Simulation Modeling for Speed Scaling Designs

[Extended Abstract]

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

ABSTRACT

This paper introduces an extensible simulation tool for the study of speed scaling designs for a single server with variable service rates. This simulator facilitates detailed performance analysis of scheduling and speed scaling policies under different background settings. Our component-based design separates the system configuration, like the power consumption model and the incoming workload, from the policy components for scheduling and speed scaling. Furthermore, the simulator allows for custom implementations of the data logger, so the instrumentation could be fine tuned to the desired level of detail for each run of the simulation. We present the design of the main components of our simulator. We then show example results to highlight the versatility of the simulator for studying different speed scaling configurations.

CCS Concepts

•**Computing methodologies** → **Simulation tools;** •**Mathematics of computing** → *Queueing theory;*

Keywords

Speed Scaling, Scheduling, Energy Consumption, Discrete Event Simulation

1. INTRODUCTION

Modern CPUs have over a dozen processing rates, which enable system designers to balance speed and power consumption based on the required performance. This tradeoff is of interest both for mobile devices, which have limited power, and for datacenters, for which heat and cost management are key considerations. Recently, there has been increased attention to the study of scheduling policies for servers with adjustable service rates both in the systems and theory communities [2, 12].

For a single-speed server, several analytical and simulation studies quantify the properties of different classes of scheduling policies. In particular, commonly used scheduling polices like First-Come-First-Serve (FCFS) and Processor Sharing (PS) were compared to size-based scheduling policies like Shortest-Remaining-Processing-Time (SRPT) with regard to queueing delay, response time, and fairness [9, 14].

For datacenters, optimizing the monetary cost of operation is the main goal. This could be translated into some combination of energy cost and the cost of losing customers or money for violating the service level aggreements (SLAs). These two metrics are inherently at odds with each other. Running at lower speeds consumes less energy, but degrades the performance. Furthermore, optimizing the average performance could have undesirable side effects like poor tail behavior, or decreased robustness in face of flash loads.

The tradeoffs between optimality, robustness, and fairness in speed scaling was first studied in [12]. For the cost function represented by the linear combination of the average response time and average energy consumption per job, [12] shows that SRPT with queue-length-based speed scaling is optimal in the worst case. The idea of queue-length-based speed scaling (*coupled speed scaling* for the rest of this paper), wherein the service rate is a function of the instantaneous system occupancy, was introduced in earlier literature [2]. In [12], the optimal speed function is shown to be the inverse of the power-consumption function.

There are many interesting questions still remaining about the interplay of scheduling and speed scaling. For example, [12] shows that coupled speed scaling magnifies unfairness for policies like SRPT and all the non-preemptive policies. However, the level of unfairness for these polices under different types of workloads is unknown.

In spite of the growing interest in the study of speed-scaling policies, to the best of our knowledge there is no public-domain tool for the simulation of scheduling with speed scaling. There are a number of simulation results presented for speed scaling designs in recent literature [7,12,13]. A recent public-domain simulation tool for a single speed server has been developed to study size-based scheduling when exact knowledge of job sizes is not available [5]. However, the simulator does not support variable service rates.

Our work: We have developed an extensible simulation tool for the study of speed scaling designs, which facilitates detailed performance analysis of scheduling and speed scaling policies under different background settings.

Design of a speed scaling simulator has several challenges. Special care is required to separate the scheduling and speed scaling decisions in order to allow for simulation of new approaches with minimal effort, and without introducing

ACM ISBN X-XXXXX-XX-X/XX/XX.
DOI: http://dx.doi.org/10.1145/0000000.0000000

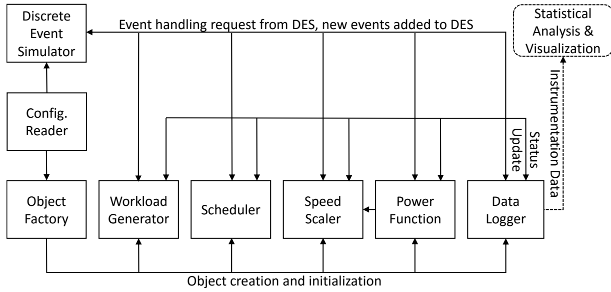


Figure 1: Data flow between the main components of the simulator.

bugs or inconsistencies into the simulation. Decoupling the scheduling decisions from speed scaling decisions was first presented in [7]. The idea was presented in the context of FSP with speed scaling, but was later generalized to other policies [6–8].

We developed our simulation tool to study the interplay between the speed scaler and the scheduler under different load regimes and power consumption models. For this reason, the extensibility of the simulation tool has been the main design focus. Furthermore, due to high variability of sampled metrics in many scenarios, long simulation runs are necessary in order to obtain statistically significant results. Therefore, efficiency in terms of run-time, memory, and storage requirements was considered in the design.

The simulator has an abstract design that models a single server with variable service rates. An infinite stream of jobs arrive to the system to receive service in some order specified by the scheduler and with some speed specified by the speed scaler. Energy consumption of the system is computed according to some power function. The entire operation of the server is tracked by the logger. Each of these components and the operation of the simulator is presented in more detail in Section 2.2.

We have used this simulator to verify previous analytical and simulation results presented in [13]. We have also gained insight into the autoscaling effect of coupled speed scaling systems under heavy load [6]. For cases where the analytical study of policies is challenging, like the average busy period length under coupled SRPT or average response time under FSP, the simulation study can be used to investigate the expected behavior of the system.

To illustrate the application of this abstract design, we present selected simulation results for coupled and decoupled speed scaling systems in Section 3.

2. SIMULATOR

This section provides an overview of the simulator (see Figure 1). We briefly discuss the design choices and then present the main components and their dependencies.

2.1 Design overview

The main focus in our design is two-fold: ease of extensibility, and efficiency for large-scale simulations. Simulating a new scheduling policy should only require the implementation of the abstract class Scheduler, without any changes to other components or the core of the simulator. The abstract design of the Logger class makes it possible to change the instrumentation targets and level of detail based on the focus of the simulation study, again without any changes required to other components. The ability to change the instrumentation level is important in order to reduce the execution time and storage requirements for long simulation runs.

Execution time for a single run of the simulator depends on the following factors: (1) duration of the run; (2) expected queue length; (3) step complexity of the scheduler and speed scaler components for handling each event; and (4) complexity of the instrumentation by the logger. The first three factors have direct dependence on the workload and the simulated policies, and usually dominate the overall complexity of the simulation. Memory usage is dominated by the implementation of the logger. The minimum requirement depends on the expected queue length, since the logger has to keep track of all jobs in the queue.

Our implementation of the simulator is written in C++, and it only uses the C++ standard libraries. The current code base is developed and compiled on a Windows machine, but could be compiled on other platforms with minimal effort [1].

The simulation scenario specifications are all modified via the configuration file, which is a human-readable text file with easy-to-interpret value pair format.

2.2 Main components

A single run of the simulator is comprised of three stages. First, the configuration file is read and the policy components and the background component objects are created and initialized. Second, the Discrete Event Simulator (DES) and the Logger are initialized. Third, according to the user specified configurations, the simulation is run until all the first N_{MAX} arrivals have departed the system, at which point the simulation terminates. If jobs are serviced in the order of their arrival (FCFS), this is at the time of departure number N_{MAX}. However, depending on the policies and the workload, termination time could be much later.

For the probe-based sampling of the response times, the third stage is repeated with a probe job inserted at random into the workload stream (similar to the algorithm presented in [9]). For each probe size, the probing is repeated until the user-specified desired confidence interval is achieved. The range of probe job sizes are also specified in the simulation configuration.

Figure 1 illustrates the main components in our simulator. Figure 2 shows selected abstract classes and example implementations.

Simulator Configuration The Simulation scenario specifications are read from a human-readable text file (`configuration.txt`). These specifications include the scheduling and speed scaling policies, the power function, the workload, and their corresponding parameters. It also specifies the logger and its parameters that include the metrics of interest and granularity of measurements.

Object Factory To make the simulator extensible, and to separate different policies from the background settings, all policy and background setting components are defined by abstract classes. The configuration component calls the Object Factory to create and initialize the objects required to execute the specified simulation scenario.

Discrete Event Simulator (DES)

The DES maintains a priority queue of events. In each iteration of the main loop, the event with the smallest timestamp is removed from the priority queue and passed on to

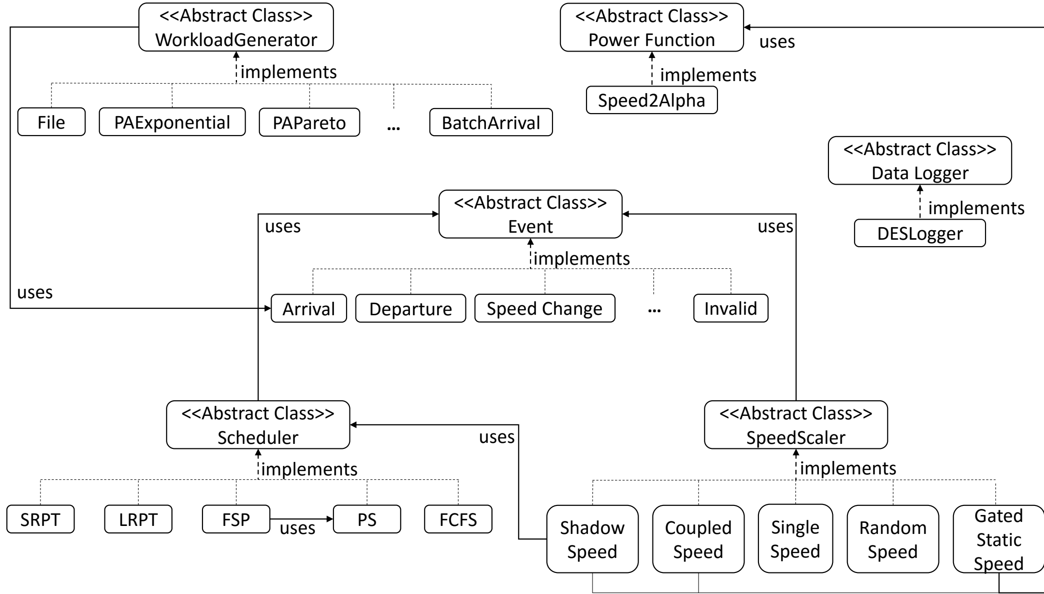


Figure 2: Abstract classes and example implementations.

the proper handlers depending on the type of the event e.g., *Arrival*, *Departure*, *Speed Change*, etc. (see Figure 2). New events may be generated as a result of handling the current event. These are inserted into the priority queue. The loop stops when all the first N_{MAX} arrivals complete their work.

Logger The logger component keeps track of the changes in the state of the system by receiving updates when other components have a status change. It plays the role of the omniscient observer. When other components need information about the current status of the system, they call the interfaces provided by the logger. For example, upon each arrival and departure, the coupled speed scaler queries the logger object for the number of jobs in the system in order to set the speed. Our *Comprehensive Logger* implementation records the status of the entire queue upon each event. For long simulations, however, the amount of data generated could be several gigabytes. For this reason, we have other implementations of the logger that only record a subset of the instrumentation such as queue length and the remaining work in the system upon each arrival and departure.

Workload Generator The arrival process is generated by the workload generator in the form of arrival events. At the initialization of the DES, the generator is asked for the first arrival event. Upon removing an arrival from the priority queue, the DES asks the workload generator to create the next arrival event. Each arrival event includes a timestamp and a job object. The job object must be initialized with the arrival time of the job, and optionally other properties like deadline and priority. Different implementations of the workload generator class allow alternative sources for the arrival sequence. For example, the workload could be generated from a trace file, or could be a Poisson process with job sizes drawn from an Exponential or a Pareto distribution.

Scheduler The scheduler component determines the order in which jobs receive service. Our current implementations include First-Come-First-Serve (FCFS), Processor Sharing (PS), Shortest Job First (SJF), Shortest Remaining Processing Time (SRPT), Longest Remaining Processing Time (LRPT), and Fair Sojourn Protocol (FSP).

Speed Scaler The speed scaling component decides the rate at which the server is processing jobs at each point in time during the simulation. This could be decided based on the system status, such as the system occupancy, or could be completely independent, like a single-speed system. The current implementations include *SingleSpeed*, *CoupledSpeed* (sets the speed to the inverse of the power function of the number of jobs in the system), *ShadowSpeed* (determines the speed based on the occupancy of a shadow scheduler that works on the same arrival process).

Power Function The power function component computes the energy consumption based on the speed of the system. It has two interfaces: one that returns the energy consumption as a function of speed, and one to compute the inverse of the power function.

Statistical Analysis and Visualization The simulator output depends on the implementation of the logger. Our current implementations output three types of log files: metric reports, job reports, and queue progress reports.

Metric reports are text files with timestamp and value pairs. This type of log records the evolution of a metric over time. For example, the `speedprofile.txt` records every speed change event, and the `byteprofile.txt` records the remaining work in the system at the time of every arrival, departure and speed change event.

Job report logs are text files in which each line shows the report of a job upon its departure. We record the size, arrival, departure, energy consumption and time under execution for each job. Metrics like response time and slowdown can be directly derived from the above. The logger could generate this report for any subset of the jobs. For example, the `probereport.txt` contains the job report for only the probe jobs inserted into the workload stream.

Queue progress reports are text files in which every line has a timestamp followed by the status of the queue at that timestamp. This log records the evolution of the remaining sizes of jobs in the queue. Figure 5 shows an excerpt from

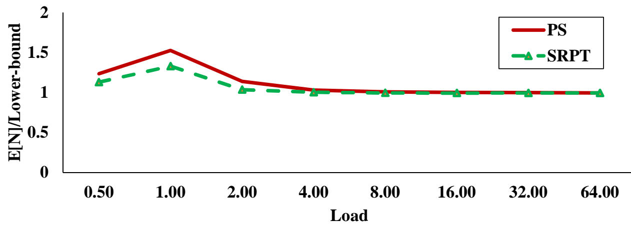


Figure 3: Comparison of queue occupancy under heavy load for coupled PS and SRPT with $s(t) = P^{-1}(n(t))$. The job sizes are Pareto(2.2)

this log.

We have implemented separate C++ programs to perform further statistical analysis on the output, including mean, variance, and frequencies for recorded metrics. We also have implemented Matlab scripts for further statistical analysis and generation of plots. A Java-based visualization of the queue progress report has been implemented by an undergraduate summer student.

3. EXAMPLE RESULTS

In this section, we show selected simulation results to demonstrate the versatility of the simulation tool for the study of a range of speed scaling policies under different settings and with different metrics.

3.1 Occupancy under heavy load

It is well known that SRPT optimizes the average occupancy, $E[N]$, in single-speed systems [11]. In fact, given any speed sample path, SRPT yields the minimal average occupancy among all work conserving scheduling policies [3].

In coupled speed scaling systems, the speed $s(t)$ at time t is based on the number of jobs in the system. The interdependency of average speeds and system occupancy results in a lower-bound for the occupancy [6]. In order to guarantee stability in the system, the average speed must be at least that of the incoming load, $\bar{s} \geq \rho$. In [13], it is observed that if the power function is convex, then by Jensen's inequality we have $E[N] = E[P(s(t))] \geq P(E[s(t)]) \geq \rho^2$.

With our simulation tool, we can investigate how far the average occupancy is from the lower-bound. Figure 3 shows the comparison of queue occupancy under heavy load for coupled PS and SRPT. As load increases, the average occupancy under PS and SRPT both converge to the lower-bound. This is in high contrast to the results in the single-speed setting, where the average occupancy under PS grows quickly in comparison to SRPT.

Figure 4 shows a snapshot of system occupancy under coupled PS and SRPT, sampled at the same points in time when the two systems work on the same workload stream. We see that with the increase of load, the occurrences where SRPT has higher occupancy than PS increases. This is counter-intuitive, since SRPT in single-speed systems always runs at lower queue occupancy (i.e., the sampled points would always remain below the 45 degree line).

3.2 Queue Progress Log

In order to better understand why the queue occupancy under SRPT exceeds that under PS, we investigate the detailed queue progress log. Figure 5 shows an excerpt of this

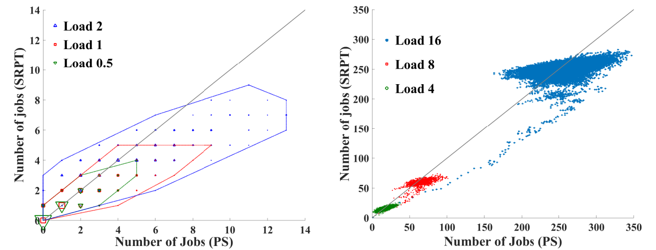


Figure 4: Occupancy $n(t)$ under coupled SRPT vs. coupled PS when the two system work on the same workload stream. Size of the marker indicates the frequency of sampled occupancy.

log for PS and SRPT on the same workload stream with load $\rho = 4$. In this short excerpt, the total remaining work under PS is much lower than under SRPT. Furthermore, the number of jobs that have not even started service is larger under SRPT. This hints at the compensation effect that PS benefits from [6]. By running at higher occupancy prior to this excerpt, PS can cope with the load in this episode at lower speed. Conversely, SRPT needs to compensate for running at lower speed prior to this excerpt in order to keep up with the incoming load.

An interesting observation is the difference in the distribution of remaining sizes in the two queues. Not surprisingly, SRPT tends to accumulate jobs with larger remaining sizes in its queue. The surprising part is the large difference in the work backlog in the two queues, and the high number of jobs that have not started service under SRPT. This hints at the possibility that coupled SRPT is nearly starving larger jobs.

3.3 Remaining bytes in the system

We further investigate the byte backlog in the queue under SRPT and PS by looking at the total remaining sizes vs. the queue occupancy. Figure 6 and 7 show the remaining bytes and the queue occupancy over time under SRPT and PS respectively. These three workloads all have average load $\rho = 4$ but with different job size distributions. For the case of Pareto job sizes, the average job size is 1 (Figure 6(c) and 7(c)). For SRPT, the job size distribution greatly influences the shape of the work backlog, while queue occupancy is less sensitive to the job size distribution. Note that the occupancy stays close to the lower-bound of ρ^2 under all the considered workloads for both SRPT and PS.

Figure 8 shows the average remaining bytes in the system as a function of load. The backlog under coupled SRPT grows very quickly with the increase of load, while the increase is much more moderate under PS.

3.4 Fairness Study

Our simulation results for remaining bytes in the system under coupled SRPT reconfirms the theoretical result on adverse effect of coupled speed scaling on slowdown of large jobs under SRPT [12]. It further shows that in comparison to coupled PS, the difference in the backlog grows rapidly with the increase of load. This suggests that the slowdown disadvantage of larger jobs under SRPT worsens under heavy load.

We can study the slowdown of different policies with the

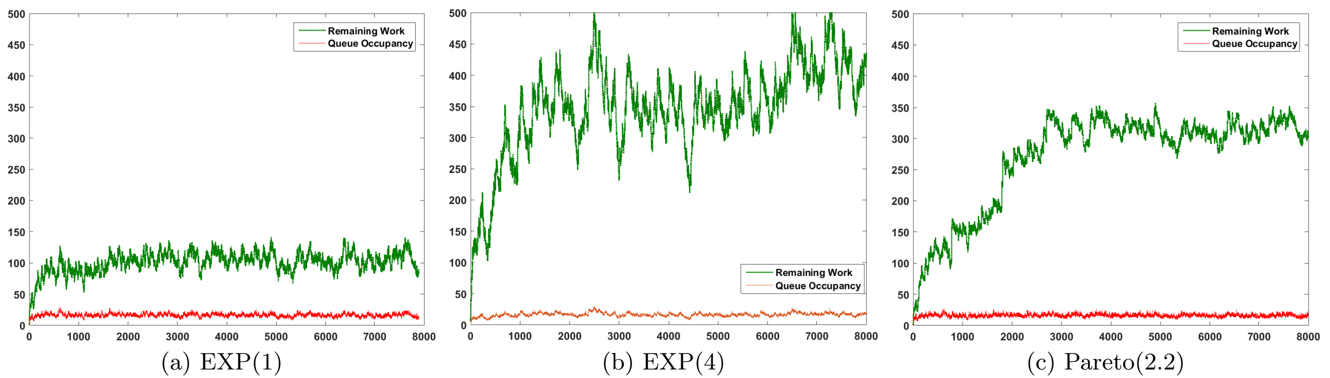
Time	Arrival (A) Departure (D) Preemption (P)	Num. started service	Num. not yet started	Sum of rem. sizes started service	Sum of rem. sizes not started service	Remaining sizes of jobs in queue (PS)												
876.991	P(0.032)	12	1	42.922	0.032	0.032	0.063	0.320	0.487	1.029	1.102	1.602	2.504	3.785	4.031	4.296	11.155	12.548
877.106	D	12	0	42.540	0.000	0.031	0.288	0.455	0.997	1.070	1.570	2.472	3.753	3.999	4.264	11.123	12.516	
877.214	D	11	0	42.165	0.000	0.257	0.424	0.966	1.039	1.539	2.441	3.722	3.968	4.233	11.092	12.485		
877.350	P(5.344)	11	1	41.716	5.344	5.344	0.216	0.383	0.925	0.998	1.498	2.400	3.681	3.927	4.192	11.051	12.444	
878.098	D	11	0	44.467	0.000	0.167	0.709	0.782	1.282	2.184	3.465	3.711	3.976	5.128	10.835	12.228		
878.611	P(4.979)	11	1	42.766	4.979	4.979	0.012	0.554	0.627	1.127	2.029	3.310	3.557	3.821	4.974	10.680	12.073	
878.649	P(1.343)	12	1	47.611	1.343	1.343	0.001	0.543	0.616	1.116	2.018	3.299	3.545	3.810	4.962	4.968	10.669	12.062
878.653	D	12	0	48.942	0.000	0.542	0.615	1.115	1.342	2.017	3.298	3.544	3.809	4.961	4.967	10.668	12.061	
880.532	D	11	0	42.434	0.000	0.073	0.573	0.800	1.475	2.756	3.002	3.267	4.419	4.424	10.126	11.519		
880.773	D	10	0	41.633	0.000	0.500	0.727	1.402	2.683	2.929	3.194	4.346	4.352	10.053	11.446			
880.877	P(2.398)	10	1	41.306	2.398	2.398	0.467	0.695	1.369	2.650	2.897	3.161	4.314	4.319	10.021	11.413		

(a) PS

Time	Arrival (A) Departure (D) Preemption (P)	Num. started service	Num. not yet started	Sum of rem. sizes started service	Sum of rem. sizes not started service	Remaining sizes of jobs in queue (SRPT)														
876.991	P(0.032)	1	12	7.877	231.066	0.032	7.877	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649		
877.000	D	1	11	7.877	231.034	7.877	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649			
877.350	P(5.344)	1	12	6.666	236.378	5.344	6.666	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649		
878.611	A(4.979)	2	12	7.462	236.013	0.796	4.979	6.666	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649	
878.649	A(1.343)	2	13	7.318	237.356	0.652	1.343	4.979	6.666	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649
878.818	D	1	13	6.666	237.356	1.343	4.979	6.666	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649	
879.177	D	1	12	6.666	236.013	4.979	6.666	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649		
880.558	D	1	11	6.666	231.034	6.666	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649			
880.877	P(2.398)	1	12	5.561	233.432	2.398	5.561	17.683	18.134	18.250	18.478	19.455	20.297	20.497	20.643	22.898	23.050	31.649		

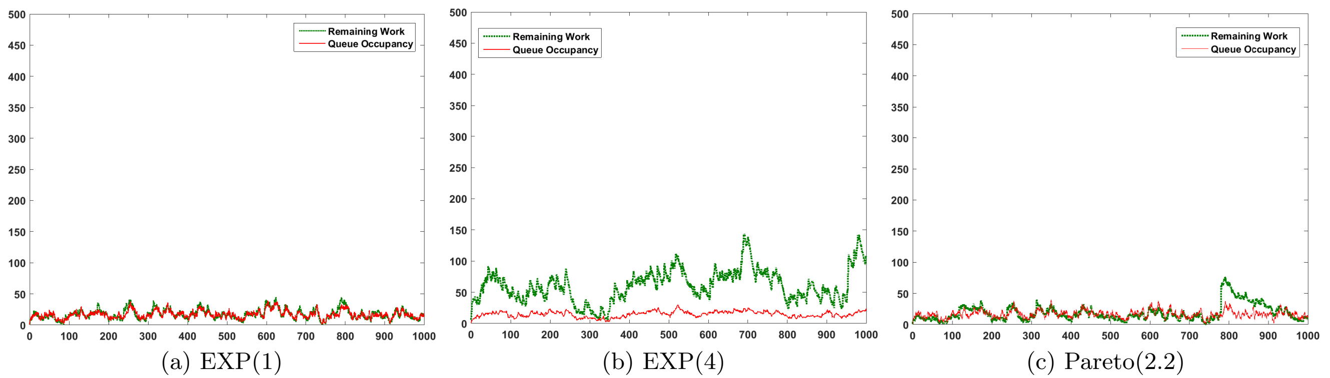
(b) SRPT

Figure 5: Comparison of queue progress log for coupled PS and SPRT given the same workload stream with load 4.



(a) EXP(1) (b) EXP(4) (c) Pareto(2.2)

Figure 6: The remaining bytes vs. queue occupancy in coupled SRPT given average load 4 with different job size distributions.



(a) EXP(1) (b) EXP(4) (c) Pareto(2.2)

Figure 7: The remaining bytes vs. queue occupancy in coupled PS given average load 4 with different job size distributions.

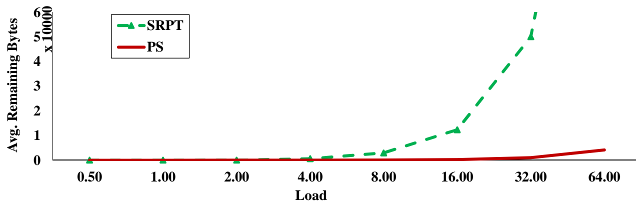


Figure 8: Comparison of average remaining bytes under coupled SRPT and PS, Pareto(2.2) job sizes.

help of our probe-based simulation. Figure 9 shows the slowdown for 6 different policies when $\rho = 2$ with Pareto(2.2) job sizes. Note that coupled PS is a symmetric queue and therefore has constant slowdown for all job sizes [10,13]. In Figure 9(a) the slowdown under SRPT for jobs larger than 5 is already more than double the slowdown under PS. Figure 9(b) highlights the slowdown disadvantage of SRPT. Under SRPT with load 2, slowdown of jobs larger than 250 is greater than slowdown of jobs under PS with load 64.

In Figure 9(a), SRPT-PS and FSP-PS dominate PS (i.e., always maintain slowdown no worse than that under PS). These decoupled speed scaling policies determine the speed based on the occupancy of a shadow PS scheduler that works on the same arrival process. The analytical result in [7] shows that for all load configurations, FSP-PS performs at least as well as PS for both slowdown and the sum of average response time and energy consumption (i.e., the cost). Further investigation is required to quantify the improvement under different load regimes.

3.5 Cost

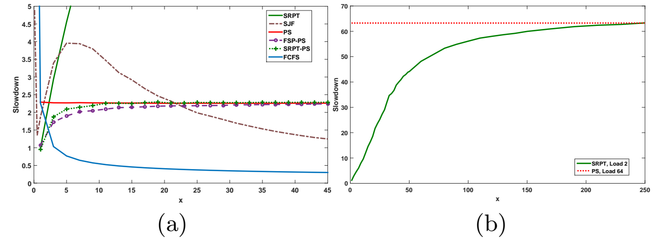
We study the cost improvements under FSP-PS and SRPT-PS in comparison to coupled PS. Figure 10 shows the ratio of cost to the lower-bound. The cost is the linear combination of average response time and energy consumption. For load ρ, the lower bound $max(2\rho, \rho^2)$ is shown in [13]. Under heavy load, both FSP-PS and SRPT-PS greatly improve the cost in comparison to PS. SRPT-PS has a slight edge on FSP-PS, but there is a slowdown tradeoff for this gain in cost (see Figure 9(a)).

4. CONCLUSIONS

This paper presented an extensible simulation tool for the study of speed scaling designs. Through example results, we have shown the versatility of the simulation tool for the study of a range of policies under different speed scaling configurations. We are further investigating the behavior of coupled and decoupled SRPT under heavy load. For future work, more policies can be implemented and studied. The current implementation of FSP could be greatly improved by incorporation of the algorithm proposed in [4].

Acknowledgments

Financial support for this work was provided by Canada's Natural Sciences and Engineering Research Council (NSERC). The authors are grateful to Philipp Woelfel for insightful discussions about speed scaling systems, and to Jennifer Williamson for building a Java-based visualization of our queue progress log output.



(a) (b)

Figure 9: Slowdown under different coupled and decoupled scheduling polices. (a) Load 2 and job sizes are Pareto(2.2).

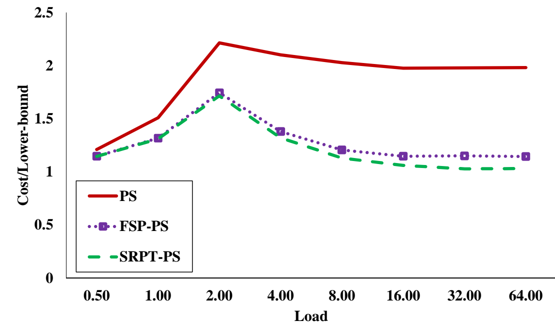


Figure 10: Comparison of cost of Coupled PS and Decoupled SRPT and FSP with speeds of PS. Pareto(2.2) job sizes.

5. REFERENCES

[1] Versatile Speed Scaling Simulator. https://github.com/bmelahi/SpeedScalingSimulator.
[2] S. Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, May 2010.
[3] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *Proc. ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 693–701, 2009.
[4] M. Dell'Amico, D. Carra, and P. Michiardi. PSBS: practical size-based scheduling. *IEEE Trans. Computers*, 65(7):2199–2212, 2016.
[5] M. Dell'Amico, D. Carra, M. Pastorelli, and P. Michiardi. Revisiting size-based scheduling with estimated job sizes. In *MASCOTS*, pages 411–420, 2014.
[6] M. Elahi and C. Williamson. Autoscaling effects in speed scaling systems. In *MASCOTS 2016 (preprint)*.
[7] M. Elahi, C. Williamson, and P. Woelfel. Decoupled speed scaling: Analysis and evaluation. *Perform. Eval.*, 73:3–17, 2014.
[8] M. Elahi, C. Williamson, and P. Woelfel. Turbocharged speed scaling: Analysis and evaluation. In *MASCOTS*, pages 41–50, 2014.
[9] M. Gong and C. Williamson. Revisiting unfairness in web server scheduling. *Comput. Netw.*, 50(13):2183–2203, 2006.
[10] F. Kelly. *Reversibility and Stochastic Networks*. Cambridge University Press, 2011.
[11] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Oper. Res.*, 16:678–690, 1968.
[12] A. Wierman, L. Andrew, and M. Lin. Speed scaling: An algorithmic perspective. In *Handbook of Energy-Aware and Green Computing - Two Volume Set.*, pages 385–405. 2012.
[13] A. Wierman, L. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *Perform. Eval.*, 69(12):601–622, 2012.
[14] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proc. ACM SIGMETRICS*, pages 238–249, 2003.