Meeting the Fairness Deadline in Speed Scaling Systems: Is Turbocharging Enough?

[Extended Abstract]

Maryam Elahi, Carey Williamson, and Philipp Woelfel
Department of Computer Science, University of Calgary
{bmelahi, carey, woelfel}@ucalgary.ca

ABSTRACT
In this work, we explore the notion of ‘turbocharging’ in speed scaling systems, and ask whether this is sufficient to preserve the strong dominance property of FSP over PS. The answer turns out to be no, but the analysis yields useful insights into the design of speed scaling systems that can outperform PS in response time, energy consumption, or perhaps both.

1. INTRODUCTION
Dynamic speed scaling has received considerable attention in the recent literature as a method for balancing performance and energy consumption in computer systems [1]. In speed scaling systems, the goal of optimizing the response times is in natural tension with minimizing the energy consumption. Running at higher speeds results in better response times, yet it requires more energy. The first analytical study of dynamic speed scaling was performed by Yao, Demers and Shenker [7]. They presented an offline algorithm to minimize the total energy consumption under the constraint that jobs have to complete before given deadlines. In many practical settings, there are no clear job completion deadlines, and the goal instead is to optimize the tradeoff between energy consumption and the mean response time. A typical formulation of the problem involves minimizing the total cost \( z \) in the following equation, where \( T \) represents response time, \( \varepsilon \) reflects energy cost, and \( \beta \) is a positive relative weighting factor:

\[
z = E[T] + E[\varepsilon]/\beta.
\]

Andrew, Lin and Wierman [2] presented the first study of fairness under policies that optimize the cost \( z \). In particular, they show that while Processor Sharing (PS) with speed scaling maintains its constant slowdown and is considered to be the criterion for fairness, speed scaling exacerbates unfairness under Shortest Remaining Processing Time (SRPT) and non-preemptive policies such as First Come First Served (FCFS).

Although PS remains the status quo for fairness, similar to the single speed model, it is suboptimal [6, 2]. In search of a scheduling policy that can improve the average response time of PS without being unfair to any job, Elahi, Williamson and Woelfel [5] investigate the Fair-Sojourn-Protocol (FSP) with speed scaling. The authors demonstrate that in the coupled speed scaling model where the speed is a function of the number of jobs in the system, FSP’s dominance over PS, namely its guarantee that no job finishes later under FSP than under PS, is violated. This is due to the premature reduction in the speed of the system under FSP, which puts it at a disadvantage in comparison to PS. In fact it is not possible to dominate PS in the coupled speed scaling model [5]. Hence, to dominate PS, we need to change the speed scaling model. One possible solution to this problem is to run the system at the rates that PS would have used for the same sequence of arrivals. This is called decoupled speed scaling [5].

In this paper, we investigate another option, which scales all CPU speeds, so that the final job completes no later than under PS. We call this approach turbocharging. The key idea is illustrated in Figure 1, which shows the execution behavior of a linear speed scaling system handling two jobs, each of unit size. PS completes both jobs at time 1, as shown in Figure 1 left. FSP completes the first job at time 0.5, and the second job at time 1.5, as shown in Figure 1 middle. Note that the latter constitutes a violation of the dominance property, since job 2 completes later under FSP than under PS. However, turbocharging the FSP rates by 50% restores the dominance property in this particular example, as shown in Figure 1 right. Our work explores whether this property holds under general workloads. We show that naive turbocharging does not suffice to preserve dominance.

We then look at a new model in which we introduce deadlines for jobs in the system based on their departure time in a PS system. We show that in the case of batch arrivals, the algorithms introduced by Yao, Demers and Shenker [7] outperform PS in optimizing the cost \( z \) in (1). Through an example, we show that PS is suboptimal, not only in response time, but also in energy consumption. The lower bound for the worst case analysis of the online case is the topic of our ongoing research.

2. SYSTEM MODEL
We consider a single-server queue with dynamically adjustable service rates. We assume that the service rates are continuous and unbounded, and there is no penalty for changing the service rate. We consider a preempt-resume model, where a job may be preempted and later resumed without any context-switching overhead. A sample path is a sequence of tuples specifying job arrival times, job sizes, and job deadlines. Let \( a_i, w_i \) and \( d_i \) denote the arrival time, size and deadline, respectively, for job \( J_i \). Size (work) of a
job is equal to the time it takes to service the job at unit rate. We let \( d_i = \infty \) if job \( J_i \) has no deadline. A batch of \( n \) jobs is a sequence of jobs arriving at the same time. For our results, we assume that all jobs arrive in one batch (i.e., at the same time) and have arbitrary sizes and no deadlines. We assume that jobs in a batch are sorted in non-decreasing order of job sizes, such that \( w_1 \leq w_2 \leq \ldots \leq w_n \).

A speed scaling function, \( r(t) \), specifies the rate of the system at time \( t \). Let \( P(r) \) denote the power required to run at rate \( r \). Then the total energy consumed by the system in the time interval \([0,t]\) is

\[
\int_0^t P(r(\tau)) \, d\tau.
\]

For coupled speed scaling, the rate of the system at time \( t \) is determined by the number of jobs remaining in the system at time \( t \), denoted by \( n(t) \), and thus is influenced by the scheduling policy. One of the more commonly considered functions to model the power consumption of the CPU is \( P(s) = s^\alpha \). The best known policy that optimizes the cost in (1) uses the speed function \( r(t) = P(n(t))^{-1} \). [2]. This complies with the common heuristic used by system designers, e.g., [4]. For decoupled speed scaling, the rate at time \( t \) is uniquely determined by the sample path and \( t \), and thus it is independent of the scheduling policy.

We consider the class of work-conserving batch scheduling policies. In batch scheduling, one batch of jobs arrives at some time when no other jobs are in the system, and no other jobs arrive before the entire batch has been processed. The scheduler learns about all of the job sizes upon their arrival.

The response time (also known as turnaround time, sojourn time, or flow time) of a job is the amount of time the job spends in the system, i.e., its departure time minus its arrival time.

We use the following definitions of dominance, as previously presented in [12].

**Definition 1.** Scheduling policy \( p' \) dominates policy \( p \) if

1. on any sample path, no job completes later under \( p' \) than under \( p \), and
2. there exists a sample path such that some job on that sample path completes earlier under \( p' \) than under \( p \).

Our goal is to devise a policy that achieves fairness by dominating PS while having comparable energy consumption in the speed scaling world. The ultimate goal is to minimize the total cost \( z \), which is the linear combination of response time and energy consumption as in (1), while maintaining fairness.

### 3. Turbocharged FSP

As an initial attempt, we investigate FSP with coupled speed scaling. In our own prior work [5], we show that no policy can dominate PS when the speed function is monotonically increasing in the number of jobs in the system (which is the case for coupled speed scaling). Therefore, FSP cannot work in the coupled speed scaling model. On the other hand, under decoupled speed scaling, where at any point in time FSP runs at the same speed as PS, FSP dominates PS. Using this strategy improves upon the response time of PS, but it is using the same amount of energy as PS.

The question is whether we can devise a policy that can improve upon both the average response time and the average energy consumption of PS. One possible idea is to use FSP with coupled speed scaling, and scale the speeds by a constant factor that is large enough so that the last job finishes at the same time as it finishes under PS. We call this approach turbocharged FSP. We show in the following that even in the simple case of batch arrivals this naive approach does not provide dominance over PS.

Let \( X_k^PS \) denote the departure time of job \( k \) under policy PS. Let \( f(n) \) denote any coupled speed scaling function. We assume all jobs arrive at time 0. Under PS, jobs leave the system in the order of shortest remaining work. A simple calculation gives us the closed form in (3) for departure time of job \( J_k \) under PS. For notational convenience, we assume a job \( J_0 \) with size \( w_0 = 0 \) exists, which does not affect the system, but allows for a simpler closed form of \( X_k^{PS} \).

\[
X_k^{PS} = \sum_{i=1}^{k} \frac{w_i}{f(n-i+1)}.
\]

By the definition of FSP, the order in which jobs depart is the same under PS and FSP. Therefore, the departure time of job \( J_k \) under FSP is:

\[
X_k^{FSP} = \sum_{i=1}^{k} \frac{w_i}{f(n-i+1)}.
\]

Define \( b_n = X_n^{FSP}/X_n^{PS} \). In order to ensure that the last job, \( J_n \), finishes at the same time under FSP as under PS, it suffices to scale up the speed function \( f(n) \) used by FSP by a factor of \( b_n \), where

\[
b_n = \frac{X_n^{FSP}}{X_n^{PS}} = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{f(n-i+1)}}
\]

\[
\times \frac{1}{\sum_{i=1}^{n} \left( \frac{w_i}{f(n-i+1)} - \frac{w_i}{f(n-i)} \right)}
\]

\[
= 1 + \frac{\sum_{i=1}^{n} (n-i) w_i (1 - \frac{1}{f(n-i+1)})}{\sum_{i=1}^{n} (n-i) w_i (1 - \frac{1}{f(n-i)})}.
\]

Since job sizes are positive, and \( f(n) \) is monotonically increasing, it is clear that \( b_n \geq 1 \). In general, \( b_n \) is an increasing function of \( n \). Also, the worst case (largest) value of \( b_n \) occurs when all jobs are homogenous in size.

To achieve dominance over PS, all departure times of jobs under turbocharged FSP need to be no later than under PS. This happens if and only if \( b_n \), the turbocharge value required to finish job \( k \) at the same time under FSP as under PS, is no greater than \( b_n \), i.e., if \( b_k \leq b_n \) for all \( k \in [1, n] \).

It is easy to see that if all jobs are of unit size and the power function \( P(s) = s^\alpha \) is used with \( \alpha = 1 \), then \( b_n = H_n \).
where \( H_n \) is the \( n \)th Harmonic Number. If we allow \( \alpha > 1 \), then \( b_n = n \frac{2}{\alpha} - n \frac{H_n}{\alpha} \), where \( H_{n,m} \) is the \( n \)th Generalized Harmonic Number. Assuming \( w_1 \cdots w_{n-1} = 1 \) and \( w_n > 1 \), Equation (4) simplifies to,

\[
b_n = 1 + \frac{H_{n,m} - n \frac{H_n}{\alpha}}{w_n + n \frac{H_n}{\alpha} - 1} = 1 + \frac{H_{n,1/\alpha} - n \frac{H_n}{\alpha}}{w_n + n \frac{H_n}{\alpha} - 1}.
\]

We can show that \( b_{n-1} > b_n \), for all \( n \) and \( w_n \).

Thus, the dominance of turbocharged FSP over PS breaks. For example, for \( \alpha = 1 \), \( n = 4 \) and \( w_n = 14 \), \( b_4 = 181/168 \), while for FSP to finish the third job before PS, it needs a turbocharge value \( b_3 = 25/12 \).

We showed that naive turbocharging of FSP does not dominate PS. Nonetheless, this analysis provides insight into the design of a possible speed scaling policy based on target deadlines derived from PS scheduling, which we explore next.

4. DEADLINES FOR FAIRNESS

In the batch arrival model, the problem of optimizing the energy consumption, while guaranteeing that jobs finish no later than under PS is equivalent to the problem of minimizing the energy consumption for jobs with deadlines. The latter problem has been studied extensively [7, 3].

Yao, Demers and Shenker [7] presented an offline algorithm that computes a schedule and a rate function so that all jobs meet their deadline. If the power function is convex, then the algorithm yields the smallest possible energy consumption under the constraint that jobs should finish before their deadlines. Their algorithm, called YDS, proceeds in iterations and considers the intensity of time intervals as defined in the following (see Definition 2). In each iteration, it identifies the time interval of maximal intensity. The jobs whose arrival and departure times fall in that interval are then scheduled in the order of their deadlines. The system rate is set to be equal to the intensity of the interval throughout the entire interval. The algorithm then repeats these steps until all jobs are scheduled.

**Definition 2.** The intensity of an interval \( I = [t_1, t_2] \) is

\[
g(I) = \frac{\sum_{j \mid [a_j, d_j] \subseteq I} w_j}{t_2 - t_1}
\]

where \( a_j \) is the arrival and \( d_j \) is the departure time of job \( j \).

We can use YDS to minimize the energy consumption of a speed scaling system while maintaining good response time behavior, if all jobs arrive in one batch. For each job, we compute the departure time that it would have under PS with a job-count based speed scaling function. We then define each job’s deadline to be its departure time under PS and use YDS to compute an optimal schedule and system rate function.

By definition, YDS dominates PS. Hence it improves upon the response time behavior of PS. Since YDS minimizes the energy consumption, it outperforms PS in minimizing the total cost \( z \). The following example shows how PS uses more energy than necessary in some scenarios.

Consider a batch of \( n \) jobs, in which all jobs are of size 1, except the last one which has size \( w_n > 1 \). The deadline of job \( j_k \) is given by \( X^{PS}_{j_k} \) as defined in (3). Assume \( \alpha = 1 \). Then \( X_{PS}^1 = \cdots = X_{PS}^{j_k} = 1 \), and \( X_{PS}^{j_k} = w_n \). For this example, the only intervals with intensity greater than zero are \( I_1 = [0, 1] \) and \( I_2 = [w_n, 1] \). These two intervals have intensities \( g(I_1) \) and \( g(I_2) \), respectively, where

\[
g(I_1) = \frac{\sum_{i=1}^{n-1} w_i}{X_{PS}^{i-1}} = \frac{n-1}{n^{\frac{1}{\alpha}}}, \text{ and } g(I_2) = \frac{\sum_{i=1}^{n} w_i}{X_{PS}^{n-1}} = \frac{(n-1) + w_n}{w_n - 1 + n^{\frac{1}{\alpha}}}.
\]

If the intensity of the first interval exceeds that of the second, then PS and YDS use identical rates, and thus incur the same energy consumption. On the other hand, if \( g(I_1) > g(I_2) \), then YDS uses a fixed rate throughout \( I_1 \) and \( I_2 \). This is lower than the rate used by PS during \( I_1 \), and higher than what PS uses during \( I_2 \). If the power function is convex, then YDS consumes less energy in the second case. This simple example confirms that YDS outperforms PS in minimizing the cost \( z \). Further investigation of the competitive ratio of PS in this model is the focus of our ongoing work.

5. CONCLUDING REMARKS

The results show that naive turbocharging of FSP does not suffice. To provide dominance over PS, we can introduce deadlines based on the departure times under PS, and use an algorithm for energy minimization for scheduling jobs with deadlines. For the case of batch arrivals, this is equivalent to YDS.

We are looking to extend this idea to the online version. We are investigating a lower bound for the competitive ratio of any policy that minimizes the energy consumption for jobs with deadlines based on their departure under PS.

6. REFERENCES