Turbocharged Speed Scaling: Analysis and Evaluation

Maryam Elahi Carey Williamson Philipp Woelfel
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4
{bmelahi, carey, woelfel}@cpsc.ucalgary.ca

***Abstract*—In speed scaling systems, the execution speed of a processor can be adjusted dynamically under operating system control to provide tradeoffs between response time, fairness, and energy consumption. In this paper, we propose and evaluate an approach called envelope-based turbocharging, applied in conjunction with Fair Sojourn Protocol (FSP) scheduling and job-count-based speed scaling. This approach restores the strict dominance of FSP over Processor Sharing (PS) in speed scaling systems, and preserves fairness. We evaluate our new approach using analysis and simulation. The simulation results show that Turbocharged FSP (T–FSP) outperforms PS in response time, and often in energy consumption as well. Furthermore, the energy consumption of T–FSP is typically within 15% of optimal.**

***Index Terms*—Speed scaling; response time; energy efficiency**

I. Introduction

Dynamic speed scaling provides tradeoffs between execution-time performance and energy consumption in modern computer systems [1]. Specifically, running the CPU at higher speeds provides better response times, but consumes more energy. Finding the right balance between these two metrics is the primary challenge in speed scaling systems.

Fairness in dynamic speed scaling systems is also important. In recent work by Andrew, Lin, and Wierman [2], the authors illustrate the inherent tradeoffs between fairness, robustness, and optimality in speed scaling systems. In particular, they show that Processor Sharing (PS) remains fair under speed scaling, since it provides the same expected slowdown for all jobs. However, speed scaling can exacerbate the unfairness of other scheduling policies, such as Shortest Remaining Processing Time (SRPT) and First Come First Serve (FCFS). While PS remains the benchmark for fairness, it performs suboptimally on the other metrics [2].

In earlier work [3], we studied a speed-scaled version of the Fair Sojourn Protocol (FSP) [4], seeking a scheduling policy that improves upon the response time of PS, without being unfair to any job. In single-speed systems, FSP is appealing because of its *dominance* property; specifically, no job finishes later under FSP than under PS. However, in a coupled (i.e., job-count-based) speed scaling model, we found that FSP's dominance over PS no longer holds [3].

In the same paper [3], we proposed *decoupled* speed scaling, wherein the speed of the system is determined by an external speed scaling function. This approach differs from the prior literature on (coupled) job-count-based speed scaling, in which the speed is determined dynamically based on the current number of jobs in the system. Under decoupled speed scaling, FSP again dominates PS [3]. However, there are many technical challenges in implementing decoupled speed scaling, since CPU speed changes may be required at arbitrary points during the execution of a given job, rather than just at job arrival and departure instants.

In this paper, we consider "turbocharging" as an alternative approach to restoring the dominance of FSP over PS [5]. In this approach, CPU speeds are scaled up so that jobs under FSP complete no later than under PS. This approach is simpler in that it once again couples CPU speed with instantaneous system occupancy, so that speed changes (when required) only occur at job arrival or departure points.

The key idea behind turbocharging is illustrated in Figure 1. This example shows the execution behavior of a linear speed scaling system handling two jobs, each of unit size. By running at speed 2, as shown in the leftmost part of Figure 1, PS completes both jobs at time 1. In contrast, speed-scaled FSP completes the first job (with speed 2) at time 0.5, and the second job (with speed 1) at time 1.5, as shown in the middle diagram. Note that the latter event violates the dominance property, since job 2 completes later under FSP than under PS. However, turbocharging the FSP rates by a factor of 1.5 restores the dominance property on this particular example, as shown in the rightmost part of Figure 1. The first job is run at speed 3, completing at time 1/3, while the second job is run at speed 1.5 to complete at time 1. Energy consumption is arguably higher, but no job finishes later than it did under PS.

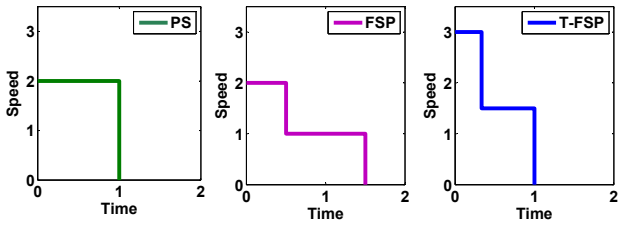

Fig. 1. Comparison of PS, FSP, and Turbocharged FSP (two jobs of size 1 at time 0, with linear job-count-based speed scaling)

In this paper, we investigate the idea of turbocharging in speed scaling systems. In particular, we explore the conditions

under which turbocharging FSP can guarantee dominance over PS. We first motivate our study through an example examining the effect of turbocharging on the response time and energy consumption under different scheduling policies. In particular, we show that naive approaches to turbocharging do not suffice to preserve the dominance property. However, we propose a new model, called *envelope-based turbocharging*, that is able to restore the dominance over PS. Finally, we compare our approach to one in which we introduce deadlines for jobs in the system based on their departure times in a PS system. We show that in the case of batch arrivals, the algorithm introduced by Yao, Demers, and Shenker [6] outperforms PS. Specifically, we show that PS is suboptimal, not only in response time, but also in energy consumption. Furthermore, Turbocharged FSP typically outperforms PS on one or both of these metrics.

The remainder of the paper is organized as follows. Section II reviews recent literature on CPU speed scaling systems. Section III presents an example to motivate our work. Section IV presents our system model. Section V presents our analytical results. Section VI presents simulation results. Finally, Section VII concludes the paper.

II. Background and Related Work

A. Scheduling in Single-Speed Systems

Scheduling policies are often evaluated based on *mean response time*, which measures the average time that a job spends in the system between arrival and departure. Under this metric, the optimal policy is Shortest Remaining Processing Time (SRPT) [7]. SRPT is a preemptive policy that always selects for service the pending job with the least remaining work. Although SRPT minimizes the mean response time, it is rarely used in practice, because it might be unfair. In particular, large jobs may be starved if priority is given to small jobs.

The unfairness of SRPT has been studied extensively [8], [9], [10], [11]. Under heavy load, SRPT may be unfair to certain jobs [8]. However, in many cases, SRPT can provide lower expected response times than PS for *all* job sizes [11]. Counter-intuitively, the jobs that receive unfair treatment under heavy load are the medium-large jobs, not the largest jobs. In fact, all scheduling policies asymptotically converge to the same slowdown value for very large jobs [12].

The design of the Fair Sojourn Protocol (FSP) policy combines the response time advantages of SRPT with the fairness properties of PS [4]. FSP selects for service the pending job that would complete the soonest under PS scheduling, and then devotes full service to this job until the next arrival or departure occurs. FSP relies upon a "virtual PS" queue in the background, and recomputes its next scheduling decision upon each job arrival or departure event.

An intriguing aspect of FSP is its strict dominance over PS. Friedman and Henderson [4] define dominance as follows:

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.

Definition 1 provides a very strong notion for fairness. Specifically, if a policy p dominates PS, then it is also fair. However, a fair policy is not guaranteed to dominate PS.

B. Scheduling in Speed Scaling Systems

The literature on speed scaling systems includes both systems work and theoretical work. In the systems community, speed scaling is known as Dynamic Voltage and Frequency Scaling (DVFS). In such work, many practical constraints arise due to limited voltage/frequency ranges, discrete speeds, limited granularity of control, the overhead of CPU governors, and diverse job characteristics [13], [14], [15], [16]. Implemented policies in practice include (coarse-granularity) threshold-based control, rush-to-idle, gated on-off, and Intel's Turbo Boost technology. In the theoretical community, speed scaling is typically job-count-based, with a continuous and unbounded range of speeds available. Since our work follows the latter paradigm, we focus on this literature next, though we do mention practical implementation issues later in the paper.

In speed scaling systems, the many tradeoffs between service rate, response time, and energy consumption have triggered extensive work on energy-efficient algorithms [1], [17]. Yao, Demers, and Shenker [6] pioneered the analytical study of dynamic speed scaling in a context where jobs have deadlines and the service rate is unbounded. An alternative approach has focused on minimizing the response time in systems, given a fixed energy budget [18], [19].

A more practical approach aims at optimizing the tradeoff between energy consumption and mean response time [20]. Several studies on this metric suggest that energy-proportional speed scaling[1] is near optimal [2], [21], [22]. The optimal (coupled) policy is SRPT with $s = P^{-1}(n\beta)$ as its job-count-based speed scaling function [2].

The use of speed scaling may alter the fairness properties of scheduling policies. Fairness under speed scaling was first formally studied by Andrew, Lin, and Wierman [2]. They use a similar definition for fairness in speed scaling systems as in single-speed systems:

Definition 2: A policy p is fair if for all x

$$\frac{E[T_p(x)]}{x} \leq \frac{E[T_{PS}(x)]}{x}.$$

Under this metric, speed scaling can magnify unfairness for SRPT and non-preemptive policies such as FCFS. The intuition is that large jobs receive lower service rates, since they are only scheduled when the smaller jobs have left the system (i.e., the system occupancy is lower).

Two observations about this fairness definition are worth noting. First, fairness is defined in terms of expectation, rather than in the strict per-job sense of dominance. Strict dominance

[1] In energy-proportional speed scaling, the power consumption $P(s)$ when the system is run at speed s is directly proportional to the number of jobs in the system (e.g., $P(s) = n$).

implies fairness, but the converse is not necessarily true. Second, Definition 2 ignores energy consumption, since it only considers job response times. In [3], PS is proven to be efficient under the coupled speed scaling model, while it is conjectured that no other policy can be fair according to Definition 2. With decoupled speed scaling, however, it is possible to improve upon the response time of PS while maintaining fairness [3].

III. Motivating Example

This section uses an example to build the intuition behind turbocharged speed scaling. At time 0, a batch of three jobs arrive (J_1, J_2, and J_3), with sizes 1, 2, and 5, respectively.

In a single-speed system, PS would schedule the jobs as shown in Figure 2, while FSP would schedule the jobs as shown in Figure 3. In these diagrams, the vertical axis represents CPU speed, while the horizontal axis represents time. Since both systems run at the same fixed rate, they both complete the same amount of work in the same amount of time. However, FSP has a response time advantage over PS, since some jobs (J_1 and J_2) complete sooner under FSP than under PS, and no job completes later under FSP than it does under PS. This example illustrates the dominance property.

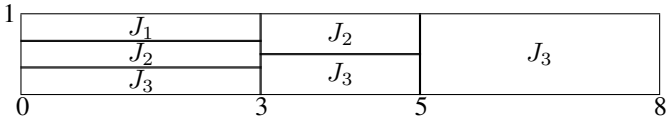

Fig. 2. PS Scheduling (no speed scaling)

Fig. 3. FSP Scheduling (no speed scaling)

Now consider a speed-scaling system, with job-count-based speed scaling, and a linear speed scaling function. PS would schedule the jobs as shown in Figure 4, while FSP would schedule the jobs as shown in Figure 5. Because FSP completes the smaller jobs more quickly, it lowers the CPU speed sooner than PS does, and ends up taking longer to complete the entire batch of jobs. In particular, job J_3 finishes later under FSP than it does under PS. The extended duration of the busy period leads to a violation of the dominance property.

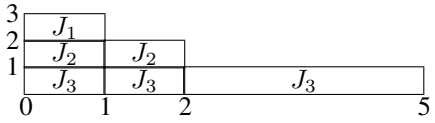

Fig. 4. PS Scheduling (speed scaling)

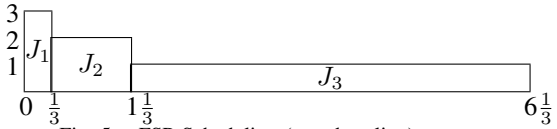

Fig. 5. FSP Scheduling (speed scaling)

One possible solution to this problem is to *turbocharge* FSP, based on the corresponding busy period durations. In this particular example, we would scale all of the CPU speeds by 1.266667 (the ratio of $6\frac{1}{3}$ and 5). The result is the job schedule in Figure 6. In this example, turbocharging is sufficient to restore the dominance property: no job finishes later under Turbocharged FSP (Figure 6) than it does under PS (Figure 4).

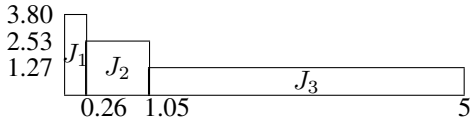

Fig. 6. Turbocharged FSP Scheduling (speed scaling)

Does this property hold in general for Turbocharged FSP? The answer turns out to be "no", at least for naive turbocharging that merely aligns the busy period durations [5]. A more elaborate example with 4 jobs (sizes 1, 1, 1, and 20) suffices to illustrate this point. Figure 7 shows that PS completes all the jobs in 20 time units, while FSP needs 21.083 time units (Figure 8). However, scaling the speeds of FSP by 5.4% to make the busy periods align in Figure 9 is not sufficient. While this naive turbocharging approach does satisfy job J_4, job J_3 still encounters a violation (1.028 versus 1). A 2.8% higher service rate is required to complete this job on time.

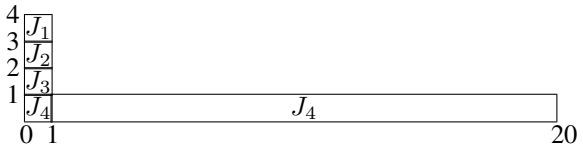

Fig. 7. PS Scheduling (4 jobs: 1, 1, 1, 20)

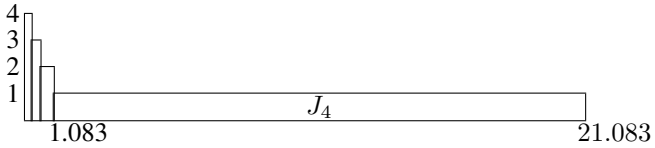

Fig. 8. FSP Scheduling (4 jobs: 1, 1, 1, 20)

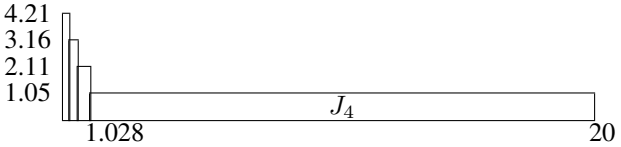

Fig. 9. Naive Turbocharged FSP Scheduling (4 jobs: 1, 1, 1, 20)

The rest of this paper explores the conditions under which turbocharging can work in speed scaling systems. In particular, we propose and evaluate an envelope-based turbocharging strategy that preserves the dominance property of FSP, and quantify its response time performance and energy cost.

IV. System Model

We consider a single-server system with dynamically adjustable service rates. We assume that the service rates are continuous and unbounded, and there is no overhead incurred for changing the service rate.

A *sample path* is a sequence of tuples specifying job arrival times, job sizes, and possibly 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 main results, we assume that all jobs arrive in one batch (i.e., at the same time) and have arbitrary sizes and no deadlines. For convenience, we assume that jobs in a batch are sorted in non-decreasing order of job sizes, such that $w_1 \leq w_2 \leq \cdots \leq w_n$. A summary of our model notation appears in Table I.

In this paper, 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 knows all of the job sizes upon their arrival. We consider a preempt-resume model, where a job may be preempted and resumed with zero context-switching overhead.

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,τ] is

$$\int_0^\tau P(r(t))dt. \tag{1}$$

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 $l(t)$, and thus is influenced by the scheduling policy. To optimize a linear combination of mean response time and energy consumption, the best known policy uses the speed function $r(t) = P^{-1}(l(t)\beta)$, where β is a weighting factor indicating the relative importance of response time and energy consumption [2]. In this paper, we assume $\beta = 1$. We consider $P(s) = s^\alpha$, which is commonly used in the literature to model the power consumption of the CPU. The typical value for the exponent is $1 \leq \alpha \leq 3$. Therefore, in the coupled speed scaling model we use $r(t) = l(t)^{\frac{1}{\alpha}}$. In speed scaling with turbocharging, the speeds are scaled by a positive factor b, such that $r(t) = b(l(t)^{\frac{1}{\alpha}})$.

In online scheduling, the scheduling decisions of a policy are independent of future arrivals. As is conventional in the performance modeling literature, we use the term *busy period* to refer to a time period during which there is always at least one job in the system

We use the PS scheduling policy as a baseline for comparison. PS is the *de facto* standard for fairness because it treats all jobs equally. At each point in time, PS gives equal service to all active jobs in the system.

V. Analytical Results for Turbocharging

In this section, we present our analytical results for turbocharged speed scaling systems. Our main focus is on FSP and its relationship to PS, although the notion of turbocharging is broadly applicable to any scheduling policy.

TABLE I
Model Notation

Symbol	Description
n	Number of jobs
$f(n)$	CPU speed as a function of number of jobs
J_i	Job i
a_i	Arrival time of job i
w_i	Work (size or service requirements) associated with job i
d_i	Deadline (if any) for job i
X_i	Departure time of job i
b_i	Turbocharging (boosting) rate of job i
t	Time in seconds
$r(t)$	CPU speed (rate) as a function of time
$l(t)$	Number of jobs as a function of time
$P(r)$	Power consumption when running at rate r
α	Exponent used in power consumption function

A. General Formulation

Let X_k^π denote the departure time of job J_k under policy π. Under PS, the server is shared equally amongst all jobs in the system. Under this policy, jobs complete and depart from the system in order of size, with CPU speeds decreasing in a staircase fashion (see Figure 4). Let $f(n)$ denote the speed that the server runs at when there are n jobs in the system. Equation (2) shows the expression for the departure time of job J_k under PS. The summation occurs over one or more components of job execution, each with a different speed. For notational convenience, we assume a job J_0 with size $w_0 = 0$. While it does not affect the system, it simplifies the expression for X_k^{PS}. The denominator inside the summation represents the system speed being used when there are $n - i + 1$ jobs in the system. The factor $(w_i - w_{i-1})$ in the numerator represents the residual service time of the i-th job, once its immediate predecessor has departed, while the factor $(n-i+1)$ represents the "stretch factor" from sharing the CPU in a PS fashion with the other remaining jobs.

$$X_k^{PS} = \sum_{i=1}^{k} \frac{(w_i - w_{i-1})(n-i+1)}{f(n-i+1)} \tag{2}$$

Under the Fair Sojourn Policy (FSP), the server is dedicated to one job at a time, in preferential order based on the virtual PS completion times. By the definition of FSP, the order in which jobs depart is the same as under PS. Therefore, the departure time of job J_k under FSP is:

$$X_k^{FSP} = \sum_{i=1}^{k} \frac{w_i}{f(n-i+1)}. \tag{3}$$

The physical interpretation of this expression is quite straightforward. The waiting time for job J_k depends on the $k-1$ smaller jobs that precede it, each of which is served at progressively lower rates. To this waiting time, one must add the service time of job J_k itself. See Figure 5 for an example.

Because FSP completes some jobs (i.e., jobs with smaller remaining sizes) more quickly than they finish under PS, the CPU speed is reduced earlier, which can lead to some jobs (i.e., jobs with larger remaining sizes) finishing *later* under

FSP than they finished under PS (see Figure 5). Such an occurrence violates the dominance property of FSP over PS.

To restore the dominance property, we 'turbocharge' FSP (T–FSP). Define b_k as the ratio between the two completion times calculated for job k. That is, $b_k = X_k^{FSP}/X_k^{PS}$.

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

$$b_n = \frac{X_n^{FSP}}{X_n^{PS}} = \frac{\sum_{i=1}^{n} \frac{w_i}{f(n-i+1)}}{\sum_{i=1}^{n} \frac{(w_i - w_{i-1})(n-i+1)}{f(n-i+1)}} \tag{4}$$

Algebraic manipulation yields,

$$b_n = 1 + \frac{\sum_{i=1}^{n} (n-i) w_i (\frac{1}{f(n-i)} - \frac{1}{f(n-i+1)})}{\sum_{i=1}^{n} w_i (\frac{n-i+1}{f(n-i+1)} - \frac{n-i}{f(n-i)})}. \tag{5}$$

For $f(n) = n^{\frac{1}{\alpha}}$ and $\alpha = 1$, the foregoing expression reduces to:

$$b_n = 1 + \frac{1}{w_n} \sum_{i=1}^{n-1} \frac{w_i}{n-i+1} \tag{6}$$

With these formulations, it is straightforward to compute the turbocharging rate for each job in the batch *a priori*, given the job sizes and the speed scaling function.

B. Insights and Observations

Several insights emerge from studying these formulations:

- The turbocharging rate b_n can never be less than 1. Since job sizes are positive, and $f(n)$ is monotonically increasing, all terms inside the summations are positive, and so is the ratio. Therefore, $b_n \geq 1$. Equality occurs only for $n = 1$. For $n > 1$, $b_n > 1$ (i.e., the last job under FSP always finishes later than under PS).
- b_n has a direct linear dependence on the sizes of the first $n-1$ jobs, but is inversely dependent on the size w_n of the last job. The term w_n appears only in the denominator, and not in the numerator, so there is an inverse relationship in the dependence on w_n. That is, if w_n is huge compared to the other jobs, very little turbocharging will be needed (since the last job will consume most of the time in both PS and FSP).
- The speed scaling function $f(n)$ has two different effects. First, it shows up in the numerator of Equation (5), where it represents the relative effect of a *system* speed change between adjacent settings. Second, it shows up in the denominator, where it represents the relative change in the *per-job* share of the CPU in the PS case.
- When $\alpha > 1$, the relative speed changes with growing system occupancy are smaller, so b_n is smaller as well. Intuitively, both the PS schedule and the FSP schedule become much longer, so that the acceleration factor required to preserve dominance becomes relatively smaller.

The following observations also arise from our analysis:

Normalization: Only relative job sizes matter, not their absolute sizes. If the sizes of jobs are scaled by some arbitrary factor $c > 0$, this factor can be moved outside the summations and canceled, so there is no net effect on b_n. We can use this fact to consider only "normalized" job sequences where $w_1 = 1$, and all other job sizes are scaled relative to job J_1.

TABLE II
GENERALIZED HARMONIC NUMBERS

n	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
1	1.000	1.000	1.000
2	1.500	1.707	1.794
3	1.833	2.284	2.487
4	2.083	2.784	3.117
5	2.283	3.232	3.702
6	2.450	3.640	4.252
7	2.593	4.018	4.775
8	2.718	4.371	5.275
9	2.829	4.705	5.756
10	2.929	5.021	6.220

Homogeneous Jobs: The worst case (largest) value of b_n occurs when all jobs are the same size. In such a system, PS runs at full speed $f(n)$ until all jobs complete simultaneously, while FSP monotonically decreases the CPU speed as each of the jobs depart. The growing discrepancy between the speed of PS and the speed of FSP can make b_n large.

Harmonic Numbers: When the speed scaling is linear $f(n) = n$, then the homogeneous job case results in $b_n = H_n$, where H_n denotes the n-th Harmonic number. This is good news for turbocharged speed scaling, since the Harmonic numbers are a slowly growing function of n. For general α, the homogeneous job case results in $b_n = \frac{H_{n,m}}{n^{\frac{\alpha-1}{\alpha}}}$, where $H_{n,m}$ denotes the n-th generalized Harmonic number, with $m = \frac{1}{\alpha}$. Table II summarizes the first 10 Harmonic numbers in each of these sequences for several different values of α.

Monotonicity: In general, b_n is an increasing function of n for homogeneous job sizes. However, b_n is also a monotonically decreasing function of w_n, when the sizes of the first $n-1$ jobs are fixed.

Job Size Variability: The converse of the earlier observation about homogeneous jobs being the worst case is that variability in the job size distribution is beneficial, since it lowers b_n. This observation is consistent with a similar property in single-speed systems, wherein SRPT scheduling has its greatest advantage over PS when job sizes are highly variable [8]. Furthermore, size-based scheduling policies tend to be robust to inexact job size information [23], [24].

C. Naive Turbocharging

As our first result, we show that even in the simple case of batch arrivals, naive turbocharging of FSP does not guarantee dominance over PS. The intuition as to how to build these scenarios comes from the foregoing analysis. One can start with $n-1$ homogeneous jobs, for a large value of n so that b_{n-1} is large. Then the size of job J_n can be increased as needed to make b_n smaller than b_{n-1}. This will create scenarios where naive turbocharging fails to guarantee dominance.

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 $b_k \leq b_n$ for all $k \in [1, n]$.

Assuming $w_1 = w_2 = \ldots = w_{n-1} = 1$ and $w_n > 1$, Equation (5) simplifies to:

$$\begin{aligned} b_n &= 1 + \frac{H_{n,m} - \frac{n}{f(n)}}{w_n + \frac{n}{f(n)} - 1} \\ &= 1 + \frac{H_{n,1/\alpha} - n^{\frac{\alpha-1}{\alpha}}}{w_n + n^{\frac{\alpha-1}{\alpha}} - 1}. \end{aligned} \tag{7}$$

In this setting, we have:

$$\begin{aligned} b_{n-1} &= \frac{X_{n-1}^{FSP}}{X_{n-1}^{PS}} \\ &= \frac{\sum_{i=1}^{n-1} \frac{w_i}{(n-i+1)^{1/\alpha}}}{\sum_{i=1}^{n-1} (w_i - w_{i-1})(n-i+1)^{\frac{\alpha-1}{\alpha}}} \\ &= n^{\frac{1-\alpha}{\alpha}}(H_{n,1/\alpha} - 1). \end{aligned} \tag{8}$$

Specifically, b_{n-1} will exceed b_n, and break the dominance of turbocharged FSP over PS, for all n and w_n such that:

$$H_{n,1/\alpha} \geq 1 + n^{\frac{\alpha-1}{\alpha}} \text{ and } w_n > \frac{(n^{\frac{\alpha-1}{\alpha}} - 1)(H_{n,m} - 1)}{H_{n,m} - 1 - n^{\frac{\alpha-1}{\alpha}}}. \tag{9}$$

When $\alpha = 1$, these conditions reduce to $H_n \geq 2$ and $w_n = \frac{H_n - 1}{H_n - 2}$. For example, recall Figure 8, in which job J_3 experienced a violation of dominance when $\alpha = 1$, $n = 4$ and $w_n = 20$. The smallest integer value of w_n for which such a violation would occur is $w_n = 14$ (i.e., $w_n > 13.05$).

One corollary from the above analysis is that for linear speed scaling, the smallest possible counter-example must have at least 4 jobs (i.e., $H_n \geq 2$). For larger values of α, even more jobs are needed (e.g., 6 for $\alpha = 2$, and 7 for $\alpha = 3$). In other words, when the system occupancy is low, naive turbocharging cannot violate the dominance property.

D. Envelope-based Turbocharging

Naive turbocharging does not always suffice to guarantee the dominance property. One reason why it could fail is that naive turbocharging only aligns the end of the busy period to that of PS. Under an *arbitrary* scheduling policy, this criterion is not sufficient to guarantee dominance, since the jobs within the busy period could complete in a *different order* than under PS. For this reason, we restrict our focus to scheduling policies, such as FSP, that complete jobs in the same order as PS. However, even under FSP, as we have shown, it is possible for some jobs (e.g., the second last job, or any earlier job) to finish later under Turbocharged FSP than it did under PS.

The obvious solution to the violation problem is to identify the *critical job* within the batch (i.e., the job J_k with the largest b_k value). For the example in Figure 8, the b_k values are $b_1 = 1.0$, $b_2 = 1.5$, $b_3 = 1.833$, and $b_4 = 1.054167$, so job J_3 is the critical job with $b_{max} = 1.833$. Note, however, that we do not need to turbocharge the *entire* batch by b_{max}. Rather, once the critical job has been completed on time, one can recompute the turbocharging rate for the remaining jobs in the batch. In the example, the turbocharging rate can be reduced to $b = 1.054167$ again *after* time 1.0 to complete the remaining job J_4 by time 20. In fact, $b = 1.05263$ suffices for this purpose.

We call this approach *envelope-based turbocharging*. If $b_{max} = b_n$, then this approach is identical to naive turbocharging, with a single turbocharging rate for the entire busy period. Otherwise, multiple turbocharging rates are used, with one b value for the first part of the busy period, and envelope-based turbocharging applied recursively on the remaining jobs in the batch. This approach tightens the speed scaling profile for the batch, while aligning the end of the busy period (i.e., the completion time of job J_n) to that of PS, so as to reduce the energy consumption.

The key to envelope-based turbocharging is the notion of *virtual batches*. One example of a virtual batch is mentioned above, where a single job of size 20 needs to be completed within 19 time units after the departure of the critical job. The same idea can be used to extend our approach to online (i.e., dynamic) arrivals of jobs within a batch. Upon each new arrival, a recomputation of turbocharging rates is required. The virtual batch excludes all previously completed jobs from the batch, but includes the new job as well as all currently active jobs (partially completed or not-yet-started) in the batch. New PS completion deadlines are computed based on the residual sizes of jobs at the arrival instant, as well as their inherited deadlines from the former (physical or virtual) batch. Some job deadlines may remain the same when new work is added, while some may move forward into the future. However, with linear or sub-linear speed scaling (for $1 \leq \alpha$), the PS deadlines can never move backwards in time.

VI. Simulation Results

This section presents simulation results to demonstrate the efficacy of turbocharged speed scaling, in comparison to other scheduling and speed scaling strategies. Section VI-A presents results for the simple case of a single static batch of jobs, while Section VI-B presents results for more general workloads with dynamic job arrivals.

A. Baseline Results

Consider a single server system, initially empty, to which a batch of 10 jobs arrive. We use a simple discrete-event simulation model of this system, with configurable scheduling policies and workloads, and adequate instrumentation to record job response time and energy consumption. We use three different batch workloads to examine the behavior of different schedulers with turbocharging. Workload 1 is a batch of 10 homogeneous jobs (all of size 1), while workload 2 is a batch of 10 jobs with a linear progression of sizes $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$, and workload 3 is a batch of 10 jobs with a multiplicative progression of sizes $(1, 2, 4, 8, 16, 32, 64, 128, 256, 512)$. These tests cover low (no) variance, medium variance, and high variance workloads.

Figures 10, 11, and 12 show the simulation results for these three batch workloads, when $\alpha = 2$. In each figure, the left-most column of graphs shows the instantaneous number of jobs in the system, for the different scheduling and speed

scaling policies. The middle column of graphs shows the instantaneous CPU speed of the system, and the right-most column of graphs shows the instantaneous volume of work remaining in the system, as well as the job departure points.

Under PS, jobs leave the system in the same order as under SRPT [4]. By definition, FSP schedules jobs based on their order of departure under PS. Therefore, in batch scheduling, FSP is equivalent to SJF scheduling.

Table III, Table IV, and Table V provide a summary of the results for mean response time and energy consumption under the simulated policies. The tables represent the low variance, medium variance, and high variance job size distributions, respectively. Within each table, we also show the effect of α, for $1 \leq \alpha \leq 3$. Beneath each α value, the two columns show the mean response time ($E[T]$) and mean energy consumption ($E[\varepsilon]$) for each policy evaluated. Within each table, the bold font shows the scheduling policy with the lowest mean response time, which is often (but not always) Turbocharged FSP (T–FSP).

The results show that the mean response time and energy consumption of FSP are no worse than those of PS when $\alpha = 1$, and 20-30% lower than those of PS for larger values of α. However, Figure 10 (a) shows that under FSP, the last few jobs depart later than under PS, and hence FSP does not guarantee dominance over PS. This is due to the repeated reductions in the speed of the system under FSP (Figure 10 (b)), leaving a larger backlog of remaining work than under PS (Figure 10 (c)). Similar behavior is observed for the other workloads in Figure 11 and Figure 12.

Turbocharging FSP restores the dominance over PS in these examples, as shown in Figure 10 (d)), for example. For each sample workload, we observe that T–FSP provides better mean response time than PS. Compared to FSP, T–FSP has much lower response time, since the system is running faster. However, scaling up the speeds in T–FSP increases the energy consumption, as evident when $\alpha > 1$. Compared to PS, T–FSP improves response time by 20-60%, though often with a slight increase in energy consumption by 0-15%.

The simulation results for the sample workloads suggest that T–FSP can improve upon the performance of PS, while preserving dominance. In some cases, it improves upon *both* response time and energy consumption compared to PS (see Table V).

Two additional policies are shown in the tables as a point of comparison. One is FSP–PS, as an example of decoupled speed scaling [3]. By construction, this policy has the same energy consumption as PS, while exploiting the size-based scheduling of FSP to improve response time. In many of the cases shown, T–FSP provides slightly better response time than FSP–PS at slightly higher energy costs (Table III and Table IV), or comparable response time with slightly lower energy costs (Table V).

The second policy for baseline comparison is YDS, an offline scheduling algorithm that optimizes energy consumption while meeting job execution deadlines [6]. We use an initial run of the PS scheduler to determine job deadlines, and then use YDS to determine the minimum energy consumption possible among all policies that meet[2] the PS deadlines. This value is shown in italics in the table.

TABLE III
MEAN RESPONSE TIME AND ENERGY CONSUMPTION FOR DIFFERENT POLICIES (10 JOBS, HOMOGENEOUS JOB SIZES)

Scheduling Policy	$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$
PS	1.00	1.00	3.16	3.16	4.64	4.64
FSP	1.00	1.00	2.25	2.25	3.00	3.00
T–FSP	**0.34**	1.00	**1.42**	3.57	**2.24**	5.39
FSP–PS	0.55	1.00	1.74	3.16	2.55	4.64
YDS	0.55	1.00	1.74	3.16	2.55	4.64

TABLE IV
MEAN RESPONSE TIME AND ENERGY CONSUMPTION FOR DIFFERENT POLICIES (10 JOBS, LINEAR JOB SIZES)

Scheduling Policy	$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$
PS	5.50	5.50	14.3	14.3	19.8	19.8
FSP	5.50	5.50	10.4	10.4	13.3	13.3
T–FSP	**2.48**	5.50	**7.17**	15.2	**10.4**	21.6
FSP–PS	2.93	5.50	7.86	14.3	11.0	19.8
YDS	4.00	5.50	8.99	13.5	12.0	18.4

Overall, the results show that T–FSP performs comparably to FSP–PS and YDS. For example, T–FSP often improves upon the response time of FSP–PS, with only a slight increase in energy consumption. Furthermore, T–FSP is typically within 15% of the optimal energy consumption of YDS, while providing mean response times 12-30% lower than YDS.

Based on these results, we argue that Turbocharged FSP provides a promising approach for speed scaling systems. It provides significant response time advantages over PS, while preserving fairness (i.e., dominance property). In terms of practical considerations, the turbocharging rates of T–FSP can be computed directly from workload information, and T–FSP incurs far fewer context switches than PS scheduling. Furthermore, the energy consumption of T–FSP is only slightly higher than that of the (offline) YDS algorithm, which provides the (optimal) lower-bound for energy consumption among all policies that meet the PS deadlines.

B. Additional Results

We next consider a single server system, initially empty, to which jobs arrive at random times. In particular, we consider

[2]Note that in the tables, only PS, T–FSP, and YDS guarantee this property.

TABLE V
MEAN RESPONSE TIME AND ENERGY CONSUMPTION FOR DIFFERENT POLICIES (10 JOBS, MULTIPLICATIVE JOB SIZES)

Scheduling Policy	$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$
PS	102.3	102.3	168.1	168.1	202.9	202.9
FSP	102.3	102.3	137.6	137.6	155.1	155.1
T–FSP	73.80	102.3	115.0	164.7	137.2	198.3
FSP–PS	**72.13**	102.3	**114.2**	168.1	**136.8**	202.9
YDS	101.9	102.3	137.3	151.7	154.8	176.9

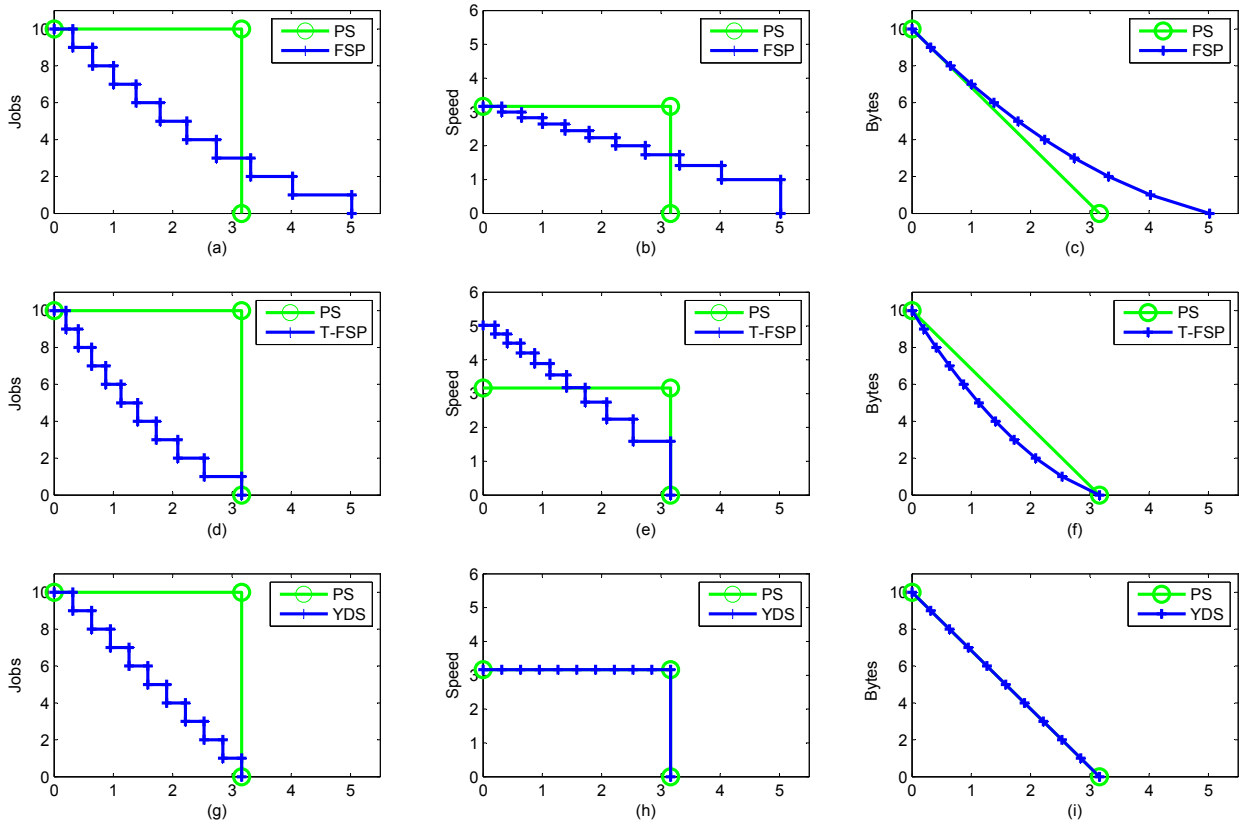

Fig. 10. Simulation results for workload 1 (10 homogeneous jobs, $\alpha = 2$)

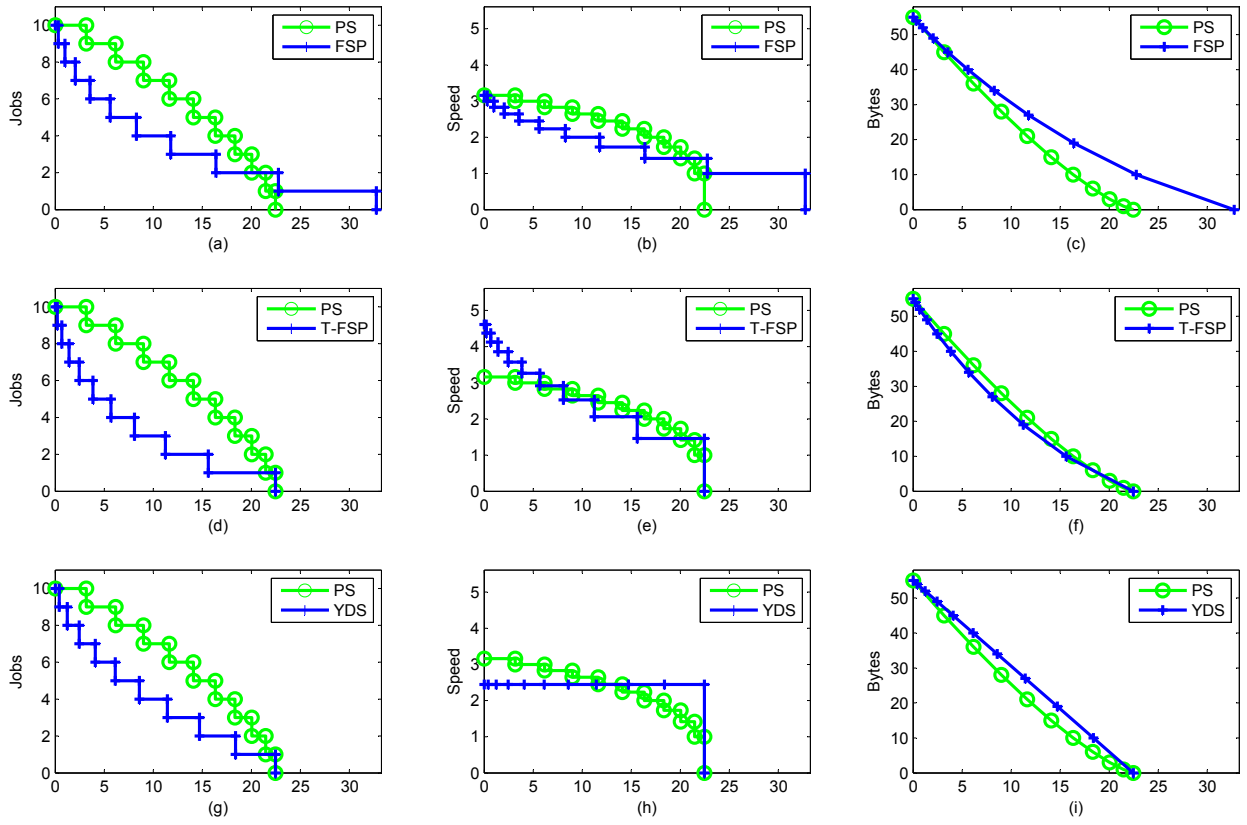

Fig. 11. Simulation results for workload 2 (10 linear jobs, $\alpha = 2$)

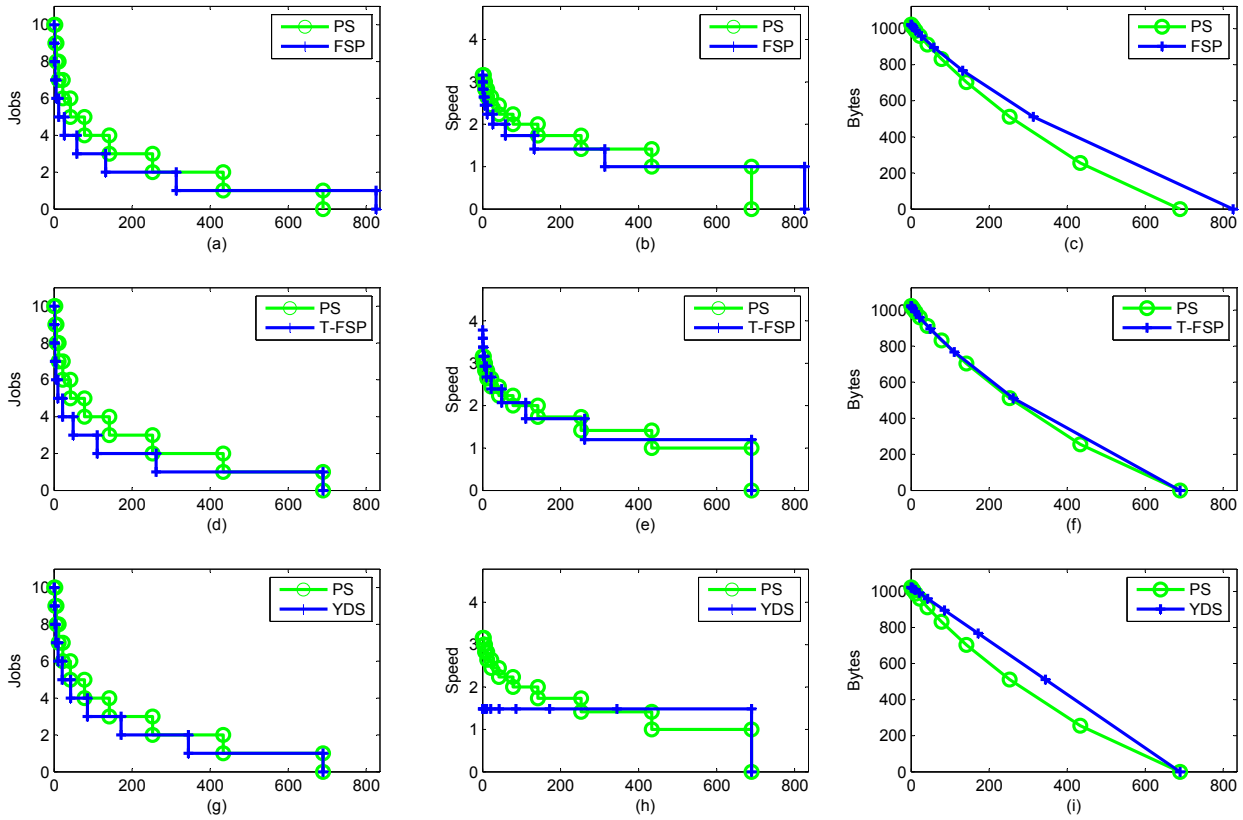

Fig. 12. Simulation results for workload 3 (10 multiplicative jobs, $\alpha = 2$)

a Poisson arrival process, with an average arrival rate of one job per second. Job sizes are exponentially distributed.

Since jobs can arrive dynamically at random times, the idea of virtual batches (see Section V-D) is used to compute the turbocharging rates for each envelope. In particular, upon each job arrival or departure, the residual sizes of all remaining active jobs are computed, based on the elapsed time (and service discipline) since the most recent speed change. For the subset of jobs that are still active under both PS and FSP, the projected completion times under PS and FSP are calculated, and these times are used to determine the turbocharging rate required for envelope-based turbocharged FSP (ET–FSP) so that no PS deadlines are violated.

We first consider a lightly loaded system, in which the average job service time is 1.0 seconds. There are 1000 jobs in total. Post-processing of this workload shows that there are 380 busy periods, with an average of 2.63 jobs per busy period. The highest system occupancy observed under PS is 6 jobs, while that under FSP and ET–FSP never exceeds 5 jobs.

Table VI shows the simulation results for PS, FSP, and ET–FSP on this workload. When $\alpha = 1$, ET–FSP provides a 22% reduction in the mean response time compared to PS, with no increase in energy consumption. When $\alpha = 2$, ET–FSP has 32% reduction in mean response time, with an increased energy cost of 2.4% compared to PS. For $\alpha = 3$, the corresponding values are 38% lower response time for ET–FSP compared to PS, with 6.9% higher energy consumption.

TABLE VI
MEAN RESPONSE TIME AND ENERGY CONSUMPTION FOR DIFFERENT POLICIES (LIGHT LOAD, 1000 JOBS, EXPONENTIAL SIZES)

Scheduling Policy	$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$
PS	1.006	1.006	1.513	1.513	1.966	1.966
FSP	1.006	1.006	1.331	1.331	1.566	1.566
ET–FSP	**0.783**	1.006	**1.029**	1.549	**1.220**	2.101

Finally, we consider a heavily loaded system, in which the average job service time is 100.0 seconds. There are 100 jobs in total. Post-processing of this workload shows that there is a single massive busy period. The highest system occupancy observed under PS when $\alpha = 1$ is 65 jobs, while that under FSP is 42 jobs, and that under ET–FSP is 28 jobs. By completing small jobs sooner, FSP and ET–FSP use lower service rates than PS throughout much of the busy period.

Table VII shows the simulation results for PS, FSP, and ET–FSP on this workload. When $\alpha = 1$, ET–FSP is able to provide a 60% reduction in the mean response time compared to PS, with exactly the same energy consumption. For $\alpha = 2$, ET–FSP provides 59% lower mean response time than PS, but an increased energy cost of 9.0%. For $\alpha = 3$, ET–FSP is 56% faster than PS, but with 15% higher energy consumption.

These results are similar to our baseline scenarios, and show that the performance advantages of envelope-based turbocharging are robust across the set of workloads considered.

TABLE VII
MEAN RESPONSE TIME AND ENERGY CONSUMPTION FOR DIFFERENT POLICIES (HEAVY LOAD, 100 JOBS, EXPONENTIAL SIZES)

Scheduling Policy	$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$
PS	105.0	105.0	695.2	695.2	1361.7	1361.7
FSP	105.0	105.0	470.4	470.4	827.8	827.8
ET–FSP	**42.0**	105.0	**282.8**	757.5	**594.3**	1565.1

VII. CONCLUSIONS

In this paper, we consider "turbocharging" in speed scaling systems, and explore whether it can preserve the strong dominance property of FSP over PS. The results show that naive turbocharging of FSP does not suffice. To provide dominance over PS, we propose envelope-based turbocharging, in which deadlines are introduced for jobs based on their departure times under PS. Our approach is initially demonstrated on batch workloads, but extends naturally to online arrivals by using virtual batches.

Our analytical and simulation results show that T–FSP provides response time performance that is superior to that of PS, with little or no additional energy cost. Furthermore, T–FSP provides better response time performance than FSP–PS and YDS, with only slightly higher energy costs. The results observed are quite robust across the mix of workloads considered.

Our ongoing work is focusing on a prototype implementation of speed scaling using the Running Average Power Limit (RAPL) functionality on the Intel Ivy-Bridge architecture [25], [26], [27]. This implementation will facilitate direct experimental comparisons between coupled, decoupled, and turbocharged speed scaling.

ACKNOWLEDGMENTS

The authors thank the anonymous MASCOTS 2014 reviewers for their constructive feedback on our paper. The authors are extremely grateful to Caitlin Ryan for completing the simulation implementation of virtual batches and envelope-based turbocharging, which made the results in Section V-D possible. Financial support for this research was provided by Alberta Innovates – Technology Futures (AITF) in the Province of Alberta, and by the Discovery Grant (DG) and the Canada Research Chairs (CRC) program of the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] S. Albers, "Energy-efficient algorithms," *Commun. ACM*, vol. 53, no. 5, pp. 86–96, May 2010.

[2] L. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," in *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2010, pp. 37–48.

[3] M. Elahi, C. Williamson, and P. Woelfel, "Decoupled speed scaling: Analysis and evaluation," in *Proc. International Conference on Quantitative Evaluation of Systems (QEST)*, 2012, pp. 2–12.

[4] E. Friedman and S. Henderson, "Fairness and efficiency in web server protocols," in *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2003, pp. 229–237.

[5] M. Elahi, C. Williamson, and P. Woelfel, "Meeting the fairness deadline in speed scaling systems: is turbocharging enough?" *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 89–91, 2012.

[6] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, 1995, pp. 374–382.

[7] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Oper. Res.*, vol. 16, pp. 678–690, 1968.

[8] N. Bansal and M. Harchol-Balter, "Analysis of SRPT scheduling: investigating unfairness," in *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2001, pp. 279–290.

[9] M. Gong and C. Williamson, "Revisiting unfairness in web server scheduling," *Comput. Netw.*, vol. 50, no. 13, pp. 2183–2203, Sep. 2006.

[10] D. Raz, H. Levy, and B. Avi-Itzhak, "A resource-allocation queueing fairness measure," in *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2004, pp. 130–141.

[11] A. Wierman and M. Harchol-Balter, "Classifying scheduling policies with respect to unfairness in an M/GI/1," in *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2003, pp. 238–249.

[12] M. Harchol-Balter, K. Sigman, and A. Wierman, "Asymptotic convergence of scheduling policies with respect to slowdown," *Perform. Eval.*, vol. 49, no. 1-4, pp. 241–256, Sep. 2002.

[13] D. Snowdon, S. Petters, and G. Heiser, "Accurate on-line prediction of processor and memoryenergy usage under voltage scaling," in *Proc. ACM/IEEE International Conference on Embedded Software (EMSOFT)*, 2007, pp. 84–93.

[14] D. Snowdon, E. Le Sueur, S. Petters, and G. Heiser, "Koala: a platform for OS-level power management," in *Proc. ACM European Conference on Computer Systems (EuroSys)*, 2009, pp. 289–302.

[15] V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with PAPI," in *Proc. International Conference on Parallel Processing Workshops (ICPPW)*, 2012, pp. 262–268.

[16] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 1994.

[17] J. George and J. Harrison, "Dynamic control of a queue with adjustable service rate," *Oper. Res.*, vol. 49, no. 5, pp. 720–731, Sep. 2001.

[18] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *J. ACM*, vol. 54, no. 1, pp. 1–39, Mar. 2007.

[19] N. Bansal, K. Pruhs, and C. Stein, "Speed scaling for weighted flow time," in *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007, pp. 805–813.

[20] S. Albers and H. Fujiwara, "Energy-efficient algorithms for flow time minimization," *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007.

[21] A. Wierman, L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems: Optimality and robustness," *Perform. Eval.*, vol. 69, no. 12, pp. 601–622, Dec. 2012.

[22] N. Bansal, H.-L. Chan, and K. Pruhs, "Speed scaling with an arbitrary power function," in *Proc. ACM-SIAM Symposium on Discrete algorithms (SODA)*, 2009, pp. 693–701.

[23] D. Lu, H. Sheng, and P. Dinda, "Size-based scheduling policies with inaccurate scheduling information," in *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2004, pp. 31–38.

[24] I. Rai, G. Urvoy-Keller, and E. Biersack, "Analysis of LAS scheduling for job size distributions with high variance," in *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2003, pp. 218–228.

[25] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using rapl," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 13–17, Jan. 2012.

[26] T. Rauber and G. Runger, "Energy-aware execution of fork-join-based task parallelism," in *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012, pp. 231–240.

[27] V. Spiliopoulos, A. Sembrant, and S. Kaxiras, "Power-sleuth: A tool for investigating your program's power behavior," in *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012, pp. 241–250.