Balancing SRPT and FCFS via Starvation Mitigation

Tung-Wei Kuo∗

Abstract
The assessment of a job’s Quality of Service (QoS) often revolves around its flow time, also referred to as response time. This study delves into two fundamental objectives for scheduling jobs: the average flow time and the maximum flow time. While the Shortest Remaining Processing Time (SRPT) algorithm minimizes average flow time, it can result in job starvation, causing certain jobs to experience disproportionately long and unfair flow times. In contrast, the First-Come-First-Served (FCFS) algorithm minimizes the maximum flow time but may compromise the average flow time. To strike a balance between these two objectives, a common approach is to minimize the ℓ₂ norm of flow time. In this study, we provide theoretical and experimental evidence that a judicious combination of SRPT and FCFS yields superior performance compared to standard heuristics for minimizing the ℓ₂ norm of flow time.

1 Introduction
In a server system, the Quality of Service (QoS) of a job is normally measured by its response time, which is defined as the amount of time between job release and job completion. In job scheduling, the response time is also called the flow time. Specifically, for any schedule $S$, let $c_i(S)$ be the completion time of job $J_i$ under $S$ and $r_i$ be the release time of $J_i$. Then the flow time of a job $J_i$ under $S$, denoted by $f_i(S)$, is defined as

$$f_i(S) = c_i(S) - r_i.$$ 

Two natural QoS objectives for a server system is the average, or equivalently, ℓ₁ norm, of job flow times, and the maximum, or ℓ_∞ norm, of job flow times. It is well-known that Shortest Remaining Processing Time (SRPT), which minimizes average flow time, may cause some jobs to have long and inequitable flow times (i.e., job starvation). In contrast, First-Come-First-Served (FCFS), which minimizes the maximum flow time [8], may deteriorate the average flow time.

Example. In Fig. 1, jobs $J_1$ and $J_2$ are released at time 1, and both have processing times of 2. All the other jobs $J_i$ are released at time $i$ and have unit processing times. Assume that under SRPT, $J_1$ is executed first. Thus, $J_2$ is starving in the sense that it has to wait until all the other jobs are completed. Under FCFS, however, the flow time of every job is only $O(1)$.

Example. Consider Fig. 2. Compared with Fig. 1, the processing times of $J_1$ and $J_2$ in Fig. 2 are increased to $n$. Under FCFS, the flow time of every job is Θ($n$), and thus FCFS’s average flow time is Θ($n$). Under SRPT, however, the average flow time is only $O(1)$.

The standard approach to balance the ℓ₁ norm and the ℓ_∞ norm of flow time is minimizing the ℓ₂ norm of flow time [6, 19]. Specifically, the ℓ₂ norm of flow time under schedule $S$ is

$$\sqrt{f_1(S)^2 + f_2(S)^2 + \cdots + f_n(S)^2}.$$ 

In this paper, we aim to minimize the ℓ₂ norm of flow time online on a single machine. Specifically, an online algorithm does not know job $J_i$ until its release. We consider preemptive (i.e., the scheduler can suspend a job and later resumes the execution of the job from the...
point of suspension) and clairvoyant (i.e., the scheduling algorithm learns the size of a job when the job is released) scheduling algorithms. For example, SRPT is a preemptive and clairvoyant scheduling algorithm. While we consider the online setting, we assume that an estimate of the number of jobs is available (e.g., via historical data). At a high level, we show that a good estimate of \( n \) can significantly improve the \( \ell_2 \) norm of flow time.

For the online problem of minimizing the \( \ell_2 \) norm of flow time, only standard algorithms, including SRPT, Shortest Job First (SJF), Shortest Execution Time First (SETF), and Round Robin (RR), have been analyzed, assuming that the machine is gifted extra speed \([6, 19]\). However, SRPT, SJF, and SETF may cause job starvation, and FCFS and RR may deteriorate average flow time (see Appendix C). These drawbacks indicate that these algorithms may balance flow time inadequately and worsen the \( \ell_2 \) norm of flow time.

Despite their obvious drawback, SRPT and FCFS stand out from the aforementioned algorithms, as they complement each other. In particular, when one of the duo achieves a poor balance, it is usually the case that the other algorithm achieves an adequate balance. For instance, some empirical results have indicated that when the job size\(^1\) distribution is heavy-tailed, the fear that SRPT may cause starvation is unfounded \([3, 6, 9, 12, 17, 26]\). Thus, when the job size distribution is heavy-tailed, SRPT not only has an optimal average flow time, it also has a decent maximum flow time. In contrast, as exemplified in Fig. 2, FCFS may have a poor average flow time when the job size distribution is heavy-tailed.

In their seminal work, Bansal and Pruhs proved that SRPT achieves near optimal \( \ell_2 \) norm of flow time under speed augmentation \([6]\). Specifically, the competitive ratio of an algorithm \( A \) for minimizing \( \text{obj} \) is \( \max_{I} \text{obj}(A, I) / \text{obj}(\text{OPT}, I) \), where \( \text{obj}(A, I) \) is the objective value achieved by algorithm \( A \) on problem instance \( I \) and \( \text{OPT} \) is an optimal algorithm. \( A \) is \( c \)-competitive if its competitive ratio is at most \( c \). In addition, \( A \) is \( s \)-speed \( c \)-competitive if it is \( c \)-competitive when it is given a machine that is \( s \) times faster than the original machine. \([6]\) proved that SRPT is \((1 + \epsilon)\)-speed \( O(1/\epsilon) \)-competitive for minimizing the \( \ell_2 \) norm of flow time for any \( \epsilon > 0 \). In contrast, FCFS may have a poor \( \ell_2 \) norm of flow time even under speed augmentation.\(^2\)

There are conditions under which FCFS can achieve an adequate balance, but SRPT cannot. Let \( P \) be the ratio of the maximum job processing time to the minimum job processing time. In Appendix E, we show that when \( P = O(1) \), FCFS is \( O(1) \)-competitive for minimizing the \( \ell_2 \) norm of flow time. SRPT, in contrast, is not \( O(1) \)-competitive for minimizing the \( \ell_2 \) norm of flow time, even if \( P = O(1) \). For example, the instance depicted in Fig. 1 shows that SRPT may have a poor \( \ell_2 \) norm of flow time, even if \( P = 2 \).

Motivated by the above complementary relationship between SRPT and FCFS, we design an algorithm that balances SRPT and FCFS. The algorithm follows SRPT initially. For any job, if it has been in the system for too long and its remaining processing time is small enough, then the job becomes a starving job. Specifically, a job \( J_i \) becomes starving at time \( t \) if

\[
\frac{t - r_i}{\text{remaining processing time of } J_i} \geq \theta,
\]

where \( \theta \) is called the starvation threshold. Whenever some jobs are starving, the algorithm processes the job that becomes starving first. The formal description of the algorithm is presented in Section 2. Throughout this paper, we use \( BAL(\theta) \) to denote the algorithm when the starvation threshold is set to \( \theta \). For example, \( BAL(0) \) is equivalent to FCFS and \( BAL(\infty) \) is equivalent to SRPT. Thus, the best starvation thresholds for the \( \ell_1 \) norm and \( \ell_\infty \) norm of flow time are \( \infty \) and 0, respectively. This leads us to the following question:

What is the best starvation threshold \( \theta \) to minimize the \( \ell_2 \) norm of flow time?

1.1 Our Results. Our analysis on \( BAL(\theta) \)'s competitive ratio suggests that \( \theta = n^{\frac{1}{2}} \) is the best starvation threshold (Theorem 3.3). Among all the aforementioned standard scheduling algorithms, SRPT and FCFS achieve the best competitive ratio of \( O(n^{\frac{1}{2}}) \) for minimizing the \( \ell_2 \) norm of flow time (see Appendix C). We prove that \( BAL(\theta) \) achieves a significantly better competitive ratio of \( O(n^{\frac{1}{3}}) \) when the best starvation threshold \( \theta \) is applied.

Recall that SRPT and FCFS complement each other and achieve near optimal \( \ell_2 \) norm of flow time on various job size distributions. Consider an offline algorithm that executes SRPT and FCFS and then outputs the better schedule. In their seminal work, Bansal and Pruhs implicitly proved that for the problem of minimizing the \( \ell_2 \) norm of flow time, the competitive ratio of every online scheduling algorithm against the above offline algorithm is \( \Omega(n^{\frac{3}{4}}) \). \([6]\). In Appendix F, we slightly modify the proof in \([6]\) to show that the
same lower bound holds even if $n$ is known a priori. This lower bound suggests that designing an online algorithm comparable to both SRPT and FCFS is challenging.

Surprisingly, our experimental findings show that when the best starvation threshold $\theta$ is applied, $BAL(\theta)$ preserves the advantages of SRPT and FCFS under various job size distributions. Specifically, our experimental study shows that when the best starvation threshold $\theta$ is applied, $BAL(\theta)$:

1. outperforms FCFS and matches SRPT when the job size distribution is heavy-tailed;
2. outperforms SRPT and matches FCFS when jobs have similar sizes;
3. outperforms SRPT and FCFS simultaneously for other job size distributions.

1.2 Related Work. In their seminal work, Bansal and Pruhs introduced the online problem of minimizing the $\ell_p$ norm of flow time [6]. They proved that SRPT and SJF are $(1 + \epsilon)$-speed $O\left(\frac{1}{\epsilon^2}\right)$-competitive and that SETF is $(1 + \epsilon)$-speed $O\left(\frac{1}{\epsilon^2}\right)$-competitive. The results have been extended to all symmetric norms of flow time [16] and identical machines [10, 15]. Moreover, Im et al. showed that RR is $O(1)$-speed $O(1)$-competitive for the $\ell_2$ norm of flow time [19]. In [1, 21], more general objective functions were considered. Specifically, for a job $J_i$ with flow time $f_i$, a cost $g_i(f_i)$ is incurred. The only restriction on $g_i$ is that $g_i$ must be non-decreasing. For this general cost minimization problem, there are $O(1)$-speed $O(1)$-competitive online algorithms [1, 21].

For the offline setting, there are $O(1)$-approximation algorithms for minimizing the $\ell_p$ norm of flow time without speed augmentation [7, 14, 20]. All known non-trivial offline algorithms for this problem are based on linear programming rounding [5, 7, 14, 20] and thus are not suitable for online scheduling.

Quite a few studies have attempted to alleviate the starvation caused by SRPT [2, 11, 13, 22, 23, 24, 25, 27]. However, these works did not provide theoretical guarantees for minimizing the $\ell_2$ norm of flow time. Among these algorithms, the one in [24] is most similar to ours. In [24], every job is associated with a priority that increases as the waiting time increases or as the remaining processing time decreases. The algorithm always executes the job that has the highest priority. We can also modify our algorithm so that it always executes the job $J_i$ that has the highest $t - r_i$ remaining processing time. Thus, the modified algorithm does not need the starvation threshold $\theta$. However, compared with SRPT, the modified algorithm may execute large jobs prematurely, leading to a poor average flow time and a poor $\ell_2$ norm of flow time (see Appendix G).

2 Definitions and the Algorithm
In the problem of minimizing the $\ell_2$ norm of flow time, there are $n$ jobs, $J_1, J_2, \ldots, J_n$, and one machine. Each job $J_i$ has a processing time $p_i$ and a release time $r_i$. As in [5, 20], we assume that $p_i$ and $r_i$ are integers. We allow job preemption and consider clairvoyant scheduling. Define $F(S) = \sum_{i=1}^{n} f_i(S)^2$. The goal is to compute online a schedule $S$ that minimizes the $\ell_2$ norm of flow time, i.e., $\sqrt{F(S)}$.

For every $t \in \mathbb{N}^3$, the time slot $[t]$ is defined as the time interval between time $t$ and time $t + 1$. Thus, we can divide time into time slots $[0], [1], [2], \ldots$. We can view each job $J_i$ as a chain of tasks $J_{i,1}, J_{i,2}, \ldots, J_{i,p_i}$, where each task has a unit processing time. Because all the processing times and release times are integers, by a simple exchange argument, we can assume that under an optimal schedule, the machine never executes more than one task in time slot $[t]$ for any $t \in \mathbb{N}$ (i.e., the machine is either idle or executing the same task throughout the entire time slot $[t]$). Thus, for every time slot $[t]$, a schedule assigns a (possibly empty) task to be executed in $[t]$. If a task $J_{i,k}$ is executed in time slot $[t]$ under schedule $S$, then its completion time, denoted by $c_{i,k}(S)$, is $t + 1$. Throughout this paper, we use $BAL(\theta), SRPT, FCFS, RR, \text{ and } OPT$ to denote the schedule obtained by $BAL(\theta), SRPT, FCFS, RR, \text{ and } OPT$ to denote the schedule obtained by $BAL(\theta), SRPT, FCFS, RR$, and an optimal schedule, respectively. If $\theta$ is clear from the context, we simply write $BAL$ instead of $BAL(\theta)$.

A job is said to be active at time $t$ under schedule $S$ if it is released by time $t$ but has not yet been completed by time $t$ under $S$. We use $A(t, S)$ to denote the index set of the active jobs under $S$ at time $t$. In this paper, for any $a, b \in \mathbb{N}$ with $a \leq b$, $[a, b]$ is defined as $\{i | i \in \mathbb{N}, a \leq i \leq b\}$. If $a > b$, then $[a, b] = \emptyset$. Moreover, we define a map as a non-negative function with a finite domain. In this paper, we use the terms “remaining processing time” and “number of remaining tasks” interchangeably. The next definition introduces the most important map in this paper.

**Definition 2.1.** For any schedule $S$, any time $t$, and any $i \in A(t, S)$, define $q_{i,S}(i)$ as the number of remaining tasks of $J_i$ at time $t$ under $S$: if $i \in [1, n] \setminus A(t, S)$, define $q_{i,S}(i) = 0$.

**Example.** Consider the schedule obtained by SRPT for the instance shown in Fig. 3. Assume that SRPT first executes $J_1$. Then, at time 3, the number of remaining tasks of $J_1$ and $J_2$ are 4 and 6, respectively. Thus, $q_{1,SRPT}(1) = 4$ and $q_{3,SRPT}(2) = 6$. Because jobs $J_3, J_4, J_5$ are not active at time 3, we have $q_{3,SRPT}(i) = 0$ for all $3 \leq i \leq 5$. 

---

In this paper, we assume $\mathbb{N}$ contains 0.
2.1 The Algorithm. Next, we formalize the algorithm \( BAL(\theta) \) introduced in Section 1. For brevity, define \( q_i(t) \) as \( q_{i, BAL}(t) \). In other words, \( q_i(t) \) is the number of remaining tasks of \( J_i \) at time \( t \) under \( BAL(\theta) \). \( BAL(\theta) \) categorizes active jobs into two types, starving and normal, according to the given starvation threshold \( \theta \). Every job is normal initially. An active job \( J_i \) is said to be starving at time \( t \) if

\[
(2.1) \quad \frac{t - r_i}{q_i(t)} \geq \theta.
\]

Moreover, once \( J_i \) becomes starving, every remaining task of \( J_i \) is said to be starving as well. For any starving job \( J_i \), define \( \gamma_i \) as the time at which \( J_i \) becomes starving (i.e., \( t_i \) is the smallest \( t \in \mathbb{N} \) that satisfies Eq. (2.1)), and define \( \gamma_i = \frac{t - r_i}{q_i(t)} \). In Algorithm 1, \( ST \) denotes the index set of the starving jobs.

When there are starving jobs, \( BAL(\theta) \) executes the starving job that has the smallest \( t_i \). In other words, \( BAL(\theta) \) executes the job that becomes starving first. Ties are broken by choosing the job that has the largest \( \gamma_i \). If there are still ties, break them by choosing the smallest job index. When there is no starving job, \( BAL(\theta) \) follows SRPT. Specifically, let \( M_{SRPT}(t) \) and \( M_{BAL}(t) \) be the sets of tasks that are executed in time slots \([0], [1], [2], \ldots, [t]\) under SRPT and \( BAL(\theta) \), respectively. Define \( M_{BAL}(-1) = \emptyset \). At time \( t \), if there is no starving job, then among the tasks in \( M_{SRPT}(t) \setminus M_{BAL}(t - 1) \), \( BAL(\theta) \) executes the one that is completed the earliest under SRPT. Observe that if tasks \( a \) and \( b \) are executed as normal tasks, and \( a \) is completed before \( b \) under \( BAL(\theta) \), then \( a \) is also completed before \( b \) under \( SRPT \). \( BAL(\theta) \) can be viewed as an interpolation between SRPT and FCFS. In Appendix D, we show that for any \( \theta > 0 \), the competitive ratio of \( BAL(\theta) \) for the problem of minimizing the \( \ell_2 \) norm of flow time cannot be worse than that of SRPT and FCFS.

3 Searching for the Best \( \theta \)

To find the best starvation threshold \( \theta \), we partition the lifetime of \( J_i \) into two periods, normal and starving. Specifically, if a job never becomes starving under \( BAL(\theta) \), it is termed a Finished-as-Normal (FaN) job; otherwise, it is termed a Finished-as-Starving (FaS) job. We further extend the definition of \( t_i \) to FaN jobs as follows: If \( J_i \) is an FaN job, \( t_i \) is defined as its completion time (i.e., \( t_i = c_i(BAL) \)). For any job \( J_i \), we then define

\[ norm_i(\theta) = t_i - r_i \]

and

\[ starv_i(\theta) = c_i(BAL) - t_i. \]

Thus,

\[ f_i(BAL) = norm_i(\theta) + starv_i(\theta) \]

and

\[ F(BAL) = \Theta \left( \sum_{i=1}^{n} norm_i(\theta)^2 + \sum_{i=1}^{n} starv_i(\theta)^2 \right). \]

To minimize the \( \ell_2 \) norm of flow time, we balance \( \sum_{i=1}^{n} norm_i(\theta)^2 \) and \( \sum_{i=1}^{n} starv_i(\theta)^2 \). Intuitively, \( norm_i(\theta) \) decreases as the starvation threshold \( \theta \) decreases. In contrast, \( starv_i(\theta) \) decreases as the starvation threshold \( \theta \) increases. The following two theorems, whose proofs are postponed to Sections 5 and 6, relate \( \sum_{i=1}^{n} norm_i(\theta)^2 \) and \( starv_i(\theta) \) to \( F(OPT) \).

---

\(^4\)All the variables associated with \( BAL(\theta) \) depend on \( \theta \) (e.g., \( t_i, \gamma_i, ST \), and \( M_{BAL}(t) \)). For simplicity, we do not explicitly show their dependency on \( \theta \) in our notation.
Theorem 3.1. For any starvation threshold \( \theta \geq 0 \),
\[
\sum_{i=1}^{n} \|n_i\|^2 = O(\theta)F(OPT).
\]

Theorem 3.2. For any starvation threshold \( \theta > 0 \),
\[
\sum_{i=1}^{n} \operatorname{starv}_i(\theta)^2 = O\left(\frac{n}{\sqrt{\theta}}\right)F(OPT).
\]

Solving \( \theta = \frac{n}{\sqrt{\theta}} \) yields \( \theta = n^{\frac{2}{3}} \). Thus, Theorems 3.1 and 3.2 suggest that, to minimize the \( L \) norm of flow time, the best starvation threshold is \( \theta = n^{\frac{2}{3}} \).

In practice, we can only estimate \( n \). Given an estimate \( \bar{n} \) of \( n \) with bounded error, the following theorem gives the competitive ratio of \( BAL(\bar{n}^{\frac{2}{3}}) \). The proof is a simple combination of the above two theorems, and can be found in Appendix D.

Theorem 3.3. Let \( \bar{n} \) be an estimate of \( n \) such that \( \beta \bar{n} \leq \bar{n} \leq \alpha n \) for some \( \frac{1}{2} \leq \beta \leq 1 \) and \( \alpha \geq 1 \). Then the competitive ratio of \( BAL(\bar{n}^{\frac{2}{3}}) \) for minimizing the \( L \) norm of flow time is \( O\left(n^{\frac{2}{3}} \left(\beta^{\frac{2}{3}} + \alpha^{\frac{2}{3}}\right)\right)\).

Theorem 3.3 along with the discussion in Appendix C show that, given a good estimate \( \bar{n} \), \( BAL(\bar{n}^{\frac{2}{3}}) \) achieves a much better competitive ratio than that of \( SRPT \) and \( FCFS \). Next, we show that \( BAL(\bar{n}^{\frac{2}{3}}) \) preserves the advantages of \( SRPT \) and \( FCFS \).

4 Experimental Study

In this section, we plot \( \sqrt{\frac{F(BAL(\theta))}{F(S)}} \), where \( S \in \{SRPT, FCFS, RR\} \). We consider \( RR \) in our experimental study because \( RR \) is \( O(1) \)-speed \( O(1) \)-competitive for the \( L \) norm of flow time [19]. Additional experimental results can be found in Appendix A, where we consider a variant of \( BAL(\theta) \) that does not need any estimate. All results are averaged over ten independent runs. We assume that the job arrival process is a Poisson process with arrival rate \( \lambda \), and we schedule jobs that arrive between time 0 and time \( T = 2^{20} \). Define \( \rho \) as the product of the average job size and the job arrival rate. Thus, \( \rho \) can be viewed as the load of the system.

We assume that the job size follows a bounded Pareto distribution parameterized by the shape parameter (i.e., the exponent of the power law), \( L \) (i.e., the lowest possible job size), and \( H \) (i.e., the highest possible job size). It has been shown that many job size distributions in different computing systems are well-modeled by a bounded Pareto distribution with a shape parameter close to one [4]. Throughout this experimental study, we set the shape parameter to 1.1. We always set \( L \) and \( H \) so that the average job size is 30.\(^5\) As a result, the system load \( \rho = 30\lambda \).

Specifically, in this experimental study, \( (L, H) \in \{(16.772, 2^{6}), (7.918, 2^{3}), (5.649, 2^{12}), (4.639, 2^{15}), (4.073, 2^{18})\} \). When \( H = 2^{15} \), the largest 2.2\% of the jobs comprise 50\% of the load; when \( H = 2^{18} \), the largest 1.1\% of the jobs comprise 50\% of the load. The heavy-tailed property of these distributions makes \( SRPT \) an adequate scheduling algorithm [3, 6, 9, 12, 17, 26]. On the other hand, when \( H = 2^{6} \), the ratio of the largest job size to the smallest is small \((P < 4)\). Thus, as discussed in Section 1, \( FCFS \) is well suited for this type of job size distribution.

4.1 Instance Partition and Settings of \( \theta \). A natural estimate of \( n \) for our problem instance is \( T\lambda \). However, it is not appropriate to set the starvation threshold based on this estimate when the system load is low. When the system load is low, the system may become idle (i.e., there are no jobs to execute) frequently. When this happens, the schedule in the future becomes independent of the schedule in the past. We can then treat the scheduling of jobs in the future as a new problem instance, and partition the original instance into idle-free instances. An idle-free instance ends once the system becomes idle, and the next idle-free instance starts when a new job is released.

Once the problem instance is partitioned into idle-free instances, we can apply scheduling algorithms to each of them separately. This allows us to determine the starvation threshold for idle-free instances rather than the original instance. At first glance, it may seem reasonable to set \( \bar{n} \) as the average number of jobs in an idle-free instance. However, this approach is not ideal because when the system load is high, the \( L \) norm of flow time is mostly influenced by the largest idle-free instances rather than the ones with only a few jobs. Therefore, in high-load scenarios, we ignore the idle-free instances that are too small. Based on these observations, we set \( \bar{n} \) as follows. Let \( I \) be the original instance and \( I_{>10} \) be the set of idle-free instances in \( I \) that contain more than ten jobs. We then calculate \( \bar{n} \) as the average number of jobs in \( I_{>10} \). Specifically,

\[
\bar{n} = \frac{\text{total number of jobs in the instances in } I_{>10}}{|I_{>10}|}.
\]

In practice, this estimate can be obtained from historical data. Following the above estimate, we then consider \( \theta = \bar{n}^2 \), where \( x \in \{\frac{1}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}, \frac{6}{6}\} \).

---

\(^5\)Our experiment involves rounding interarrival times and job sizes to generate integral inputs. Specifically, if a number falls between 0 and 1 (exclusive), it is rounded up to 1. If a number is greater than 1, it is rounded down to the nearest integer.
4.2 Comparison to SRPT: $H \in \{2^{12}, 2^{15}, 2^{18}\}$. When $H$ is large, Fig. 4 shows that $BAL(\tilde{n}^2)$ and SRPT have almost identical performance. In other words, setting the starvation threshold $\theta$ to $\tilde{n}^2$ preserves the advantage of SRPT for large $H$. In contrast, Fig. 5 shows that when $H$ is large, SRPT outperforms $BAL(\tilde{n}^2)$, which in turn outperforms $BAL(\tilde{n})$. This is because when the starvation threshold $\theta$ is too low (e.g., $\theta \leq \tilde{n}^2$), some large jobs become starving prematurely. On the other hand, when $\theta \geq \tilde{n}^2$, starvation is mitigated adequately and $BAL(\theta)$ maintains the superiority of SRPT for heavy-tailed job size distributions.

4.3 Comparison to SRPT: $H \in \{2^6, 2^9\}$. When $H$ or $P$ is small, $BAL(\tilde{n}^2)$ outperforms SRPT. The improvement of $BAL(\tilde{n}^2)$ over SRPT reaches its peak when the system is slightly overloaded. Specifically, when $H = 2^6$ and $\lambda^{-1} = 28$, setting the starvation threshold $\theta$ to $\tilde{n}^2$ reduces SRPT’s $\ell_2$ norm of flow time by more than 70%. This situation is captured by Fig. 1, in which starvation mitigation greatly reduces SRPT’s $\ell_2$ norm of flow time. The drawback of SRPT during a minor overload scenario was also discussed in [6], and it was addressed by speed augmentation. Our experiments demonstrate that careful mitigation of starvation can effectively overcome this drawback.

A system normally cannot be overloading for too long (e.g., due to system upgrade) [26]. Thus, the system load is usually less than 100%. Even when the system load is less than 100%, Fig. 4 shows that $BAL(\tilde{n}^2)$ outperforms SRPT for small $H$. It is worth noting that when the job arrival process is Poisson, the system may still experience transient overloads even if $\rho$ is moderately low (i.e., $75% \leq \rho \leq 100\%$). During these transient overloads, SRPT unnecessarily delays jobs that are only slightly larger than the others, which deteriorates the $\ell_2$ norm of flow time. In contrast, $BAL(\tilde{n}^2)$ avoids the unnecessary delays by starvation mitigation and thus achieves a better performance.

4.4 Comparison to FCFS: $H \in \{2^9, 2^{12}, 2^{15}, 2^{18}\}$. Next, we compare $BAL(\tilde{n}^2)$ with FCFS. For large or medium $H$, Fig. 4 shows that $BAL(\tilde{n}^2)$ outperforms FCFS. Specifically, when $H = 2^{18}$, setting the starvation threshold $\theta$ to $\tilde{n}^2$ reduces FCFS’s $\ell_2$ norm of flow time by more than 90%. This is because when FCFS is executing large jobs, many small jobs are starving. In contrast, $BAL(\tilde{n}^2)$ gives priority to small jobs and thus has a much better performance.

When the system load is low (e.g., $\rho = 75\%$), $BAL(\tilde{n}^2)$ still outperforms FCFS. When $\rho = 75\%$, the mean job interarrival time is 40 but the size of a large job is much greater than 40. Thus, under FCFS, many small jobs are starving during the execution of a large job, even under low load.

It is important to highlight that for $H = 2^9$, $BAL(\tilde{n}^2)$ reduces the $\ell_2$ norm of flow time of both SRPT and FCFS by more than 30% during a minor overload scenario (when $27 \leq \lambda^{-1} \leq 29$). When $H = 2^9$, the largest 10% of the jobs comprise less than 50% of the load and $P > 60$. This shows that when the job size distribution is not heavy-tailed and not all jobs have similar sizes, $BAL(\tilde{n}^2)$ outperforms both SRPT and FCFS simultaneously.
4.5 Comparison to FCFS: $H \in \{2^6\}$. When $H$ or $P$ is small, Fig. 4 shows that the performance of $BAL(\tilde{n}^{\hat{x}})$ is almost identical to that of FCFS. This shows that $BAL(\tilde{n}^{\hat{x}})$ effectively mitigates the starvation caused by SRPT. In contrast, Fig. 5 shows that when $H$ or $P$ is small, FCFS is better than $BAL(\tilde{n}^{\hat{x}})$, which is in turn better than $BAL(\tilde{n}^{\hat{x}})$. This is because when the starvation threshold $\theta$ is too high (e.g., $\theta \geq \tilde{n}^{\hat{x}}$), some jobs wait too long before being viewed as starving jobs, even if they are only slightly larger than the others. On the other hand, when $\theta \leq \tilde{n}^{\hat{x}}$, starvation is mitigated adequately and $BAL(\theta)$ preserves the advantage of FCFS.

4.6 Comparison to RR. Fig. 6 shows that $BAL(\tilde{n}^{\hat{x}})$ outperforms RR regardless of the load and the job size distribution. We first focus on the case where $H$ is large. When $H$ is large, RR allocates processing time to large jobs even when many small jobs are waiting to be processed. This unnecessarily increases the flow times of many small jobs and deteriorates the $\ell_2$ norm of flow time.

When $H$ or $P$ is small, the frequent switches among...
jobs under RR deteriorates the performance. As a result, \( BAL(\tilde{n}^2) \) outperforms RR when \( H \) is small. The drawback caused by frequently switching job executions is more severe when the system is overloading. Specifically, when the system is overloading and \( H = 2^6 \), \( BAL(\tilde{n}^2) \) achieves a 40% reduction in the \( \ell_2 \) norm of flow time. Finally, recall that the performance of \( BAL(\tilde{n}^2) \) is almost identical to SRPT and FCFS when \( H = 2^{18} \) and \( H = 2^6 \), respectively. Thus, Fig. 6 shows that RR is worse than SRPT and FCFS under various settings.

5 Proof of Theorem 3.1

To minimize the \( \ell_2 \) norm of flow time (and many other reasonable objective functions), we should not waste any time slot. Specifically, at every time \( t \in \mathbb{N} \), if some active jobs are not completed, then one of them must be executed in time slot \([t, t+1)\). A schedule that satisfies the above property is called a work-conserving schedule. Clearly, for any time \( t \), all work-conserving schedules have the same total number of remaining tasks over all active jobs. Specifically, for any two work-conserving schedules \( S_1 \) and \( S_2 \) and any time \( t \), we have \( \sum_{t=1}^{\infty} q_t s_t(i) = \sum_{t=1}^{\infty} q_t s_t(i) \). Thus, we define \( g(t) \) as the total number of remaining tasks overall active jobs at time \( t \) under any work-conserving schedule. For example, consider the instance shown in Fig. 3 at time \( t = 3 \). Then we have \( g(3) = 10 \). The following lemma gives the relationship between \( g(t) \) and \( F(\mathcal{OPT}) \).

**Lemma 5.1.** For any work-conserving schedule \( S \), \( F(\mathcal{OPT}) \geq \sum_{t \in \mathbb{N}} q_t(t) = \sum_{t=1}^{\infty} c_i(S) q_t s_t(i) \).

**Proof.** For any work-conserving schedule \( S \), we have

\[
F(\mathcal{OPT}) = \sum_{i=1}^{n} c_i(\mathcal{OPT})^{-1} f_i(\mathcal{OPT}) \\
\geq \sum_{i=1}^{n} c_i(\mathcal{OPT})^{-1} \sum_{t=r_i}^{\infty} p_t \geq \sum_{i=1}^{n} c_i(\mathcal{OPT})^{-1} \sum_{t=r_i}^{\infty} q_t \mathcal{OPT}(i) \\
= \sum_{i \in N} q_t(i) = \sum_{i=1}^{n} c_i(S) q_t s_t(i).
\]

\( \square \)

To prove Theorem 3.1, we use Lemma 5.1 to upper bound \( \sum_{i=1}^{n} \text{norm}_i(\theta)^2 \). Let \( I = \{ i | \text{norm}_i(\theta) = 1, i \in [1, n] \} \). Obviously, \( \sum_{i \in I} \text{norm}_i(\theta)^2 \leq F(\mathcal{OPT}) \). Thus, we only need to consider job indices that are not in \( I \). Let \( i \in [1, n] \setminus I \). Define \( T_i = [r_i, t_i - 1] \). Because \( \text{norm}_i(\theta) = t_i - r_i = |T_i| \), we have

\[
\text{norm}_i(\theta)^2 = |T_i|^2 = \Theta \left( \sum_{t \in T_i} (t - r_i) \right).
\]

When \( t \in T_i \), \( \frac{t - r_i}{q_i(t)} < \theta \), and thus \( t - r_i < \theta q_i(t) \). As a result, \( \text{norm}_i(\theta)^2 = O(\theta) \sum_{t \in T_i} q_t(i) \). We then have

\[
\sum_{i \in [1, n] \setminus I} \text{norm}_i(\theta)^2 = O(\theta) \sum_{i \in [1, n] \setminus I} \sum_{t \in T_i} q_t(i)
\]

by Lemma 5.1

\( \Theta \left( (\theta) F(\mathcal{OPT}) \right) \).

6 Proof of Theorem 3.2

Let \( u(t) \) be the number of starving tasks at time \( t \). Thus, by the design of \( BAL(\theta) \), \( c_i(BAL) \leq t_i + u(t_i) \). Therefore, \( \text{starv}_i(\theta) = c_i(BAL) - t_i \leq u(t_i) \). Define \( \theta^* = \max_{t \in \mathbb{N}} u(t) \). We then have \( \text{starv}_i(\theta) \leq u(\theta^*) \) and thus \( \sum_{i=1}^{n} \text{starv}_i(\theta)^2 = O(n \cdot u(\theta^*)) \). As a result, to prove Theorem 3.2, it is sufficient to show

\( (*) \quad F(\mathcal{OPT}) = \Omega \left( \sqrt{\theta^*} \cdot u(\theta^*)^2 \right) \).

6.1 The First Lower Bound of \( F(\mathcal{OPT}) \). By Lemma 5.1, we have

\( (\ast) \quad F(\mathcal{OPT}) \geq \sum_{i=1}^{n} t_i q_i(i) \geq \sum_{i=1}^{n} t_i q_i(i) \geq \theta \sum_{i=1}^{n} q_i(i)^2, \)

where the last inequality holds because by the design of \( BAL(\theta) \), \( t_i - r_i \geq \theta q_t(i) \) for any FaS job \( J_i \). For any FaN job \( J_i \), we have \( q_t(i) = 0 \).

6.2 The Second Lower Bound of \( F(\mathcal{OPT}) \). Observe that to minimize \( F(S) \) when all jobs are released at the same time, jobs should be executed in increasing order of their processing times. To derive the second lower bound of \( F(\mathcal{OPT}) \) by this idea, jobs that are not active at time \( t^* \) under \( \mathcal{OPT} \) are ignored (and thus their flow times are not counted in \( F(\mathcal{OPT}) \)). For every job \( J_i \) that is active at time \( t^* \) under \( \mathcal{OPT} \), we gift \( F(\mathcal{OPT}) \) by setting \( r_i = t^* \) and \( p_t = q_t \cdot \mathcal{OPT}(i) \). We then execute jobs in increasing order of their new processing times. The above idea will be used in Lemma 6.2.

6.3 Proof of Eq. (\ast). For any two maps \( f \) and \( f' \), we write \( f \) dominating \( f' \), or \( f' \) is dominated by \( f \), if \( \text{dom} f' = \text{dom} f \) and \( f'(i) \leq f(i), \forall i \in \text{dom} f \). We prove the following lemma in Section 6.4.

**Lemma 6.1.** There is a map \( m_{OPT} \) dominated by \( q_{\cdot \cdot OPT} \) such that

- **P1:** \( \sum_{i=1}^{n} m_{OPT}(i)^2 \leq \sum_{i=1}^{n} q_i(i)^2 \), and
- **P2:** \( \sum_{i=1}^{n} m_{OPT}(i) \geq \frac{1}{\theta^*} \cdot u(\theta^*) \).

\( \ast \)For any map \( f \), \( \text{dom} f \) denotes the domain of \( f \).
By Eq. (6.2) and P1, we have

\[ F(\text{OPT}) \geq \sum_{i=1}^{n} m_{\text{OPT}}(i)^2. \]

Because \( m_{\text{OPT}} \) is dominated by \( q_{*\cdot \text{OPT}} \), we can use the technique developed for the second lower bound to prove the following lemma, whose proof is given in Appendix B.1.

**Lemma 6.2.** Let \( m_{\text{OPT}} \) be the map defined in Lemma 6.1. Reindex jobs so that

\[ m_{\text{OPT}}(1) \leq m_{\text{OPT}}(2) \leq \cdots \leq m_{\text{OPT}}(n). \]

We then have

\[ F(\text{OPT}) \geq \sum_{i=1}^{n} \left( \sum_{h=1}^{i} m_{\text{OPT}}(h) \right)^2. \]

We are now ready to prove Eq. (\#). Like Lemma 6.2, we reindex jobs so that Eq. (6.4) holds. We then have

\[
\begin{aligned}
F(\text{OPT}) &= \Omega \left( \sum_{i=1}^{n} \left( \sum_{h=1}^{i} m_{\text{OPT}}(h) \right)^2 + \sum_{i=1}^{n} m_{\text{OPT}}(i)^2 \right) \\
&= \Omega \left( \sqrt{\theta} \left( \sum_{i=1}^{n} \left( \sum_{h=1}^{i} m_{\text{OPT}}(h) \right)^2 \right) \right) \left( \sum_{i=1}^{n} m_{\text{OPT}}(i)^2 \right) \\
&= \Omega \left( \sqrt{\theta} \sum_{i=1}^{n} \left( \sum_{h=1}^{i} m_{\text{OPT}}(h) \right) m_{\text{OPT}}(i) \right) \\
&= \Omega \left( \sqrt{\theta} \left( \sum_{i=1}^{n} m_{\text{OPT}}(i) \right)^2 \right) = \Omega \left( \sqrt{\theta} \right) u(t^*)^2,
\end{aligned}
\]

where the first equality follows from Lemma 6.2 and Eq. (6.3), the second equality follows from the AM–GM inequality, the third equality follows from the Cauchy-Schwarz inequality, and the last equality follows from P2 in Lemma 6.1.

### 6.4 Proof of Lemma 6.1

Note that Lemma 6.1 is trivial when \( u(t^*) = 0 \). Thus, in the following proof, we assume \( u(t^*) \geq 1 \). We first introduce the following shorthand notations.

**Definition 6.1.** For any map \( f \), define \( S(f) = \sum_{i \in \text{dom } f} f(i) \), and define \( f^2 \) as a map such that \( \text{dom } f^2 = \text{dom } f \) and \( f^2(x) = f(x)^2 \) for any \( x \in \text{dom } f^2 \).

Informally, to prove Lemma 6.1, we have to construct a map \( m_{\text{OPT}} \) dominated by \( q_{*\cdot \text{OPT}} \) such that \( S(m_{\text{OPT}}) \) is sufficiently small but \( S(m_{\text{OPT}}^2) \) is sufficiently large. Thus, for any \( c > 0 \), we say that a map \( f \) is a \( c \)-proper map if \( f \) satisfies the following two constraints:

1. \( S(f^2) \leq \sum_{i=1}^{n} q_t(i)^2 \),
2. \( S(f) \geq c \cdot u(t^*) \).

**Proof Overview of Lemma 6.1.** To prove Lemma 6.1, it is sufficient to construct a \( \frac{1}{16} \)-proper map dominated by \( q_{*\cdot \text{OPT}} \). We reduce this task to the task of finding a \( \frac{1}{4} \)-proper map dominated by \( q_{*\cdot \text{SRPT}} \). To this end, we first analyze the relationship between \( q_{*\cdot \text{OPT}} \) and \( q_{*\cdot \text{SRPT}} \) (Lemma 6.3). We then prove that the aforementioned reduction is feasible (Lemma 6.4). Finally, we construct a \( \frac{1}{8} \)-proper map dominated by \( q_{*\cdot \text{SRPT}} \) (Lemma 6.5).

#### 6.4.1 The Relationship Between \( q_{*\cdot \text{OPT}} \) and \( q_{*\cdot \text{SRPT}} \)

Recall that SRPT always executes the active job \( J_i \) that has the smallest \( q_{t, \text{SRPT}}(i) \). In other words, SRPT avoids executing the active job that has the largest number of remaining tasks. It is not difficult to show that, for any positive integer \( k \), among all work-conserving schedules, SRPT maximizes the sum of the top-\( k \) largest numbers of remaining tasks at any time. Lemma 6.3 formalize the above statement using majorization. Roughly speaking, if map \( f \) majorizes map \( g \), then for any positive integer \( k \), the sum of the top-\( k \) largest outputs of \( f \) is at least that of \( g \).

To define majorization, we first introduce the following definition, which sorts the domain of a map in decreasing order of their outputs.

**Definition 6.2.** Let \( f \) be any map. Define \( \pi_f \) as a function that maps any \( k \in [1, |\text{dom } f|] \) to the element in \( \text{dom } f \) that has the \( k \)th largest output of \( f \) (ties can be broken arbitrarily). Thus, \( \text{dom } f = \{ \pi_f(1), \pi_f(2), \ldots, \pi_f(|\text{dom } f|) \} \) and \( f(\pi_f(1)) \geq f(\pi_f(2)) \geq \cdots \geq f(\pi_f(|\text{dom } f|)) \).

Next, we define the sum of the top-\( k \) outputs of a map \( f \), denoted by \( S_k(f) \).

**Definition 6.3.** Let \( f \) be any map. Define

\[
S_k(f) = \begin{cases} 
 0 & \text{if } k = 0 \\
 1 \sum_{j=1}^{k} f(\pi_f(j)) & \text{if } k \in [1, |\text{dom } f|] \\
 S(f) & \text{if } k > |\text{dom } f| 
\end{cases}
\]

**Definition 6.4.** For any two maps \( f \) and \( g \), \( f \) majorizes \( g \) if the following two conditions are met:

1. \( S(f) = S(g) \),
2. \( S_k(f) \geq S_k(g), \forall k \in \mathbb{N} \).

The next lemma formalizes the previous discussion on SRPT. The proof is based on the simple property that SRPT avoids executing the active jobs that have the most remaining tasks and can be found in Appendix B.2.
Specifically, for any $t$ remaining tasks of an FaS job once it becomes starving.

Let $f$ and $g$ be such that $f$ majorizes $g$. For any $c > 0$, if there is a $c$-proper map dominated by $f$, then there is a $\frac{1}{c}$-proper map dominated by $g$.

Proof. We only give a proof sketch here. The complete proof can be found in Appendix B.3. To prove Lemma 6.4, it suffices to prove that for any map $f'$ dominated by $f$, there is a map $g'$ dominated by $g$ such that $S(g'^2) \leq S(f'^2)$ and $S(g') \geq \frac{1}{2}S(f')$. Thus, if $f'$ is a $c$-proper map, then $g'$ is a $\frac{1}{c}$-proper map.

The construction of $g'$ is purely combinatorial. Initially, $g' = g$ and we find a set $I \subseteq \text{dom } f'$ such that $\sum_{i \in I} f'(i) \geq \frac{1}{2} S(f')$. We then construct a family $\{B_i\} \subseteq I$ of mutually disjoint subsets of $\text{dom } g'$, such that for each $i \in I$,

$$\frac{f'(i)}{2} \leq \sum_{b \in B_i} g'(b) \leq f'(i).$$

To this end, we may decrease $g'$. For any $b \notin \bigcup_{i \in I} B_i$, we set $g'(b) = 0$. $g'$ is then the desired map.

A $\frac{1}{4}$-Proper Map Dominated by $q_{16, \text{SRPT}}$.

Lemma 6.5. There is a $\frac{1}{4}$-proper map dominated by $q_{16, \text{SRPT}}$.

Proof. We only give a proof sketch here. The full proof, which also uses Lemma 6.4, can be found in Appendix B.4. At a high level, we show that there is a map $h$ that majorizes $q_{16, \text{SRPT}}$, and then construct a $1$-proper map $h'$ dominated by $h$. The proof then follows from Lemma 6.4.

We first analyze the relationship between $q_{16, \text{SRPT}}$ and $q_t$. Consider a map $\hat{q}_t$ that freezes the number of remaining tasks of an FaS job once it becomes starving. Specifically, for any $t \in \mathbb{N}$ and $i \in [1, n]$, define

$$\hat{q}_t(i) = \begin{cases} q_t(i) & \text{if } (J_i \text{ is an FaN job}) \\ q_t(i) & \text{if } (J_i \text{ is an FaS job and } t < t_i) \\ q_t(i) & \text{if } J_i \text{ is an FaS job and } t > t_i. \end{cases}$$

We stress that the definition of $\hat{q}_t(i)$ can be applied to $t > c_i(BA\text{C})$. Because BAL$(\theta)$ follows SRPT when no jobs are starving, it is not difficult to show that for any $t \in \mathbb{N}$, $\hat{q}_t$ dominates $q_{16, \text{SRPT}}$ (Lemma B.5).

Because $S(\hat{q}_t)$ may be greater than $q(t^*) = S(q_{16, \text{SRPT}})$, $\hat{q}_t$ may not majorize $q_{16, \text{SRPT}}$. Thus, we construct a map $h$ by decreasing the smallest non-zero outputs of $\hat{q}_t$ so that $S(h) = q(t^*)$. Because $\hat{q}_t$ dominates $q_{16, \text{SRPT}}$, and only the smallest non-zero outputs of $\hat{q}_t$ are decreased, it can be shown that $h$ majorizes $q_{16, \text{SRPT}}$ (Lemma B.6).

To construct a $1$-proper map $h'$ dominated by $h$, consider the set $I_S = \{i | J_i \text{ is an FaS job and } t_i \leq t^*\}$. We then set

$$h'(i) = \begin{cases} h(i) & \text{if } i \in I_S \\ 0 & \text{if } i \in [1, n] \setminus I_S. \end{cases}$$

Clearly, $h'$ is dominated by $h$. In addition,

$$S(h'^2) = \sum_{i \in I_S} h(i)^2 \leq \sum_{i \in I_S} \hat{q}_t(i)^2 = \sum_{i \in I_S} q_t(i)^2 \leq \sum_{i=1}^n q_t(i)^2.$$ 

Moreover, let $I_N = [1, n] \setminus I_S$. Thus,

$$S(h') = S(h) - \sum_{i \in I_N} h(i) \geq S(h) - \sum_{i \in I_N} \hat{q}_t(i) = q(t^*) - \sum_{i \in I_N} q_t(i) = u(t^*),$$

where the inequality follows because $h(i) \leq \hat{q}_t(i) = q_t(i)$ for any $i \in I_N$. Thus, $h'$ is a $1$-proper map dominated by $h$ (Lemma B.7).

We are now ready to prove Lemma 6.1. Consider Lemma 6.4 and fix $f = q_{16, \text{SRPT}}$ and $g = q_{16, \text{OPT}}$. By Lemma 6.3, $f$ majorizes $g$. By Lemma 6.5, there is a $\frac{1}{4}$-proper map dominated by $f$. Thus, by Lemma 6.4, there is a $\frac{1}{4}$-proper map dominated by $g = q_{16, \text{OPT}}$.

7 Concluding Remarks
To minimize the $\ell_2$ norm of flow time, prior work applied speed augmentation to standard online algorithms, including SRPT and RR. Despite the near-optimal performance under speed augmentation, our experimental results show that SRPT and RR balance job flow times inadequately under various practical scenarios. In particular, when all jobs have similar sizes, SRPT and RR are worse than FCFS, and when the job size distribution is heavy-tailed, RR is worse than SRPT.

In this paper, we show that balancing job flow times without speed augmentation is possible when $\theta = \frac{n}{2 \pi}$, resulting in a significantly better competitive ratio and outperforming RR, SRPT, and FCFS in various practical scenarios.
References

[1] Spyros Angelopoulos, Giorgio Lucarelli, and Thang Nguyen Kim. Primal–dual and dual-fitting analysis of online scheduling algorithms for generalized flow-time problems. *Algorithmica*, 81(9):3391–3421, 2019.

[2] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Pias: Practical information-agnostic flow scheduling for commodity data centers. *IEEE/ACM Transactions on Networking*, 25(4):1954–1967, 2017.

[3] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. *SIGMETRICS Perform. Eval. Rev.*, 29(1):279–290, June 2001.

[4] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2001.

[5] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014.

[6] Nikhil. Bansal and Kirk R. Pruhs. Server scheduling to balance priorities, fairness, and average quality of service. *SIAM Journal on Computing*, 39(7):3311–3335, 2010.

[7] Jatin Batra, Naveen Garg, and Amit Kumar. Constant factor approximation algorithm for weighted flow-time on a single machine in pseudopolynomial time. *SIAM Journal on Computing*, 0(0):FOCS18–158–FOCS18–188, 2020.

[8] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *ACM-SIAM SODA*, 1998.

[9] Ernst W Biersack, Bianca Schroeder, and Guillaume Urvoy-Keller. Scheduling in practice. *ACM SIGMETRICS Performance Evaluation Review*, 34(4):21–28, 2007.

[10] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with ε resource augmentation. In *ACM STOC*, 2004.

[11] Ludmila Cherkasova. Scheduling strategy to improve response time for web applications. In Peter Slott, Marian Bubak, and Bob Hertzberger, editors, *High-Performance Computing and Networking*, pages 305–314. Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[12] Mark Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in web servers. In *USITS*, 1999.

[13] Pamela Delgado, Diego Didona, Florin Dinu, and Willy Zwaenepoel. Job-aware scheduling in eagle: Divide and stick to your probes. In *ACM SoCC*, 2016.

[14] Uriel Feige, Janardhan Kulkarni, and Shi Li. A polynomial time constant approximation for minimizing total weighted flow-time. In *ACM-SIAM SODA*, 2019.

[15] Kyle Fox and Benjamin Moseley. Online scheduling on identical machines using SRPT. In *ACM-SIAM SODA*, 2011.

[16] Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. All-Norms and All-$L_p$-Norms Approximation Algorithms. In *FSTTCS*, 2008.

[17] Mor Harchol-Balter, Nikhil Bansal, Bianca Schroeder, and Mukesh Agrawal. SRPT scheduling for web servers. In *Job Scheduling Strategies for Parallel Processing*, pages 11–20, 2001.

[18] Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. Some simple inequalities satisfied by convex functions. *Messenger Mathematics*, 58:145–152, 1929.

[19] Sungjin Im, Janardhan Kulkarni, and Benjamin Moseley. Temporal fairness of round robin: Competitive analysis for $l_k$-norms of flow time. In *ACM SPAA*, 2015.

[20] Sungjin Im and Benjamin Moseley. Fair scheduling via iterative quasi-uniform sampling. *SIAM Journal on Computing*, 49(3):658–680, 2020.

[21] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Online scheduling with general cost functions. *SIAM Journal on Computing*, 43(1):126–143, 2014.

[22] Wenxin Li, Sheng Chen, Keqiu Li, Heng Qi, Renhai Xu, and Song Zhang. Efficient online scheduling for coflow-aware machine learning clusters. *IEEE Transactions on Cloud Computing*, 2020.

[23] Rahul Mangharam, Mustafa Demirhan, Ragunathan Rajkumar, and Dipankar Raychaudhuri. Size matters: size-based scheduling for MPEG-4 over wireless channels. In *SPIE Multimedia Computing and Networking*, 2003.

[24] E. Modiano. Scheduling algorithms for message transmission over a satellite broadcast system. In *IEEE MILCOM*, 1997.

[25] Mario Pastorelli, Damiano Carra, Matteo Dell’Amico, and Pietro Michiardi. Hsp: Bringing size-based scheduling to hadoop. *IEEE Transactions on Cloud Computing*, 5(1):43–56, 2017.

[26] Bianca Schroeder and Mor Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Trans. Internet Technol.*, 6(1):20–52, February 2006.

[27] Zhiyuan Wang, Jiancheng Ye, Dong Lin, Yipei Chen, and John C.S. Lui. Designing approximate and deployable SRPT scheduler: A unified framework. In *IEEE/ACM IWQOS*, 2021.

[28] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *SFC*, 1977.
Appendix A  Additional Experimental Study:

**BAL(θ) Without Estimate**

We design a variant of \( BAL(\tilde{n}^{\frac{1}{2}}) \) that does not need any estimate of \( n \). In this variant, we adjust \( \theta \) dynamically and keep track of the number of jobs released by time \( t \) in the current idle-free instance using a counter \( n(t) \). We then set the starvation threshold at time \( t \) as \( \theta = n(t)^{\frac{1}{2}} \). Once a job \( J_i \) becomes starving at time \( t \) (i.e., \( \frac{t-r_i}{q_i(t)} \geq n(t)^{\frac{1}{2}} \)), \( J_i \) will be viewed as a starving job until \( J_i \) is completed, even if \( \theta \) increases after time \( t \). The result is shown in Fig. 7.

Before we explain the results in Fig. 7, we first discuss an advantage of the estimate \( \tilde{n} \) introduced in Section 4.1. Recall that when the system load is close to one, the entire instance is partitioned into multiple idle-free instances. The estimate \( \tilde{n} \) in Section 4.1 is then the average number of jobs in an idle-free instance (excluding the tiny and negligible idle-free instances). As a result, \( \tilde{n} \) underestimates the number of jobs in the largest idle-free instances, which in effect reduces the starvation threshold for these largest idle-free instances. As shown in Fig. 1, when \( P \) is small and the system load is low or slightly greater than 100%, FCFS is an adequate schedule, and thus a smaller starvation threshold yields a better performance. The reduced starvation threshold, which is a by-product of the natural estimate \( \tilde{n} \), then improves the performance of \( BAL(\tilde{n}^{\frac{1}{2}}) \) when \( P \) is small and the system load is close to one. Nevertheless, Fig. 7 shows that the performance of this variant is almost identical to SRPT when \( H \) is large, and yet it outperforms SRPT for small \( H \). Like \( BAL(\tilde{n}^{\frac{1}{2}}) \), when \( H = 2^6 \) and the system is slightly overloading, this variant reduces SRPT’s \( \ell_2 \) norm of flow time by more than 70%. When \( H = 2^{18} \), this variant reduces FCFS’s \( \ell_2 \) norm of flow time by more than 90%.

Note that this variant initially has a small starvation threshold. Thus, it may be tempting to think that when \( H \) is large, this variant may complete some large jobs prematurely. However, this is unlikely to occur when the job arrival process is Poisson and \( H \) is large. This is because under a Poisson arrival process, \( n(t) \) increases steadily after a huge job arrives, and thus \( \theta \) is also increased steadily. Therefore, large jobs seldom become starving prematurely. As a result, when \( H \) is large, this variant and SRPT have almost identical performance.

---

**Figure 7: Comparison of \( BAL(\theta) \) to SRPT and FCFS without estimate.**
Appendix B  Missing Proofs in Section 6

B.1  Proof of Lemma 6.2  To derive a lower bound of \( F(\OPT) \), we remove some tasks from the instance at time \( t^* \) so that under \( \OPT \), every job \( J_i \) has exactly \( n_{OPT}(i) \) remaining tasks at time \( t^* \). The above modification is achievable because \( q_{r,OPT} \) dominates \( n_{OPT} \). Let \( i^* \) be the smallest integer such that \( n_{OPT}(i^*) > 0 \). Further assume that starting from time \( t^* \), the goal of \( \OPT \) becomes to minimize

\[
\sum_{i=1}^{n} (c_i'(OPT) - t^*)^2,
\]

where \( c_i'(OPT) \) is the completion time of \( J_i \) under \( \OPT \) in the modified instance. Observe that for all \( i \geq i^* \), because \( q_{r,OPT}(i) > n_{OPT}(i) \), \( J_i \)'s original release time is at most \( t^* \). Thus, Eq. (2.5) is a lower bound of \( F(\OPT) \). By Eq. (6.4), to minimize Eq. (2.5), \( \OPT \) should execute \( J_1, J_{i^*+1}, \ldots, J_n \) in increasing order of their job indices. Thus, for any \( i \geq i^* \), \( c_i'(OPT) - t^* = \sum_{h=i}^{\infty} n_{OPT}(h) \), which implies Lemma 6.2.

B.2  Proof of Lemma 6.3. To prove Lemma 6.3, we will consider the restriction of some map \( f \) to some subset of \( \dom f \). Specifically, for any map \( f \) and any set \( S \subseteq \dom f \), the restriction of \( f \) to \( S \), denoted by \( f|_S \), is a map from \( S \) to \( \mathbb{N} \) such that \( f|_S(i) = f(i) \) for any \( i \in S \). The following definition adds an element \( x \) and its associated output \( y \) to a map \( f \).

**Definition B.1.** For any map \( f \), any \( x \not\in \dom f \), and any \( y \in \mathbb{N} \), define \( f \cup \{(x, y)\} : \dom f \cup \{x\} \to \mathbb{N} \) as a map such that \( (f \cup \{(x, y)\})(i) = f(i) \) if \( i \in \dom f \) and \( (f \cup \{(x, y)\})(x) = y \).

The following definition considers the union of two disjoint maps.

**Definition B.2.** For any two maps \( f \) and \( \mathcal{h} \) such that \( \dom f \cap \dom \mathcal{h} = \emptyset \), define \( f \cup \mathcal{h} : \dom f \cup \dom \mathcal{h} \to \mathbb{N} \) as a map such that \( (f \cup \mathcal{h})|_{\dom f} = f \) and \( (f \cup \mathcal{h})|_{\dom \mathcal{h}} = \mathcal{h} \).

**Example.** Assume \( f(1) = 1 \), \( f(2) = 2 \), \( h(3) = 3 \), and \( h(4) = 4 \). Further assume \( \dom f = \{1, 2\} \) and \( \dom \mathcal{h} = \{3, 4\} \). We then have \( \dom (f \cup \mathcal{h}) = \{1, 2, 3, 4\} \) and \( (f \cup \mathcal{h})(1) = 1 \), \( (f \cup \mathcal{h})(2) = 2 \), \( (f \cup \mathcal{h})(3) = 3 \), and \( (f \cup \mathcal{h})(4) = 4 \).

**Lemma B.1.** Let \( f \) and \( \mathcal{g} \) be any two maps such that \( S_k(f) \geq S_k(g) \) for any \( k \in \mathbb{N} \). Let \( x \not\in \dom f \cup \dom \mathcal{g} \). Then for any \( k, y \in \mathbb{N} \), \( S_k(f \cup \{(x, y)\}) \geq S_k(g \cup \{(x, y)\}) \).

**Proof.** Assume that in \( \dom f \) (respectively, \( \dom g \)), there are \( p_f \) (respectively, \( p_g \)) elements \( i \) satisfying \( f(i) \geq y \) (respectively, \( g(i) \geq y \)).

**Case 1:** \( p_f \leq p_g \).
- If \( k \leq p_g \), then \( S_k(f \cup \{(x, y)\}) \geq S_k(f) \geq S_k(g) = S_k(g \cup \{(x, y)\}) \).
- If \( k \geq p_g + 1 \), then \( S_k(f \cup \{(x, y)\}) = S_{k-1}(f) + y \geq S_{k-1}(g) + y = S_k(g \cup \{(x, y)\}) \).

**Case 2:** \( p_g < p_f \).
- If \( k \leq p_g \), then \( S_k(f \cup \{(x, y)\}) = S_k(f) \geq S_k(g) = S_k(g \cup \{(x, y)\}) \).
- If \( p_g + 1 \leq k \leq p_f \), then \( S_k(f \cup \{(x, y)\}) = S_k(f) \geq S_{p_g}(f) + y(k - p_g) \geq S_{p_g}(g) + y(k - p_g) \geq S_k(g \cup \{(x, y)\}) \).
- If \( k \geq p_f + 1 \), then \( S_k(f \cup \{(x, y)\}) = S_{k-1}(f) + y = S_{k-1}(g) + y = S_k(g \cup \{(x, y)\}) \).

**Lemma B.2.** Let \( f \), \( g \), and \( \mathcal{h} \) be any three maps such that \( \dom f \cap \dom \mathcal{h} = \emptyset \), \( \dom g \cap \dom \mathcal{h} = \emptyset \), and \( S_k(f) \geq S_k(g), \forall k \in \mathbb{N} \). Then \( S_k(f \cup \mathcal{h}) \geq S_k(g \cup \mathcal{h}), \forall k \in \mathbb{N} \).

**Proof.** We prove Lemma B.2 by induction on \( |\dom \mathcal{h}| \). When \( |\dom \mathcal{h}| = 1 \), the lemma holds due to Lemma B.1. Assume the lemma holds when \( |\dom \mathcal{h}| = z \). When \( |\dom \mathcal{h}| = z + 1 \), pick any \( x \in \dom \mathcal{h} \). Consider the map \( \mathcal{h}' = \mathcal{h}|_{\dom \mathcal{h} \setminus \{x\}} \). Thus, \( |\dom \mathcal{h}'| = z \) and by the induction hypothesis, we then have \( S_k(f \cup \mathcal{h}') \geq S_k(g \cup \mathcal{h}') \) for any \( k \in \mathbb{N} \). By Lemma B.1, we then have \( S_k((f \cup \mathcal{h}') \cup \{(x, h(x))\}) \geq S_k((g \cup \mathcal{h}') \cup \{(x, h(x))\}) \) for any \( k \in \mathbb{N} \). The proof then follows from \( (f \cup \mathcal{h}') \cup \{(x, h(x))\}) = f \cup \mathcal{h}' \) and \( (g \cup \mathcal{h}') \cup \{(x, h(x))\}) = g \cup \mathcal{h}' \).

We are now ready to prove Lemma 6.3. The proof is by induction on \( t \). Clearly, the lemma holds when \( t = 0 \). Assume that the lemma holds when \( t = \tau \). Thus,

\[
S_k(q_{\tau,SRPT} | A(\tau, SRPT)) \geq S_k(q_{\tau, S} | A(\tau, S)), \forall k \in \mathbb{N}.
\]

Because SRPT always executes the job with the least remaining tasks, and \( S \) is work-conserving, we then have \( S_k(q_{\tau + 1, SRPT} | A(\tau, SRPT)) \geq S_k(q_{\tau + 1, S} | A(\tau, S)), \forall k \in \mathbb{N} \). Let \( I_{\tau+1} \) be the index set of the jobs released at time \( \tau + 1 \). By Lemma B.2, for any \( k \in \mathbb{N} \), we have

\[
S_k(q_{\tau + 1, SRPT} | A(\tau, SRPT) \cup I_{\tau + 1}) \geq S_k(q_{\tau + 1, S} | A(\tau, S) \cup I_{\tau + 1}),
\]

which implies

\[
S_k(q_{\tau + 1, SRPT}) \geq S_k(q_{\tau + 1, S}), \forall k \in \mathbb{N}.
\]

Finally, because both SRPT and \( S \) are work-conserving, we have \( S(q_{\tau + 1, SRPT}) = S(q_{\tau + 1, S}) \), which completes the proof.
The proof proceeds as follows: For $I \subseteq S$ and based on $g$, we write $\text{dom } I = \{ a_k | k \in [d_f + 1, \lvert \text{dom } f \rvert] \}$, $I_0 = \{ a_k | \lvert \text{dom } f \rvert \}$, $I_1 = \{ a_k | \lvert \text{dom } f \rvert \}$, and $I_2 = \{ a_k | \lvert \text{dom } f \rvert \}$. Note that by Eq. (2.6), $b_k$ exists for any $k \in [1, d_f]$. The proof proceeds as follows: For $I_1$ (respectively, $I_2$), we will construct a map $g'$ (respectively, $g_2$) that is dominated by $g$. The desired map $g'$ will be constructed based on $g_1$ and $g_2$.

### B.3.2 Construction of $g_2$.

If $I_2 \neq \emptyset$, we construct another map $g_2$ dominated by $g$. Initially, $g_2(i) = 0$ for any $i \in \text{dom } g$. We rewrite $I_2$ as $\{a_{\kappa(1)}, a_{\kappa(2)}, \ldots, a_{\kappa(|I_2|)}\}$ so that $\kappa(1) \leq \kappa(2) \leq \cdots \leq \kappa(|I_2|)$. For brevity, for any positive integers $x$ and $y$ with $x \leq y \leq d_f$, define

$$S_{x,y}(g) = \sum_{k \in [x,y]} g(b_k).$$

Define $y(0) = \kappa(1) - 1$. The construction of $g_2$ proceeds in rounds. In the $j$th $(j \in [1, \lvert I_2 \rvert])$ round, we set

$$x(j) = \max(\kappa(j), y(j - 1) + 1).$$

We set $y(j)$ to be the smallest integer such that

$$S_{x(j),y(j)}(g) \geq \frac{f'(a_{\kappa(j)})}{2}. \tag{2.10}$$

We then set

$$g_2(b_k) = g(b_k), \forall k \in [x(j), y(j)]. \tag{2.11}$$

**Example.** Consider the maps $f$, $f'$, and $g$ shown in Fig. 8. We then have $I_2 = \{a_1, a_2, a_3\}$. Thus, $\kappa(1) = 1$, $\kappa(2) = 2$, $\kappa(3) = 8$.

1. In Round 1, we set $x(1) = \kappa(1) = 1$ and $y(1) = 3$. Observe that $f'(a_{\kappa(1)}) = 10$ and

$$\frac{f'(a_{\kappa(1)})}{2} \leq S_{x(1),y(1)}(g) < f'(a_{\kappa(1)}).$$

2. In Round 2, we set $x(2) = y(1) + 1 = 4$ and $y(2) = 6$. Observe that $f'(a_{\kappa(2)}) = 10$ and

$$\frac{f'(a_{\kappa(2)})}{2} \leq S_{x(2),y(2)}(g) < f'(a_{\kappa(2)}).$$

3. In Round 3, we set $x(3) = \kappa(3) = 8$ and $y(3) = 9$. Observe that $f'(a_{\kappa(3)}) = 5$ and

$$\frac{f'(a_{\kappa(3)})}{2} \leq S_{x(3),y(3)}(g) < f'(a_{\kappa(3)}).$$

□
B.3.3 Correctness of the Construction of $g_2$. To prove the above construction is correct, it suffices to prove that in every round $j$, $x(j)$ is valid (i.e., $x(j) \leq d_g$) and $y(j)$ is valid (i.e., there exists $y(j)$ that satisfies Eq. (2.10)). We first prove the following lemma.

**Lemma B.4.** If $x(j)$ and $y(j)$ are valid, then $S_{x(j),y(j)}(g) < f'(a_{\kappa(j)})$.

**Proof.** First, we have

$$f'(a_{\kappa(j)}) \geq g(b_{\kappa(j)}) \geq g(b_{x(j)}).$$

Thus, $S_{x(j),x(j)}(g) < f'(a_{\kappa(j)})$. Moreover, for any $h \in [1, \lceil \log_2 g \rceil - 1]$, we have $g(b_h) \geq g(b_{h+1})$. Thus, if $S_{x(j),h}(g) < f'(a_{\kappa(h)})/2$, then $S_{x(j),h+1}(g) < f'(a_{\kappa(h)})$. The proof then follows from the definition of $y(j)$. □

Next, we prove that the following statements $X(j)$ and $Y(j)$ hold for any $j \in [1, |I_2|]$ by induction on $j$:

$X(j)$: $x(j) \leq d_g$.

$Y(j)$: $S_{x(j),d_g}(g) \geq \sum_{h=j}^{I_2} f'(a_{\kappa(h)})$.

Observe that $X(j)$ and $Y(j)$ imply that $x(j)$ and $y(j)$ are valid, respectively.

When $j = 1$, $x(1) = \kappa(1) \leq d_f \leq d_g$. Thus, $X(1)$ holds. In addition, we have

$$S_{x(1),d_g}(g) = S_{s(1),d_g}(g) = S(g) - S_{s(1)-1}(g) \geq S(f) - S_{s(1)-1}(f) = \sum_{h=\kappa(1)}^{d_f} f(a_h) \geq \sum_{h=\kappa(1)}^{d_f} f'(a_{\kappa(h)}) \geq \sum_{h=\kappa(1)}^{I_2} f'(a_{\kappa(h)}).$$

Thus, $Y(1)$ holds.

Assume $X(j)$ and $Y(j)$ hold when $j = z$ for some $z \in [1, |I_2| - 1]$. To prove $X(z + 1)$ and $Y(z + 1)$ hold, we first consider the case where $\kappa(z + 1) \geq y(z) + 1$. In this case, $x(z + 1) = k(z + 1) \leq d_f \leq d_g$. Thus, $X(z + 1)$ holds. In addition, we have

$$S_{x(z+1),d_g}(g) = S_{s(z+1),d_g}(g) = S(g) - S_{s(z+1)-1}(g) \geq S(f) - S_{s(z+1)-1}(f) = \sum_{h=\kappa(z+1)}^{d_f} f(a_h) \geq \sum_{h=\kappa(z+1)}^{d_f} f'(a_{\kappa(h)}) \geq \sum_{h=z+1}^{I_2} f'(a_{\kappa(h)}).$$

Thus, $Y(z + 1)$ holds.

Next, we consider the case where $y(z) + 1 > \kappa(z + 1)$. Thus, $x(z + 1) = y(z) + 1$. We have

$$(2.12) S_{x(z),d_g}(g) - S_{x(z),y(z)}(g) \geq \sum_{h=z}^{I_2} f'(a_{\kappa(h)}) - S_{x(z),y(z)}(g)$$

by Lemma B.4.

Thus, both $X(z+1)$ and $Y(z+1)$ hold. By mathematical induction, $X(j)$ and $Y(j)$ hold for any $j \in [1, |I_2|]$.

B.3.4 Properties of $g_2$. Clearly, $g_2$ is dominated by $g$. By Eq. (2.9), $[x(j), y(j)] \cap [x(j'), y(j')] = \emptyset$ if $j \neq j'$. In addition, because we have $g_2(b_k) > 0$ only when $k \in [x(j), y(j)]$ for some $j \in [1, |I_2|]$, we then have

$$(2.13) S(g_2) = \sum_{j=1}^{I_2} \sum_{k \in [x(j), y(j)]} g_2(b_k) \geq \frac{|I_2|}{2} \sum_{i \in I_2} f'(i).$$

In addition, we have

$$(2.14) \sum_{i \in I_2} f'(i)^2 \geq \sum_{k \in [x(j), y(j)]} \sum_{j=1}^{I_2} \sum_{k \in [x(j), y(j)]} g_2(b_k)^2 \geq S(g_2^2).$$

B.3.5 Construction of $g'$ Based on $g_1$ and $g_2$. If $\sum_{i \in I_2} f'(i) \geq \sum_{i \in I_2} f'(i')$, we set $g' = g_1$. Otherwise, we set $g' = g_2$. Because both $g_1$ and $g_2$ are dominated by $g$, $g'$ is dominated by $g$. Observe that $S(f') = \sum_{i \in I_2} f'(i) + \sum_{i \in I_2} f'(i')$. Thus, if $g' = g_1$, by Eq. (2.7), we have $S(g') = S(g_1) = \sum_{i \in I_2} f'(i) \geq \frac{1}{2} S(f')$. By Eq. (2.8), we have $S(f^2) \geq \sum_{i \in I_2} f'(i) = S(g_2^2)$. Finally, if $g' = g_2$, by Eq. (2.13), we have $S(g') = S(g_2) \geq \frac{1}{2} \sum_{i \in I_2} f'(i) \geq \frac{1}{4} S(f')$. By Eq. (2.14), we have $S(f^2) \geq \sum_{i \in I_2} f'(i)^2 > S(g_2^2) = S(g_2^2)$. Therefore, $S(f') > S(g_2^2)$.
B.4 Proof of Lemma 6.5.

B.4.1 The Relationship Between $q_i, \text{SRPT}$ and $q_i$.

**Lemma B.5.** For any $t \in \mathbb{N}$, $\tilde{q}_t$ dominates $q_i, \text{SRPT}$.

**Proof.** It suffices to show that for all $i \in [1,n]$, $q_i, \text{SRPT}(i) \leq \tilde{q}_t(i)$.

**Case 1:** $J_i$ is an FaN job or $J_i$ is an FaS job with $t \leq t_i$. In this case, by Line 14 of Algorithm 1, for any $t' < t$, if $\text{BAL}$ executes a task $J_i, k$ in time slot $[t']$, then it must be the case that $\text{SRPT}$ executes $J_i, k$ in some time slot $[t'']$ with $t'' \leq t'$. Thus, $q_i, \text{SRPT}(i) \leq \tilde{q}_t(i)$.

**Case 2:** $J_i$ is an FaS job with $t > t_i$. In this case, we have $\tilde{q}_t(i) = q_i(i) \geq q_i, \text{SRPT}(i) \geq q_i, \text{SRPT}(i)$.

**B.4.2 $TR[\tilde{q}_t, q(t^*)]$: A Map that Majorizes $q^*, \text{SRPT}$.** As mentioned earlier, we will use Lemma 6.4 to construct a $\frac{1}{2}$-proper map dominated by $q^*, \text{SRPT}$. Note that because $S(\tilde{q}_t^*)$ may be greater than $q(t^*) = S(q^*, \text{SRPT})$, $\tilde{q}_t^*$ may not majorize $q^*, \text{SRPT}$. Thus, we increase the smallest outputs of $\tilde{q}_t^*$ so that $S(\tilde{q}_t^*) = q(t^*)$. We call this operation the truncation of $\tilde{q}_t^*$ at $q(t^*)$.

**Definition B.3.** Let $f$ be any map such that $S(f) \geq 1$. Let $c$ be any integer in $[1, S(f)]$. Let $lu(f,c)$ be the least integer such that $S_{lu(f,c)}(f) \geq c$. The **truncation** of $f$ at $c$, denoted by $TR[f,c]$, is a map with domain $\text{dom} f$ such that $TR[f,c](\pi_f(k))$

\[
\begin{cases}
  f(\pi_f(k)) & \text{if } k \in [1, lu(f,c) - 1] \\
  c - S_{k-1}(f) & \text{if } k = lu(f,c) \\
  0 & \text{if } k \in [lu(f,c) + 1, |\text{dom} f|]
\end{cases}
\]

Finally, $TR[f,c]$ is said to be a valid truncation if $c \in [1, S(f)]$.

**Example.** Assume $f(i) = 10i$ and $\text{dom} f = [1, 10]$. We then have $S_3(f) = 100 + 90 + 80 = 270$ and $S_4(f) = 100 + 90 + 80 + 70 = 340$. If $c = 300$, then $lu(f,c) = 4$, $TR[f,c](10) = 100$, $TR[f,c](9) = 90$, $TR[f,c](8) = 80$, and $TR[f,c](7) = 30$. For all $i \in [1,6]$, $TR[f,c](i) = 0$.

**Proof.** First note that because $q(t^*) \geq u(t^*) \geq 1$ and $q(t^*) \leq S(\tilde{q}_t^*)$, $TR[\tilde{q}_t^*, q(t^*)]$ is a valid truncation. By Fact B.1, $S(TR[\tilde{q}_t^*, q(t^*)]) = q(t^*) = S(q^*, \text{SRPT})$. It is then sufficient to prove

\[
(2.15) \quad S_k(TR[\tilde{q}_t^*, q(t^*)]) \geq S_k(q^*, \text{SRPT}), \forall k \in \mathbb{N}.
\]

To prove (2.15), we consider the following two cases.

**Case 1:** $k \leq lu(\tilde{q}_t^*, q(t^*)) - 1$. The case where $k = 0$ is trivial. Thus, we assume $1 \leq k \leq lu(\tilde{q}_t^*, q(t^*)) - 1$. By the definition of truncation, we have

\[
S_k(TR[\tilde{q}_t^*, q(t^*)]) = S_k(\tilde{q}_t^*), \forall k \in [1, lu(\tilde{q}_t^*, q(t^*)) - 1].
\]

The proof then follows from Lemma B.5. Specifically, by Lemma B.5, we have

\[
S_k(\tilde{q}_t^*) \geq S_k(q^*, \text{SRPT}), \forall k \in [1, n].
\]

**Case 2:** $k \geq lu(\tilde{q}_t^*, q(t^*))$. In this case, we have $S_k(TR[\tilde{q}_t^*, q(t^*)]) = q(t^*) \geq S_k(q^*, \text{SRPT})$.

**B.4.3 A 1-Proper Map Dominated by $TR[\tilde{q}_t^*, q(t^*)]$.** By Lemma 6.4 and Lemma B.6, to prove Lemma 6.5, it suffices to prove the following lemma.

**Lemma B.7.** Let $h = TR[\tilde{q}_t^*, q(t^*)]$. There is a 1-proper map $h'$ dominated by $h$.

**Proof.** Let $I_S = \{i | J_i$ is an FaS job and $t_i \leq t^* \}$. Define a map $h'$ as follows.

\[
(2.16) \quad h'(i) = \begin{cases}
  h(i) & \text{if } i \in I_S \\
  0 & \text{if } i \in [1, n] \setminus I_S
\end{cases}
\]

Clearly, $h'$ is dominated by $h$. It is then sufficient to prove the following inequalities.

\[
(2.17) \quad S(h'^2) \leq \sum_{i=1}^{n} q_i(i)^2
\]

\[
(2.18) \quad S(h') \geq u(t^*)
\]

**Proof of Eq. (2.17).**

\[
S(h'^2) = \sum_{i \in I_S} TR[\tilde{q}_t^*, q(t^*)](i)^2 \leq \sum_{i \in I_S} \tilde{q}_t^*(i)^2 = \sum_{i \in I_S} q_i(i)^2 \leq \sum_{i=1}^{n} q_i(i)^2,
\]

where the first inequality holds because, by Fact B.1, $TR[\tilde{q}_t^*, q(t^*)]$ is dominated by $\tilde{q}_t^*$. 

"
Proof of Eq. (2.18). Let $I_N = [1, n] \setminus I_S$. We then have

$$S(h') = S(h) - \sum_{i \in I_N} h(i)$$

$$\geq S(h) - \sum_{i \in I_N} q_{t^*}(i)$$

$$= q(t^*) - \sum_{i \in I_N} q_{t^*}(i)$$

$$= u(t^*),$$

where the inequality follows because $h(i) \leq \hat{q}_{t^*}(i) = q_{t^*}(i)$ for any $i \in I_N$, the second equality holds because $S(h) = S(TR[\hat{q}_{t^*}, q(t^*)]) = q(t^*)$, and the third equality holds because $q(t^*) = \sum_{i \in I_S} q_{t^*}(i) + \sum_{i \in I_N} q_{t^*}(i)$ and $u(t^*) = \sum_{i \in I_S} q_{t^*}(i)$.

We are now ready to prove Lemma 6.5. Consider Lemma 6.4 and fix $f = TR[\hat{q}_{t^*}, q(t^*)]$ and $g = q_{t^*, SRPT}$. By Lemma B.6, $f$ majorizes $g$. By Lemma B.7, there is a 1-proper map dominated by $f$. Thus, by Lemma 6.4, there is a $\frac{1}{4}$-proper map dominated by $g = q_{t^*, SRPT}$. 

Appendix C Discussion on Standard Scheduling Algorithms

C.1 Discussion on RR. The lower bound instance of RR in [6] shows that RR’s competitive ratio for minimizing the $\ell_2$ norm of flow time is $\Omega(\sqrt{n}/\log n)$. The root cause is that RR frequently switches job execution, and thus delays the completion of many jobs. As a result, RR may have a poor average flow time, which in turn deteriorates the $\ell_2$ norm of flow time.

C.2 Discussion on SRPT.

**Proposition C.1.** Let $S$ be a schedule that minimizes the average flow time. Then $F(S) \leq n \cdot F(OPT)$.

**Proof.**

$$n \cdot F(OPT) = \left(\sum_{i=1}^{n} 1^2\right) \left(\sum_{i=1}^{n} f_i(OPT)^2\right)$$

$$\geq \left(\sum_{i=1}^{n} f_i(OPT)^2\right)^2 \geq \left(\sum_{i=1}^{n} f_i(S)^2\right)^2$$

$$\geq \sum_{i=1}^{n} f_i(S)^2 = F(S),$$

where the first inequality follows from the Cauchy-Schwarz inequality and the second inequality follows from the assumption that $S$ minimizes the average flow time. □

It is well-known that SRPT minimizes the average flow time. Thus, we have the following corollary.

**Corollary C.1.** SRPT is $O(\sqrt{n})$-competitive for minimizing the $\ell_2$ norm of flow time.

Observe that the instance shown in Fig. 1 not only shows that SRPT may cause job starvation, it also shows that the competitive ratio in Corollary C.1 is asymptotically tight by comparing to SRPT.

C.3 Discussion on FCFS.

**Proposition C.2.** Let $S$ be a schedule that minimizes the maximum flow time. Then $F(S) \leq n \cdot F(OPT)$.

**Proof.**

$$n \cdot F(OPT) \geq n \max_i f_i(OPT)^2 \geq n \max_i f_i(S)^2 \geq F(S),$$

where the second inequality follows from the assumption that $S$ minimizes the maximum flow time. □

Because FCFS minimizes the maximum flow time [8], we have the following corollary.

**Corollary C.2.** FCFS is $O(\sqrt{n})$-competitive for minimizing the $\ell_2$ norm of flow time.

Observe that the instance shown in Fig. 2 not only shows that FCFS may deteriorate average flow time, it also shows that the competitive ratio in Corollary C.2 is asymptotically tight by comparing to SRPT.

C.4 Discussion on SJF and SETF. Under SJF, the server always executes the job that has the smallest index. Under SETF, the server always executes the job that has been executed the least so far. Take Fig. 1 as an example, and assume that ties are broken by choosing the job with the smallest index. Under SJF, $J_1$ is completed first, and $J_2$ has to wait until all the other jobs are completed. Under SETF, both $J_1$ and $J_2$ have to wait until all the other jobs are completed. Thus, both SJF and SETF cause job starvation. By comparing to FCFS, Fig. 1 shows that the competitive ratios of SJF and SETF for minimizing the $\ell_2$ norm of flow time are $\Omega(\sqrt{n})$. Next, we give an instance to show that the competitive ratios of SJF and SETF are $\Omega(n)$. In this instance, for every job $J_i$, $p_i = n - i + 1$. We set $r_1 = 1$. For $i \geq 2$, we set $r_i = r_{i-1} + p_{i-1} - 1$. Fig. 9 shows such an instance when $n = 7$.

![Figure 9: A lower bound instance of SJF and SETF when $n = 7$.](image)

It is easy to see that under SJF or SETF, every job is completed after $J_n$ is released. Thus, for every job $J_i$ with $i \leq \frac{n}{2}$, its flow time under SJF or SETF is $\Omega(1 + 2 + 3 + \cdots + \frac{n}{2}) = \Omega(n^2)$. As a result, the $\ell_2$ norm of flow time of SJF or SETF is $\Omega(\sqrt{n^3})$. It is easy to see that for every job $J_i$, $f_i(FCFS) = n$. Therefore, $F(OPT) \leq F(FCFS) = \Theta(n^3)$. We then have the following result.

**Proposition C.3.** The competitive ratios of SJF and SETF for minimizing the $\ell_2$ norm of flow time are $\Omega(n)$.

In hindsight, the root cause of such a poor performance is that SETF and SJF may have poor average flow time and poor maximum flow time simultaneously.
Appendix D  Competitive Ratio of BAL(θ)

In this section, we first prove Theorem 3.3. We then prove that for any $\theta > 0$, the competitive ratio of BAL(θ) for the problem of minimizing the $\ell_2$ norm of flow time cannot be worse than that of SRPT and FCFS. Specifically, we have the following result.

**Theorem D.1.** For any $\theta \geq 0$, BAL(θ) is $O(\sqrt{n})$-competitive for minimizing the $\ell_2$ norm of flow time.

**D.1 Proof of Theorem 3.3.**

$$F(BAL(\tilde{n}^2)) = \sum_{i=1}^{n} \left( \text{norm}_i(\tilde{n}^2) + \text{starv}_i(\tilde{n}^2) \right)^2$$

$$= O\left( \sum_{i=1}^{n} \text{norm}_i(\tilde{n}^2) + \sum_{i=1}^{n} \text{starv}_i(\tilde{n}^2) \right)^2$$

by Theorems 3.1 and 3.2

$$= O\left( \frac{(\alpha n)^2 + \frac{n}{\beta n^2}}{2} \right) F(OPT)$$

$$= O\left( \frac{(\alpha n)^2 + \beta^{-2} n^2}{2} \right) F(OPT).$$

Thus, the competitive ratio of BAL(\tilde{n}^2) for minimizing the $\ell_2$ norm of flow time is $O\left( \frac{\alpha n}{\beta^{-2} n^2} \right)$.

**D.2 Proof of Theorem D.1.** For every job $J_i$, define $X_i$ as the set of tasks that are executed between time $r_i$ and time $c_i(BAL)$ under BAL. In other words,

$$X_i = \{ J_{i,k} | r_i < c_{i,k} \text{(BAL)} \leq c_i(BAL) \}.$$ 

Thus $|X_i| = f_i(BAL)$. Further divide $X_i$ into the following two sets according to the release time.

$$X_{i,1} = \{ J_{i,k} | J_{i,k} \in X_i \land r_i \leq r_i \}$$

$$X_{i,2} = \{ J_{i,k} | J_{i,k} \in X_i \land r_i > r_i \}$$

We first upper bound $|X_{i,2}|^2$ in the following lemma.

**Lemma D.1.** For any $i \in [1,n]$, $|X_{i,2}|^2 = O(F(OPT))$.

**Proof.** Because tasks in $X_{i,1}$ are released by time $r_i$ and are completed after time $r_i$ under BAL, we have $q(r_i) \geq |X_{i,1}|$. Thus, $|X_{i,1}|^2 \leq q(r_i) = O\left( \sum_{t=0}^{n} q(t) \right) = O(F(OPT))$, where the first equality follows from $q(t+1) \geq q(t) - 1$ for any $t \in \mathbb{N}$, and the second equality follows from Lemma 5.1.

We further divide $X_{i,2}$ into the following two sets.

$$X_{i,2}^S = \{ J_{i,k} | J_{i,k} \in X_{i,2} \land J_{i,k} \text{ is executed as a normal task} \}$$

$$X_{i,2}^N = \{ J_{i,k} | J_{i,k} \in X_{i,2} \land J_{i,k} \text{ is executed as a starving task} \}$$

**Lemma D.2.** For any $i \in [1,n]$, $|X_{i,2}^N| = O(f_i(SRPT)).$

**Proof.** Let $J_{i,k} \in X_{i,2}^N$. It suffices to show $r_i < c_{i,k} \text{(SRPT)} < c_i(BAL)$. The first inequality holds because $J_{i,k} \in X_{i,2}$ and thus $r_i < r_i < c_{i,k} \text{(SRPT)}$. Next, consider the second inequality and note that $c_i(SRPT) = c_i(BAL)$. Because $J_{i,k}$ is executed as a normal task, $J_i$ cannot be a starving job when $J_{i,k}$ is executed. In other words, when $J_{i,k}$ is executed, both $J_{i,k}$ and $J_{i,p}$ are normal tasks. By the design of Line 14 in Algorithm 1, the fact that BAL executes $J_{i,k}$ before $J_{i,p}$ then implies $c_{i,k}(SRPT) < c_{i,p}(SRPT)$. □

By Lemma D.2 and Corollary C.1, we then have

**Lemma D.3.** $\sum_{i=1}^{n} |X_{i,2}^N|^2 = O(n F(OPT)).$

Next, we upper bound $|X_{i,2}^S|$. Define $f_{i,k}(S) = c_{i,k}(S) - r_i$. The following lemma is a simple application of Lemma 5.1.

**Lemma D.4.** For any work-conserving schedule $S$,

$$F(OPT) \geq \sum_{i=1}^{n} \sum_{k=1}^{p_i} f_{i,k}(S).$$

**Proof.** By Lemma 5.1, we have $F(OPT) \geq \sum_{i=1}^{n} \sum_{t=r_i}^{c_i(S)} q_{t,S}(i)$. Recall that $q_{t,S}(i)$ counts the number of remaining tasks of $J_i$ at time $t$ under $S$. Thus, in $\sum_{i=1}^{n} \sum_{t=r_i}^{c_i(S)} q_{t,S}(i)$, a task $J_{i,k}$ is counted from time $r_i$ to time $c_{i,k}(S) - 1$. Thus, $J_{i,k}$ is counted $(c_{i,k}(S) - 1) - r_i + 1 = f_{i,k}(S)$ times. As a result, $F(OPT) \geq \sum_{i=1}^{n} \sum_{t=r_i}^{c_i(S)} q_{t,S}(i) = \sum_{i=1}^{n} \sum_{k=1}^{p_i} f_{i,k}(S)$.

□

To prove Theorem D.1, it is then sufficient to prove the following lemma.

**Lemma D.5.** For any $i \in [1,n]$, $|X_{i,2}^S|^2 = O(n F(OPT))$.

**Proof.** Observe that Lemma D.5 and Lemma D.4 imply $\sum_{i=1}^{n} |X_{i,2}^S|^2 = O(n^2 F(OPT))$. □

**D.2.1 Proof of Lemma D.5.** For any task $J_{i,k}$, we partition $X_{i,2}^S$ into two sets by comparing the completion time of the task in $X_{i,2}^S$ to the completion time of $J_{i,k}$.

Specifically, define

$$X_{i,2,k}^S = \{ J_{i,k} | J_{i,k} \in X_{i,2}^S \land c_{i,k} \text{(BAL)} < c_{i,k} \text{(BAL)} \}$$

We further divide $X_{i,2,k}^S$ into the following two sets.

$$X_{i,2,k}^N = \{ J_{i,k} | J_{i,k} \in X_{i,2,k}^S \land J_{i,k} \text{ is executed as a normal task} \}$$

$$X_{i,2,k}^N = \{ J_{i,k} | J_{i,k} \in X_{i,2,k}^S \land J_{i,k} \text{ is executed as a starving task} \}$$
\[ X_{i,2,k}^{S} = \{ j_{i',k'} : j_{i',k'} \in X_{i,2}^{S} \land c_{i',k'}(BAL) > c_{i,k}(BAL) \} \]

Note that because \( X_{S}^{2} \) is a subset of \( X_{i,2} \), tasks in \( X_{i,2}^{S} \) cannot be from \( J_{i} \). Thus, for any task in \( X_{i,2}^{S} \), its completion time under \( BAL \) cannot be \( c_{i,k}(BAL) \). Thus, \( |X_{i,2,k}^{S}| = |X_{i,2,k}^{S}| + |X_{i,2,k}^{BAL}| \). Moreover, because tasks in \( X_{i,2,k}^{S} \) are executed between time \( r_{i} + 1 \) and time \( c_{i,k}(BAL) - 1 \) under \( BAL \), we have \( |X_{i,2,k}^{S}| \leq (c_{i,k}(BAL) - 1) - (r_{i} + 1) < f_{i,k}(BAL) \). Thus, to prove Lemma D.5, it suffices to prove

\[
(\star) \quad |X_{i,2,k}^{S}|^{2} = O(n) \sum_{k=1}^{p_{i}} |X_{i,2,k}^{S}|.
\]

Note that for any FaS job \( J_{i'} \), its number of starving tasks is maximized when it becomes starving. Thus, for any FaS job \( J_{i'} \), it has at most \( q_{i'}(i') \) starving tasks throughout the course of the execution of \( BAL(\theta) \). Because \( X_{i,2}^{S} \) only contains starving tasks, \( J_{i'} \) has at most \( q_{i'}(i') \) tasks in \( X_{i,2}^{S} \). We further use the following lemma to upper bound \( q_{i}(i) \).

**Lemma D.6.** Let \( J_{i'} \) be a job that has some task in \( X_{i,2}^{S} \). Then \( q_{i'}(i') \leq q_{i}(i) \).

**Proof.** Let \( t = t_{i'} \). Because \( J_{i'} \) has some task in \( X_{i,2}^{S} \), we have \( r_{i'} > r_{i} \) and thus

\[
t - r_{i'} < t - r_{i}.
\]

It is then sufficient to show that

\[
\frac{t - r_{i'}}{q_{i'}(i')} \geq \frac{t - r_{i}}{q_{i}(i)} \tag{4.19}
\]

**Case 1: \( J_{i} \) is an FaN job.** Thus, when \( J_{i} \) becomes starving, \( J_{i} \) is a normal job. Therefore \( \frac{t - r_{i'}}{q_{i'}(i')} \geq \theta > \frac{t - r_{i}}{q_{i}(i)} \).

**Case 2: \( J_{i} \) is an FaS job.** In this case, \( t_{i'} \leq t_{i} \) must hold; otherwise, if \( t_{i'} > t_{i} \), then by the design of \( BAL(\theta) \), all the starving tasks of \( J_{i} \) must be executed after \( J_{i} \) is completed. Thus, \( J_{i} \) cannot have any task in \( X_{i,2}^{S} \). As a result, we only need to consider two possible subcases: \( t_{i'} < t_{i} \) and \( t_{i'} = t_{i} \). If \( t_{i'} < t_{i} \), then at time \( t_{i'} \), \( J_{i} \) is normal and \( J_{i} \) is starving. Thus, Eq. (4.19) holds for the same reason as in Case 1. If \( t_{i'} = t_{i} \), then by the fact that \( BAL(\theta) \) executes a starving task of \( J_{i} \) before \( J_{i} \) is completed and the tie-breaking rule of \( BAL(\theta) \), we have \( \gamma_{i'} \geq \gamma_{i} \), which implies Eq. (4.19).

By Lemma D.6, every job \( J_{i'} \) has at most \( q_{i'}(i) \leq p_{i} \) tasks in \( X_{i,2}^{S} \). Therefore,

\[
|X_{i,2,k}^{S}| \leq n \cdot p_{i} \tag{4.20}
\]

Next, we upper bound \( |X_{i,2,k}^{S}| \) to lower bound the right-hand side of Eq. (\( \star \)).

**Lemma D.7.** Let \( J_{i'} \) be a job that has some tasks in \( X_{i,2,k}^{S} \). Then \( t_{i'} \geq c_{i,k}(BAL) \).

**Proof.** For the sake of contradiction, assume \( t_{i'} < c_{i,k}(BAL) \).

**Case 1: \( J_{i} \) is an FaN job or \( J_{i} \) is an FaS with \( t_{i'} < t_{i} \).** In this case, because \( t_{i'} < c_{i,k}(BAL) \), by the design of \( BAL(\theta) \), \( J_{i} \) is completed before \( J_{i,k} \) is completed under \( BAL \). Thus, \( J_{i} \) cannot have any task in \( X_{i,2,k}^{S} \), which contradicts to the assumption of \( J_{i'} \).

**Case 2: \( J_{i} \) is an FaS job with \( t_{i'} \geq t_{i} \).** In this case, the assumption that \( J_{i} \) has a task in \( X_{i,2}^{S} \) implies that \( t_{i'} = t_{i} \) and that after time \( t_{i'} \), \( J_{i} \) cannot be executed until \( J_{i'} \) is completed. Because \( t_{i'} < c_{i,k}(BAL) \), \( J_{i} \) cannot have any task in \( X_{i,2,k}^{S} \), which contradicts to the assumption of \( J_{i'} \). \( \square \)

**Lemma D.8.** For any job \( J_{i'} \), \( J_{i'} \) has at most \( p_{i} - k \) tasks in \( X_{i,2,k}^{S} \).

**Proof.** Assume that \( J_{i'} \) has at least one task in \( X_{i,2,k}^{S} \). By Lemma D.7, \( t_{i'} \geq c_{i,k}(BAL) \). Thus, by Lemma D.6, \( q_{i'}(i') \leq q_{i'}(i) \leq q_{i,k}(BAL) = p_{i} - k \). Thus, \( J_{i'} \) has at most \( p_{i} - k \) starving tasks throughout the course of the execution of \( BAL(\theta) \). Because \( X_{i,2,k}^{S} \) only contains starving tasks, \( J_{i'} \) has at most \( p_{i} - k \) tasks in \( X_{i,2,k}^{S} \).

By Lemma D.8, we have

\[
|X_{i,2,k}^{S}| \leq n(p_{i} - k). \tag{4.21}
\]

**Proof of Eq. (\( \star \)).** Let \( k^{*} = p_{i} - \left\lfloor \frac{|X_{i,2}^{S}|}{2n} \right\rfloor \). By Eq. (4.20), \( k^{*} \geq p_{i} - \left\lfloor \frac{p_{i}}{2} \right\rfloor \geq 1 \). Therefore, task \( J_{i,k}^{*} \) exists. By Eq. (4.21), we have

\[
|X_{i,2,k^{*}}^{S}| = |X_{i,2}^{S}| - |X_{i,2,k^{*}}^{S}| \geq |X_{i,2}^{S}| - n(p_{i} - k^{*}) \geq |X_{i,2}^{S}| - n \left( p_{i} - \left\lfloor \frac{|X_{i,2}^{S}|}{2n} \right\rfloor \right) \geq \frac{|X_{i,2}^{S}|}{2n} \geq \frac{|X_{i,2}^{S}|}{2}.
\]

Therefore,

\[
n \sum_{k=1}^{p_{i}} |X_{i,2,k}^{S}| \geq n \sum_{k=k^{*}}^{p_{i}} |X_{i,2,k}^{S}| \geq n \sum_{k=k^{*}}^{p_{i}} \|X_{i,2,k^{*}}^{S}\| \geq n(p_{i} - k^{*} + 1) |X_{i,2,k^{*}}^{S}| \geq \left( \left\lfloor \frac{|X_{i,2}^{S}|}{2n} \right\rfloor + 1 \right) |X_{i,2}^{S}| - \left( \left\lfloor \frac{|X_{i,2}^{S}|}{2} \right\rfloor \right)^{2},
\]

where the second inequality follows from \( X_{i,2,p_{i}}^{S} \supseteq X_{i,2,p_{i}-1}^{S} \supseteq \cdots \supseteq X_{i,2,1}^{S} \).
Appendix E  Analysis of FCFS

Let $p_{\text{max}} = \max_i p_i$, $p_{\text{min}} = \min_i p_i$, and $P = \frac{p_{\text{max}}}{p_{\text{min}}}$. This section is devoted to proving the following result.

**Theorem E.1.** When $P = O(1)$, FCFS is $O(1)$-competitive for minimizing the $\ell_2$ norm of flow time.

Clearly, the following theorem implies Theorem E.1.

**Theorem E.2.** FCFS is $O(\sqrt{P})$-competitive for minimizing the $\ell_2$ norm of flow time.

To prove Theorem E.2, it suffices to prove

\begin{equation}
\sum_{i=1}^{n} f_i(\text{FCFS})^2 = O(P) \sum_{i=1}^{n} f_i(\text{OPT})^2.
\end{equation}

Observe that $f_i(S)^2 = \Theta \left( 1 + \sum_{t=r_i}^{\infty} (t-r_i) \right)$. For any set of job indices $J$, define $A_{\ell}(J)$ as $\sum_{t \in J} (t-r_i)$. Recall that $A(t,S)$ is the set of active job indices at time $t$ under schedule $S$.

Thus, for any schedule $S$, we have

$$
\sum_{i=1}^{n} f_i(S)^2 = \Theta \left( n + \sum_{i=1}^{n} \sum_{t=r_i}^{-1} A_{\ell}(\{i\}) \right) = \Theta \left( n + \sum_{t \in \mathbb{N}} A_{\ell}(A(t,S)) \right).
$$

Therefore, to prove Eq. (5.22), it suffices to prove that for any time $t$,

\begin{equation}
A_{\ell}(A(t,\text{FCFS})) = O(P)A_{\ell}(A(t,\text{OPT})).
\end{equation}

For brevity, in the following proof, we omit the subscript $t$ in $A_{\ell}$ and the input $t$ in $A(t,S)$. We consider the following two cases.

**Case 1:** $A(\text{OPT}) \setminus A(\text{FCFS}) \neq \emptyset$. Under FCFS, jobs are processed in increasing order of their release times. Therefore, if $J_1$ is completed before $J_2$ under FCFS, then $r_1 \leq r_2$ and $A(\{1\}) = t-r_1 \geq t-r_2 = A(\{2\})$. We then have the following simple fact about FCFS.

**Fact E.1.** Let $J_1 \in A(\text{OPT}) \setminus A(\text{FCFS})$. Let $J_2 \in A(\text{FCFS}) \setminus A(\text{OPT})$. Then $A(\{1\}) \geq A(\{2\})$.

We have the following lemma about non-preemptive and work-conserving schedules. We postpone the proof to the end of this section. Note that FCFS is non-preemptive and work-conserving, and OPT is work-conserving.

**Lemma E.1.** Let $S$ be any work-conserving schedule. Let $S_{NW}$ be any non-preemptive and work-conserving schedule. If $A(S) \setminus A(S_{NW}) \neq \emptyset$, then

$$
|A(S_{NW}) \setminus A(S)| = O(P)|A(S) \setminus A(S_{NW})|.
$$

We are now ready to prove Eq. (5.23).

$$
\begin{align*}
\text{Age}(A(\text{FCFS})) &= \text{Age}(A(\text{FCFS}) \cap A(\text{OPT})) + \text{Age}(A(\text{FCFS}) \setminus A(\text{OPT})) \\
&\leq \text{Age}(A(\text{FCFS}) \cap A(\text{OPT})) + O(P)\text{Age}(A(\text{OPT}) \setminus A(\text{FCFS})) \\
&= O(P)\text{Age}(\text{OPT}),
\end{align*}
$$

where the inequality follows from Fact E.1 and Lemma E.1.

**Case 2:** $A(\text{OPT}) \setminus A(\text{FCFS}) = \emptyset$. In other words, $A(\text{OPT}) \subseteq A(\text{FCFS})$. Because FCFS is non-preemptive, at any time $t$, there can be at most one job $J_h$ that is partially processed (i.e., FCFS has completed some, but not all, tasks of $J_h$). We first consider the case where $J_h$ does not exist. Thus, FCFS does not complete any task of the jobs in $A(\text{FCFS})$ by time $t$. Because $A(\text{OPT}) \subseteq A(\text{FCFS})$ and both FCFS and OPT are work-conserving, we then have $A(\text{OPT}) = A(\text{FCFS})$. Thus, Eq. (5.23) holds.

We then consider the case where $J_h$ exists. Because $J_h$ is the only partially processed job at time $t$ under FCFS and $A(\text{OPT}) \subseteq A(\text{FCFS})$,

\begin{equation}
\text{Age}(A(\text{FCFS})) = \text{Age}(A(\text{FCFS}) \cap A(\text{OPT})) + \text{Age}(A(\text{FCFS}) \setminus A(\text{OPT})) \\
= \text{Age}(A(\text{OPT})) + \text{Age}(A(\text{FCFS}) \setminus A(\text{OPT})) \\
\leq \text{Age}(A(\text{OPT}) \setminus A(\text{FCFS})) + O(P)\text{Age}(A(\text{FCFS})) \\
= O(P)\text{Age}(\text{OPT}).
\end{equation}

Because $J_h$ is partially processed by FCFS, we then have

\begin{equation}
\text{Age}(\{h\}) \geq \text{Age}(\{i\}), \forall i \in A(\text{FCFS}).
\end{equation}

Assume that FCFS completes $y$ tasks of $J_h$ by time $t$. Thus, OPT completes $y$ tasks of jobs in $A(\text{FCFS})$ by time $t$. As a result, OPT completes at most $\frac{y}{p_{\text{min}}} \leq P$ jobs in $A(\text{FCFS})$ by time $t$. Thus,

\begin{equation}
|A(\text{FCFS}) \setminus A(\text{OPT})| \leq P.
\end{equation}

As a result,

\begin{align*}
\text{Age}(A(\text{FCFS})) &= \text{Age}(A(\text{FCFS}) \cap A(\text{OPT})) + \text{Age}(A(\text{FCFS}) \setminus A(\text{OPT})) \\
&\leq \text{Age}(A(\text{FCFS}) \cap A(\text{OPT})) + O(P)\text{Age}(\{h\}) \\
&\leq \text{Age}(A(\text{FCFS}) \cap A(\text{OPT})) + P \cdot \text{Age}(A(\text{OPT})) \\
&= O(P)\text{Age}(\text{OPT}).
\end{align*}

**E.1 Proof of Lemma E.1** Let $z$ be the number of remaining tasks of jobs in $A(S_{NW})$ at time $t$ under
Because both schedules are work-conserving, \( z \) is also the number of remaining tasks of jobs in \( A(S) \) at time \( t \) under \( S \). We categorize the remaining tasks as follows.

\[
S_{NW} \quad z_{NW,I} = \text{the numbers of remaining tasks of jobs in } A(S_{NW}) \cap A(S) \text{ at time } t \text{ under } S_{NW} \\
S_{NW,D} = \text{the numbers of remaining tasks of jobs in } A(S_{NW}) \setminus A(S) \text{ at time } t \text{ under } S_{NW} \\
S_I = \text{the numbers of remaining tasks of jobs in } A(S) \cap A(S_{NW}) \text{ at time } t \text{ under } S \\
S_D = \text{the numbers of remaining tasks of jobs in } A(S) \setminus A(S_{NW}) \text{ at time } t \text{ under } S
\]

Thus,

\[
(5.27) \quad z = z_{NW,I} + z_{NW,D} = z_I + z_D.
\]

Clearly,

\[
(5.28) \quad |A(S) \setminus A(S_{NW})| \geq \left\lceil \frac{z_D}{p_{\text{max}}} \right\rceil.
\]

On the other hand, because \( S_{NW} \) is non-preemptive, there is at most one job in \( A(S_{NW}) \) that is processed partially. As a result,

\[
(5.29) \quad |A(S_{NW}) \setminus A(S)| \leq \left\lfloor \frac{z_{NW,D}}{p_{\text{min}}} \right\rfloor.
\]

Next, we establish the relationship between \( z_{NW,D} \) and \( z_D \). Because \( S_{NW} \) is non-preemptive, for any subset \( J \) of \( A(S_{NW}) \), at most one job in \( J \) is partially processed at time \( t \) under \( S_{NW} \). Let \( J = A(S_{NW}) \cap A(S) \). Thus, \( z_{NW,I} > (\sum_{i \in J} p_i) - p_{\text{max}} \geq z_I - p_{\text{max}} \). This inequality together with Eq. (5.27) implies

\[
(5.30) \quad z_{NW,D} < z_D + p_{\text{max}}.
\]

As a result,

\[
\begin{align*}
|A(S_{NW}) \setminus A(S)| & \leq \left\lceil \frac{z_{NW,D}}{p_{\text{min}}} \right\rceil \\
& \leq \left\lfloor \frac{z_D + p_{\text{max}}}{p_{\text{min}}} \right\rfloor \leq \frac{z_D}{p_{\text{min}}} + 1 \\
& = \frac{P \cdot z_D}{p_{\text{max}}} + P + 1 \\
& \leq P|A(S) \setminus A(S_{NW})| + P + 1 \\
& = O(P)|A(S) \setminus A(S_{NW})|,
\end{align*}
\]

where the last equality follows from the assumption that \( A(S) \setminus A(S_{NW}) \neq \emptyset \). This completes the proof of Lemma E.1.

Recall that FCFS minimizes the maximum flow time. In fact, when \( P \) is small, FCFS is also good at minimizing the average flow time. Specifically, we have the following theorem.

**Theorem E.3.** When \( P = O(1) \), FCFS is \( O(1) \)-competitive for minimizing the average flow time.

**Proof.** First note that minimizing the average flow time is equivalent to minimizing the total flow time. We then prove it by simplifying the proof for Theorem E.2. In particular, the definition of \( Age_t(J) \) becomes \( |J| \). Observe that under the new definition, Eq. (5.23), which can be proved similarly as before, implies the theorem.

\[\Box\]
Appendix F  A Lower Bound of the Competitive Ratio for Minimizing the $l_2$ Norm of Flow Time

Theorem F.1. Even if $n$ is known a priori, the competitive ratio of any randomized algorithm against an oblivious adversary for minimizing the $l_2$ norm of flow time is $\Omega(n^{0.5})$.

The proof is based on the proof of [6, Theorem 1]. In [6], Bansal and Pruhs used Yao's minimax principle, and they gave a distribution that has two possible instances, $I_1$ and $I_2$, where $I_1$ has fewer jobs than $I_2$. In our proof, we simply add small dummy jobs to the end of $I_1$ so that both instances have the same number of jobs.

Proof. Let $A$ be any deterministic algorithm. Without loss of generality, assume that $A$ knows $n$ a priori. We will give a distribution over two problem instances of the same number of jobs, and show that $A$ has a competitive ratio of $\Omega(n^{0.5})$ on average with respect to the distribution. The proof then follows from Yao's minimax principle [28]. The inputs are parameterized by an integer $L$. The two possible instances, denoted by $I_1$ and $I_2$, are identical before time $L^5$. In addition, both instances have a probability of 0.5 to occur. At time 0, a large job $J_1$ with processing time $L^3$ arrives. At time $t \in \{0, L^3, 2L^3, 3L^3, \ldots, (L^3-1)L^2\}$, a medium job with processing time $L^2$ arrives. Thus, there are $L^3$ medium jobs. In $I_1$, at time $t \in \{L^5 + L^3, L^5 + L^3 + 1, L^5 + L^3 + 2, L^5 + L^3 + 3, \ldots, 2L^5 + L^3 - 1\}$, a small job with unit processing time arrives. In $I_2$, at time $t \in \{L^5, L^5 + 1, L^5 + 2, L^5 + 3, \ldots, 2L^5 - 1\}$, a small job with unit processing time arrives. As a result, $n = 1 + L^3 + L^5 = \Theta(L^3)$ in both instances. Let $t' = L^5 - 1$. Because $I_1$ and $I_2$ are identical before time $L^5$, each instance still has a probability of 0.5 to occur at time $t'$, even if $n$ is known. Let $S$ be the schedule obtained by $A$. We divide the proof into two cases.

Case 1: $A$ does not finish $J_1$ by time $t'$. Thus, $J_1$ has flow time $\Omega(L^5)$ and $F(S) = \Omega((L^5)^2)$. At time $t'$, with probability 0.5, the instance is $I_1$, and we consider the schedule $FCFS$ obtained by applying FCFS to $I_1$. It is easy to see that $F(FCFS) = \Theta((L^3)^2 + ((L^3)^2)L^3 + (1^2)L^5) = \Theta(L^6)$. Therefore, in this case, $A$'s average competitive ratio is $\Omega(\sqrt{L})$.

Case 2: $A$ finishes $J_1$ by time $t'$. In this case, $A$ has to spend $\Theta(L^3)$ units of time on medium jobs after time $L^5$. At time $t'$, with probability 0.5, the instance is $I_2$, and we consider the schedule $SRPT$ obtained by applying $SRPT$ to $I_2$. Clearly, $F(SRPT) = \Theta((L^5)^2) = \Theta((L^5)^2)$. We then consider $F(S)$ in the following two subcases:

Case 2A: $S$ spends at most $L^3/2$ units of time on medium jobs between time $L^5$ and time $1.5L^5$. In this subcase, $\Omega(L^{5/2})$ medium jobs are not finished under $S$ by time $1.5L^5$. These medium jobs have flow time $\Omega(L^5)$, and thus $F(S) = \Omega((L^5)^2)$. Therefore, in this subcase, $A$'s average competitive ratio is $\Omega(\sqrt{L})$.

Case 2B: $S$ spends more than $L^3/2$ units of time on medium jobs between time $L^5$ and time $1.5L^5$. In this subcase, at time $1.5L^5$, at least $L^3/2$ small jobs have arrived but are not finished under $S$. Let $S_{small}$ be the set of the small jobs that are not finished by time $1.5L^5$ under $S$, including the small jobs that arrive after time $1.5L^5$. Thus, $|S_{small}| = \Theta(L^3)$. By a simple exchange argument, it can be shown that after time $1.5L^5$, the best strategy for $A$ to minimize $\sum_{i \in S_{small}} f_i(S)^2$, which is a lower bound of $F(S)$, is to execute jobs in $S_{small}$ in increasing order of their time.

Finally, because $L = \Theta(n^{1/5})$, the average competitive ratio of $A$ is $\Omega(n^{1/10})$ in all cases.
Appendix G Discussion on the Priority-Based Scheduling Algorithm

We give an instance to show that the priority-based scheduling algorithm mentioned in Section 1.2 may perform poorly, compared with SRPT. Recall that in every time slot \( t \), the priority-based scheduling algorithm executes a task of the job that has the highest priority at time \( t \), where the priority of a job \( J_i \) at time \( t \) is defined as \( \frac{t - r_i}{\text{number of remaining tasks of } J_i \text{ at time } t} \). In addition, define the age of job \( J_i \) at time \( t \) as \( t - r_i \).

The instance is parameterized by an integer \( k \).
There are \( k + 1 \) different classes of jobs, \( C_0, C_1, \ldots, C_k \). Every job in \( C_h \) has a processing time of \( P_h = 2^{k-h} \). \( C_0 \) has two jobs, and both of them are released at time 0. All the other classes of jobs than \( C_0 \) have the same total processing time of \( X = 5 \cdot 2^k \). Let \( R_{h,i} \) be the release time of the \( i \)th job in \( C_h \). We set \( R_{0,1} = 0 \) and \( R_{1,1} = 2^k \). For any \( 2 \leq h \leq k \), \( R_{h,1} = R_{h-1,1} + X \).

For every \( 1 \leq h \leq k \), and \( 2 \leq i \leq \frac{X}{P_h} \), we set \( R_{h,i} = R_{h-1,i} + P_h \). Observe that if we remove one job in \( C_0 \), then every job can be executed once it is released. Fig. 10 shows the instance when \( k = 3 \). Clearly, \( n = \Theta(2^k) \) and \( X = \Theta(n) \).

We first consider the schedule obtained by SRPT. Under SRPT, the last completed job is from \( C_0 \), and every other job can be executed once it is released. Thus,
\[ F(SRPT) = (P_0)^2 + (kX + 2P_0)^2 + \sum_{h=1}^{k} \left( \frac{X}{P_h} \right) \cdot P_h^2 = O(k^2X^2) = O((\log n)^2n^2). \]

Let \( S \) be the schedule obtained by the priority-based scheduling algorithm. We have the following lemma, whose proof is postponed to the end of this section.

**Lemma G.1.** For any \( 2 \leq h \leq k \), at time \( R_{h,1} \) under \( S \), one active job is from \( C_h \), and all the other active jobs are from \( C_{h-1} \). Moreover, the total remaining processing time of the jobs from \( C_{h-1} \) at time \( R_{h,1} \) under \( S \) is \( 2^k \).

The above lemma shows that compared with SRPT, the priority-based scheduling algorithm unnecessarily delay many small jobs and thus deteriorates the performance. Specifically, by Lemma G.1, at time \( R_{h,1} \) under \( S \), there are \( \frac{X}{P_h} = \Theta(n) \) active jobs in \( C_{k-1} \). All these jobs have flow times of \( \Omega(2^k) = \Omega(n) \) under \( S \). Thus,
\[ F(S) = \Omega(n^3) \text{ and} \]
\[ \frac{F(S)}{F(SRPT)} = \Omega\left( \frac{n}{(\log n)^2} \right). \]

**Proof of Lemma G.1.** We prove the lemma by induction on \( h \). We first consider the case where \( h = 2 \).
Clearly, at time \( 2^k \), one job in \( C_0 \) is completed under \( S \). In addition, \( S \) starts to execute the second job in \( C_0 \) at time \( 2^k \). At time \( 2^k + 2^k \), both jobs in \( C_0 \) are completed. As a result, at time \( R_{2,1} \), one active job is from \( C_2 \), and all the other active jobs are from \( C_1 \). Observe that there is no idle time slot under \( S \) before all jobs are completed. Moreover, if we remove one job in \( C_0 \), which has size \( 2^k \), then every job can be executed once it is released. Thus, for any \( 1 \leq h \leq k \), \( 1 \leq i \leq \frac{X}{P_h} \), at time \( R_{h,i} \), the total remaining processing time of active jobs is \( 2^k + P_h \). Thus, at time \( R_{2,1} \) the total remaining processing time of jobs from \( C_1 \) is \( 2^k \). This completes the proof for the induction basis.

Assume that the lemma holds for \( h = v \) for some \( v \in [2, k - 1] \). We prove the lemma also holds for \( h = v + 1 \). Let \( \tau = R_{v+1,1} + 3 \cdot 2^v \). Let \( T_v \) be the time interval between \( \tau \) and \( R_{v+1,1} - 1 \). By the induction hypothesis, during \( T_v \) under \( S \), all the active jobs are from \( C_{v-1} \) and \( C_v \), and their total remaining processing time is at most \( 2^k + P_v \). Observe that \( S \) always executes jobs in the same class following their releasing order. Thus, during \( T_v \), the age of any job in \( C_v \) is at most \( 2^k + P_v \). A job is said to be intact if \( S \) has not executed any task of the job. Therefore, during \( T_v \), the priority of any intact job in \( C_v \) is at most \( \frac{2^k + P_v}{P_v} = 2^v + 1 \).

Next, we lower bound the priority of any active job in \( C_{v-1} \) during \( T_v \). Clearly, the age of any active job in \( C_{v-1} \) after time \( \tau \) is at least \( 3 \cdot 2^v \). Thus, the priority of any active job in \( C_{v-1} \) during \( T_v \) is at least \( \frac{3^v}{P_{v-1}} = 3 \cdot 2^v - 1 = 2^v + 2^{v-1} > 2^v + 1 \), where the inequality follows from \( v \geq 2 \). Thus, starting from time \( \tau \), \( S \) cannot execute any intact job in \( C_v \) before all the jobs in \( C_{v-1} \) are completed. Clearly, at any time under \( S \), among all the active jobs from \( C_v \), at most one of them is not intact. Thus, \( S \) can start to execute the remaining jobs in \( C_{v-1} \) before time \( \tau + P_v \). By the induction hypothesis, at time \( \tau \), the total remaining processing time of jobs from \( C_{v-1} \) is at most \( 2^k \). Thus, all the remaining jobs from \( C_{v-1} \) can be completed before time \( \tau + P_v + 2^k < R_{v+1,1} \). As a result, at time \( R_{v+1,1} \) under \( S \), one active job is from \( C_{v+1} \), and all the other active jobs are from \( C_v \). By an argument similar to that for the induction basis, the total remaining processing time of jobs from \( C_v \) at time \( R_{v+1,1} \) under \( S \) is \( 2^k \). Thus, the proof for the induction step is completed.