Split proof scheduling

1 Setting

There are n jobs with sizes $p \in R_+^n$ to be processed on a server with capacity 1. Jobs arrive at times d_i (unknown to the server) and are completed by some time C_i, and we are interested in minimizing expected total flow

$F = \sum F_i = \sum (C_i - d_i)$.

We say scheduling mechanism is split proof if max waiting time of any two jobs is at least the waiting time of a single job with size equal two the sum of sizes of the two jobs.

2 Exponential Scheduling

Consider the following online scheduling algorithm. When a job of size p_i arrives, we generate a random score $v_i \sim exp(\lambda = p_i)$. Then at any time the server processes the job with highest score.

Additionally, scores of jobs that were processed for some time t' are adjusted as $v_i' = \frac{vp_i}{p_i - t'}$. Computationally, we only need to do this reassignment when a new job arrives. Also note that for any t' the distribution for v_i' is $v_i' \sim exp(\lambda = p_i - t')$ (scaling property of exponential distribution).

Theorem 2.1. *Exponential scheduling is split-proof*

Proof. This is a pretty simple argument □

3 Flow guarantees for exponential scheduling

Theorem 3.1. *When $d_i = 0$ for all i, exponential scheduling gives a 2-approximation for optimal flow.*

Proof. We prove this by proving 2-approximation for total delay, $D = \sum (C_i - p_i)$.

$$D = \sum_i \sum_{j \neq i} D_{ij},$$

where D_{ij} is delay that job j imposes on job i. Suppose jobs are sorted in ascending order. Than, in optimal schedule we have $D_{ij} + D_{ji} = min(p_i, p_j)$.

For exponential scheduling, for any two jobs, we have $\mathbb{P}[i > j] = \frac{p_i}{p_i + p_j}$. ($i > j$ means i is scheduled after j). Thus, we have $D_{ij} + D_{ji} = \frac{2p_i p_j}{p_i + p_j} \leq 2min(p_i, p_j)$. Done. □

Unfortunately, this does not extend to arbitrary release dates. Consider the following instance: on $t = 1$, n jobs are released, $n-1$ of them are of size 1 and the last one is of size A. Then, on turn $t = n-1$ new jobs of size 1 start arriving at every integer time, for a long-long time. For large enough n, w.h.p. exponential scheduling would result in a queue of size A by turn $t = n-1$ and so there is no constant factor approximation.

Conjecture 3.2. *If server is sped up by a factor of 2, exponential scheduling is 1-competitive with the optimal schedule.*

(Really, I would be happy with any constant factor approximation and any speed up).

3.1 Some useful facts

I list a few things that I tried in case they're useful.

1) One could try to show, that for significantly large speed-up, N_t, number of jobs in queue at times t is always smaller for the exp scheduling. This is unfortunately, not true, the instance that breaks it for any speed-up is $p_i = i$.

2) Since we know that Prus's algo that processes the least processed job is competitive under speed-up, we can try and show that exp scheduling is competitive with it rather than with SRPT. I couldn't find a way to use this.

3) It turns out that a simpler randomized algo is provably always worse than exp scheduling and might be simpler to analyze: for every job you sample it's virtual size $p'_i \sim u[0, p_i]$ and process shortest remaining virtual size. Couldn't do anything with this either.