Decoupled Speed Scaling: Analysis and Evaluation

Maryam Elahi, Carey Williamson, Philipp Woelfel
Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, AB, Canada T2N 1N4
{bmelahi, carey, woelfel}@ucalgary.ca

Abstract—**In this paper, we introduce the notion of decoupled speed scaling, wherein the speed scaling function is completely decoupled from the scheduling policy used in a simple single-server computer system. As an initial result, we first demonstrate that the Fair Sojourn Protocol (FSP) scheduling policy does not work properly with coupled (native) speed scaling, but that it can and does work well with decoupled speed scaling. We then compare the performance of PS, SRPT, and FSP scheduling policies under decoupled speed scaling, and demonstrate significant advantages for FSP. Our simulation results suggest that it might be possible to simultaneously achieve fairness, robustness, and near optimality with decoupled speed scaling.**

I. Introduction

Many modern processors support dynamic speed scaling, which allows the CPU speed (e.g., clock frequency, voltage) to be adjusted dynamically via software control in the operating system [1]. The primary motivation for this feature is energy efficiency in the presence of highly-varying workloads. That is, the processor can be run at or near full speed to complete work quickly when demand is high, but be reduced to a low-power or quiescent mode when demand is low or absent.

The dynamic adjustment of the processor speed is often referred to as "CPU speed scaling" in the literature, though the technically precise term is Dynamic Voltage and Frequency Scaling (DVFS). Modern processors typically support on the order of a dozen discrete operating states, rather than just two (e.g., On/Off in gated speed scaling) or three (e.g., On/Sleep/Off for a typical laptop).

The performance implications of speed scaling designs are interesting, even in the single processor case. An early paper on this topic was by Weiser, Welch, Demers and Shenker in 1994 [2], who used simulation to explore the tradeoffs between response time and CPU energy consumption for empirical Unix workloads. This work pre-dated the widely-cited Yao, Demers, and Shenker (YDS) paper [3] on speed scaling in 1995, which provided the first formal treatment of the speed scaling problem, including several heuristic algorithms, as well as proofs that they were within a constant factor of optimal. The latter paper triggered substantial follow-on work by Albers [4], [5], Bansal [6], [7], [8], and others on improved algorithms, tighter bounds, and alternative metrics for evaluating speed scaling designs.

In speed scaling systems, the traditional performance metrics of throughput and response time become secondary to energy consumption, or a weighted combination of energy consumption and response time. The typical formulation[1] of the problem involves optimizing the total cost z, where:

$$z = E[T] + E[\varepsilon]/\beta. \tag{1}$$

In this expression, T represents response time, ε reflects energy cost, and β is a relative weighting factor.

Andrew, Lin and Wierman [10] presented an intriguing paper on this topic in ACM SIGMETRICS 2010. In their work, the authors demonstrate that there are inherent tradeoffs between optimality, fairness, and robustness in speed scaling systems. In particular, they prove that it is possible to provide *any two* of these properties simultaneously, but not all three. For example, Shortest Remaining Processing Time (SRPT) scheduling with dynamic speed scaling can provide optimal total cost, and is robust to uncertainties in the workload estimation, but it is unfair to large jobs. Conversely, Processor Sharing (PS) with dynamic speed scaling is fair to all jobs, and robust, but its mean response time and energy consumption can be much worse than SRPT.

In our paper, we further investigate the tradeoffs inherent in speed scaling systems. In particular, we consider *decoupled speed scaling*, wherein the speed of the system is determined by an external speed scaling function that is agnostic about the current state of the system under a particular scheduling policy, and only depends on the incoming jobs and their sizes. This approach differs from the well-studied job-count-based speed scaling (coupled speed scaling), where the speed is determined dynamically based on the current state of the system, namely the number of jobs remaining in the system.

In particular, our speed scaling function is determined by a virtualized execution of a reference scheduling policy, such as PS, in the background. While this "virtual PS" idea has been used previously in the design of scheduling policies such as Weighted Fair Queueing (WFQ), Generalized Processor Sharing (GPS), and FSP, to the best of our knowledge we are the first to apply this principle to the speed scaling function itself. Furthermore, our approach is not restricted to using PS as the reference scheduler.

This idea is conceptually elegant, and simplifies the analysis of speed scaling systems. In particular, it facilitates the analysis of additional scheduling policies, including the Fair Sojourn

[1] An alternative cost function that is starting to receive more attention lately is the *energy-delay product* [9]. Because energy consumption is invariant under our definition of decoupled speed scaling, our results are directly applicable to either formulation of the weighted cost model.

Protocol (FSP) [11], and relaxes some of the restrictive assumptions in [10] regarding the structure of speed scaling systems (i.e., "natural" speed scaling).

This notion of decoupled speed scaling enables us to fix the speeds and then compare the behaviors (e.g., response time, fairness) of scheduling policies on a level playing field. The resulting system has time-varying capacity, similar to stochastic capacity systems, which are well-studied in the literature (e.g., [12]). However, there are fundamental differences as well. In decoupled speed scaling, the capacity of the system varies deterministically, rather than stochastically. Furthermore, for a given speed scaling function, the CPU speed is identical at any point in time for any scheduling policy that governs the system. Thus the energy component of the cost function becomes equal for the policies under comparison, and the sole remaining focus is on the response time $E[T]$.

Our paper makes several key contributions. First, we define and propose a new approach to speed scaling system design, which we call decoupled speed scaling. Second, we show that all policies are efficient in coupled speed scaling systems based on job count. Third, we show that the FSP scheduling policy is ill-suited for job-count-based coupled speed scaling, but that it can and does work well with decoupled speed scaling. Finally, using simulation, we compare the performance of PS, SRPT, and FSP scheduling policies under coupled and decoupled speed scaling. Our simulation results demonstrate pronounced advantages for FSP, which lead us to speculate that it might be possible to attain fairness, robustness, and near optimality with decoupled speed scaling. Note that the latter result does not directly contradict the results in [10], since the underlying assumptions about speed scaling differ.

The remainder of the paper is organized as follows. Section II presents a brief description of prior related work. Section III provides a simple pedagogical example to help motivate our work. Section IV presents our formal system model, while Section V and Section VI present our theoretical results for coupled and decoupled speed scaling systems, respectively. Section VII presents simulation results to validate the models. Finally, Section VIII concludes the paper.

II. Background and Related Work

Our paper builds upon substantial prior work in scheduling and speed scaling system design. In this section, we provide a concise summary of prior related work, particularly that most relevant to our current paper.

A. Scheduling Policies

Processor Sharing (PS) is a well-known scheduling policy that epitomizes fairness [13], [14]. PS shares a single processor simultaneously among n active jobs in the system by devoting a service rate of $1/n$ to each job. As jobs arrive and depart, PS dynamically adjusts the service rate provided to each job, while the aggregate rate is always fixed at unity.

Shortest Remaining Processing Time (SRPT) is a preemptive scheduling policy that optimizes mean response time. Using advance knowledge of job size information, the SRPT policy always selects for service the job that has the least remaining service time. With this approach, the mean waiting time and the mean response time are minimized [15], [16], [17], [18]. SRPT scheduling has generated significant research interest, particularly in the context of request scheduling in Web servers [19], [20], [21], [22], [23]. Under certain job size distributions, SRPT even has a counter-intuitive "all can win" property, where all jobs prefer SRPT to PS [19]. However, this policy is sometimes unfair [24]. Furthermore, the performance of SRPT can deteriorate if it does not have accurate job size information [25].

Several other scheduling policies attempt to exploit the response time advantage of size-based scheduling, while also considering fairness and practical issues. Examples include Foreground-Background (FB) [26], Least Attained Service (LAS) [27], Resource Allocation Queueing Fairness Measure (RAQFM) [28], and the Fair Sojourn Protocol (FSP) [11]. FB and LAS approximate the effectiveness of SRPT, without the need to know job sizes in advance. RAQFM balances fairness based on the size and seniority of a job. The FSP scheduling policy combines aspects of PS and SRPT. It 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 event. To do so, the FSP algorithm conceptually runs a "virtual PS" queue in the background, and recomputes its next scheduling decision upon each job arrival or departure event [11].

The FSP policy is a central focus in our work. In the non-speed-scaling world, FSP (provably) has several desirable properties [11], including strict dominance[2] over PS. Specifically, in the execution of any job on any sample path, no job is worse off (in terms of response time) under FSP than it is under PS. One of the goals in our paper is to integrate FSP into speed scaling systems.

B. Speed Scaling Systems

The tradeoffs between energy consumption and performance metrics such as mean response time have stimulated extensive work on energy-efficient algorithms [5], as well as dynamic service rate control [29] and power management [30]. Yao, Demers and Shenker [3] 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 or temperature budget [6].

A more recent approach aims at optimizing the cost function in Equation (1), which is a linear combination of the energy consumption and the average response time [4]. Several studies on this metric suggest that the optimal dynamic speed scaling function should be a function of n, the number of jobs in the system. Later, Bansal, Chan and Pruhs [8] showed that SRPT with the speed scaling function $P^{-1}(n+1)$ is 3-competitive for

[2]While this criterion may seem overly restrictive, it is an attractive property that is amenable to formal analysis. We leave to future work the consideration of alternative (weaker) forms of dominance, and the analysis of more general scheduling policies.

an arbitrary power function P. Andrew, Lin and Wierman [10] show that SRPT with speed scaling function $P^{-1}(n\beta)$ is 2-competitive, and is optimal among the class of "natural" speed scaling functions.

Other researchers have raised concern about the cost function in Equation (1). Although it directly reflects changes in the mean response time and energy consumption, it does not reflect the relative magnitude of these changes. The energy-delay product is an alternative cost function that emphasizes relative change. This metric has received attention in the recent literature [9].

Fairness in dynamic speed scaling designs was first formally studied by Andrew *et al.* [10]. In their model, they assume that the rate of the server is determined as a function of the number of jobs in the system (We call this job-count-based speed scaling). In this model, the slowdown value [31] for PS at load ρ is no longer $1/(1-\rho)$, but the value is still a constant for all job sizes (see Proposition 15 in [10]). They argue that the slowdown of PS remains the right criterion for fairness, and use a similar definition for fairness as introduced for the non-speed-scaling world [19].

Definition 1: A policy p is fair if for all job sizes x

$$\frac{E[T_p(x)]}{x} \leq \frac{E[T_{PS}(x)]}{x} \tag{2}$$

Based on Definition 1, Andrew *et al.* show that speed scaling exacerbates unfairness under SRPT, as well as for non-preemptive policies such as FCFS. This definition, however, solely considers the response time of jobs and does not take energy consumption into account. In Section V-B, we formally prove that PS is efficient under this model, and then conjecture that no policy can be fair with this definition (see Theorem 9 and Conjecture 10). However, if we change the model and allow a decoupled speed scaling function, we show that FSP strictly dominates PS, and is also fair, based on Definition 1 (see Section VI).

III. Motivating Example

The purpose of this section is to establish the intuition behind the concept of decoupled speed scaling. In our work, we explore the role of scheduling in speed scaling systems, and in particular the tradeoffs between efficiency, fairness, and cost. Efficiency refers to how closely the system achieves optimality, such as minimizing response time. Fairness refers to whether jobs with different characteristics (e.g., size) are treated similarly or not. Cost refers to the aggregate energy consumption for the system when executing a given workload, or a combination of metrics as in Equation (1).

To illustrate the issues, consider a simple single processor system, initially empty, to which a small set of jobs arrive, as shown in Table I. We use a simple discrete-event simulation model of this system, with configurable scheduling policies, and adequate instrumentation to record job response times and speed scaling dynamics. We use this example to motivate and explain the ideas behind decoupled speed scaling.

Figure 1 shows an example of our simulation results for traditional coupled speed scaling, using the workload in Table I.

TABLE I
Example Workload

Job ID	Arrival Time	Job Size
1	1.0	5
2	2.2	2
3	2.8	3
4	3.5	1
5	4.7	4

The top row of graphs shows the instantaneous number of jobs in the system, for each of three scheduling policies (FCFS, PS, and SRPT, from left to right). The second row of graphs shows the instantaneous amount of work remaining in the system, as well as the job departure points. In these examples, the CPU speed is scaled linearly[3] based on the number of active jobs in the system.

Multiple insights emerge from studying Figure 1. The first (and most obvious) observation is that the three scheduling algorithms produce different profiles for the number of jobs in the system. Since the three policies complete the jobs at different times and in different orders, they provide different response times. This is in fact the main rationale for different scheduling policies. The second observation is that the profiles for remaining work are quite similar across the three scheduling policies. This result makes sense since the workload (i.e., job arrival time, job size) presented to each scheduler is the same. However, there are some perceptible differences (e.g., slopes, departure points), since the speed scaling decisions (based on the number of active jobs) differ across policies. The third observation from Figure 1 is that the scheduling algorithms incur different energy costs. For example, FCFS runs the CPU at a moderate speed for most of the simulation, while SRPT tends to run the CPU at a lower speed, since there are fewer jobs in the system. If power consumption is proportional to the square of the CPU speed (a typical modeling assumption in the literature), then scheduling algorithms such as SRPT might be advantageous. However, the fourth and final observation is that SRPT is not optimal for completion time. While SRPT finishes jobs quickly, maintains fewer jobs in the system, and runs the CPU at a lower speed, it also needs to run longer to complete all of the jobs. FCFS and PS scheduling both finish sooner, in this particular example. While this example is small, it does illustrate the basic tradeoffs between response time, fairness, and energy cost.

For comparison, Figure 2 presents results for decoupled speed scaling, in which the speed scaling function used by a scheduler is provided externally. In these examples, the speed scaling function chosen is that derived from PS scheduling (i.e., the CPU speeds implied by Figure 1(b)). That is, the speed that an arbitrary scheduling policy p uses at time t is the same as the speed that PS would be using if it was executing

[3]Our results also hold for other job-count-based speed scaling policies, including the square root of the number of jobs in the system (a recommended policy from the literature). The assumption of linear scaling here merely simplifies the pedagogical and graphical presentation of the example.

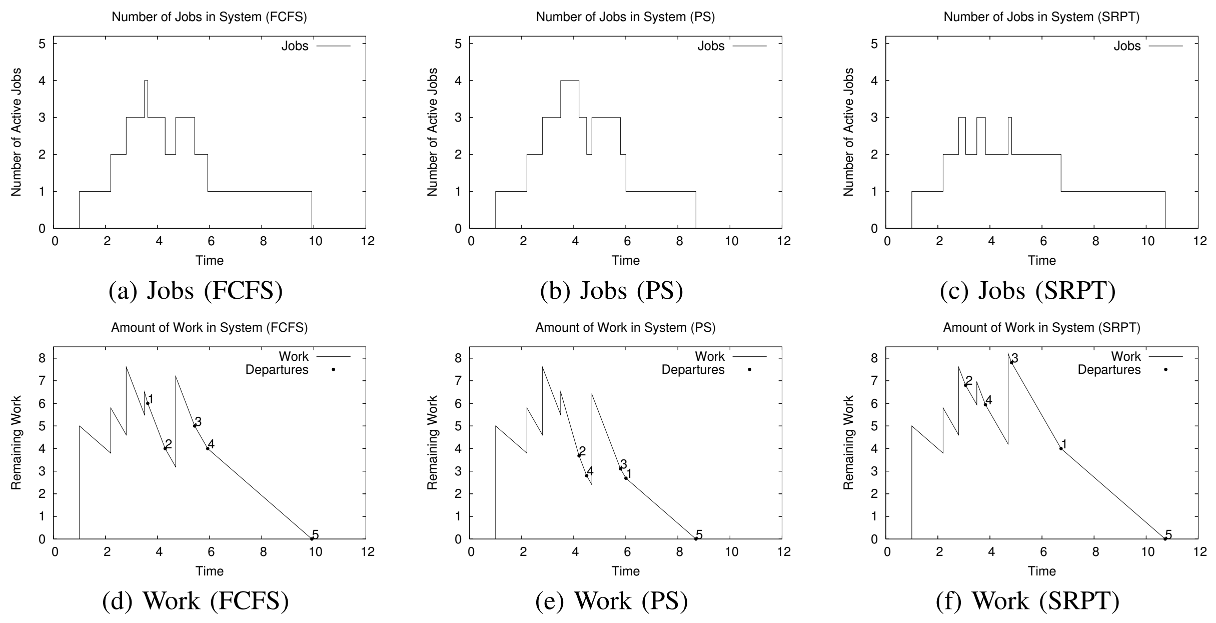


Fig. 1. Example Results for Dynamic Speed Scaling (Coupled)

TABLE II
COMPARISON OF COUPLED AND DECOUPLED SPEED SCALING

Item	Coupled Speed Scaling	Decoupled Speed Scaling
CPU speed	Depends on current number of jobs in system	Depends on external function (e.g., occupancy of virtual PS system)
Busy period duration	Depends on scheduling policy	Same for all work-conserving scheduling policies
Energy consumption	Depends on scheduling policy	Same for all work-conserving scheduling policies
Mean response time	Depends on scheduling policy	Depends on scheduling policy
Fairness	Depends on scheduling policy	Depends on scheduling policy

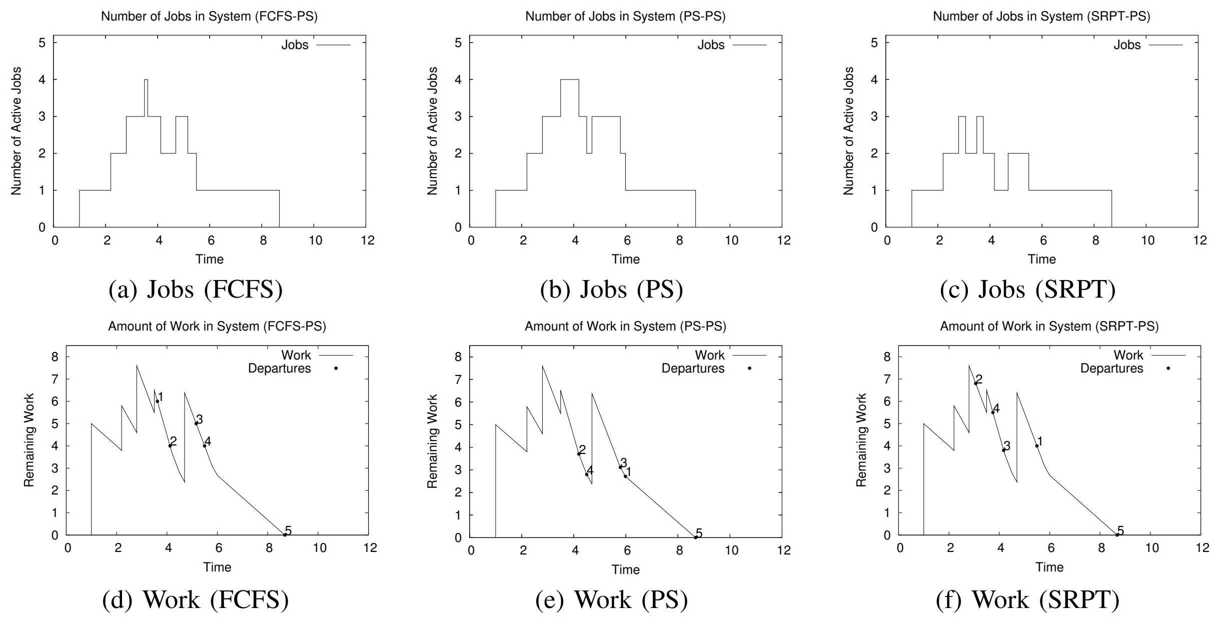


Fig. 2. Example Results for Dynamic Speed Scaling (Decoupled, PS-driven)

on the same workload. (This requires a simple "virtual PS" background computation, similar to FSP.) We select PS as the reference speed scaling function because of its well-defined fairness properties (i.e., egalitarian treatment of jobs).

Figure 2 highlights four new observations. First, the behavior of PS scheduling in Figure 2 is preserved (cf. Figure 1) when using its own speed scaling function. This result is straightforward. Second, the results for FCFS and SRPT scheduling are transformed when they are driven by the external speed scaling function of PS. In particular, all three scheduling policies now have exactly the same busy period structure and duration. That is, they complete the same amount of work in the same amount of time, since they are work-conserving policies. Third, all three scheduling policies consume exactly the same amount of energy, since the CPU speed is adjusted to the same rates at the same times as in PS scheduling. Fourth, the departure points of jobs differ across these three scheduling policies. Thus, the mean job response times are different.

This simple example demonstrates that with decoupled speed scaling, the response times of jobs can be altered without changing the energy consumption. That is, we can alter $E[T]$ in Equation (1) without affecting $E[\varepsilon]$. It is shown that SRPT scheduling is optimal for mean response time [16], [18]. However, we have not yet determined the optimal speed scaling function to use. In fact, PS may not be the best external speed scaling function to use. For instance, using FCFS (i.e., Figure 1(a)) as the speed scaling function on this example workload would reduce energy consumption by 16%, while increasing the response time of SRPT scheduling by 17%. Similarly, using SRPT (i.e., Figure 1(c)) as the speed scaling function would reduce energy consumption by 30%, while increasing SRPT response time by 36%. Choosing suitably among these (or other) configurations would depend on the value of β in Equation (1).

Table II summarizes our main observations about coupled and decoupled speed scaling. The key insight is that decoupled speed scaling enables "apples to apples" comparisons between scheduling policies under fixed energy consumption. Furthermore, the external speed scaling function can be very general (e.g., constant, derived from a particular scheduling policy, or completely contrived). This approach provides great flexibility for the analysis and evaluation of speed scaling designs.

IV. System Model

We consider a single-server queue with dynamically adjustable, continuous, and unbounded service rates. For most of the results, we consider an arbitrary arrival process and arbitrary job sizes. A *sample path* is a sequence of tuples specifying job arrival times and job sizes.

Let $r(t)$ be the rate of the system at time t, and let $P(r)$ denote the power required to run at rate r. Then the total energy consumed by the system by time t is

$$\int_0^t P(r(\tau))d\tau. \tag{3}$$

A speed scaling function specifies the value of $r(t)$. For coupled speed scaling, the rate of the system at time t can be determined by the entire system state at time t, and thus is influenced by the scheduling policy. 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 online scheduling policies $\mathcal{P}$. Online means that scheduling decisions of a policy $p \in \mathcal{P}$ 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 consider a preempt-resume model, where a job may be preempted and later resumed without any context-switching overhead. In particular, we consider the behavior of the following scheduling policies:

- *Processor Sharing (PS)*: At each point in time, equal service is given to all jobs in the system.
- *Shortest Remaining Processing Time (SRPT)*: At any point in time, service is given to the job with the least remaining work in the system.
- *Fair Sojourn Protocol (FSP)*: The times at which jobs complete under PS is computed, and then full service is devoted to the job with earliest PS completion time.

The *response time*, T, of a job is the time between its arrival to the system and its departure from the system (also known as turnaround time, sojourn time, or flow time). The *energy consumption* of a job is denoted by ε. This value depends on the CPU speeds used during the execution of the job, and the time spent executing at each speed, as given by Equation (3). Our goal is to minimize the *total cost*, which is the linear combination of response time and energy consumption as in Equation (1).

Let $T_p(\sigma)$ denote the random variable representing the response time under policy p, for a job chosen at random from sample path σ. Similarly, $T_p(x,\sigma)$ and $S_p(x,\sigma) = T_p(x,\sigma)/x$ denote the response time and slowdown [31], [32], respectively, under policy p for a job chosen at random from all jobs of size x on sample path σ. For brevity, we omit σ and write T_p, $T_p(x)$, $S_p(x)$, and ε_p, when the sample path is apparent from the context.

Table III summarizes the notation used in our paper.

A. Efficiency and Fairness

To analyze policies without any distributional assumptions, we use the following definitions of dominance and efficiency, as previously presented in [11]. Later, we extend the definitions to facilitate our analysis with distributional models.

Definition 2: 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 3: A scheduling policy p is efficient if there is no other policy p' that dominates it.

TABLE III
SUMMARY OF NOTATION

Symbol	Description
σ	A sample path, which is a sequence of tuples specifying job arrival times and job sizes
$n_p(t,\sigma)$	Number of jobs in the system at time t for policy p on sample path σ
$s(n_p)$	Speed scaling function (job-count-based) that specifies the rates under policy p
$s_p(t,\sigma)$	Speed used by policy p at time t on sample path σ
T	Response time
$T(x,\sigma)$	Random variable representing response time for a job chosen at random from all jobs of size x on sample path σ (We write $T(x)$ instead of $T(x,\sigma)$ if it is clear from the context which sample path we mean.)
$S(x,\sigma)$	Random variable representing slowdown $T(x,\sigma)/x$ (We write $S(x)$ instead of $S(x,\sigma)$ if it is clear from the context which sample path we mean.)
ε	Energy consumption
β	Weighting factor
z	Total cost for speed scaling system
$E[\cdot]$	Expectation for specified performance metric
$P(r)$	Power required to run at speed r
p-q	An arbitrary scheduling policy p driven by the speed scaling function obtained from a reference scheduler q with the job-count based speed scaling function P^{-1}
z_p^s	Total cost for scheduling policy p with the speed scaling function s

Definition 4: A scheduling policy p' strictly dominates policy p if, for every busy period on every sample path, every job (except the final one of the busy period) completes strictly earlier under p' than under p.

Definition 5: A strict performance measure is a function π that maps scheduling policies to real numbers, such that if policy p' dominates p, then $\pi(p') < \pi(p)$.

As stated in [11], average response time and average slowdown are examples of strict performance measures. Specifically, over any finite time interval, the weighted average of response times constitutes a strict performance measure, provided that non-zero weights are used for every job.

These definitions can be used in the analysis of the expected response times of policies when speed scaling is used. However, note that dominance may only provide a partial ordering of policies, so not all policies are directly comparable using these definitions. Also, these definitions focus solely on response time as the relevant metric, and do not incorporate energy consumption.

Another metric of interest is slowdown, which is particularly relevant in the definition of fairness (see Definition 1). To further study fairness in speed scaling designs, we extend the notion of dominance and efficiency to slowdown as well.

Definition 6: The slowdown of scheduling policy p' dominates the slowdown of policy p, if for any given sample path σ, and for all job sizes x, $E[T_{p'}(x,\sigma)] \leq E[T_p(x,\sigma)]$, and there exists a sample path σ' for which there exists a job size x such that $E[T_{p'}(x,\sigma')] < E[T_p(x,\sigma')]$.

Definition 7: A scheduling policy p is slowdown efficient if there is no other policy p' whose slowdown dominates the slowdown of p.

V. Coupled Speed Scaling Results

In this section, we consider coupled (native) speed scaling functions that are based on the number of jobs in the system, as illustrated in our earlier example in Figure 1. In this model, the rate of the server is decided by a job-count-based speed scaling function s, which is a strictly increasing function of the number of jobs in the system. Let $n_p(t,\sigma)$ denote the number of jobs in the system at time t, when using policy p on sample path σ. Then the rate of the system at time t is given by $s(n_p(t,\sigma))$, where s is a mapping $\mathbb{N} \cup \{0\} \to [0,\infty)$ that is strictly increasing. In the rest of the paper, we use $s(n_p)$ to refer to the coupled job-count-based speed scaling function for policy p.

In this model, our key result is that no policy can dominate another one. We start by showing that FSP's properties are not preserved in this speed scaling model, and then formally prove that all policies are efficient.

A. FSP with Job-count Speed Scaling

In non-speed-scaling systems, the FSP policy is interesting because of its strict dominance over PS. Specifically, no job finishes later under FSP than under PS, and all jobs (except those ending busy periods) finish strictly earlier under FSP than under PS.

A natural question is whether we can apply FSP in speed scaling systems. Through the following examples, we demonstrate that the answer to this question is no. For simplicity, the examples assume simple job-count-based speed scaling of the form $s(n) = n$, though the observations apply more generally for other speed scaling functions.

There are two fundamental problems that arise when FSP is naively ported to speed scaling systems. First, the dominance of FSP over PS is not preserved with job-count-based speed scaling. Second, the standard implementation of FSP is undefined in some scenarios.

To understand the first issue, consider a trivial sample path with two jobs, each of unit size, which both enter the system at time 0. Under PS scheduling, the CPU runs at rate 2, and both jobs complete simultaneously at time 1. Under FSP scheduling, one (arbitrarily chosen) job is granted exclusive full service at rate 2 until it completes at time 0.5, and then the other job (the sole remaining job in the system) receives full service at rate 1 until it completes at time 1.5. Since the latter job completes later under FSP than it does under PS, the strict dominance property of FSP is violated. One possible solution to this problem is to run the second job at rate 2 (i.e., the rate that PS would have used for this workload). We call this solution *decoupled speed scaling*, and present its analysis in Section VI. A second solution is to scale all CPU speeds by 50% in this example, so that the final job completes no later than under PS. We call the latter *turbocharging*, and defer its analysis to future work.

The second issue is subtle, yet even more fundamental to the operation of FSP, and requires a slightly more elaborate example. Consider a sample path with four jobs, each of size 4, and all arriving at time 0. Under PS, all four jobs receive concurrent service, with aggregate rate 4, and finish simultaneously at time 4. Under FSP, the first job receives full service at rate 4 and finishes at time 1. Then the second job receives full service at rate 3, and finishes $\frac{4}{3}$ time units later at time $2\frac{1}{3}$. Next, the third job receives full service at rate 2, and finishes 2 time units later at time $4\frac{1}{3}$. Note that this completion time is later than it finishes under PS, and thus the strict dominance property of FSP is again violated. Even more interesting, the fourth and final job never receives any service under FSP, since the "virtual PS" queue used to drive FSP's decision-making (see the original algorithm in [11]) contains *no jobs* at time $4\frac{1}{3}$. This anomaly arises because the start time of the fourth job under FSP is beyond its point of completion under PS. Hence the FSP policy is ill-defined in this scenario for job-count-based speed scaling.

The obvious question is whether we can devise another algorithm to achieve the properties of FSP (namely, efficiency and strict dominance over PS) in speed scaling systems. In the following subsection, we show that *no policy can dominate any other policy* in the job-count-based speed scaling model. Later in the paper (Section VI), we show that the standard implementation of FSP is well defined and it dominates PS, when speed scaling is decoupled.

B. Efficiency with Job-count Speed Scaling

We next establish that with coupled job-count-based speed scaling, no policy can dominate any other one.

In the following, let $s_p(t, \sigma)$ denote the CPU speed at time t under scheduling policy p on sample path σ.

Lemma 8: With job-count-based speed scaling, if policy p dominates policy p', then for any sample path provided to both p and p', $s_p(t, \sigma) \leq s_{p'}(t, \sigma)$ at any time t.

Proof: Consider a policy p that dominates p', and let σ be some arbitrary sample path. We need to show that the rate used by p at time t never exceeds the rate used by p' at the same time t. For the purpose of a contradiction, suppose there is a time t for which $s_p(t, \sigma) > s_{p'}(t, \sigma)$. Since s is strictly increasing in n, the preceding inequality implies that $n_p(t, \sigma) > n_{p'}(t, \sigma)$. Since both policies were provided with the same sample path, the latter inequality implies that at time t, there must be at least one job that has completed under p', yet is still in the system under policy p. This contradicts the assumption that p dominates p'. ■

Theorem 9: With job-count-based speed scaling, all policies are efficient.

Proof: To prove that all policies are efficient, we show that no policy p can dominate another policy p'. We again proceed with a proof by contradiction.

Suppose there is a policy p that dominates p'. Then there exists a sample path σ such that all jobs complete under p no later than under p', and at least one job finishes earlier under p than under p'. Let t denote the first point in time when a job j on σ completes under p, while under p' it finishes at some time $t' > t$. In other words, all jobs completed before time t leave the system at exactly the same time under both p and p'. This implies that at every point in time until time t, the same number of jobs are in the system under p and p'. Hence, the system rate is equal under both policies.

Note that under either policy job j cannot be the very last job to leave the system. If j were the last job to leave under p', then p' would devote full service to j immediately after the departure preceding t. This would imply that j leaves the system under p' no later than t, contradicting the assumption. If j were the last job under p then it would also be the last job under p', leading to the same contradiction.

At time t, the number of jobs in the system under p decreases by 1, because of the departure of job j, while the number of jobs in the system under p' does not change. Since up to time t no job has finished sooner under p' than under p, at time t the number of jobs in the system under p becomes strictly less than the number of jobs under p', and will remain so throughout the time interval $[t, t')$. (Note that by the assumption that p dominates p', at any point when some job $j' \neq j$ leaves p', that job will also leave or already have left p.) Since the speed scaling function is strictly increasing in the number of jobs in the system, the rate under policy p is less than the rate under p' throughout $[t, t')$. Consequently, the service rate under p is strictly less than the service rate under p' throughout $[t, t')$.

Now, let t^* be the end of p's busy period containing job j. Since, by Lemma 8, the service rate of p never exceeds the rate of p', and throughout $[t, t')$ the service rate of p is strictly lower than the service rate of p', the average service rate of p over the interval $[0, t^*]$ is strictly lower than the average service rate of p' over the same interval. Hence, the total amount of work processed under p in $[0, t^*]$ is less than the total amount of work processed under p' in the same interval, and thus under p' all jobs that arrive in $[0, t^*]$ finish strictly before t^*. This contradicts the assumption that under p no job completes later than under p'. ■

C. Fairness in Job-count Speed Scaling

We showed that all policies, including PS, in the job-count-based speed scaling model are efficient. According to the definition of fairness by Friedman and Henderson [11], a policy is fair if it weakly dominates PS. Therefore, no other policy p can be fair in this model. However, if we consider Definition 1, we also want to see if PS is slowdown efficient (see Definition 7).

As stated below, we conjecture that PS is slowdown efficient for an M/GI/1 queue. Any policy that improves the response time of a particular class of jobs eventually reduces the average speed of the system, and therefore degrades the response time of other jobs (i.e., those already in the system, and any new arrivals). If a policy keeps more jobs in the system than PS, in order to increase the rate, then it must be keeping jobs in the system longer than PS, so their response time degrades.

We summarize our conjecture and its implications here:

Conjecture 10: For an M/GI/1 queue with job-count-based rates, PS is slowdown efficient.

Andrew *et al.* [10, Proposition 15] showed that the slowdown of PS with speed scaling remains constant.

Proposition 11: In an $M/GI/1$ queue with controllable rates, for any symmetric policy p,

$$E[T_p(x)] = x(E[T_p]/E[X]). \tag{4}$$

Now suppose Conjecture 10 is true. Then there is no policy p such that $\max_x E[T_p(x)]/x < E[T_{PS}]/E[X]$. Therefore, we obtain the following:

Corollary 12: For an $M/GI/1$ queue with job-count-based rates, if PS is slowdown efficient, then

$$\min_p \max_x \frac{E[T_p(x)]}{x} = \frac{E[T_{PS}(x)]}{x} = (E[T_{PS}]/E[X]). \tag{5}$$

VI. Decoupled Speed Scaling Results

In this section, we return our focus to decoupled speed scaling, which was motivated and explained using the example in Figure 2. Note that decoupled speed scaling is "unnatural", based on the definition in [10]. That is, it is possible for the CPU speed to change during the execution of a job, even though no other job arrivals or departures occur. It is also possible for the CPU speed to remain constant while several arrivals or departures occur. Examples of these phenomena are evident in Figure 2, particularly for job 5 under either FCFS or SRPT scheduling. In general, the monotonicity property inherent in coupled speed scaling is no longer guaranteed to hold with decoupled speed scaling.

In this decoupled speed scaling model, we show that FSP works, and that it dominates PS. Specifically, if the implementation of FSP with speed scaling runs both the FSP queue and the virtual PS queue at the same speed (which could be obtained from PS with job-count-based speed scaling or by some other external speed function), then the efficiency and strict dominance of FSP over PS are preserved.

A. Efficiency of FSP

We first note that FSP has the "no reversals" property, which was observed in [11] for the single speed model, and also in systems with speed scaling:

Lemma 13: For a fixed sample path in an FSP system, let t be a point in time at which two jobs A and B are in the system, and job A receives service. Then job B receives no service until job A is completed. In particular, job A finishes before job B.

Proof: Recall that at any point in time FSP serves the job that will finish earliest under PS. Note that this is the job that has the least amount of remaining work under PS. (We assume there are no ties, but it is straightforward to extend the proof to handle ties. For example, if there is a tie, PS gives preference to the job with the earliest arrival time.) Therefore, at time t, under PS the remaining work of job A is less than that of job B. From the fact that PS gives an equal share of service to all jobs in the system it is immediate that after t and at least until A is finished, the remaining work of A will be less than that of B. Hence, FSP does not service B until A is finished. ■

Theorem 14: No scheduling policy p dominates FSP if both use the same decoupled speed function (i.e., FSP is efficient for decoupled speed scaling).

Proof: Fix some sample path σ, such that no job under p finishes later than under FSP. Let $r(t)$ be the rate of the system at time t, under both FSP and p. We show that no job finishes earlier under p than under FSP.

Let $J_1, J_2, \ldots$ be the jobs in the order in which they finish under FSP. Assume J_i arrives at time a_i and has size w_i, and departs at time d_i under FSP. Let T_i^q be the set of real numbers so that at every time $t \in T_i^q$, job J_i receives service under policy q. (T_i^q can be viewed as the union of one or more time intervals.) We show by induction on i that $T_i^{FSP} = T_i^p$ for all $i = 1, 2, \ldots$. Therefore, all jobs finish at the same time under both protocols.

For $i = 1$, the claim follows from the fact that, by Lemma 13, the first job to finish under FSP, J_1, must receive uninterrupted service beginning with its arrival. Clearly, the same is true for J_1 under p, or else J_1 would finish later under p than under FSP. Therefore, $T_1^p = T_1^{FSP} = [a_1, d_1]$.

Now suppose the hypothesis is true for $J_1, ..., J_{i-1}$. Due to Lemma 13, the only jobs that under FSP can receive service between the arrival and departure of job J_i are jobs $J_1, ..., J_{i-1}$ (Otherwise, if some other job J^* received service while J_i was in the system, J^* would finish before J_i). Hence, we have $T_i^{FSP} = [a_i, d_i] - (T_1^{FSP} \cup T_2^{FSP} \cup \ldots \cup T_{i-1}^{FSP})$. This implies

$$w_i = \int_{\tau \in T_i^{FSP}} r(\tau) d\tau. \tag{6}$$

Since $T_{i'}^{FSP}$=$T_{i'}^p$ for $1 \leq i' < i$, we know that under p no job in $J_i, J_{i+1}, \cdots$ gets any service in $T_1^p \cup \cdots \cup T_{i-1}^p$. Therefore, the maximal amount of work that can be done on other jobs during the time interval $[a_i, d_i]$ is $\int_{\tau \in T_i^{FSP}} r(\tau) d\tau$, which, by (6), is w_i. Hence, the only way job J_i can get enough service during the interval $[a_i, d_i]$ to complete is if T_i^{FSP}=T_i^p. ■

Theorem 15: FSP strictly dominates PS if both use the same decoupled speed scaling function.

Proof: The proof is basically the same as the proof of Theorem 3 presented in [11]. The only modification is in the inductive step. The amount of work done between the m^{th} and $(m+1)^{th}$ event is not just the elapsed time between the two events, but the integral over time of the speeds used for execution. The fundamental result does not change since FSP mirrors the same speeds as PS. ■

B. Analysis of Cost under FSP with PS Job-count-based Speeds

In this section, we study the cost of FSP, as defined in Equation (1). In particular, we compare $z_{FSP}^{s(n_{PS})}$ and $z_{PS}^{s(n_{PS})}$, which are the cost of running PS with coupled speed scaling, and the cost of running FSP with the same speeds as PS, i.e., $s(n_{PS})$.

When FSP runs at the speeds of PS, both systems use the same amount of energy. So to compare the cost of FSP with PS, we only need to consider their average response time.

As described in Section IV-A, average response time is a *strict performance measure* (Definition 5). Based on Theorem 15, if FSP runs at the speeds of PS, then FSP dominates PS, so $E[T_{FSP}] \leq E[T_{PS}]$. Therefore we have:

$$z_{FSP}^{s(n_{PS})} \leq z_{PS}^{s(n_{PS})} \tag{7}$$

This result is significant for two reasons. First, it indicates that FSP is superior to PS with respect to cost in decoupled speed scaling systems. Further work is needed to determine whether FSP is itself optimal, either for scheduling, speed scaling, or overall cost. Second, the result raises a question as to whether PS is the right comparison point for fairness in speed scaling designs. Again, further work is required to investigate this issue.

VII. Simulation Results

This section presents simulation results validating the formal theoretical results established in this paper.

The simulation results are obtained using a custom-built discrete-event simulator written in C++. The simulator has configurable policies for scheduling (e.g., FCFS, PS, SRPT, FSP, LRPT) and speed scaling (e.g., coupled, decoupled).

The input to the simulator consists of a two-column workload file specifying the arrival time and size of each job, as in Table I. The arrival times are generated by a Poisson process and job sizes are exponentially distributed. We have observed qualitatively similar results for other job size distributions that we have simulated (e.g., the uniform distribution).

The mean job arrival rate is $\lambda = 1$ job per time unit. In the graphs that are presented, the mean job size is 80 units, though Table IV shows results for other workload settings. In all cases, the default service rate capacity of the system being modeled is 100. Thus the default workload represents 80% offered load in the traditional (non-speed-scaling) sense. However, the speed scaling system is inherently dynamic, adjusting its (unbounded) service capacity to deal with instantaneous workload demands. We use the square root of the number of active jobs as the default policy for job-count-based speed scaling.

The primary performance metrics reported are job response time and slowdown (i.e., normalized response time). The latter metric is plotted versus job size to illustrate the efficiency, fairness, and convergence properties of different scheduling and speed scaling policies. The results are reported for 100,000 consecutive jobs from within a simulation run length of 1,000,000 jobs, so that simulation warmup and cooldown effects have minimal impact.

Figure 3 shows the first set of simulation results, for coupled speed scaling. By design, the graphs are arranged similarly to Figure 8 in [10], to provide validation of our simulation environment. In Figure 3(a) the horizontal axis shows the main body of the (exponential) job size distribution, while in Figure 3(b), the horizontal axis represents relative job size within the empirical CDF of the full job size distribution. In both graphs, the vertical axis shows the mean slowdown as a function of job size. Note that job sizes are binned non-uniformly across the data to dampen statistical noise and improve visual acuity. Bins with fewer than 30 sample jobs are not plotted in Figure 3(a), while the full observed distribution is plotted in Figure 3(b).

Figure 3 clearly demonstrates the unfairness of the SRPT scheduling policy under coupled speed scaling. SRPT neither dominates PS, nor is dominated by PS. In particular, SRPT provides a pronounced slowdown advantage for small jobs, similar slowdown for intermediate-sized jobs, and worse slowdown for larger jobs. These results also demonstrate that dynamic speed scaling exacerbates the unfairness of SRPT scheduling [10]. The unfairness is most pronounced for jobs in the upper 10% of the job size distribution.

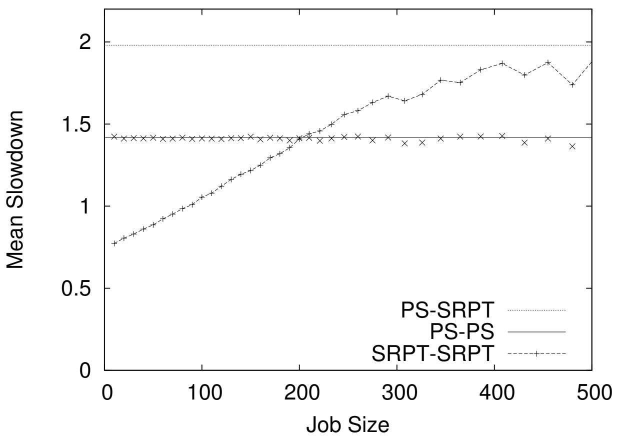
Our results in Figure 3 are qualitatively and quantitatively similar to those in [10]. For example, the slowdown of SRPT for large jobs remains above that of PS, and does not exhibit the "asymptotic convergence of slowdown" property from the non-speed-scaling world [32]. However, one additional empirical observation from our simulation results is that the unfairness of SRPT for large jobs appears to be upper bounded by the results for PS-SRPT with decoupled speed scaling (i.e., PS scheduling based on SRPT's speed scaling function for the same workload). A line for PS-SRPT has been added to Figure 3(a) and (b) to emphasize this point.

Figure 4 presents simulation results for PS scheduling with decoupled speed scaling. In particular, we consider three different speed scaling functions (PS, FSP, SRPT). In all three cases, PS is perfectly fair across job sizes. However, the mean slowdown value varies significantly depending on the speed scaling function provided. Among the three speed scaling policies presented, SRPT provides the highest mean slowdown (about 40% higher than PS-PS), since it tends to operate at lower speeds. For this workload, the slowdown results for PS-FSP are very close to those for PS-SRPT. At higher loads (see Table IV), there is greater separation between the policies. In general, the PS-FSP results are (as expected) between those for PS-PS and PS-SRPT, but are much closer to PS-SRPT than to PS-PS.

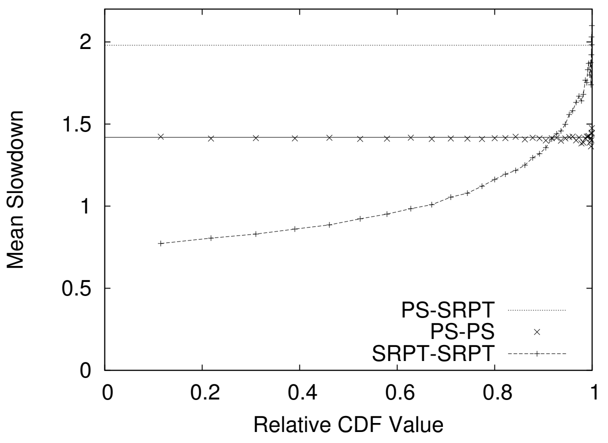
Figure 5 presents simulation results illustrating the dominance of FSP over PS in decoupled speed scaling. The horizontal line labeled PS-PS shows the native speed scaling results for PS, which provides egalitarian service for all job sizes. The second horizontal line above this shows the results for PS-FSP, which is the PS scheduling policy driven by the external speed scaling function of FSP. PS is still perfectly fair

TABLE IV
SUMMARY OF SIMULATION RESULTS FOR DIFFERENT WORKLOADS

Scheduling Policy	Mean Job Size 80		Mean Job Size 100		Mean Job Size 200	
	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$	$E[T]$	$E[\varepsilon]$
PS-SRPT	1.589	1.24E+04	2.428	1.54E+04	14.911	4.33E+04
PS-FSP	1.559	1.24E+04	2.345	1.54E+04	10.696	4.34E+04
PS-PS	1.145	2.78E+04	1.561	3.47E+04	4.634	9.38E+04
FSP-PS	0.859	2.78E+04	1.092	3.47E+04	2.415	9.38E+04
FSP-FSP	1.018	1.24E+04	1.374	1.54E+04	4.290	4.34E+04
SRPT-SRPT	1.012	1.24E+04	1.362	1.54E+04	4.246	4.33E+04



(a) Slowdown vs Job Size



(b) Slowdown vs CDF Value

Fig. 3. Simulation Results for PS and SRPT Scheduling with Coupled Speed Scaling

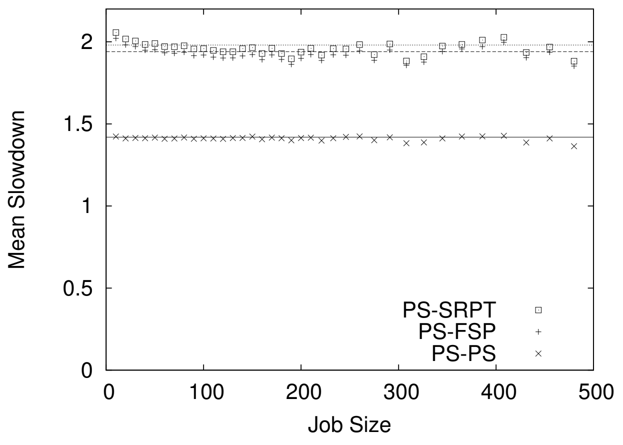


Fig. 4. Simulation Results for PS Scheduling with Decoupled Speed Scaling

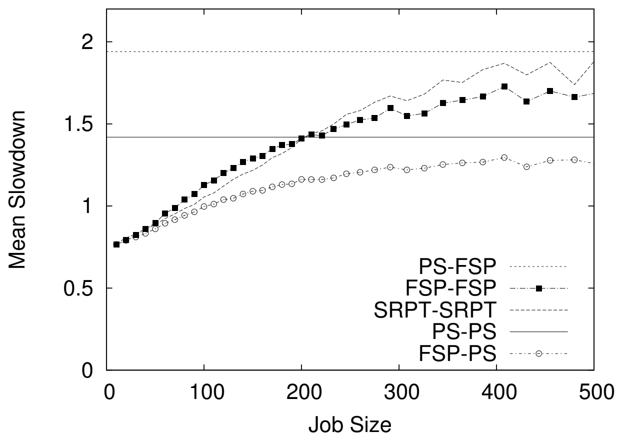


Fig. 5. Simulation Results for PS and FSP Scheduling with Decoupled Speed Scaling

in this scenario, though the increase in mean slowdown reflects the (generally) lower system occupancy and CPU speeds associated with FSP. The two dotted curves demonstrate the dominance of FSP over PS. This dominance is evident both for native FSP speed scaling (FSP-FSP versus PS-FSP) and PS speed scaling (FSP-PS versus PS-PS). Under FSP scheduling, no job ever finishes later than it does under PS, given the same speed scaling function.

The performance advantages of FSP over PS are very pronounced. In Figure 5, the mean slowdown for PS is about 40-50% worse than it is for FSP. For reference purposes, the SRPT-SRPT line from Figure 3(a) is also overlaid onto Figure 5. Comparing the SRPT-SRPT results to the FSP-FSP results (which has similar mean slowdown to FSP-SRPT) reveals that SRPT has a slight advantage for smaller jobs, while FSP has a pronounced advantage for larger jobs. This observation suggests that FSP scheduling can restore fairness under decoupled speed scaling, without compromising much on response time optimality compared to SRPT.

As a closing observation, note that the FSP-FSP entry in Table IV shows the simulation results for running FSP and the virtual PS with speeds determined based on (the square root of) the number of jobs in the FSP queue, rather than the virtual PS queue. Interestingly, under this scheme, the response time and the energy consumption (i.e., the cost) closely follows that of SRPT-SRPT. We thus conjecture that it is possible to achieve fairness, robustness, and near optimality under decoupled speed scaling.

VIII. Conclusions

In this paper, we consider the notion of decoupled speed scaling, wherein the speed scaling function is fully decoupled from the scheduling policy in a dynamic speed scaling system. We propose this approach as a new paradigm for the analysis and evaluation of speed scaling systems, and use it to compare and evaluate PS, SRPT, and FSP policies. The key advantage of decoupled speed scaling is that it enables direct comparisons between scheduling policies under fixed energy consumption. From a practical viewpoint, this approach provides great flexibility for the design and analysis of speed scaling systems.

We provide theoretical and simulation results in the paper. We demonstrate that the FSP scheduling policy, which dominates PS in the non-speed-scaling world, is ill-defined under coupled (native) speed scaling based on job count. In addition, all policies are efficient under coupled speed scaling, thus no policy can dominate PS. With decoupled speed scaling, however, FSP again dominates PS, and is provably efficient. Simulation results demonstrated a notable performance advantage for FSP, compared to PS.

Based on our analysis and evaluation, we conjecture that it is indeed possible to attain fairness, robustness, and near optimality with decoupled speed scaling. A rigorous proof, however, remains elusive, and is the immediate focus for our ongoing work.

Acknowledgment

The authors thank the anonymous reviewers for their encouraging comments and constructive suggestions, which helped to improve the clarity of the final version. Financial support for this work was provided by Canada's Natural Sciences and Engineering Research Council (NSERC), and Alberta Innovates – Technology Futures (AITF) in the Province of Alberta.

References

[1] D. C. Snowdon, E. Le Sueur, S. M. 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.

[2] 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.

[3] 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.

[4] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," in *Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2007, pp. 289–298.

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

[6] 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.

[7] 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.

[8] 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.

[9] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Perform. Eval.*, vol. 67, no. 11, pp. 1155–1171, Nov. 2010.

[10] L. L. H. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," in *Proc. ACM SIGMETRICS*, 2010, pp. 37–48.

[11] E. J. Friedman and S. G. Henderson, "Fairness and efficiency in web server protocols," in *Proc. ACM SIGMETRICS*, 2003, pp. 229–237.

[12] P. Nain and D. Towsley, "Optimal scheduling in a machine with stochastic varying processing rate," *IEEE Trans. Autom. Control*, vol. 39, pp. 1853–1855, 1994.

[13] B. Avi-Itzhak and H. Levy, "On measuring fairness in queues," *Advances in Applied Probability*, vol. 36, no. 3, pp. 919–936, 2004.

[14] E. Hahne, "Round-robin scheduling for max-min fairness in data networks," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 7, pp. 1024–1039, 1991.

[15] R. Schassberger, "The steady-state appearance of the M/G/1 queue under the discipline of shortest remaining processing time," *Advances in Applied Probability*, vol. 22, no. 2, pp. pp. 456–479, 1990.

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

[17] L. Schrage and L. Miller, "The queue M/G/1 with the shortest remaining processing time discipline," *Oper. Res.*, vol. 14, pp. 670–684, 1966.

[18] D. Smith, "A new proof of the optimality of the shortest remaining processing time discipline," *Oper. Res.*, vol. 26, pp. 197–199, 1976.

[19] N. Bansal and M. Harchol-Balter, "Analysis of SRPT scheduling: investigating unfairness," in *Proc. ACM SIGMETRICS*, 2001, pp. 279–290.

[20] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection scheduling in web servers," in *Proc. USENIX Symposium on Internet Technologies and Systems - Volume 2*, ser. USITS'99, 1999, pp. 22–22.

[21] M. Crovella, M. Harchol-Balter, and S. Park, "The case for SRPT scheduling in web servers," MIT Laboratory for Computer Science, Tech. Rep. MIT-LCS-TR-767, Oct. 1998.

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

[23] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-based scheduling to improve web performance," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 207–233, May 2003.

[24] A. Wierman and M. Harchol-Balter, "Classifying scheduling policies with respect to unfairness in an M/GI/1," in *Proc. ACM SIGMETRICS*, 2003, pp. 238–249.

[25] D. Lu, H. Sheng, and P. Dinda, "Size-based scheduling policies with inaccurate scheduling information," in *Proc. IEEE/ACM MASCOTS*, 2004, pp. 31–38.

[26] P. Brown, "Comparing FB and PS scheduling policies," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 3, pp. 18–20, Dec. 2006.

[27] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack, "Analysis of LAS scheduling for job size distributions with high variance," in *Proc. ACM SIGMETRICS*, 2003, pp. 218–228.

[28] D. Raz, H. Levy, and B. Avi-Itzhak, "A resource-allocation queueing fairness measure," in *Proc. ACM SIGMETRICS*, 2004, pp. 130–141.

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

[30] A. Wierman, L. L. H. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. IEEE INFOCOM*, 2009, pp. 2007–2015.

[31] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. E. Gehrke, "Online scheduling to minimize average stretch," in *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999, pp. 433–442.

[32] 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.