Shortest-Elapsed-Time-First on a Multiprocessor

Neal Barcelo[1], Sungjin Im[2], Benjamin Moseley[2], and Kirk Pruhs[1]

[1] Department of Computer Science, University of Pittsburgh
ncb30,kirk@cs.pitt.edu
[2] Computer Science Department, University of Illinois im3,bmosele2@illinois.edu

"I would like to call it a corollary of Moore's Law that the number of cores will double every 18 months." — Anant Agarwal, founder and chief technology officer of MIT startup Tilera

Abstract. We show that SETF, the idealized version of the uniprocessor scheduling algorithm used by Unix, is scalable for the objective of fractional flow on a homogeneous multiprocessor. We also give a potential function analysis for the objective of weighted fractional flow on a uniprocessor.

1 Introduction

At the hardware level, Moore's law continues unabated, with the number of transistors per chip doubling about every 1.5 to 2 years. However, we are in the midst of a revolutionary change in the effect of Moore's law on the software layers of the information technology stack. Instead of an exponential increase in processor speed over time, these layers are now expected to see an exponential increase in the number of processors over time. MIT startup Tilera now produces chips with up to 100 processors, and the expectation is that chips with 1000 processors will be available within the decade.

The natural research question motivating our research is whether the standard priority scheduling algorithms used for uniprocessors will be appropriate in the multiprocessor setting. In particular, we consider scheduling algorithm Shortest Elapsed Time First (SETF), as it is the idealized version of Unix's uniprocessor scheduling algorithm. (Of course the implementation in Unix has many practical kludges/modifications, such as maintaining equivalence queues of jobs that have been processed about the same amount so as to logarithmically bound the number of preemptions per job). For a uniprocessor using SETF for scheduling, all jobs that have been processed the least share the processing equally; it is useful to think of SETF giving higher priority to jobs that have been processed less. The natural generalization of SETF to a homogeneous multiprocessor setting assigns jobs to processors in priority order (recall jobs that have been processed less have higher priority); the x jobs of the next priority are either assigned to x processors, or evenly share the remaining unassigned processors if there are less than x previously unassigned processors.

Two natural quality of service measures for individual jobs are integer flow and fractional flow. The integer flow of a job is the total time a job has to wait to be completed. The fractional flow of a job is the integral over times between when a job arrives and when it is completed of the fraction of the job that is uncompleted. Integer flow is a more appropriate objective if no benefit is gained from a job being partially completed, and fractional flow is a more appropriate objective if some benefit is gained from partially completing a job. The corresponding two natural scheduling objectives are the integer flow of the schedule, which is the sum of the integer flow of the jobs, and fractional flow of the schedule, which is the sum of the fractional flow of the jobs.

On a uniprocessor, SETF is known to be scalable, $(1+\epsilon)$-speed $O(1)$-competitive, for the standard objective of integer flow [1], and it is known that speed augmentation is required to achieve bounded competitiveness in a general operating system setting requiring a nonclairvoyant scheduler, that is one that does not know the size of the jobs [2]. To the best of our knowledge, there are no results in the literature explicitly analyzing the fractional flow for nonclairvoyant algorithms on a uniprocessor (although it is possible that such results might be derivable from results on integer flow).

The main result of this paper is to show in Section 2 that for a homogeneous multiprocessor, SETF is universally[3] scalable for the objective of fractional flow.

The analysis in [1] shows that SETF is locally competitive for integer flow on a uniprocessor, that is, at all points in time the increase for the quality of service objective for SETF is not too much greater than the increase for an arbitrary schedule. But it is straightforward to see that no online scheduling algorithm can be locally competitive for either fractional or integer flow on a homogeneous multiprocessor. Thus the next logical approach is to try to use an amortized local competitiveness argument using the so called "standard potential function" method for these sorts of scheduling problems (for more background, see [3]). However, this standard approach is not immediately applicable in this setting as this approach requires a reasonably simple algebraic expression for the online algorithm's future cost given no more job arrivals, and after some thought, one can see that a simple algebraic expression does not exist for SETF's future costs on a multiprocessor. For fractional flow, we are able to surmount this difficultly by using a potential function based on an algebraic expression for SETF's future costs on a uniprocessor. The primary differences between our potential function and the "standard potential function" are that it takes the difference of future costs between the work remaining in the optimal schedule and the online algorithm instead of the future cost of the difference in remaining work, and additionally our potential function discounts the optimal's future costs. These modifications are necessary to get the running condition to hold; however, these modifications cause the potential function to jump when jobs arrive. Fortunately, we are able to complete the analysis by showing that the aggregate increase in these jumps

[3] An algorithm is said to be universally scalable if it is $(1+\epsilon)$-speed $O(f(\epsilon))$ competitive for any fixed constant $\epsilon > 0$ and the algorithm is not parameterized by ϵ. Here f is a function of only ϵ.

can be bounded by total processing times of all the jobs. Unfortunately we are unable to make this approach work for integer flow for SETF. Although one can resort to a technique to convert an algorithm that is fractionally scalable to an algorithm that is integrally scalable (see [4] for details). This technique combined with our analysis shows that a variation of SETF is scalable for integer flow.

There are two closely related results in the literature. It was known that if newly arriving jobs were randomly assigned to a processor, and if each processor ran SETF, that the resulting algorithm is universally scalable in expectation for integer flow [5]. Roughly this analysis combines the fact that SETF is universally scalable on a uniprocessor, with the fact that randomly assigning jobs roughly balances the processor loads (although the fact that there will be some unevenness in the loads in part explains why the competitive ratio that is proved is quite large, something like $O(\frac{1}{\epsilon^7})$). It is also known that the algorithm Late Arrival Processor Sharing (LAPS) is existentially[4] scalable for integer flow on a homogeneous multiprocessor [6].

There are often situations where one would like the operating system to view some jobs as being more important than other jobs. One way to formalize this is to assume that jobs have weights specifying their importance, and then consider the objective of minimizing the weighted fractional or integral flow of the jobs. WSETF is a natural generalization of SETF, where jobs are prioritized by the ratio of their weight to the time that jobs have been processed. It was shown in [7] that WSETF is scalable for a uniprocessor using a local competitiveness argument. In Section 3, we show that WSETF is scalable using an amortized local competitiveness argument using a potential function. As in our analysis of SETF, the starting point for the design of the potential function was an algebraic expression for the future cost of WSETF. However, we again had to make modifications to the "standard potential function" in order for the running condition to hold. We believe that our analysis is at least modestly interesting for a couple reasons. When one is analyzing algorithms in non-work-conserving scheduling settings, there is usually no hope of using a local competitiveness argument. In this context, a scheduling environment is said to be non-work-conserving if at any given time two reasonable scheduling algorithms could have completed a different amount of total work thus far. The lack of a potential function analysis of SETF and WSETF meant that these algorithms could not be used to design algorithms in non-work-conserving scheduling settings. For example, the analysis of nonclairvoyant speed scaling algorithms for a speed scalable processor in [8] considered the Late Arrival Processor Sharing Algorithm (LAPS) instead of SETF because a potential function analysis was known for LAPS [9]. It is our hope that our potential functions for SETF and WSETF will be useful in other non-work-conserving scheduling settings. Although in fairness we need to mention that we were unable to adapt our potential function analysis of WSETF for a uniprocessor to the multiprocessor setting because we do not

[4] An algorithm is said to be existentially scalable if it is $(1+\epsilon)$-speed $O(f(\epsilon))$ competitive for any fixed constant $\epsilon > 0$ and the algorithm is parameterized by ϵ. Here f is a function of only ϵ.

know how to bound the aggregate increases in the potential function when jobs arrive. But it is our hope that that one further idea would be enough to surmount this issue, and allow the application of this potential function (or some variation thereof) to non-work-conserving scheduling settings. Note that an existentially scalable algorithm, Weighted Late Arrival Processor Sharing, is known for the objective of integer flow on a homogeneous multiprocessor [10].

There is currently a debate within the architectural community as to whether a homogeneous multiprocessor or a heterogeneous multiprocessor is a better design [11]. There are advantages to each option. [12] points out that some standard priority scheduling algorithms, such as Highest Density First and WSETF, are not scalable for a heterogeneous multiprocessor, and that it is not clear whether other standard priority algorithms, such as Shortest Remaining Processing Time, Shortest Job First, and SETF, are scalable. So while this paper certainly does not settle the issue, taken together with [12], the results in this paper indicate that one advantage of homogeneous multiprocessors over heterogeneous multiprocessors is that they seem to be easier to schedule, and that in fact the standard uniprocessor scheduling algorithms should perform similarly well on a homogeneous multiprocessor as on a uniprocessor.

1.1 Basic Definitions

The input consists of n jobs. We let r_i denote the release time of job i, p_i denote the size of job i, and in some instances, w_i denote the weight of job i. An online scheduler does not learn about job i until time r_i. At time r_i, a nonclairvoyant scheduler learns the weight w_i but not the size p_i. For each time t, the online algorithm must choose some job i to run such that $r_i \geq t$. We assume that the processor has unit speed, so a job of size p_i, takes p_i units of time to complete. If C_i is the completion time for job i, then $\int_{t=r_i}^{C_i} w_i \, dt$ is the weighted integer flow for job i. The integer flow of a schedule is the sum over the jobs of the integer flow of each job. The weighted fractional flow of job i, is $\int_{t=r_i}^{\infty} w_i \cdot \frac{p_i(t)}{p_i} \, dt$, where $p_i(t)$ represents the remaining processing time of job i. The fractional flow of a schedule is the sum over the jobs of the fractional flow of each job. If the schedule is not obvious from context, we superscript a variable with the name of the schedule that is referred to.

An algorithm A is s-speed c-competitive if

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

where $A_s(I)$ denotes the cost of algorithm A on input I with a speed s processor, $\mathrm{OPT}_1(I)$ denotes the cost of the optimal schedule with a speed 1 processor, and the maximum is taken over all possible inputs. A class $\{A_{(1+\epsilon)}\}$ of algorithms is existentially scalable if for all $\epsilon > 0$, $A_{(1+\epsilon)}$ is $(1+\epsilon)$-speed $O(f(\epsilon))$-competitive for some function f that only depends on ϵ. An algorithm A is universally scalable if for all $\epsilon > 0$, A is $(1+\epsilon)$-speed $O(f(\epsilon))$-competitive for some function f.

To show that an algorithm A is $(c+d)$-competitive using a locally amortized competitiveness argument, one finds a potential function Φ such that the following conditions hold [3]:

Boundary condition: Φ is initially 0 and finally non-negative.
Completion condition: Φ does not increase due to completion of jobs by A or OPT.
Arrival condition: Φ does not increase by more than $d \cdot \mathrm{OPT}$ due to arrival of jobs.
Running condition: At all times t when no job arrives or is completed, we have,

$$\frac{d}{dt}A + \frac{d}{dt}\Phi(t) \leq c\frac{d}{dt}\mathrm{OPT}$$

Here $\frac{d}{dt}A$ denotes the increase in the objective in A's schedule, while $\frac{d}{dt}\mathrm{OPT}$ denotes the increase in the objective in OPT's schedule. $(c+d)$-competitiveness follows by integrating these conditions over time.

2 SETF on a Homogeneous Multiprocessor

As our first result, we show in Theorem 1 that SETF is universally scalable on a homogeneous multiprocessor for the objective of fractional flow using an amortized local competitiveness argument.

Theorem 1. SETF *is* $(1+\epsilon)$*-speed* $(1+\frac{5}{\epsilon})$*-competitive on a homogeneous multiprocessor for the objective of fractional flow.*

Proof. We use A to denote SETF. Let m denote the number of homogeneous mutliprocessors. We let $q_j^A(t)$ denote the amount of job j that has been processed up to time t. Note that $q_j^A(t) + p_j^A(t) = p_j$. Let, $(x)^+$ return x when x is positive, and 0 otherwise. Then, define $p_{i,j}^A(t) := (\min(p_i, p_j) - q_j^A(t))^+$. This represents the amount of time job i must wait on job j assuming no more jobs arrive. Note that it is possible that $i = j$. Similarly for OPT, $p_{i,j}^O(t) := (\min(p_i, p_j) - q_j^O(t))^+$. We let $Q_A(t)$ and $Q_O(t)$ denote the algorithm A's queue and OPT's queue, at time t respectively. Finally, let $Z_i^A(t) := \sum_{j \in Q_A(t)} p_{i,j}^A(t)$. Similarly, $Z_i^O(t) := \sum_{j \in Q_O(t)} p_{i,j}^O(t)$. We use an amortized local competitiveness argument. We define the potential function $\Phi(t)$ as follows.

$$\begin{aligned}
\Phi(t) &= \frac{1}{m\epsilon} \sum_{i \in Q_A(t)} \frac{p_i^A(t)}{p_i} \left(Z_i^A(t) + m p_i^A(t) - Z_i^O(t) \right) \\
&= \frac{1}{m\epsilon} \sum_{i \in Q_A(t)} \frac{p_i^A(t)}{p_i} \Big(\sum_{j \in Q_A(t)} (\min(p_i, p_j) - q_j^A(t))^+ + m p_i^A(t) \\
&\qquad - \sum_{j \in Q_O(t)} (\min(p_i, p_j) - q_j^O(t))^+ \Big)
\end{aligned}$$

Boundary condition: The boundary condition is trivially satisfied, as there are no jobs contributing to Φ at $t = 0$ or when all jobs have been finished.

Job completion: Fix some job $i \in Q_A(t)$. Consider first when A completes job i. Note that at this time, $p_i^A(t) = 0$ and therefore there is no change in Φ from removing this term from the sum. Next, consider when A completes some job $j \neq i$. Since, $q_j^A(t) = p_j$, $p_{i,j}^A(t) = 0$, so there is no change in Φ from removing this term. Similarly, the completion of a job by OPT does not change Φ.

Job arrival: We first show the following lemma.

Lemma 1. *Consider any job $i \in Q_A(t)$ and time t. Then it is the case that $Z_i^A(t) - Z_i^O(t) \leq mp_i$.*

Proof. Fix time t. Let $J(t)$ denote the set of all jobs in A's queue that have been processed less than job i's total processing time. More formally, we have $J(t) = \{j \in Q_A(t) \mid q_j^A(t) < p_i\}$. If $|J(t)| \leq m$, then there are at most m terms contributing to $Z_i^A(t)$ each of which have value at most p_i and so the desired result holds. So suppose $|J(t)| > m$. Consider the earliest time $t' \leq t$ such that at any time $\tau \in [t', t]$, $|J(\tau)| > m$. By definition of t', at time $t' - \delta$, there are at most m jobs that have elapsed processing times at most p_i. Now consider all jobs, denoted by S, which arrive during $[t', t]$. Note that for any time $\tau \in [t', t]$, for any job j that is run, $q_j^A(\tau) < p_i$ since $|J(\tau)| > m$. Therefore, $J(t) \subseteq J(t' - \delta) \cup S$. Consider $J(t)$'s contribution to $Z_i^A(t) - Z_i^O(t)$ at time t. Let $t'' = t' - \delta$.

$$\begin{aligned}
&\sum_{j \in J(t)} (\min(p_i, p_j) - q_j^A(t))^+ - (\min(p_i, p_j) - q_j^O(t))^+ \\
&\leq \sum_{j \in J(t)} (\min(p_i, p_j) - q_j^A(t)) - (\min(p_i, p_j) - q_j^O(t)) && (1)\\
&= \sum_{j \in J(t)} (q_j^O(t) - q_j^A(t)) \\
&\leq \sum_{j \in J(t'')} (q_j^O(t'') - q_j^A(t'')) + \sum_{j \in J(t'')} (q_j^O(t) - q_j^O(t'')) - (q_j^A(t) - q_j^A(t'')) \\
&\quad + \sum_{j \in S} (q_j^O(t) - q_j^A(t)) && (2)\\
&\leq mp_i && (3)
\end{aligned}$$

Inequality (1) holds as based on the definition of $J(t)$ the first term in the sum will always be positive. (2) holds by noting that $J(t) = J(t') \cup S$ and rearranging terms while letting $\delta \to 0$. Finally, (3) is true because the first sum is less than mp_i as there are at most m terms of value p_i. Further, $\sum_{j \in J(t'')} (q_j^A(t) - q_j^A(t'')) + \sum_{j \in S} q_j^A(t)$ represents the total work that SETF did during this interval and $\sum_{j \in J(t'')} (q_j^O(t) - q_j^O(t'')) + \sum_{j \in S} q_j^O(t)$ cannot be more than the work that OPT did during this interval, therefore their difference is non-positive.

Given this lemma, note that when job i arrives, $\varPhi$ increases by at most $\frac{2}{\epsilon}p_i$ and so summing over all arrivals, the increase is at most $\frac{4}{\epsilon}\mathrm{OPT}$ since $p_i/2$ is a lower bound for job i's fractional flow time in any schedule.

Running Condition: First note that $\frac{d}{dt}A = \sum_{i\in Q_A(t)} \frac{p_i^A(t)}{p_i}$. Also, $\frac{d}{dt}\mathrm{OPT} = \sum_{i\in Q_O(t)} \frac{p_i^O(t)}{p_i}$. We now bound the change in $\varPhi$ at some time t when no jobs arrive or complete. We have that,

$$\frac{d}{dt}\varPhi(t) = \frac{1}{m\epsilon}\sum_{i\in Q_A(t)}\Big(\frac{d\frac{p_i^A(t)}{p_i}}{dt}\cdot(Z_i^A(t)+mp_i^A(t)-Z_i^O(t)) \\ + \frac{p_i^A(t)}{p_i}\cdot\frac{d(Z_i^A(t)+mp_i^A(t)-Z_i^O(t))}{dt}\Big)$$

First consider the change of $\frac{p_i^A(t)}{p_i}$. This occurs only when job i is being processed by SETF. Since SETF runs at speed $(1+\epsilon)$, $\frac{p_i^A(t)}{p_i}$ is decreasing at a rate of $(1+\epsilon)\frac{1}{p_i}$. To bound the overall rate of increase in $\varPhi$ this can have, we ignore the positive terms $Z_i^A(t)$ and $mp_i^A(t)$ and consider only $-Z_i^O(t)$. Then, the rate of increase in $\varPhi$ due to change in $\frac{p_i^A(t)}{p_i}$ is bounded by

$$\frac{1}{m\epsilon}(1+\epsilon)\frac{1}{p_i}\sum_{j\in Q_O(t)}(\min(p_i,p_j)-q_j^O(t))^+$$

To bound this sum, there are two cases to consider. First, consider all jobs j such that $p_j < p_i$. Then, we have that

$$\frac{1}{p_i}(\min(p_i,p_j)-q_j^O(t))^+ = \frac{1}{p_i}p_j^O(t) \le \frac{p_j^O(t)}{p_j}$$

For all jobs j such that $p_j \ge p_i$, we have that

$$\frac{1}{p_i}(\min(p_i,p_j)-q_j^O(t))^+ = \left(\frac{p_i-q_j^O(t)}{p_i}\right)^+ = \left(1-\frac{q_j^O(t)}{p_i}\right)^+ \le \frac{p_j^O(t)}{p_j}$$

So, in total we have that

$$\begin{aligned}&\frac{1}{m\epsilon}(1+\epsilon)\frac{1}{p_i}\sum_{j\in Q_O(t)}(\min(p_i,p_j)-q_j^O(t))^+\\ &\le \frac{1}{m\epsilon}(1+\epsilon)\sum_{j\in Q_O(t)}\frac{p_j^O(t)}{p_j}\\ &= \frac{1+\epsilon}{m\epsilon}\frac{d}{dt}\mathrm{OPT}\end{aligned}$$

Since there are at most m such jobs as i running, the total rate of increase in Φ due to change in $\frac{p_i^A(t)}{p_i}$ is bounded by $(1+\frac{1}{\epsilon})\frac{d}{dt}\mathrm{OPT}$.

We now turn our attention to the change of $(Z_i^A(t)+mp_i^A(t)-Z_i^O(t))$ for any job $i \in Q_A(t)$. Note that $p_i^A(t) > 0$, i.e. $q_i^A(t) < p_i$. If SETF is working on job i, then $mp_i^A(t)$ decreases at a rate of $m(1+\epsilon)$. Otherwise, if SETF does not work on i at time t, then there must exist m jobs j such that $q_j^A(t) < p_i$ that SETF is working on. In either case, $Z_i^A(t)+mp_i^A(t)$ decreases at a rate of $m(1+\epsilon)$. On the other hand, $Z_i^O(t)$ can increase at a rate of at most m. Therefore, the rate of change of Φ due to change in $(Z_i^A(t)+mp_i^A(t)-Z_i^O(t))$ is bounded by,

$$\frac{1}{m\epsilon}\sum_{i\in Q_A(t)}\frac{p_i^A(t)}{p_i}(-m(1+\epsilon)+m) = -\sum_{i\in Q_A(t)}\frac{p_i^A(t)}{p_i} = -\frac{d}{dt}A$$

So, in total, we have that

$$\frac{d}{dt}A + \frac{d}{dt}\Phi(t) \le \frac{d}{dt}A + \left(1+\frac{1}{\epsilon}\right)\frac{d}{dt}\mathrm{OPT} - \frac{d}{dt}A = \left(1+\frac{1}{\epsilon}\right)\frac{d}{dt}\mathrm{OPT}$$

We note that one can achieve an existentially scalable nonclairvoyant algorithm for integer flow by maintaining the invariant that each job is either done or has processed $(1+\epsilon)$ times as much as SETF would have processed it on $(1+\epsilon)$ slower processors.

3 WSETF on a Uniprocessor

We now show that WSETF is scalable on a single processor for the objective of weighted fractional flow. Recall the WSETF shares the processor equally among all jobs that have maximal ratio between weight and the amount that the job has been processed.

Theorem 2. WSETF *is* $(1+\epsilon)$*-speed* $(1+\frac{3}{\epsilon})$*-competitive on a uniprocessor for the objective of weighted fractional flow.*

Proof. We use A to denote the algorithm WSETF. Let $q_j^A(t)$ denote the amount of job j that has been processed up to time t. Let $p_{i,j}^A(t) := (\min(\frac{w_j}{w_i}p_i, p_j) - q_j^A(t))^+$ and $p_{i,j}^O(t) := (\min(\frac{w_j}{w_i}p_i, p_j) - q_j^O(t))^+$. We again use an amortized local competitiveness argument. Consider the following potential function $\Phi(t)$.

$$\begin{aligned}\Phi(t) &= \frac{1}{\epsilon}\sum_{i\in Q_A(t)}\frac{w_i\cdot p_i^A(t)}{p_i}(Z_i^A(t)+p_i^A(t)-Z_i^O(t))\\ &= \frac{1}{\epsilon}\sum_{i\in Q_A(t)}\frac{w_i\cdot p_i^A(t)}{p_i}\Big(\sum_{j\in Q_A(t)}(\min(\frac{w_j}{w_i}p_i,p_j)-q_j^A(t))^+ + p_i^A(t)\\ &\qquad\qquad - \sum_{j\in Q_O(t)}(\min(\frac{w_j}{w_i}p_i,p_j)-q_j^O(t))^+\Big)\end{aligned}$$

It is worth noting that $\sum_{i \in Q_A(t)} \frac{w_i \cdot p_i^A(t)}{p_i}(Z_i^A(t) + p_i^A(t))$ represents the approximate future cost of WSETF assuming no more jobs arrive. We now verify that all four conditions hold.

Boundary condition: The boundary condition is trivially satisfied, as there are no jobs contributing to Φ at $t = 0$ or when all jobs have been finished.

Job completion: Consider first when A completes job i. Note that at this time, $p_i^A(t) = 0$ and therefore there is no change in Φ from removing this term from the sum. Next, consider when A completes job j. Since, $q_j^A(t) = p_j$, $p_{i,j}^A(t) = 0$, so there is no change in Φ from removing this term. Similarly, the completion of job i or job j by OPT does not change Φ.

Job arrival: We first show the following lemma.

Lemma 2. *Consider any job $i \in A(t)$ and time t. Then it is the case that $Z_i^A(t) - Z_i^O(t) \leq 0$.*

Proof. Fix time t. Consider the earliest time $t' \leq t$ such that at any time $\tau \in [t', t] = I$, WSETF works only on jobs j such that $q_j(\tau) \leq \frac{w_j}{w_i} p_i$. By definition of t', at time $t' - \epsilon$, all unfinished jobs have elapsed processing times at least $\frac{w_j}{w_i} p_i$, which thus contribute zero to $Z_i^A(t)$, so we can ignore those jobs. Now consider all jobs, denoted by S, which arrive during $[t', t]$. Consider S's contribution to $Z_i^A(t) - Z_i^O(t)$ at time t,

$$
\begin{aligned}
&\sum_{j \in S} (\min(\frac{w_j}{w_i} p_i, p_j) - q_j^A(t))^+ - (\min(\frac{w_j}{w_i} p_i, p_j) - q_j^O(t))^+ \\
\leq &\sum_{j \in S} (\min(\frac{w_j}{w_i} p_i, p_j) - q_j^A(t)) - (\min(\frac{w_j}{w_i} p_i, p_j) - q_j^O(t)) \qquad (4) \\
= &\sum_{j \in S} q_j^O(t) - q_j^A(t) \qquad (5)
\end{aligned}
$$

Note that (4) holds as based on the definition of S and I, for any job $j \in S$, $q_j^A(t) \leq \frac{w_j}{w_i} p_i$. Now consider the term in (5), $\sum_{j \in S} q_j^O(t) - q_j^A(t)$. First note that $\sum_{j \in S} q_j^A(t)$ captures the total work that WSETF did during the interval, and further $\sum_{j \in S} q_j^O(t)$ cannot exceed the amount of work that OPT did during the same interval. Therefore, this term is non-positive.

Given this lemma, note that when job i arrives, Φ increases by at most $\frac{1}{\epsilon}(w_i \cdot p_i)$. So, summing over all arrivals, Φ increases by at most $\frac{1}{\epsilon} \sum_i w_i \cdot p_i \leq \frac{2}{\epsilon}\mathrm{OPT}$ as desired.

Running Condition: First note that $\frac{d}{dt} A = \sum_{i \in Q_A(t)} w_i \cdot \frac{p_i^A(t)}{p_i}$. Also, $\frac{d}{dt}\mathrm{OPT} = \sum_{i \in Q_O(t)} w_i \cdot \frac{p_i^O(t)}{p_i}$. We now bound the change in Φ at some time t when no job arrives or is completed. We have that,

$$\frac{d}{dt}\Phi(t) = \frac{1}{\epsilon}\sum_{i\in Q_A(t)}\Big(\frac{d\frac{w_i p_i^A(t)}{p_i}}{dt}\cdot(Z_i^A(t)+p_i^A(t)-Z_i^O(t)) + \frac{w_i p_i^A(t)}{p_i}\cdot\frac{d(Z_i^A(t)+p_i^A(t)-Z_i^O(t))}{dt}\Big) \tag{6}$$

First consider the change of $\frac{w_i p_i^A(t)}{p_i}$. This occurs only when job i is being processed by A. We assume without loss of generality that A works on a single job at each time t. Then, since WSETF runs at speed $(1+\epsilon)$, $\frac{w_i p_i^A(t)}{p_i}$ is decreasing at a rate of $(1+\epsilon)\frac{w_i}{p_i}$. To bound the overall rate of increase in Φ this can have, we ignore the positive terms $Z_i^A(t)$ and $p_i^A(t)$ and consider only $-Z_i^O(t)$. Then, the rate of increase in Φ due to change in $\frac{w_i p_i^A(t)}{p_i}$ is bounded above by

$$\frac{1}{\epsilon}(1+\epsilon)\frac{w_i}{p_i}Z_i^O(t) = \left(1+\frac{1}{\epsilon}\right)\frac{w_i}{p_i}\sum_{j\in Q_O(t)}\left(\min(\frac{w_j}{w_i}p_i, p_j) - q_j^O(t)\right)^+$$

To bound this sum, we again consider two cases. First, consider all jobs j such that $\frac{w_j}{p_j} > \frac{w_i}{p_i}$. Then,

$$\frac{w_i}{p_i}(\min(\frac{w_j}{w_i}p_i, p_j) - q_j^O(t))^+ = \frac{w_i}{p_i}(p_j - q_j^O(t)) \le w_j\cdot\frac{p_j^O(t)}{p_j}$$

Now, for all jobs j such that $\frac{w_j}{p_j} \le \frac{w_i}{p_i}$, we have that

$$\frac{w_i}{p_i}(\min(\frac{w_j}{w_i}p_i, p_j) - q_j^O(t))^+ = \frac{w_i}{p_i}(\frac{w_j}{w_i}p_i - q_j^O(t))^+ = (w_j - \frac{w_i}{p_i}q_j^O(t))^+ \le w_j\cdot\frac{p_j^O(t)}{p_j}$$

Combining these, we have that

$$\left(1+\frac{1}{\epsilon}\right)\frac{w_i}{p_i}\sum_{j\in Q_O(t)}\left(\min(\frac{w_j}{w_i}p_i, p_j) - q_j^O(t)\right)^+ \le \left(1+\frac{1}{\epsilon}\right)\frac{d}{dt}\mathrm{OPT}$$

We now turn our attention to the change of $(Z_i^A(t)+p_i^A(t)-Z_i^O(t))$ for any job $i\in Q_A(t)$. Note that $p_i^A(t) > 0$, i.e. $q_i^A(t) < p_i$. Thus if WSETF does not work on i at time t, then there must exist a job j such that $\frac{q_j^A(t)}{w_j} < \frac{p_i}{w_i}$ that WSETF is working on. In either case, $Z_i^A(t)+p_i^A(t)$ decreases at a rate of $1+\epsilon$. On the other hand, $Z_i^O(t)$ can increase at a rate of at most 1. Therefore, the rate of change in Φ due to change in $(Z_i^A(t)+p_i^A(t)-Z_i^O(t))$ is bounded above by

$$\frac{1}{\epsilon}\sum_{i\in Q_A(t)}\frac{w_i p_i^A(t)}{p_i}(-(1+\epsilon)+1) = -\sum_{i\in Q_A(t)}\frac{w_i p_i^A(t)}{p_i} = -\frac{d}{dt}A$$

So, in total, we have that

$$\frac{d}{dt}A + \frac{d}{dt}\Phi(t) \leq \frac{d}{dt}A + \left(1 + \frac{1}{\epsilon}\right)\frac{d}{dt}\mathrm{OPT} - \frac{d}{dt}A = \left(1 + \frac{1}{\epsilon}\right)\frac{d}{dt}\mathrm{OPT}$$

Our analysis of WSETF does not extend to a homogeneous multiprocessor because we do not know how to bound the jumps in the potential function when jobs arrive, in part because when a job i arrives the increase in the potential involves terms of the form $p_i w_j$. We are able to surmount this difficulty in our analysis of SETF because all jobs have equal weight, and the sum of the processing times is a lower bound to optimal.

Acknowledgment

Supported in part by an IBM Faculty Award, and NSF grants CCF-1016684, CCF-0830558, CCF-1115575, CNS-1012070, CNS-1115575, and CNS-1253218

References

1. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**(4) (2000) 617–643
2. Motwani, R., Phillips, S., Torng, E.: Non-clairvoyant scheduling. Theor. Comput. Sci. **130**(1) (1994) 17–47
3. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News **42**(2) (June 2011) 83–97
4. Chadha, J.S., Garg, N., Kumar, A., Muralidhara, V.N.: A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In: Proceedings of the 41st annual ACM symposium on Theory of computing. STOC '09 (2009) 679–684
5. Chekuri, C., Khanna, S., et a l., Goel, A.: Multi-processor scheduling to minimize flow time with resource augmentation. In: In Proc. 36th Symp. Theory of Computing (STOC), ACM (2004) 363–372
6. Edmonds, J., Pruhs, K.: Scalably scheduling processes with arbitrary speedup curves. In: SODA. (2009) 685–692
7. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. ACM Trans. Algorithms **3**(4) (November 2007)
8. Chan, H.L., Edmonds, J., Lam, T.W., Lee, L.K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for flow and energy. Algorithmica **61**(3) (2011) 507–517
9. Chan, H.L., Edmonds, J., Pruhs, K.: Speed scaling of processes with arbitrary speedup curves on a multiprocessor. Theory Comput. Syst. **49**(4) (2011) 817–833
10. Bansal, N., Krishnaswamy, R., Nagarajan, V.: Better scalable algorithms for broadcast scheduling. In: ICALP (1). (2010) 324–335
11. Merrit, R.: Cpu designers debate multi-core future. EE Times (February 2010)
12. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling heterogeneous processors isn't as easy as you think. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '12, SIAM (2012) 1242–1253