The Price of Non-Clairvoyance
Online Scheduling to Minimize Average Slowdown

J. Könemann
jochen@cmu.edu

GSIA
Carnegie Mellon University

Joint with N. Bansal, K. Damdhere, A. Sinha
Outline

- Problem definition.
Outline

• Problem definition.
• How well can we do? Lower bounds.
Outline

- Problem definition.
- How well can we do? Lower bounds.
- An algorithm and its analysis.
Single machine scheduling

- Have scheduling instance $J = \{J_1, \ldots, J_n\}$ with processing times $p_1, \ldots, p_n$ and release dates $r_1, \ldots, r_n$. 
Single machine scheduling

- Have scheduling instance $J = \{J_1, \ldots, J_n\}$ with processing times $p_1, \ldots, p_n$ and release dates $r_1, \ldots, r_n$.
- Can run one job at a time and are allowed to preempt running jobs.
Single machine scheduling

- Have scheduling instance \( J = \{ J_1, \ldots, J_n \} \) with processing times \( p_1, \ldots, p_n \) and release dates \( r_1, \ldots, r_n \).
- Can run one job at a time and are allowed to preempt running jobs.
- A feasible schedule \( S \) for \( J \) executes a unique job \( S(t) \) at each time \( t \in [1, T] \) s.t.
  \[
  \{ t \in [1, T] : S(t) = J_i \} = p_i
  \]
  for all \( 1 \leq i \leq n \).
Performance Measures

- Popular performance measure is the time that a job spends in the system.

**Response time of job** $J_i$:

$$rt_i = f_i - r_i$$

More recently: try to introduce fairness. Time in system should scale with job-size.

**Slowdown of Job** $J_i$:

$$sl_i = rt_i$$

Useful metric to evaluate web-server performance: [Harchol-Balter '98]
Performance Measures

- Popular performance measure is the time that a job spends in the system.
  
  **Response time of job** $J_i$:
  
  $$rt_i = f_i - r_i$$
  
- More recently: try to introduce fairness. Time in system should scale with job-size.
  
  **Slowdown of Job** $J_i$:
  
  $$sl_i = rt_i/p_i$$
Performance Measures

- Popular performance measure is the time that a job spends in the system.
  Response time of job $J_i$:
  \[ rt_i = f_i - r_i \]

- More recently: try to introduce fairness. Time in system should scale with job-size.
  Slowdown of Job $J_i$:
  \[ sl_i = rt_i / p_i \]

Useful metric to evaluate web-server performance: [Harchol-Balter ’98]
Model

• Online scheduling: we know job $J_i$ only at time $r_i$. 
Model

- Online scheduling: we know job $J_i$ only at time $r_i$.
- Non-clairvoyance: don’t know $p_i$ when $J_i$ arrives. 
  $\Rightarrow$ realistic model for many systems applications.
Model

- Online scheduling: we know job $J_i$ only at time $r_i$.
- Non-clairvoyance: don’t know $p_i$ when $J_i$ arrives.  
  $\Rightarrow$ realistic model for many systems applications.
- For scheduling instance $J$, let
  $$SL(J) \quad \text{total slowdown of our algorithm}$$
  $$SL^*(J) \quad \text{total slowdown of an optimum offline algorithm}$$
Competitive analysis

- Measure quality of our algorithm in an adversarial model: adversary chooses instance $J$. Look at ratio

$$\rho := \frac{SL(J)}{SL^*(J)}$$
Competitive analysis

• Measure quality of our algorithm in an adversarial model: adversary chooses instance \( J \). Look at ratio

\[
\rho := \frac{SL(J)}{SL^*(J)}
\]

• Turns out that adversary is too strong to say anything meaningful.

[Kalyan. and Pruhs ’95]: Resource augmentation. Weaken adversary and give our algorithm \( k \) times faster processor:

\[
\rho_k := \frac{SL_k(J)}{SL^*(J)}
\]
Previous work (online, clairvoyant)

[Bender et al. ’98]  \( \text{Min } \max_i s_l_i \text{ and } \max_i r_t_i \)

[Muthukrishnan et al. ’99] \( \text{SRPT is } 2\text{-competitive for } SL \)

[Chekuri, Khanna, Zhu ’01] \( O(\log^2(B)) \) comp. ratio for weighted \( r_t \), \( B := \frac{p_{\text{max}}}{p_{\text{min}}} \)

[Chekuri&Khanna ’02] Quasi-polynomial time approx.-scheme for weighted \( r_t \)

[Becchetti et al. ’01] \( 1 + \epsilon \)-comp. ratio with speed-up \( 1 + 1/\epsilon \) for weighted \( r_t \)
Previous work (online, non-clairvoyant, avg rt)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kalyan.&amp;Pruhs ’95]</td>
<td>Introduce speedup. Deterministic algorithm. $(1 + 1/\epsilon)$-comp. with $(1 + \epsilon)$-speedup.</td>
</tr>
<tr>
<td>[Berman, Coulston ’99]</td>
<td>Improve [K&amp;P ’95], $2/v$-comp. with $v$ speedup.</td>
</tr>
</tbody>
</table>
Our results

- Lower bounds (deterministic and randomized):
  \[
  \text{SL, } k\text{-speed} \quad \Omega(n/k^3)
  
  \text{SL, } 1\text{-speed, } \frac{p_{\max}}{p_{\min}} \leq B \quad \Omega(B)
  
  \text{SL, } k\text{-speed, } \frac{p_{\max}}{p_{\min}} \leq B, r_i = 0 \quad \Omega(\log(B)/k)
  \]

- Algorithms (concentrate on bd. job sizes):
  \[
  \text{SL, } \frac{p_{\max}}{p_{\min}} \leq B, r_i = 0, 1\text{-speed} \quad O(\log(B))
  
  \text{SL, } \frac{p_{\max}}{p_{\min}} \leq B, (1 + \epsilon) \log_{1+\epsilon}(B)\text{-speed} \quad O(\log_{1+\epsilon}^2(B))
  \]
Our results

- Lower bounds (deterministic and randomized):
  \[
  \begin{align*}
  &\text{SL, } k\text{-speed} & \Omega(n/k^3) \\
  &\text{SL, 1-speed, } \frac{p_{\max}}{p_{\min}} \leq B & \Omega(B) \\
  &\text{SL, } k\text{-speed, } \frac{p_{\max}}{p_{\min}} \leq B, r_i = 0 & \Omega(\log(B)/k)
  \end{align*}
  \]

- Algorithms (concentrate on bd. job sizes):
  \[
  \begin{align*}
  &\text{SL, } \frac{p_{\max}}{p_{\min}} \leq B, r_i = 0, 1\text{-speed} & O(\log(B)) \\
  &\text{SL, } \frac{p_{\max}}{p_{\min}} \leq B, (1 + \epsilon) \log_{1+\epsilon}(B)\text{-speed} & O(\log^2_{1+\epsilon}(B))
  \end{align*}
  \]

This presentation, use \(\epsilon = 1\)
Outline

- Problem definition.
- **How well can we do? Lower bounds.**
- An algorithm and its analysis.
Lower bound for bd. job sizes (deterministic)

- Assume $B = 2^N$. Consider instance with $N$ jobs:
  
  $$p_i = 2^i \text{ and } r_i = 0$$
Lower bound for bd. job sizes (deterministic)

- Assume $B = 2^N$. Consider instance with $N$ jobs:
  
  $$p_i = 2^i$$ and $$r_i = 0$$

- How does SRPT do on this instance?
Lower bound for bd. job sizes (deterministic)

• Assume $B = 2^N$. Consider instance with $N$ jobs:
  \[ p_i = 2^i \text{ and } r_i = 0 \]

• How does SRPT do on this instance?

\[
SL_{SRPT} = \sum_{i=0}^{N} 1 + \frac{1}{2^i} \sum_{j<i} 2^j = O(\log B).
\]
Lower bound for bd. job sizes (deterministic)

- Assume $B = 2^N$. Consider instance with $N$ jobs:
  \[ p_i = 2^i \text{ and } r_i = 0 \]
- How does SRPT do on this instance?
  \[
  SL_{SRPT} = \sum_{i=0}^{N} 1 + \frac{1}{2^i} \sum_{j < i} 2^j = O(\log B).
  \]
- How does arbitrary deterministic algorithm $A$ do?
  ⇒ Can figure out order of jobs that receive $2^i$ work.
  Let $J_i$ be the $(N - i)^{th}$ job in this order.
Lower bound for bd. job sizes (deterministic)

- Assume $B = 2^N$. Consider instance with $N$ jobs:
  \[ p_i = 2^i \text{ and } r_i = 0 \]

- How does SRPT do on this instance?
  \[
  SL_{SRPT} = \sum_{i=0}^{N} 1 + \frac{1}{2^i} \sum_{j<i} 2^j = O(\log B).
  \]

- How does arbitrary deterministic algorithm $A$ do?
  \[ \Rightarrow \text{Can figure out order of jobs that receive } 2^i \text{ work.} \]
  Let $J_i$ be the $(N - i)^{th}$ job in this order.
  \[
  SL_A = \sum_{i=0}^{N} \frac{1}{2^i} (N - i) 2^i = \Omega(\log^2 B).
  \]
Randomization does not help either

- Recall Yao’s minimax principle:

\[
\min_A E_{\mathcal{J}} \left[ \frac{SL_A(J)}{SL^*(J)} \right] \leq \max_J E_A \left[ \frac{SL_A(J)}{SL^*(J)} \right]
\]

\(A\): distribution over deterministic algorithms \\
\(\mathcal{J}\): distribution over instances \\
This gives you a bound on the competitive ratio of any randomized algorithm.
Randomization does not help either

• Recall Yao’s minimax principle:

\[
\min_A E_J \left[ \frac{SL_A(J)}{SL^*(J)} \right] \leq \max_J E_A \left[ \frac{SL_A(J)}{SL^*(J)} \right]
\]

\(A\): distribution over deterministic algorithms
\(J\): distribution over instances
This gives you a bound on the competitive ratio of any randomized algorithm.

• Proof-Idea: Pick random distribution on instances \(J\) and show that any deterministic algorithm \(A\) has

\[
E_J \left[ \frac{SL_A(J)}{SL^*(J)} \right] \geq \rho
\]
Randomization does not help either

- Recall Yao’s minimax principle:

\[
\min_A \mathbb{E}_\mathcal{J} \left[ \frac{\text{SL}_A(J)}{\text{SL}^*(J)} \right] \leq \max_J \mathbb{E}_A \left[ \frac{\text{SL}_A(J)}{\text{SL}^*(J)} \right]
\]

\(A\): distribution over deterministic algorithms
\(\mathcal{J}\): distribution over instances
This gives you a bound on the competitive ratio of any randomized algorithm.

- Proof-Idea: Pick random distribution on instances \(\mathcal{J}\) and show that any deterministic algorithm \(A\) has

\[
\mathbb{E}_\mathcal{J} \left[ \frac{\text{SL}_A(J)}{\text{SL}^*(J)} \right] \geq \rho
\]

\(\Rightarrow\) no algorithm can have comp. ratio less than \(\rho\)
Randomization does not help either (ctd)

- Random instance on $N = \log(B)$ jobs:
  Pick random $\pi \in \sigma_N$ and let $p_{\pi(i)} = 2^i$.
Randomization does not help either (ctd)

- Random instance on $N = \log(B)$ jobs:
  Pick random $\pi \in \sigma_N$ and let $p_{\pi(i)} = 2^i$.
- Let $t_i$ s.t. $p_{\pi(i)}(t_i) \geq 2^i$. Define event for all $i < j$:

  $$A_{ij} : p_{\pi(j)}(t_i) \geq 2^i$$
Randomization does not help either (ctd)

- Random instance on $N = \log(B)$ jobs:
  Pick random $\pi \in \sigma_N$ and let $p_{\pi(i)} = 2^i$.
- Let $t_i$ s.t. $p_{\pi(i)}(t_i) \geq 2^i$. Define event for all $i < j$:
  $$A_{ij} : p_{\pi(j)}(t_i) \geq 2^i$$
- Claim: $\Pr[A_{ij}] = 1/2$
Randomization does not help either (ctd)

- Random instance on $N = \log(B)$ jobs: Pick random $\pi \in \sigma_N$ and let $p_{\pi(i)} = 2^i$.
- Let $t_i$ s.t. $p_{\pi(i)}(t_i) \geq 2^i$. Define event for all $i < j$:
  \[ A_{ij} : p_{\pi(j)}(t_i) \geq 2^i \]
- Claim: $\Pr[A_{ij}] = 1/2$
- Hence
  \[
  E[1_A(J_{\pi(i)})] \geq \sum_{j > i} \Pr[A_{ij}] = \frac{N - i}{2}
  \]
  \[
  \Rightarrow E[SL_A] \geq \frac{1}{2} \sum_{i=1}^{N} (N - i) = \Omega(\log^2(B))
  \]
Proof of Claim

- Idea: Decide on processing time of jobs based on working of algorithm $A$
- Let $J_1, \ldots, J_N$ be order in which jobs receive $2^i$ work in algorithm $A$. 

\[ A_{ij} \text{ holds iff } J_i(j) \text{ comes earlier than } J_j(i) \text{ in this order.} \]

That's true for half of all permutations!

Lower bound follows from earlier fact that $SRPT$ has $SL_{SRPT} = O(\log(B))$.
Proof of Claim

- Idea: Decide on processing time of jobs based on working of algorithm $A$
- Let $J_1, \ldots, J_N$ be order in which jobs receive $2^i$ work in algorithm $A$.
- $A_{ij}$ holds iff $J_{\pi(j)}$ comes earlier than $J_{\pi(i)}$ in this order.
Proof of Claim

• Idea: Decide on processing time of jobs based on working of algorithm $A$

• Let $J_1, \ldots, J_N$ be order in which jobs receive $2^i$ work in algorithm $A$.

• $A_{ij}$ holds iff $J_{\pi(j)}$ comes earlier than $J_{\pi(i)}$ in this order. 
  $\Rightarrow$ That’s true for half of all permutations!
Proof of Claim

- Idea: Decide on processing time of jobs based on working of algorithm $A$
- Let $J_1, \ldots, J_N$ be order in which jobs receive $2^i$ work in algorithm $A$.
- $A_{ij}$ holds iff $J_{\pi(j)}$ comes earlier than $J_{\pi(i)}$ in this order.
  $\Rightarrow$ That’s true for half of all permutations!
- Lower bound follows from earlier fact that SRPT has

$$S_L_{SRPT} = O(\log(B))$$
Outline

- Problem definition.
- How well can we do? Lower bounds.
- An algorithm and its analysis.
An algorithm for dynamic instances

- Know: SRPT is approximately optimum policy for mean slowdown.

⇒ big jobs do not delay small jobs!
An algorithm for dynamic instances

- Know: SRPT is approximately optimum policy for mean slowdown.

⇒ big jobs do not delay small jobs!

- Here: have no knowledge about job sizes.
An algorithm for dynamic instances

- Know: SRPT is approximately optimum policy for mean slowdown.

  ⇒ big jobs do not delay small jobs!

- Here: have no knowledge about job sizes.
- Idea: use knowledge about age of job in system instead.
Algorithm: Details

- From the fact that we can preempt:
  Can think of $\log(B)$-speed machine as $\log(B)$ single-speed machines: $M_1, \ldots, M_N$. 
Algorithm: Details

- From the fact that we can preempt:
  Can think of $\log(B)$-speed machine as $\log(B)$ single-speed machines: $M_1, \ldots, M_N$.
- Execute job $J$ on machine $M_i$ at time $t$ if
  \[ 2^{i-1} \leq p_J(t) < 2^i \]
Algorithm: Details

- From the fact that we can preempt:
  Can think of $\log(B)$-speed machine as $\log(B)$ single-speed machines: $M_1, \ldots, M_N$.

- Execute job $J$ on machine $M_i$ at time $t$ if
  \[ 2^{i-1} \leq p_J(t) < 2^i \]

- ... Jobs move from $M_1$ to $M_N$ as they age
Algorithm: Details

- From the fact that we can preempt:
  Can think of $\log(B)$-speed machine as $\log(B)$ single-speed machines: $M_1, \ldots, M_N$.
- Execute job $J$ on machine $M_i$ at time $t$ if
  $$2^{i-1} \leq p_J(t) < 2^i$$
- ... Jobs move from $M_1$ to $M_N$ as they age
- Within each machine run First-Come-First-Serve (FCFS).
Algorithm: Details

- From the fact that we can preempt:
  Can think of \( \log(B) \)-speed machine as \( \log(B) \) single-speed machines: \( M_1, \ldots, M_N \).
- Execute job \( J \) on machine \( M_i \) at time \( t \) if
  \[
  2^{i-1} \leq p_J(t) < 2^i
  \]
- ... Jobs move from \( M_1 \) to \( M_N \) as they age
- Within each machine run First-Come-First-Serve (FCFS).
- Refer to our algorithm as Aging FCFS (AFCFS)
An example
An example
An example
An example
An example
Outline

- Problem definition.
- How well can we do? Lower bounds.
- Algorithm: Aging FCFS.
  - Overview
  - Analyze SPT
  - Analyze AFCFS
Analysis: **Overview**

- We compare ourselves to **Shortest-Processing-Time (SPT)**: always run the job that has minimum $p_j$. Use FCFS to break ties.

[From Becchetti et al. '01]: SPT is $O(1)$-speed, $O(1)$-competitive for avg. slowdown.

Derive bounds on slowdown of AFCFS by examining analysis of SPT.

For simplicity: assume job-sizes are powers of 2.
Analysis: Overview

- We compare ourselves to Shortest-Processing-Time (SPT): always run the job that has minimum $p_j$. Use FCFS to break ties.

- From [Becchetti et al. ’01]: SPT is $O(1)$-speed, $O(1)$-competitive for avg. slowdown.
Analysis: Overview

- We compare ourselves to Shortest-Processing-Time (SPT): always run the job that has minimum $p_j$. Use FCFS to break ties.
- From [Becchetti et al. ’01]: SPT is $O(1)$-speed, $O(1)$-competitive for avg. slowdown.
- Derive bounds on slowdown of AFCFS by examining analysis of SPT.
Analysis: Overview

- We compare ourselves to Shortest-Processing-Time (SPT): always run the job that has minimum $p_j$. Use FCFS to break ties.
- From [Becchetti et al. ’01]: SPT is $O(1)$-speed, $O(1)$-competitive for avg. slowdown.
- Derive bounds on slowdown of AFCFS by examining analysis of SPT.
- For simplicity: assume job-sizes are powers of 2.
Outline

- Problem definition.
- How well can we do? Lower bounds.
- Algorithm: Aging FCFS.
  - Overview
  - Analyze SPT
  - Analyze AFCFS
Analysis of SPT

- At time $t$ SPT has worked on at most one of the active jobs of size $2^i$. 

Idea: In AFCFS bound slowdown of job $J$ of size $2^k$ that comes in at time $t$ by previous expression.

Problem: Jobs of size $2^j$ for $j > k$ will hurt us a little.
Analysis of SPT

- At time $t$ SPT has worked on at most one of the active jobs of size $2^i$.
- Let $n_i(t) \in \mathbb{R}^+$ remaining unfinished jobs of size $2^i$.
Analysis of SPT

- At time $t$ SPT has worked on at most one of the active jobs of size $2^i$.
- Let $n_i(t) \in \mathbb{R}^+$ remaining unfinished jobs of size $2^i$.
- Suppose, job $J$ of size $2^k$ comes in at time $t$. What is its slowdown?
Analysis of SPT

- At time $t$ SPT has worked on at most one of the active jobs of size $2^i$.
- Let $n_i(t) \in \mathbb{R}^+$ remaining unfinished jobs of size $2^i$.
- Suppose, job $J$ of size $2^k$ comes in at time $t$. What is its slowdown?

$$s_{\text{SPT}}(J) \geq \sum_{j \leq k} 2^{j-k} n_j(t)$$
Analysis of SPT

- At time $t$ SPT has worked on at most one of the active jobs of size $2^i$.
- Let $n_i(t) \in \mathbb{R}^+$ remaining unfinished jobs of size $2^i$.
- Suppose, job $J$ of size $2^k$ comes in at time $t$. What is its slowdown?

$$\text{sl}_{SPT}(J) \geq \sum_{j \leq k} 2^{j-k} n_j(t)$$

- Idea: In AFCFS bound slowdown of job $J$ of size $2^k$ that comes in at time $t$ by previous expression.
Analysis of SPT

- At time $t$ SPT has worked on at most one of the active jobs of size $2^i$.
- Let $n_i(t) \in \mathbb{R}^+$ remaining unfinished jobs of size $2^i$.
- Suppose, job $J$ of size $2^k$ comes in at time $t$. What is its slowdown?

$$s_{\text{SPT}}(J) \geq \sum_{j \leq k} 2^{j-k} n_j(t)$$

- Idea: In AFCFS bound slowdown of job $J$ of size $2^k$ that comes in at time $t$ by previous expression.
- Problem: Jobs of size $2^j$ for $j > k$ will hurt us a little.
Lessons for analyzing AFCFS

• Recall: for job $J$ of size $2^k$ at time $t$

$$s \ll SPT(J) \geq \sum_{j \leq k} 2^{j-k} n_j(t)$$

• Idea: Reuse $n_j(t)$ term up to $N^2$ times in accounting for slowdown of AFCFS.
Outline

- Problem definition.
- How well can we do? Lower bounds.
- Algorithm: **Aging FCFS**.
  - Overview
  - Analyze SPT
  - Analyze AFCFS
Analysis of AFCFS

- Modified Bound for job of size $2^k$:

$$s_1(J) \leq \sum_{j \leq k} 2^{j-k} n_j(t) + \sum_{j > k} u_{jk}(t)$$

where $0 \leq u_{jk}(t) \leq n_j(t)$ are potentials to be defined.
Analysis of AFCFS

- Modified Bound for job of size $2^k$:

$$s_1(J) \leq \sum_{j \leq k} 2^{j-k} n_j(t) + \sum_{j > k} u_{jk}(t)$$

where $0 \leq u_{jk}(t) \leq n_j(t)$ are potentials to be defined.

- Intuitively, $u_{jk}(t)$ is the portion of slowdown due to jobs of size $2^j$ that is unused by jobs of size $2^k$.
Analysis of AFCFS

- Modified Bound for job of size $2^k$:

$$s_1(J) \leq \sum_{j \leq k} 2^{j-k} n_j(t) + \sum_{j > k} u_{jk}(t)$$

where $0 \leq u_{jk}(t) \leq n_j(t)$ are potentials to be defined.

- Intuitively, $u_{jk}(t)$ is the portion of slowdown due to jobs of size $2^j$ that is unused by jobs of size $2^k$.

- Whenever job of size $2^k$ arrives:
  1. decrease $u_{jk}(t)$ for $j > k$ to reflect charge
  2. increase $u_{kj}(t)$ for all $j \leq k$ to allow smaller jobs to charge to $J$
Analysis of AFCFS

- Modified Bound for job of size $2^k$:

$$s_1(J) \leq \sum_{j \leq k} 2^{j-k} n_j(t) + \sum_{j > k} u_{jk}(t)$$

where $0 \leq u_{jk}(t) \leq n_j(t)$ are potentials to be defined.

- Intuitively, $u_{jk}(t)$ is the portion of slowdown due to jobs of size $2^j$ that is unused by jobs of size $2^k$.

- Whenever job of size $2^k$ arrives:
  1. decrease $u_{jk}(t)$ for $j > k$ to reflect charge
  2. increase $u_{kj}(t)$ for all $j \leq k$ to allow smaller jobs to charge to $J$

- Whenever SPT finishes job of size $2^k$:
  decrement $u_{jk}(t)$ for $j \leq k$. 
Details: lessons for analyzing AFCFS

• Interpretation of $u_{jk}(t)$: Charges used to pay for slowdown of $J$ of size $2^k$ due to current jobs of size $2^j$ in AFCFS by appropriate updates.
Details: lessons for analyzing AFCFS

- Interpretation of $u_{jk}(t)$: Charges used to pay for slowdown of $J$ of size $2^k$ due to current jobs of size $2^j$ in AFCFS by appropriate updates.
- In $S\underline{L}_{SPT}$, $\lceil n_j(t) \rceil$ is the contribution of some job of size $2^j$. 

Skip proof details. – p.26
Details: lessons for analyzing AFCFS

- Interpretation of $u_{jk}(t)$: Charges used to pay for slowdown of $J$ of size $2^k$ due to current jobs of size $2^j$ in AFCFS by appropriate updates.

- In $S\subseteq_{SPT}$, $[n_j(t)]$ is the contribution of some job of size $2^j$.

- Crux of proof argues that copying $n_j(t)$ up to $N^2$ times, we can upper bound $\sum_{k\leq j} u_{jk}(t)$.

Skip proof details.
Potential updates: details

- Initially: $u_{ij}(0) = n_{i}(0) = 0$
Potential updates: details

- Initially: $u_{ij}(0) = n_i(0) = 0$
- [Job of size $2^k$ arrives]
  1. $u_{jk}(t) = u_{jk}(t) - 1/N$ for $j > k$
  2. $u_{kj}(t) = u_{kj}(t) + 1$ for $j \leq k$
Potential updates: details

- Initially: $u_{ij}(0) = n_i(0) = 0$

- [Job of size $2^k$ arrives]
  1. $u_{jk}(t) = u_{jk}(t) - 1/N$ for $j > k$
  2. $u_{kj}(t) = u_{kj}(t) + 1$ for $j \leq k$

- [Passage of time] When SPT is working on job of size $2^k$:

\[
\frac{d}{dt} u_{kj}(t) = \frac{1}{2^k}
\]

for all $j \leq k$
Maintain slowdown bound inductively

Claim: Let $w(k)$ be the total work that has to be done on active jobs until they reach machine $M_{k+1}$. Then,

$$
\frac{w(k)}{2^k} \leq \sum_{j > k} u_{jk}(t) + \sum_{j \leq k} 2^{j-k} n_j(t)
$$
Maintain slowdown bound inductively

Claim: Let \( w(k) \) be the total work that has to be done on active jobs until they reach machine \( M_{k+1} \). Then,

\[
\frac{w(k)}{2^k} \leq \sum_{j>k} u_{jk}(t) + \sum_{j\leq k} 2^{j-k} n_j(t)
\]

• true initially
Maintain slowdown bound inductively

Claim: Let $w(k)$ be the total work that has to be done on active jobs until they reach machine $M_{k+1}$. Then,

$$\frac{w(k)}{2^k} \leq \sum_{j>k} u_{jk}(t) + \sum_{j \leq k} 2^{j-k} n_j(t)$$

- true initially
- Job $J$ of size $2^j$ arrives...
  1. $j \leq k$: lhs increase by $2^{j-k}$, same on rhs since $n_j$ increases also by 1
  2. $j > k$: lhs increase by 1, rhs increase same since $u_{jk}$ increases by 1
Maintain slowdown bound inductively..

- When nothings arrives → passage of time:
  \[
  \frac{d}{dt} lhs = 2^{-k}.
  \]
  SPT is working on job of size \(2^j\)...
  
  1. \(j \leq k\):
     \[
     \frac{d}{dt} rhs = 2^{-k}\] since \(\frac{d}{dt} n_j(t) = 2^{-j}\)
  2. \(j > k\):
     \[
     \frac{d}{dt} rhs = \frac{d}{dt} u_j < 2^{-k}\]
Maintain slowdown bound inductively.

- When nothings arrives → passage of time:
  \[ \frac{d}{dt} l h s = 2^{-k} . \]

  SPT is working on job of size \( 2^j \) ...

  1. \( j \leq k \): \( \frac{d}{dt} r h s = 2^{-k} \) since \( \frac{d}{dt} n_j(t) = 2^{-j} \)

  2. \( j > k \): \( \frac{d}{dt} r h s = \frac{d}{dt} u_j < 2^{-k} \)

This shows:

Slowdown of a job of size \( 2^k \) is bounded by

\[
\sum_{j>k} u_{jk}(t) + \sum_{j\leq k} 2^{j-k} n_j(t)
\]
Bounding the total slowdown

- There is an active job $J$ of size $2^j$ that contributes $[u_{jk}(t)]$ to $SL_{SPT}$.
Bounding the total slowdown

- There is an active job $J$ of size $2^j$ that contributes $[u_{jk}(t)]$ to $S_{SPT}$.
- We use $J$’s slowdown at most $N^2$ times: whenever we use $J$, we decrease $u_{jk}$ by $1/N$ and when $J$ arrives, we increase at most $N$ potentials $u_{jj}, \ldots, u_{jN}$.
Bounding the total slowdown

- There is an active job $J$ of size $2^j$ that contributes $[u_{jk}(t)]$ to $S\mathcal{L}_{SPT}$.
- We use $J$’s slowdown at most $N^2$ times: whenever we use $J$, we decrease $u_{jk}$ by $1/N$ and when $J$ arrives, we increase at most $N$ potentials $u_{jj}, \ldots, u_{jN}$.

$$\rightarrow S\mathcal{L}_{AFCFS} = O(\log^2(B))S\mathcal{L}_{SPT}$$
Open problems

• Close gap between lower-bound and upper-bound: maybe better lower bound that uses non-zero release-dates?
• Non-clairvoyance is largely unexplored. There are many open questions out there!