(I))} \leq c,$$

where $A_s(I)$ is the schedule produced by algorithm A with sm unit speed processors on input I, and $\mathrm{Opt}_1(I)$ is an optimal schedule for m unit speed processors on input I [Kalyanasundaram and Pruhs 2000; Phillips et al. 2002]. Since in the context of preemptive scheduling, a speed s processor is always at least as useful as s unit speed processors, an s-processor c-competitive algorithm A can easily be converted into an s-speed c-competitive algorithm. We call an algorithm A *universally speed/processor scalable* if for every $\epsilon > 0$, there is a constant c_ϵ such A is $(1+\epsilon)$-speed/processor c_ϵ-competitive [Pruhs et al. 2004; Pruhs 2007]. We we call a family of algorithms A_ϵ *existentially speed/processor scalable* if for every $\epsilon > 0$, there is a constant c_ϵ such algorithm A_ϵ is $(1+\epsilon)$-speed/processor c_ϵ-competitive. A scalable algorithm is $O(1)$-competitive on inputs I where $\mathrm{Opt}_1(I)$ is approximately $\mathrm{Opt}_{1+\epsilon}(I)$, which intuitively are inputs that do not fully load the server. So as the load increases, the performance of a scalable algorithm should be reasonably close to the performance of the optimal algorithm up until the server is almost fully loaded. For a more detailed explanation, see Pruhs et al. [2004] and Pruhs [2007].

The nonclairvoyant algorithm Shortest Elapsed Time First (SETF) is universally speed scalable [Kalyanasundaram and Pruhs 2000] for scheduling jobs on a single processor (or for scheduling jobs on a multiprocessor if all jobs are fully parallelizable) for the objective of total response time. SETF shares the processor equally among all processes that have been processed the least to date. Intuitively, SETF gives priority to more recently arriving jobs, until they have been processed as much as older jobs, at which point all jobs are given equal priority. The process scheduling algorithm used by most standard operating systems, for example, Unix, essentially schedules jobs in way that is consistent with this intuition. No nonclairvoyant scheduling algorithm can be $O(1)$-competitive for total response time if compared against the optimal schedule with the same speed [Motwani et al. 1994]. The intuition is that one can construct adversarial instances where the load is essentially the capacity of the system, and there is no time for the nonclairvoyant algorithm to recover from any scheduling mistakes.

One important issue that arises when scheduling jobs on a multiprocessor is that jobs can have widely varying degrees of parallelism. That is, some jobs may be considerably sped up when simultaneously run on to multiple processors, while some jobs may not be sped up at all (this could be because the underlying algorithm is inherently sequential in nature, or because the process was not coded in a way to make it easily parallelizable). To investigate this issue, we adopt the following general model used in

Edmonds [2000]. Each job consists of a sequence of phases. Each phase consists of a positive real number that denotes the amount of work in that phase, and a speedup function that specifies the rate at which work is processed in this phase as a function of the number of processors executing the job. The speedup functions may be arbitrary, other than we assume that they are nondecreasing (a job doesn't run slower if it is given more processors), and sublinear (a job satisfies Brent's theorem, that is, increasing the number of processors doesn't increase the efficiency of computation).

The most obvious scheduling algorithm in the multiprocessor setting is Equipartition (Equi), which splits the processors evenly among all processes. Equi is analogous to the Round Robin or Processor Sharing algorithm in the single processor setting. In what is generally regarded as a quite complicated analysis, it is shown in Edmonds [2000] that Equi is a $(2+\epsilon)$-processor $(\frac{2s}{\epsilon})$-competitive for total response time. It is also known that, even in the case of a single processor, speed at least $2+\epsilon$ is required in order for Equi to be $O(1)$-competitive for total response time [Kalyanasundaram and Pruhs 2000].

1.1. Our Results

In this article we introduce a family of nonclairvoyant algorithms, parameterized by a real $\beta \in (0, 1]$, which we call $\text{LAPS}_{\langle\beta,s\rangle}$. The subscript of s denotes that the algorithm is using sm processors. Arguably the processor augmentation parameter s is not a property of the algorithm, but we include it for convenience. We then show that $\text{LAPS}_{\langle\beta,s\rangle}$ is existentially processor scalable for scheduling jobs with sublinear nondecreasing speedup curves with the objective of total response time.

$LAPS_{\langle\beta,s\rangle}$ (Latest Arrival Processor Sharing) Definition. Let n_t be the number of jobs alive at time t. The processors are equally partitioned among the $\lceil \beta n_t \rceil$ jobs with the latest arrival times (breaking ties arbitrarily but consistently).

Note that $\text{LAPS}_{\langle\beta,s\rangle}$ is a generalization of Equi since $\text{LAPS}_{\langle 1,s\rangle}$ identical to Equi_s. But as β decreases, $\text{LAPS}_{\langle\beta,s\rangle}$, in a manner reminiscent of SETF, favors more recently released jobs. The main result of this article, which we prove in Section 3, is then as follows.

THEOREM 1.1. *$\text{LAPS}_{\langle\beta,s\rangle}$, with $s = (1+\beta+\epsilon)$ times as many processors, is an $\left(\frac{4s}{\beta\epsilon}\right)$-competitive algorithm for scheduling processes with sublinear nondecreasing speedup curves for the objective of average response time. Trivially the same result holds $\text{LAPS}_{\langle\beta,s\rangle}$ is given processors that are s times as fast.*

Essentially this shows, perhaps somewhat surprisingly, that a nonclairvoyant scheduling algorithm can perform roughly as well in the setting of scheduling jobs with arbitrary speedup curves on a multiprocessor as it can when scheduling jobs on a single processor. Our proof of Theorem 1.1 uses a simple amortized local competitiveness argument with a simple potential function. When $\beta = 1$, that is when $\text{LAPS}_{\langle\beta,s\rangle} = \text{Equi}_s$, we get as a corollary of Theorem 1.1 that Equi is $(2+\epsilon)$-processor $(\frac{2s}{\epsilon})$-competitive, matching the bound given in Edmonds [2000], but with a much easier proof.

Theorem 1.1 also improves the best known competitiveness result for broadcast/multicast pull scheduling. It is easiest to explain broadcast scheduling in context of a Web server serving static content. In this setting, it is assumed that the Web server is serving content on a broadcast channel. So if the Web server has multiple unsatisfied requests for the same file, it need only broadcast that file once, simultaneously satisfying all the users who issued these requests. Edmonds and Pruhs [2003] showed how to convert any s-speed c-competitive nonclairvoyant algorithm for scheduling jobs

with arbitrary speedup curves into a $2s$-speed c-competitive algorithm for broadcast scheduling. Using this result, and the analysis of Equi from Edmonds [2000], Edmonds and Pruhs [2003] showed that a version of Equi $(4+\epsilon)$-speed $O(1)$-competitive for broadcast scheduling with the objective of average response time. Using Theorem 1.1 we can then deduce that a broadcast version of $\text{LAPS}_{\langle\beta,s\rangle}$ is $(2+\epsilon)$-speed $O(1)$-competitive for broadcast scheduling with the objective of average response time.

1.2. Related Results

For the objective of total response time on a single processor, the competitive ratio of every deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $\Omega(\log n)$ [Motwani et al. 1994]. There is a randomized algorithm, Randomized Multi-Level Feedback Queues, that is $O(\log n)$-competitive against an oblivious adversary [Kalyanasundaram and Pruhs 2003; Becchetti and Leonardi 2004]. The online clairvoyant algorithm Shortest Remaining Processing time is optimal for total response time. The competitive analysis of SETF_s for single processor scheduling was improved for cases when the speed augmentation is large [Berman and Coulston 1999].

Variations of Equipartition are built into many technologies. For example, the congestion control protocol in the TCP Internet protocol essentially uses Equipartition to balance bandwidth to TCP connections through a bottleneck router. Extensions of the analysis of Equi in Edmonds [2000] to analyzing TCP can be found in Edmonds et al. [2003] and Edmonds [2004]. Other extensions to the analysis of Equi in Edmonds [2000] for related scheduling problems can found in Robert and Schabanel [2007b, 2008, 2007a]. In our results here, we essentially ignore the extra advantage that the online algorithm gains from having faster processors instead of more processors. Edmonds [2000] gives a better competitive ratio for Equi in the model with faster processors.

There are many related scheduling problems with other objectives, and/or other assumptions about the processor and job environment. Surveys can be found in Pruhs et al. [2004] and Pruhs [2007].

2. PRELIMINARIES

In this section, we review the formal definitions introduced in Edmonds [2000]. An instance consists of a collection $J = \{J_1, \ldots, J_n\}$ where *job* J_i has a *release/arrival time* r_i and a sequence of phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i}\rangle$. Each phase is an ordered pair $\langle w_i^q, \Gamma_i^q\rangle$, where w_i^q is a positive real number that denotes the amount of *work* in the phase and Γ_i^q is a function, called the *speedup function*, that maps a nonnegative real number to a nonnegative real number. $\Gamma_i^q(p)$ represents the rate at which work is processed for phase q of job J_i when run on p processors running at speed 1. If these processors are running at speed s, then work is processed at a rate of $s\Gamma_i^q(p)$.

A schedule specifies for each time, and for each job: (1) a nonnegative real number specifying the number of processors assigned to that job, and (2) a nonnegative real speed. The number of processors assigned at any time can be at most m, the number of processors. Note that, formally, a schedule does not specify an assignment of copies of jobs to processors.

A nonclairvoyant algorithm only knows when processes have been released and finished in the past, and which processes have been run on each processor each time in the past. In particular, a nonclairvoyant algorithm does not know w_i^q, nor the current phase q, nor the speedup function Γ_i^q.

The *completion time* of a job J_i, denoted C_i, is the first point of time when all the work of the job J_i has been processed. Note that in the language of scheduling, we are assuming that preemption is allowed, that is, a job maybe be suspended and later

restarted from the point of suspension. A job is said to be *alive* at time t, if it has been released, but has not completed, that is, $r_i \leq t \leq C_i$. The *response/flow time* of job J_i is $C_i - r_i$, which is the length of the time interval during which the job is active. Let n_t be the number of active jobs at time t. Another formulation of total flow time is $\int_0^\infty n_t dt$.

A phase of a job is *parallelizable* if its speedup function is $\Gamma(p) = p$. Increasing the number of processors allocated to a parallelizable phase by a factor of s increases the rate of processing by a factor of s. A phase is *sequential* if its speedup function is $\Gamma(p) = 1$, for all $p \geq 0$. The rate that work is processed in a sequential phase is independent of the number of processors, even if it is zero. A speedup function Γ is *nondecreasing* if and only if $\Gamma(p_1) \leq \Gamma(p_2)$ whenever $p_1 \leq p_2$. A speedup function Γ is *sublinear* if and only if $\Gamma(p_1)/p_1 \geq \Gamma(p_2)/p_2$ whenever $p_1 \leq p_2$. We assume all speedup functions Γ in the input instance are nondecreasing and sublinear.

Let A be an algorithm and J an instance. We denote the schedule output by A with speed s processors on J as $A_s(J)$. Let $\text{Opt}(J)$ be the optimal schedule with unit speed processors on input J. We let $F(S)$ denote the total response time incurred in schedule S.

3. ANALYSIS OF LATE ARRIVAL PROCESSOR SHARING

This section will be devoted to proving Theorem 1.1, that $\text{LAPS}_{\langle\beta,s\rangle}$ is scalable. We will assume that the online algorithm has sm unit speed processors while the adversary has m unit speed processors.

Following the lead of Edmonds [2000] and Robert and Schabanel [2008], the first step in our proof is to prove that there is a worst-case instance that contains only sequential and parallelizable phases.

LEMMA 3.1. *Let A be a nonclairvoyant scheduler. Let J be an instance of jobs with sublinear nondecreasing speedup functions. Then there is a job set J' that with only sequential and parallelizable phases such that $F(A(J')) = F(A(J))$ and $F(\text{Opt}(J')) \leq F(\text{Opt}(J))$.*

PROOF. We explain how to modify J to obtain J'. We perform the following modification for each time t and each job J_i that A runs during the infinitesimal time $[t, t+dt]$. Let dw be the infinitesimal amount of work processed by A during this time, and Γ the speedup function for the phase containing dw. Let p_a denote the number of processors allocated by A to dw at time t. So the amount of work in dw is $\Gamma(p_a)dt$. Let p_o denote the number of processors allocated by Opt to dw. It is important to note that Opt may not process dw at time t. If $p_0 \leq p_a$, we then modify J by replacing this dw amount of work with a sequential phase with work $dw' = dt$. If $p_o > p_a$, we then modify J by replacing this dw amount of work with parallelizable phase with work $dw' = p_a dt$. Note that by construction, A will not be able to distinguish between the instances J and J' during the time period $[t, t+dt]$. Hence, since A is nonclairvoyant $A(J') = A(J)$. We are now left to argue that $F(\text{Opt}(J')) \leq F(\text{Opt}(J))$. We will accomplish this by giving a schedule X for J' that has total response time at most $F(\text{Opt}(J))$.

First consider the case that $p_o \leq p_a$. Because the speedup function Γ of the phase containing the work dw is nondecreasing, it took $\text{Opt}(J)$ more than time dt to finish the work dw. The schedule X will start working on the work dw' with p_o processors when $\text{Opt}(J)$ started working on the work dw, and then after X completes dw', X can let these p_0 processors idle until $\text{Opt}(J)$ completes dw.

Now consider that case that $p_o \geq p_a$. Again the schedule X will start working on dw' when $\text{Opt}(J)$ started working on dw. We now want to argue that X can complete dw' with p_o processors in less time than it took $\text{Opt}(J)$ to complete dw with p_o processors. It took time $\frac{p_a dt}{p_o}$ time for X to complete dw' since the $p_a dt$ work in dw' is parallelizable.

It took $\mathrm{Opt}(J)$ time $\frac{\Gamma(p_a)dt}{\Gamma(p_o)}$ to complete the $\Gamma(p_a)dt$ work in dw. The fact X completes dw' before $\mathrm{Opt}(J)$ completes dw follows since $\frac{p_a}{p_o} \leq \frac{\Gamma(p_a)}{\Gamma(p_o)}$ since $p_o \geq p_a$ and Γ is sublinear. □

By Lemma 3.1, it is sufficient to consider instances that contain only sequential and parallelizable phases. So for the rest of the proof we fix such an instance. Our goal is to bound the number N_t of jobs alive under Opt at time t in terms of what is happening under $\mathrm{LAPS}_{\langle\beta,s\rangle}$ at this same time. This requires the introduction of a fair amount of notation. Let n_t denote number of jobs alive under $\mathrm{LAPS}_{\langle\beta,s\rangle}$ at time t. Let m_t denote the number of these that are within a parallelizable phase at this time and let ℓ_t denote the same except for sequential phases. Let N_t, M_t, and L_t denote the same numbers except under Opt. Let $\widehat{N}_t$ denote the number jobs at time t that $\mathrm{LAPS}_{\langle\beta,s\rangle}$ has not completed, but for which $\mathrm{LAPS}_{\langle\beta,s\rangle}$ is ahead of Opt. Let $\widehat{\ell}_t$ denote the number jobs that $\mathrm{LAPS}_{\langle\beta,s\rangle}$ has not completed at time t, and either $\mathrm{LAPS}_{\langle\beta,s\rangle}$ is ahead of Opt on this job at this time, or $\mathrm{LAPS}_{\langle\beta,s\rangle}$ is executing a sequential phase on this job at this time.

We note some relationships between these job counts. Clearly $\widehat{N}_t \leq N_t$ since Opt has not completed these $\widehat{N}_t$ jobs. $\int_0^\infty L_t dt = \int_0^\infty \ell_t dt$ since each integral is simply the sum of the work of all sequential phases of all jobs. Finally note that $\widehat{\ell}_t \leq \widehat{N}_t + \ell_t$ since each of the $\widehat{\ell}_t$ jobs is either in a sequential phase or is included in the count $\widehat{N}_t$. Thus we can conclude that the total cost to Opt is bounded as follows.

$$\begin{aligned} F\left(\mathrm{Opt}(J)\right) &= \int_0^\infty N_t dt \\ &= \frac{1}{2}\int_0^\infty (N_t + (M_t + L_t))dt \\ &\geq \frac{1}{2}\int_0^\infty (\widehat{N}_t + 0 + \ell_t)dt \\ &\geq \int_0^\infty \frac{\widehat{\ell}_t}{2} dt \end{aligned}$$

To prove c-competitiveness using an amortized local competitiveness argument we need to define a potential function Φ_t such that the following conditions hold:

Boundary. Φ is initially and finally 0, that is, $\Phi_0 = \Phi_\infty = 0$.
Arrival. Φ_t does not increase when a new job arrives.
Completion. Φ_t does not increase when either the online algorithm or the adversary complete a job.
Running. For all times t when no job arrives or is completed,

$$n_t + \frac{d\Phi_t}{dt} \leq \frac{c\widehat{\ell}_t}{2}. \tag{1}$$

By integrating the running condition over time, and using the boundary, arrival, and completion conditions, one can conclude that

$$\begin{aligned} F(\mathrm{LAPS}_{\langle\beta,s\rangle}) &= \int_0^\infty n_t dt \\ &= \int_0^\infty n_t dt + [\Phi_\infty - \Phi_0] \\ &\leq \int_0^\infty \left(n_t + \frac{d\Phi_t}{dt}\right) dt \end{aligned}$$

$$\leq \int_0^\infty \left(\frac{c\widehat{\ell}_t}{2}\right) dt$$
$$\leq c \cdot F(\text{Opt}).$$

For more information on amortized local competitiveness arguments, see Edmonds [2000], Pruhs [2007], and Pruhs et al. [2004].

We define the potential function Φ_t as follows. Let J_i denote the i^{th} of the n_t jobs currently alive under $\text{LAPS}_{\langle\beta,s\rangle}$ at time t, sorted by their arrival times r_i (breaking ties arbitrarily but consistently). So J_1 is the earliest arriving job, and J_{n_t} is the latest arriving job, among the jobs alive for $\text{LAPS}_{\langle\beta,s\rangle}$ at time t. Let x_i denote the amount of parallelizable work of J_i that has been completed by Opt before time t, but that was not completed by $\text{LAPS}_{\langle\beta,s\rangle}$ before time t. Let $\gamma = \frac{2}{\epsilon m}$. The potential function is then

$$\Phi_t = \gamma \sum_{i=1}^{n_t} i \cdot \max(x_i, 0). \tag{2}$$

The boundary conditions for Φ_t are trivially satisfied. If a new job J_j arrives, then the value of the potential function does not increase because $\text{LAPS}_{\langle\beta,s\rangle}$ will not be behind on that job (i.e., $x_j = 0$). If $\text{LAPS}_{\langle\beta,s\rangle}$ completes job J_j, then $j \max(x_j, 0) = 0$ since $x_j = 0$, and removing job J_j from the summation will not increase the coefficient i of any other job. Opt completing a job J_j has no effect on the potential function at all.

To establish inequality (1), consider an infinitesimal period of time $[t, t + dt]$ during which no jobs arrive or are completed by either Equi or Opt. Consider how much Φ_t can increase due Opt's processing during this period. Without loss of generality, Opt processes only parallelizable work. Opt processes this parallelizable work at rate at most m. This increases the sum of the x_i's for these jobs by a total of at most $m\,dt$. Opt can increase Φ_t the most by working only on the most recently arrived job because its coefficient is maximal. Since the most recently arrived job has coefficient n_t in Φ_t, the rate of increase in Φ_t due to Opt's processing is at most $\gamma m n_t$.

We now need to bound how much Φ_t must decrease due to $\text{LAPS}_{\langle\beta,s\rangle}$'s processing during the same infinitesimal period of time $[t, t+dt]$. The algorithm $\text{LAPS}_{\langle\beta,s\rangle}$ works on the $f_t = \lceil \beta n_t \rceil$ jobs with the latest arrival times. Ideally, for these jobs, the term $\max(x_i, 0)$ in the potential function decreases at a rate of $\frac{sm}{f_t}$. However, there are two possible reasons that this desired decrease will not occur. The first possible reason is that $\text{LAPS}_{\langle\beta,s\rangle}$ has processed one of these jobs more than Opt has at this time. For such jobs, $x_i \leq 0$ and hence $\max(x_i, 0)$ is already 0. The second possible reason is that the job is in a sequential phase under $\text{LAPS}_{\langle\beta,s\rangle}$ at this time. Because x_i measures only the work in parallelizable phases, any processing that $\text{LAPS}_{\langle\beta,s\rangle}$ does on a sequential phase does not decrease $\max(x_i, 0)$. Recall that we defined $\widehat{\ell}_t$ to be the number jobs that have at least one of these properties. In the worst case, these $\widehat{\ell}_t$ jobs are those that arrive the most recently. Let us for the moment assume that $\widehat{\ell}_t \leq f_t$. In this case, $\text{LAPS}_{\langle\beta,s\rangle}$ effectively decreases the term $\max(x_i, 0)$ only for the jobs with coefficients in the range $[n_t - f_t + 1, n_t - \widehat{\ell}_t]$. The value of $\max(x_i, 0)$ decreases for these jobs at a rate of $\frac{sm}{f_t}$. Hence, the decrease in Φ_t due to $\text{LAPS}_{\langle\beta,s\rangle}$'s processing is at least

$$\gamma \sum_{i=n_t-f_t+1}^{n_t-\widehat{\ell}_t} i \cdot \frac{dx_i}{dt}$$
$$= \gamma \sum_{i=n_t-f_t+1}^{n_t-\widehat{\ell}_t} i \cdot \left(-\frac{sm}{f_t}\right)$$

$$
\begin{aligned}
&= \frac{-sm\gamma}{2f_t}((n_t - \widehat{\ell}_t)(n_t - \widehat{\ell}_t + 1) - (n_t - f_t)(n_t - f_t + 1)) \\
&= \frac{sm\gamma}{2f_t}(2n_t\widehat{\ell}_t - \widehat{\ell}_t^2 + \widehat{\ell}_t - 2n_t f_t + f_t^2 - f_t) \\
&\leq \frac{sm\gamma}{2f_t}(2n_t\widehat{\ell}_t - 2n_t f_t + f_t^2 - f_t) \\
&\leq \frac{sm\gamma n_t\widehat{\ell}_t}{f_t} - sm\gamma n_t + \frac{sm\gamma f_t}{2} - \frac{sm\gamma}{2} \\
&= \frac{sm\gamma n_t\widehat{\ell}_t}{\lceil \beta n_t \rceil} - sm\gamma n_t + \frac{sm\gamma \lceil \beta n_t \rceil}{2} - \frac{sm\gamma}{2} \\
&\leq \frac{sm\gamma n_t\widehat{\ell}_t}{\beta n_t} - sm\gamma n_t + \frac{sm\gamma(\beta n_t + 1)}{2} - \frac{sm\gamma}{2} \\
&= \frac{sm\gamma\widehat{\ell}_t}{\beta} - sm\gamma n_t + \frac{sm\gamma\beta n_t}{2}.
\end{aligned}
$$

Substituting back our bounds on the decrease in Φ_t due to LAPS$_{\langle\beta,s\rangle}$'s processing, and the increase in Φ_t due to Opt's processing, back into (1), we get

$$
\begin{aligned}
n_t + \frac{d\Phi_t}{dt} &\leq n_t + \left((\gamma m n_t) + \left(\frac{sm\gamma\widehat{\ell}_t}{\beta} - sm\gamma n_t + \frac{sm\gamma\beta n_t}{2}\right)\right) \\
&= \left(1 + \gamma m - sm\gamma + \frac{sm\gamma\beta}{2}\right) n_t + \frac{sm\gamma\widehat{\ell}_t}{\beta} \\
&\leq \frac{sm\gamma\widehat{\ell}_t}{\beta} \\
&= \frac{2s\widehat{\ell}_t}{\beta\epsilon} \\
&= \frac{c \cdot \widehat{\ell}_t}{2}.
\end{aligned}
$$

The last inequality follows since by substituting in $\gamma = \frac{2}{\epsilon m}$ and $s = 1+\beta+\epsilon$

$$
1 + \gamma - s\gamma + \frac{s\gamma\beta}{2} = 1 + \frac{2}{\epsilon m} - 2\frac{1+\beta+\epsilon}{\epsilon m} + \frac{(1+\beta+\epsilon)\beta}{\epsilon m}
$$

which one can verify is not positive by multiplying through by ϵ, and collecting like terms.

Now consider that case in which $\widehat{\ell}_t \geq f_t$. In this case all of the $f_t = \lceil \beta n_t \rceil$ jobs being processed LAPS$_{\langle\beta,s\rangle}$ might be in sequential phases or have $\max(x_i, 0) = 0$ and hence LAPS$_{\langle\beta,s\rangle}$'s processing might not decrease Φ_t. Evaluating inequality (1), we find that

$$
\begin{aligned}
n_t + \frac{d\Phi_t}{dt} &\leq n_t + \gamma m n_t \\
&= \left(1 + \frac{2}{\epsilon}\right) n_t \\
&\leq \frac{2(1+\beta+\epsilon)}{\epsilon\beta} \lceil \beta n_t \rceil
\end{aligned}
$$

$$= \frac{2s}{\beta\epsilon} \cdot f_t$$
$$\leq \frac{c \cdot \widehat{\ell_t}}{2}.$$

4. CONCLUSION

The LAPS algorithm that we introduced in this article, has found application in several subsequent papers. In was used in Chan et al. [2009a] as the job selection algorithm in a $O(1)$-competitive speed scaling algorithm on a single processor with the objective of minimizing a linear combination of response time and energy. LAPS was used instead of the more obvious choice of SETF because the analysis of speed scaling algorithms generally requires amortized local competitiveness arguments, and it is not clear what potential function one should use with SETF. The potential function used in Chan et al. [2009a] is a modification of the potential function that we used here. A modification of LAPS was used in Chan et al. [2009b] as the job selection algorithm in a $O(\log m)$-competitive speed scaling algorithm on a multiprocessor processor with the objective of minimizing a linear combination of response time and energy. Finally Bansal et al. [2009] showed that the broadcast version of LAPS is scalable for broadcast scheduling, answering a decade old open question of whether such an algorithm exists. A scalable algorithm for broadcasting scheduling of unit work pages was given in Im and Moseley [2010].

Contemporaneously and subsequent to this research, other existentially scalable algorithms were discovered for broadcast scheduling [Im and Moseley 2010; Bansal et al. 2009; Chekuri et al. 2009; Chekuri and Moseley 2009]. It is a very interesting open question whether there is a universally scalable algorithm for the problem considered in this article, and for broadcast scheduling. We conjecture, at least for the problem considered in this article, that a universally scalable algorithm does not exist, although it is not at all clear how to prove this.

ACKNOWLEDGMENT

We thank Nicolas Schabanel and Julien Robert for helpful discussions.

REFERENCES

BANSAL, N., KRISHNASWAMY, R., AND NAGARAJAN, V. 2009. Better scalable algorithms for broadcast scheduling. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming (ICALP'09)*.

BECCHETTI, L. AND LEONARDI, S. 2004. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM 51,* 4, 517–539.

BERMAN, P. AND COULSTON, C. 1999. Speed is more powerful than clairvoyance. *Nordic J. Comput. 6,* 2, 181–193.

CHAN, H.-L., EDMONDS, J., LAM, T. W., LEE, L.-K., MARCHETTI-SPACCAMELA, A., AND PRUHS, K. 2009a. Nonclairvoyant speed scaling for flow and energy. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*. 255–264.

CHAN, H.-L., EDMONDS, J., AND PRUHS, K. 2009b. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*. 1–10.

CHEKURI, C., IM, S., AND MOSELEY, B. 2009. Minimizing maximum response time and delay factor in broadcast scheduling. In *Proceedings of the European Symposium on Algorithms*.

CHEKURI, C. AND MOSELEY, B. 2009. Online scheduling to minimize the maximum delay factor. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 1116–1125.

EDMONDS, J. 2000. Scheduling in the dark. *Theor. Comput. Sci. 235*, 109–141.

EDMONDS, J. 2004. On the competitiveness of AIMD-TCP within a general network. In *Proceedings of the Latin American Symposium on Theoretical Informatics*. 567–576.

EDMONDS, J., DATTA, S., AND DYMOND, P. 2003. TCP is competitive against a limited adversary. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*. 174–183.

EDMONDS, J. AND PRUHS, K. 2003. Multicast pull scheduling: When fairness is fine. *Algorithmica 36,* 3, 315–330.

IM, S. AND MOSELEY, B. 2010. An online scalable algorithm for average flowtime in broadcast scheduling. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*.

KALYANASUNDARAM, B. AND PRUHS, K. 2000. Speed is as powerful as clairvoyance. *J. ACM 47,* 4, 617–643.

KALYANASUNDARAM, B. AND PRUHS, K. 2003. Minimizing flow time nonclairvoyantly. *J. ACM 50,* 4, 551–567.

MERRITT, R. 2008. CPU designers debate multi-core future. *EE Times*.

MOTWANI, R., PHILLIPS, S., AND TORNG, E. 1994. Non-Clairvoyant scheduling. *Theor. Comput. Sci. 130*, 17–47.

PHILLIPS, C., STEIN, C., TORNG, E., AND WEIN, J. 2002. Optimal time-critical scheduling via resource augmentation. *Algorithmica 32,* 2, 163–200.

PRUHS, K. 2007. Competitive online scheduling for server systems. *SIGMETRICS Perform. Eval. Rev. 34,* 4, 52–58.

PRUHS, K., SGALL, J., AND TORNG, E. 2004. Online scheduling. In *Handbook on Scheduling*, CRC Press.

ROBERT, J. AND SCHABANEL, N. 2007a. Non-Clairvoyant batch sets scheduling: Fairness is fair enough. In *Proceedings of the European Symposium on Algorithms*. 741–753.

ROBERT, J. AND SCHABANEL, N. 2007b. Pull-based data broadcast with dependencies: be fair to users, not to items. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*.

ROBERT, J. AND SCHABANEL, N. 2008. Non-clairvoyant scheduling with precedence constraints. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 491–500.

Received September 2009; revised May 2010; accepted October 2011