

Fairness and efficiency in web server protocols*

MinghongLin
(Presentation for CS286b)

1 Motivation

This paper is motivated by the question: For a server under Processor Sharing policy, Is it possible to reduce the response time of every job, simply by changing the scheduling policy?

- Many systems can be shared among multiple jobs simultaneously. Some version of processor sharing is used widely. (File transfers share the link. Time-sharing on computer. Processor sharing on web server.)
- Typical justification for PS is “fairness”: all jobs receive similar processing resources.
- However, it may be very inefficient.
Consider the case n jobs of size 1 arrive at the same time.
 - Under PS the sojourn time for every job is n .
 - Schedule them in any order, we get a longest sojourn time of n . All other jobs receive better service.
- SRPT has been recommended for better performance. it minimizes mean response time.
- Improvement is not guaranteed. Large jobs can do much worse under SRPT compared to PS.
- Can we find a policy which is 1) more efficient than PS, 2) “as fair as PS”?
- Then, what is “efficiency”? what is “fairness”?

That is one of the contribution of this paper! The contributions are:

- Develop a partial ordering on policies, use it to define “efficiency” and “fairness”. This partial ordering provides a framework to classify some policies(like PS) as being dominated, and therefore inefficient.
- Show that PS is inefficient, SRPT is unfair under their framework.
- Present a new policy(FSP) which is both fair and efficient.

2 Model

- A single server, no assumption on arrival process / job size distribution. (Since it is based on sample path arguments.)
- The only assumption: When a job arrives, we know its size. (not blind)
- Define a sample path as the sequence of arrival times and job lengths. $((t_1, l_1), \dots, (t_n, l_n))$.
- A scheduling policy can be defined as a function ω , $\omega(i, t)$ is the work rate on job i at time t . We consider the policies in which
 1. $\omega(i, t)$ is non-negative and piece wise continuous.

*Eric J. Friedman and Shane G. Henderson, ACM SIGMETRICS Perform. Eval. Rev, 2003

2. $\sum_i \omega(i, t) = \mu = 1$ for the jobs present in the system. (conservation server)
3. $\omega(i, t)$ cannot depend on arrivals after time t . (online policy)

Then the completion time of job i is t given by $\int_{t_i}^t \omega(i, s) ds = l_i$.

- Goal: minimizing some weighted expectation of the sojourn times. (mean sojourn time/mean slowdown/...)

Now the following is the first contribution.(the partial ordering)

3 Efficiency and Fairness

Definition 1 A policy p' dominates policy p if no job completes later under p' than under p on any sample path, and there is at least one job on at least one sample path that completes earlier under p' than under p .

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

if some policy is dominated by another one, then it(inefficient policy) will be no better under any reasonable measure. So

Definition 3 A reasonable loss measure is a mapping $\pi : P \rightarrow R$ such that if p' dominates p , then $\pi(p') \leq \pi(p)$.

For example, expected sojourn time and expected slowdown are reasonable measures, as is any weighted average of sojourn times.

We can also define strict version of these notations:

Definition 4 A policy p' strictly dominates policy p if, for every busy period on every sample path, every job except the terminal one completes strictly earlier under p' than under p .

(explain why except the last one)

Definition 5 A strict loss measure is a mapping $\pi : P \rightarrow R$ such that if p' strictly dominates p , then $\pi(p') < \pi(p)$.

For example, expected sojourn time and expected slowdown are strict measures, as is any nonzero weighted average of sojourn times.

Next we define PS as the standard of fairness:

Definition 6 A policy p is fair if it dominates PS.

Remark

- SRPT is efficient: $\min E[T]$, can't be dominated.
- All nonpreemptive policies are efficient! (??)
- SRPT is unfair: consider the sample path $((0, 3), (1, 1), \dots, (n, 1))$. can not dominate PS.
- PS is inefficient. We will introduce a policy called Fair Scheduling Protocol, which (strongly) dominates PS.

4 FSP

Intuition For any sample path under PS, consider two concurrent jobs a and b where a will complete first at time t . If we were to trade some of b 's processing time before t to a for the same amount of a 's processing time after t , then a would complete earlier and b 's completion time would be unchanged.

Definition 7 FSP computes the time at which jobs would complete under PS and then order the jobs in terms of earliest (PS) completion times. FSP then devotes its full attention to the (uncompleted) job with the earliest (PS) completion time.

Some facts:

- Consider any two sample paths of arrivals that are identical up to time t and two jobs a, b that both arrive before t . Then under PS they complete in the same order on both sample paths.
- There are no “reversals” under FSP. A reversal occurs when job B enters the system before job A , and the processing of job A is interrupted to perform some processing on job B .
- FSP works like SRPT in many cases. e.g. all jobs arrives at the same time.
- Give two examples, compared to PS/SRPT.
- How to implement this policy?—“Run” a PS policy background to provide the order information. We can use the technique similar to Discrete Time Simulator.

5 Efficiency and fairness of FSP

Theorem 1 *FSP is efficient.*

Proof: Consider an arbitrary policy p' and assume that there is a sample path under which p' improves upon FSP, so that all jobs complete either at the same time as in FSP, or earlier. We will show that no job can complete strictly earlier under p' than under FSP.

Consider the first job to complete under FSP. By the fact of no reversals, the service of this job is uninterrupted. So it completes at the same time under both p' and FSP.

Assume that the first n jobs that complete under p' and FSP do so at the same time. Now consider the $(n + 1)^{th}$ job to complete under FSP. Again by the “no reversals” property, the only jobs that can have received service while the $(n + 1)^{th}$ job was present in the system are those that have already completed under FSP (and p'). Thus, the $(n + 1)^{th}$ job to complete must do so as early as possible, conditional upon the serving of the n previously completed jobs. These n jobs completed at the same times under both p' and FSP, so that the $(n + 1)^{th}$ job completes at the same time under both p' and FSP.

Theorem 2 *FSP strongly dominated PS.*

Proof: The proof in the paper is based on the “Discrete Time Simulator”. Let w and x be vectors indicating the remaining work for each job in the system under PS and FSP. The vector are ordered in the same way that $w_1 \leq w_2 < \dots < w_n$. It may be the case that $x_i = 0$ while $w_i > 0$ for same i . they prove that the statement, “ $\sum_{j=1}^i x_j \leq \sum_{j=1}^i w_j$ holds for all i , and it is not equal unless $i = n$ ”, is true not matter which kind of event occurs.

6 Simulation

The simulations compare the mean slowdown among PS/FSP/SRPT. Respect to mean slowdown, the result shows that FSP is better than SRPT and PS with very high probability.