

Minimizing Mean Response Time Subject to Fairness

Eric J. Friedman*
Shane G. Henderson
Gavin Hurley

Operations Research and Industrial Engineering, Cornell
(*Current: visiting EECS, U.C. Berkeley)

Work supported in part by National Science Foundation Grant Nos. ANI-9730162
(Friedman) and DMI-0085165 (Henderson).

Motivation: Web Serving

- Scheduling (preemptive) jobs of known size on a single server.
- Usually processor sharing (PS)
- Shortest remaining processing time (SRPT) minimizes mean sojourn time (easily a factor of 6).
- But, SRPT is not “fair,” and can “starve large jobs”.
- However, Bansal and Harchol-Balter (2002) – very rare in many situations and SRPT outperforms PS for all size jobs!
- (Our interest was kindled by their results!)

Fairness

- **Definition:** A protocol is fair if no job completes later under the given protocol than it would have under PS.
- (Note: Wierman and Harchol Balter (2004) use a weaker definition based on mean sojourn time stratified by job size.)
- Previous result: Fair Sojourn Protocol (FSP): fair and close to SRPT in mean sojourn time. (Only known fair protocol prior to the current work.)
- Questions: Are there other fair protocols? Do they improve on SRPT?

Summary of Results

- Result 1: A complete (and computationally efficient) characterization of all fair protocols.
- Result 2: Two new fair protocols
 - Pessimistic FSP (PFSP) and Optimistic FSP (OFSP)
 - Both outperform FSP
 - PFSP usually outperforms OFSP.
- (non)Result 3: More sophisticated protocols had negligible improvement on PFSP. (PFSP is so close to SRPT that the potential gain is not very significant anyway.)

Why PS is Inefficient

- Consider a sample path in which n jobs of length 1 arrive at the same time.
 - Under PS all complete at $t = n$
 - Under random scheduling all, except the terminal one, complete before $t = n$
- Consider two concurrent jobs a and b (under PS)
 - a will complete first at time t .
 - trade some of b 's processing time before t to a for the same amount of a 's processing time

- then a would complete earlier and b 's completion time would be unchanged.
- Observation: sharing processor time is always suboptimal.

The Fair Scheduling Protocol (FSP)

- Motivated by the inefficiency of PS
- Intuitively: swap time from jobs that complete later to those that complete earlier.
- Formally: Serve packets in order of their PS completion time.
- Essentially fair queuing, where each job is treated as its own packet stream.
- **Theorem:** FSP is fair. (F&Henderson 2002)

The Slack System

- At time t track all jobs that arrived before t and had not finished (under PS) before t .
- Order by PS completion time. $i \in \{1, \dots, m\}$. (Note: order unchanged by arrivals.)
- v_i : remaining processing time for job i under PS.
- w_i : remaining processing time of job i under the current protocol.
- slack vector: $s_i = (m - i)v_i + \sum_{j=1}^n v_j - \sum_{j=1}^n w_j$

- **Theorem:** A protocol is fair if and only if $s(t)$ is non-negative for all t .
- FSP: serve the first job in the list.

Understanding the Slack Vector

- Consider the first job in the list. If no further jobs arrive, it will complete at time mv_1 in the virtual PS-schedule.
- ———
- If a protocol chooses serves job 1 to completion, it will complete at time w_1
- thus we define $s_1 = mv_1 - w_1$.
- ———

- Second job will complete at time $v_1 + (m - 1)v_2$ (under PS)
- at time $w_1 + w_2$ in a schedule which serves only jobs 1 and 2.
- Thus $s_2 = v_1 + (m - 1)v_2 - (w_1 + w_2)$.
- ———
- PS completes the i th job at time $(m - i)v_i + \sum_{j=1}^i v_j$ since when the n th job completes, jobs 1 through n have completed, and furthermore jobs $n + 1, \dots, m$ have received an amount of service equal to v_n .
- ———

Updating the Slack System

- **Job i serviced for Δt and no new jobs arrive:**
- $w_i \rightarrow w_i - \Delta t$, others unchanged.
- for all j , $v_j \rightarrow v_j - \Delta t/m$.
- for all $j < i$, $s_j \rightarrow s_j - \Delta$, others unchanged.
- _____

- **New Job arrives with service time x such that $v_i < x < v_{i+1}$:**
- Insert after i , with $v_{i+1} = w_{i+1} = x$, others unchanged.
- $s_{i+1} \rightarrow s_i + (m - i)(x - v_i)$
- Slack of jobs j that precede it increase by v_j and those after it are unchanged.

FSP PFSP and OFSP

- Idea: try to mimic SRPT – serve the shortest job, that doesn't violate fairness.
- **Definition:** A job is **servable** at time t if a fair protocol can serve it at time t , independent of future jobs arriving.
- **Definition:** A job is **completable** at time t if a fair protocol can serve it to completion starting, if no future jobs arrive.
- A job is servable if and only if $s_i > 0$ for all $i < n$.
- A job n is completable if and only if $s_i \geq w_n$ for all $i < n$.

PFSP and OFSP

- **Theorem:** PFSP and OFSP are both fair.
- **Proof:** This follows easily from the slack system.
- **Theorem:** OFSP and PFSP are not comparable. OFSP and FSP are not comparable.
- Both depend on whether OFSP's gamble pays off – do new jobs arrive in time.

- **Conjecture:** PFSP dominates FSP.
- **Theorem:** If there are no further arrivals PFSP minimizes the average response time over all fair protocols.

Competitive analysis

Theorem 1 [Motwani et. al.] *PS is $\Omega(n/\log n)$ competitive to the optimal (not necessarily fair) protocol (SRPT).*

Theorem 2 *The FSP protocol is $\Omega(n/\log n)$ -competitive to the optimal (not necessarily fair) protocol.*

Theorem 3 *All fair scheduling protocols are $\Omega(n^{1/2})$ competitive to the optimal (not necessarily fair) protocol. (**)*

Theorem 4 *The FSP protocol is $\Omega(n/\log n)$ -competitive to the optimal fair protocol.*

Conjecture 1 *OFSP and PFSP are $O(1)$ -competitive to the optimal fair protocol.*

Simulations

- M/GI/1, arrival rate λ .
- IID jobs sizes, mean size 1 – truncated Pareto with $\alpha = 2$ (instead of 1.1 to reduce simulation time.)

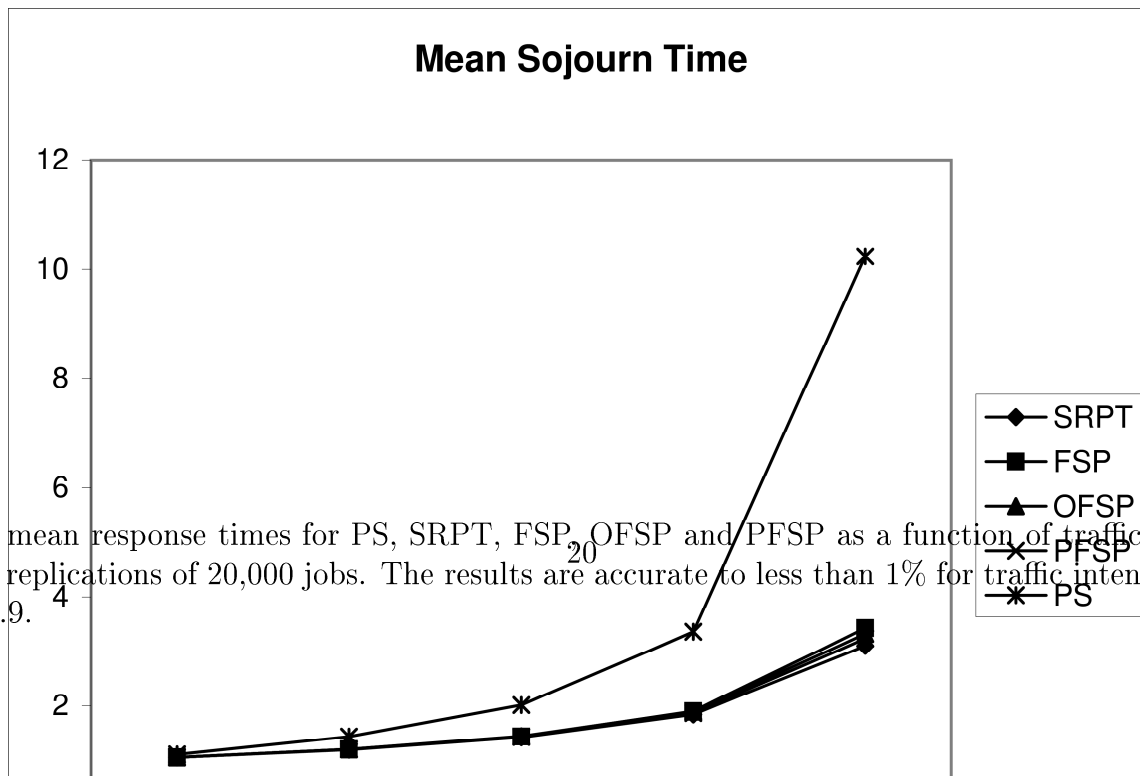


Figure 1: Estimated mean response times for PS, SRPT, FSP, OFSP and PFSP as a function of traffic intensity. The results show averages for 50 replications of 20,000 jobs. The results are accurate to less than 1% for traffic intensity 0.1 through to 2% for traffic intensity 0.9.

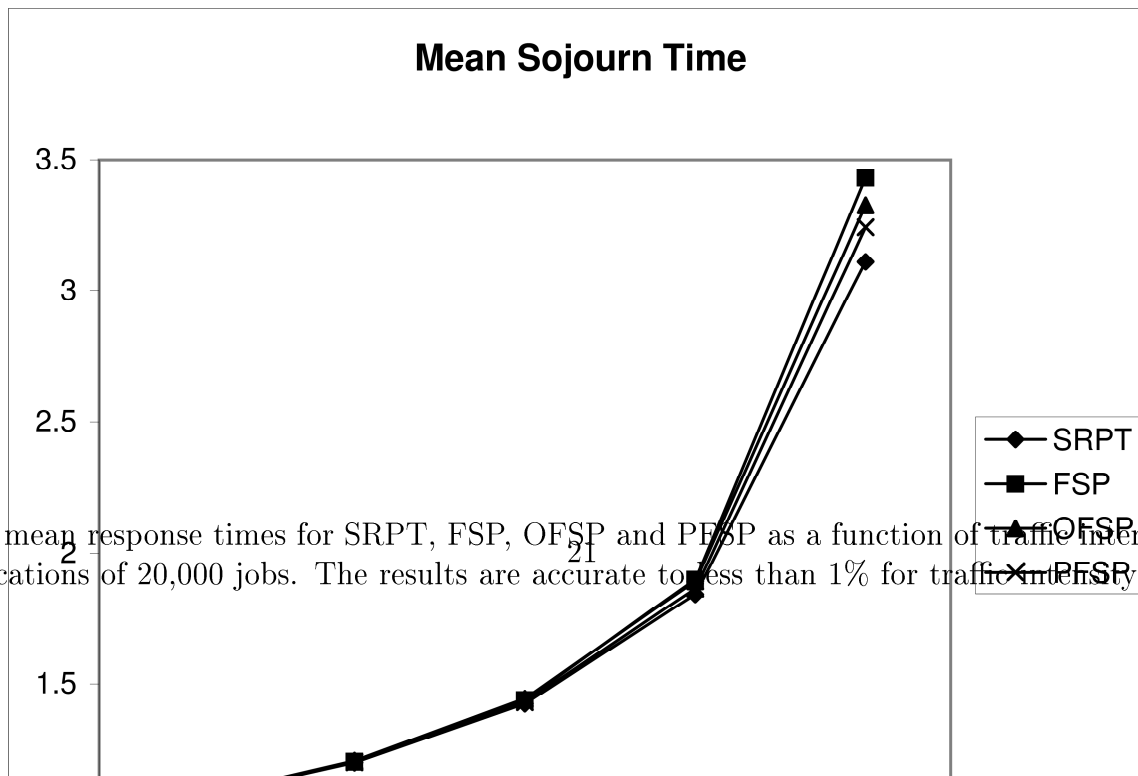


Figure 2: Estimated mean response times for SRPT, FSP, OFSP and PFSP as a function of traffic intensity. The results show averages for 50 replications of 20,000 jobs. The results are accurate to less than 1% for traffic intensity 0.1 through to 2% for traffic intensity 0.9.

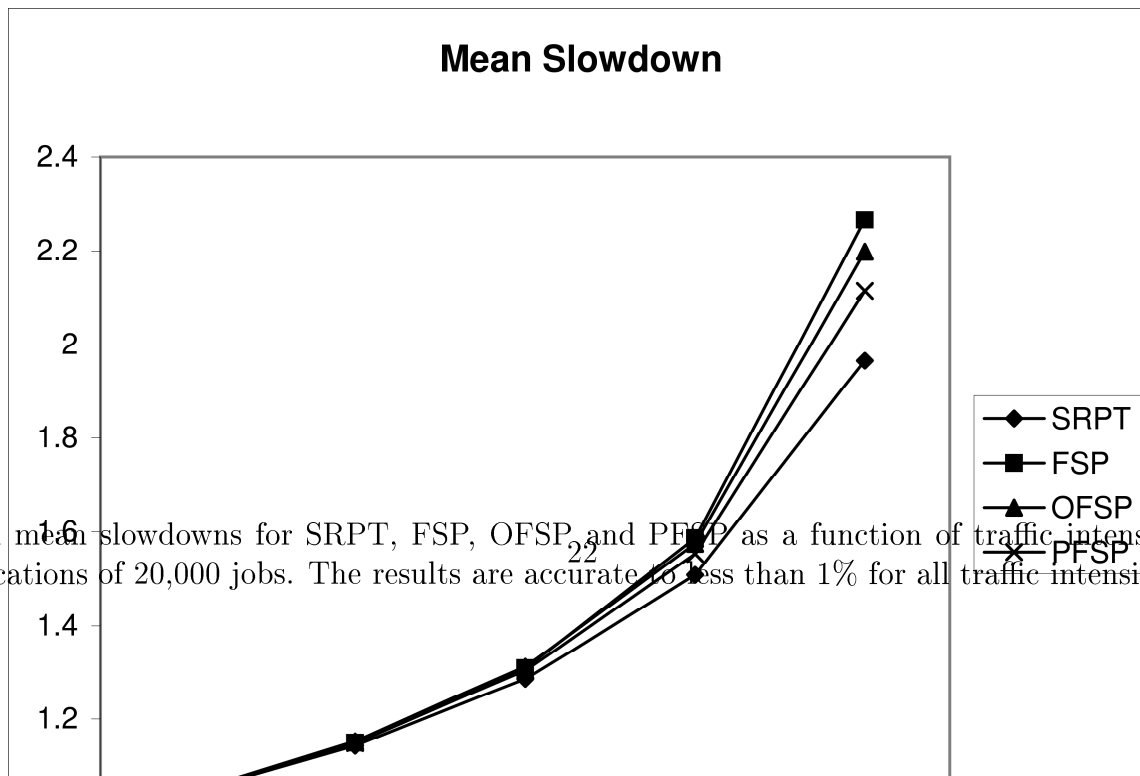


Figure 3: Estimated mean slowdowns for SRPT, FSP, OFSP, and PFSP as a function of traffic intensity. The results show averages for 50 replications of 20,000 jobs. The results are accurate to less than 1% for all traffic intensities.

Simulations – Summary

- FSP close to SRPT, but non-negligible gap under heavy loads.
- OFSP and PFSP close the gap considerably.
- PFSP almost always wins.
- Note: PFSP not very complex to implement.
- More sophisticated (i.e., adaptive FSP) did not beat PFSP.

Conclusions and Future Research

- Open questions: PFSP over FSP, online bounds.
- Trace driven simulations
- Multi-server systems.
- Unknown job sizes
- Pre-emption costs.
- TCP considerations.
- Implementation into apache? smarter web servers, etc...