Stochastic Scheduling of Heavy-tailed Jobs

Sungjin Im*[1], Benjamin Moseley[2], and Kirk Pruhs†[3]

1 **Electrical Engineering and Computer Science**
University of California, Merced
`sim3@ucmerced.edu`

2 **Computer Science and Engineering**
Washington University in St. Louis
`bmoseley@wustl.edu`

3 **Computer Science**
University of Pittsburgh
`kirk@cs.pitt.edu`

Abstract

We revisit the classical stochastic scheduling problem of nonpreemptively scheduling n jobs so as to minimize total completion time on m identical machines, $P \mid\mid \mathbb{E}\sum C_j$ in the standard 3-field scheduling notation. Previously it was only known how to obtain reasonable approximation if jobs sizes have low variability. However, distributions commonly arising in practice have high variability, and the upper bounds on the approximation ratio for the previous algorithms for such distributions can be even inverse-polynomial in the maximum possible job size. We start by showing that the natural list scheduling algorithm Shortest Expected Processing Time (SEPT) has a bad approximation ratio for high variability jobs. We observe that a simple randomized rounding of a natural linear programming relaxation is a $(1+\epsilon)$-machine $O(1)$-approximation assuming the number of machines is at least logarithmic in the number of jobs. Turning to the case of a modest number of machines, we develop a list scheduling algorithm that is $O(\log^2 n + m\log n)$-approximate. Our results together imply a $(1+\epsilon)$-machine $O(\log^2 n)$-approximation for an arbitrary number of machines. Intuitively our list scheduling algorithm finds an ordering that not only takes the expected size of a job into account, but also takes into account the probability that job will be big.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, Sequencing and scheduling

Keywords and phrases stochastic scheduling, completion time, approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.474

1 Introduction

Scheduling jobs on identical machines with the objective of minimizing total completion time is a well studied class of scheduling problems as well as being one of the most basic multiple machines scheduling settings. Basic versions of these problems and their varianst are reasonably well understood in both the online and offline settings. For example, if the processing times of the jobs are available to the scheduler, then list scheduling the jobs in increasing order of their sizes yields an optimal nonpreemptive schedule [4]. See [16]

* Supported in part by NSF grant CCF-1409130.
† Supported in part by NSF grants CCF-1115575, CNS-1253218, CCF-1421508, and an IBM Faculty Award.

for various approximation results. Unfortunately, in many systems, the scheduler may not know a priori the exact processing times of jobs. But often the scheduler may know a priori, from past invocations of the job, stochastic information about a job's size. Thus there is significant research on such stochastic scheduling problems (see for example [9]).

We revisit the classical stochastic scheduling problem of *non-preemptively* scheduling a collection $\mathcal{J}$ of n jobs so as to minimize the total expected completion time on m identical machines, $P \mid\mid \mathbb{E}\sum C_j$ in the standard 3-field scheduling notation. The algorithm and optimal solution are assumed to be nonanticipatory. In the stochastic setting, a nonanticipatory scheduler only knows a priori the probability distribution on the size P_j of each job j. If a machine is not running a job at a particular time, the nonanticipatory scheduler may optionally assign a job to that machine at that time; if it assigns a job, then the job must be run to completion, which is when the nonanticipatory scheduler finally learns the realized size of the job. Note that the scheduler can be dynamic in the sense that it can make scheduling decisions at time t based on the revealed processing times of jobs scheduled up to time t. If a job j is started at time S_j and is still being scheduled at time t then the scheduler only knows the job's size is at least $t - S_j$ and if the job j completes by time t, the scheduler knows its realized size.

In contrast to the worst-case offline and online settings, this problem is not well understood in the stochastic setting for general distributions. The most natural nonanticipatory algorithm is Shortest Expected Processing Time (SEPT), which always assigns a job with the minimum expected size to a machine that is free. SEPT is known to be an optimal nonanticipatory algorithm on a single machine [10]. As is standard in stochastic settings, approximation is relative to the optimal nonanticipatory algorithm. For identical machines, SEPT is optimal if job sizes are exponentially distributed or are stochastically comparable in pairs [15, 14]. However, for general distributions, even the question of whether SEPT has a reasonable approximation ratio independent of the variability of the jobs' processing times for identical machines was open.

We start by resolving this open question by showing that the approximation ratio of SEPT is $\Omega(n^{1/4})$. For full details see Section 2. It is useful to consider this instance as it also represents the types of instances that are hard to design and analyze algorithms for. In this instance most of the jobs are small with high probability. The remaining jobs are either of size zero, or are (very) big. The expected number of jobs that become big is approximately m. If the high variability jobs are run first (as SEPT does given the proper settings of the parameters) then there can be a large difference in the cost depending on whether m jobs become big, clogging up the machines for a long period of time, or whether only $m - 1$ jobs become big, leaving one machine to finish off the small jobs. This instance demonstrates that the problem is delicate/fragile in the sense that small changes in the input, or changes in the random realizations of sizes for a small number of jobs, can have a large impact on the objective.

This demonstrates that the main difficulty in these types of stochastic scheduling problems is handling high variability jobs. A standard research approach for problems, that are difficult when some parameter is large, is to seek algorithms that perform reasonably when this parameter is small. To the best of our knowledge, this is the approach taken by all previous research for these types of stochastic scheduling problems. For example, [8, 13] give algorithms for our problem $P \mid\mid \mathbb{E}\sum C_j$, and generalizations thereof, and show that they have approximation ratios that are roughly linear in the squared coefficient of variation, which is the variance divided by the square of the expectation. So these algorithms would provide reasonable approximation if the sizes of the jobs did not have high variability.

Unfortunately, the distributions that most commonly arise in practice, Zipf distributions with small α parameters, have high variability [3, 5, 1]. In a Zipf distribution the probability that a job has size s is proportional to $1/s^{\alpha}$. Zipf distributions with $\alpha \in (1, 2)$ have a squared coefficient of variation which is $\tilde{\Omega}(P^{\alpha-1})$ where P is the maximum possible job size. For example, for the common case that $\alpha \approx 2$, this gives approximation ratio approximately $O(P)$, which is quite weak as it is achieved by every algorithm that doesn't unnecessarily idle a processor.

The starting point for our research was to investigate whether one can obtain reasonable approximation when jobs may have high variability. It was clear a priori that success would require the development of new analysis techniques. The approaches in the stochastic scheduling literature all require low variability. The approaches in the deterministic scheduling literature are all essentially based on volume arguments, in which one bounds the total volume of work that will be processed by the algorithm before a specific job completes. Unfortunately, on instances such as the lower bound instance for SEPT, similar volume based arguments again only lead to approximate ratios depending on the squared coefficient of variation [8, 13]. Further the most natural candidate algorithms, like SEPT, all perform badly.

1.1 Our Results and Contributions

Our main result is a polynomial-time $(1+\epsilon)$-machine $O(\frac{1}{\epsilon^2}\log^2 n)$-approximation algorithm for any number of machines. Our analysis is broken into two parts depending on the number of machines as compared to the number of jobs. The most challenging case is when the number of machines is small. In Section 4 we develop a novel polynomial-time list scheduling algorithm LS. Here list scheduling means that the algorithm initially computes an ordered list of the jobs, and whenever a machine becomes free, the algorithm assigns the next job on the list to the available machine. In Section 5 we show the approximation ratio for LS is $O(\log^2 n + m \log n)$. When m is $O(\log n)$ this implies an $O(\log^2 n)$-approximation *without* resource augmentation. Adopting the viewpoint of the approximation algorithms community that a poly-log approximation is reasonable, albeit at the top end of reasonable, then LS is reasonable for a smallish/poly-log number of machines. Our main result is based on two, related, insights:

- We can construct a lower bound for optimum using two characteristics of the jobs:
  - The expected size of a job (so the same characteristic that SEPT uses), and
  - the probability that a job becomes big.
- We can construct a list using these same two parameters so that the list scheduling algorithm LS will never be too far off from our lower bound.

In most stochastic approximation literature, finding a good lower bound for the adaptive adversary is crucial for the analysis, and we believe our lower bound is worth further investigation for other stochastic scheduling problems. As mentioned before, prior to our work, the best known approximation ratio for arbitrary job size distributions was the maximum possible job size, even when there are only two machines. A more detailed overview can be found in Section 3.

We then consider the case that the number of machines is not small relative to n. In our lower bound instance for SEPT it is the case that the objective could be significantly affected by a small change in the amount of resources available. This is a clear signal that a resource augmentation analysis might be useful. If the number of machines is large, then allowing the algorithm some modest resource/machine augmentation seems reasonable.

We show in Section 6 that a simple randomized rounding of a natural linear programming relaxation is a $(1+\epsilon)$-machine $O(1)$-approximation provided that $m \geq \Omega((1/\epsilon^2)\log n)$; here the algorithm is allowed to use $(1+\epsilon)m$ machines and is compared against the optimal adaptive algorithm that can only use m machines. The analysis follows by showing, using standard concentration arguments, that if there are at least this many machines, then with high probability it is never that case that all machines are "clogged up".

Finally, we show in Section 2 that the approximation ratio of SEPT is $\Omega(n^{1/4})$.

1.2 Other Related Work

Most of the related results in the literature also hold for the more general problem where jobs have weights and the objective is the weighted sum of completion times. For a single machine, the algorithm WSEPT (running jobs with high weight to expected size ratio) is 2-approximate, and this approximation ratio is best possible [10]. Turning back to identical machines, it is known that WSEPT is asymptotically optimal; that is, the approximation tends to one as the number of jobs tends to infinity [14, 15]. [8] gives a list scheduling policy based on a linear programming relaxation where the approximation is linear in the squared coefficient of variation. It is known that this is the best approximation ratio possible if jobs must be irrevocably assigned to machines a priori [13]. This approach was extended to an online setting in [11], to allow the possibility of precedence constraints in [12], and to allow the possibility of related machines in [13]. The fact that the approximation results also hold for weighted completion time is in part explained by the fact that they are based on linear programming formulations, which easily incorporate weights. [7] gives a combinatorial algorithm for the setting that job arrive online in a list, and must be assigned to machines as they arrive, and again show an approximation ratio that is linear in the square coefficient of variation. For deterministic sizes, there is a polynomial time approximation scheme [2].

2 Lower Bound for SEPT

▶ **Theorem 1.** *The algorithm SEPT rule has an approximation ratio $\Omega(n^{1/4})$.*

Proof. We first describe the example. There are m machines where m is greater than a sufficiently large constant such that $\exp(-m/16) < 1/m^8$. There are two types of jobs.

- Type-1: There are $2m^2$ jobs, and each job has size 1 with probability $1/m$, and 0 otherwise.
- Type-2: There are $m^4/4$ jobs, and each job has size m^2 with probability $1/m^3$, and 0 otherwise.

Note that all jobs have the same expected size. Therefore, SEPT rule can schedule jobs in arbitrarily order. Suppose it first schedules Type-1 jobs. By applying a standard Chernoff bound (for example, Theorem 20 with $\mu = 2m$ and $\delta = 1/2$), with a probability of at least $1 - \exp(-m/4) \geq 1 - 1/m^8$, at least m jobs will have size 1, thereby delaying all Type-2 jobs after time 1. Hence the total completion time of SEPT will be $\Omega(m^4)$ in expectation.

In contrast, suppose the adversary schedule Type-2 jobs first. The adversary can learn at an infinitesimally small time, say $1/m^4$, the empty machines. Let $\mathcal{E}$ denote the event that the number of such machines is at least $m/2$. Observe that $\Pr[\mathcal{E}] \geq 1 - \exp(-m/16) \geq 1 - 1/m^8$ by Theorem 20 with $\mu = m/4$ and $\delta = 1$. Since the total job size is at most $O(m^6)$ in all cases, and the total number of jobs is $O(m^4)$, the expected total completion time of the adversary in the event of $\neg\mathcal{E}$ is at most $O(m^6 \cdot m^4 \cdot \frac{1}{m^8}) = O(m^2)$. Now consider the case that the event $\mathcal{E}$ occurs. The adversary distributes Type-1 jobs evenly on the empty machines it discovered at an infinitesimally small time. Since there are at least $m/2$ empty machines, and there are $2m^2$ Type-1 jobs to schedule, no machine is assigned more than $O(m)$ Type-1 jobs. Hence, even in the worst case where all Type-1 jobs have size 1, every Type-1 job is completed by time $O(m)$. In sum, we have shown the adversary's total completion time is $O(m^3)$ in expectation. Since $n = \Theta(m^4)$, the gap follows. ◀

3 Intuitive Overview of the Design and Analysis of the Algorithm LS

We now give an informal overview of the intuition behind the intertwined design and analysis of the algorithm LS (occasionally oversimplifying some issues).

The initial starting point is the way in which we estimate the total completion time. Let τ_k be the time that the $\frac{n}{2^k}$th to last job completes. Let G_i be those jobs that complete between τ_{k-1} and τ_k. By rounding down the completion times in G_k to τ_{k-1} we obtain an estimate $\sum_k \tau_k n/2^k$ of the total competition time that is accurate within a constant factor. To see this note that the decrease in total completion time for the $n/2^k$ jobs in G_k can be charged to a $[\tau_{k-1}, \tau_k]$ portion of the competition times of the $n/2^k$ jobs that complete after τ_k. It will be convenient to consider job starting times, instead of job competition times. This is, without any real loss of generally, as the sum of the starting times differs from the sum of the completion times by only the sum of the processing times, and the expected sum of the processing times is the same for all algorithms. Then intuitively our algorithm needs to determine the jobs in G_k, which roughly one would expect should be the $n/2^k$ jobs in positions $[n - n/2^{k-1}, n - n/2^k]$ in the algorithm's list, so that these jobs are all likely to start by a deadline τ_k that is as early as possible. Let us for the moment assume that the algorithm knows the "correct" value of the deadline τ_k. The algorithm must then solve the following informal subproblem:

Key Subproblem (k, τ): Given a cardinality k and a deadline τ, which set $E_{k,\tau}$ of $n/2^k$ jobs should be excluded so as to maximize the probability that the remaining set $A_{k,\tau}$ of $n - n/2^k$ jobs can all start by time τ?

We give an algorithm SPLIT for selecting $E_{k,\tau}$. We then show that if SPLIT isn't likely to start all jobs in $A_{k,\tau}$ by deadline τ_k then the optimal adaptive algorithm likely has a comparable number of jobs unfinished by deadline τ/Δ. Here Δ is a parameter that we will eventually set to $m + \log n$. This relaxed deadline contributes a Δ factor to the approximation ratio. It will be convenient to call a job small if it has size at most τ/Δ, and call a job big otherwise.

Our algorithm SPLIT certainly should exclude those jobs with high expected processing time. Here all expected processing times will be conditioned on the fact that the job is small, since all big jobs are equally bad for the optimal adaptive algorithm. Our lower bound for SEPT suggests that we should also exclude those jobs that are most likely to be big as these jobs are the ones most likely to clog up the machines. The algorithm SPLIT splits the $n/2^k$ exclusions in $E_{k,\tau}$ equally between the $n/2^{k+1}$ jobs with the highest expected processing times, the $n/2^{k+1}$ jobs with highest probability of being big. The algorithm SPLIT then list schedules the jobs in $A_{k,\tau}$ in an arbitrary order.

The key part of our analysis of SPLIT, and of almost all algorithm analyses of such stochastic problems, is lower bounding optimal. Here we are able to lower bound optimal using the same exact two job characteristics, the probability that a job is big and the expected size, that SPLIT uses. The analysis is split into two cases. The first case is when the aggregate expected size jobs in $A_{k,\tau}$ is at least $\tau/2$. In this case, there is sufficient probability mass on small sizes (this is where we need that Δ is sufficiently large) so that a standard lower tail bound can be used to show with high probability the aggregate size of the small jobs is close to expectation. The result is then established using the fact that the aggregate sizes divided by m is a lower bound for optimal. The second case is when the aggregate expected size of jobs in $A_{k,\tau}$ is at most $\tau/2$. Then with high probability the small jobs from $A_{k,\tau}$ don't have sufficient aggregate size to keep one machine busy until time τ. Thus if SPLIT has all machines busy at time τ, then SPLIT must have seen at least m big jobs. But as SPLIT excluded the jobs most likely to be big, the optimal algorithm also likely saw m big jobs, and thus still have all machines busy at time τ/Δ.

It is natural to try to extend the algorithm SPLIT to create a list for LS by first picking in arbitrary order the jobs in $\mathcal{J} - E_{1,\tau_1}$, which are intuitively the $n/2$ jobs most likely startable by τ_1, then picking in arbitrary order the jobs in $\mathcal{J} - E_{1,\tau_1} - E_{2,\tau_2}$, which are intuitively the $n/4$ jobs in the set of $3n/4$ jobs most likely startable by τ_2 that were not previously picked, and on the phase k, picking in arbitrary order the jobs in $\mathcal{J} - \cup_{i \le k} E_{i,\tau_i} = \cap_{i \le k} A_{i,\tau_i}$. There are two difficulties with this approach. We end up surmounting both difficulties in a similar fashion.

The first difficulty is that we do know know a priori the "right" values for the τ_k's. Using standard transformations we can without loss of generality assume that the range of possible times is polynomially bounded, and that we can restrict our attention to τ_k being one of the logarithmically many times that are an integer power of two. We then modify our solution $E_{k,\tau}$ to the subproblems (k, τ) by excluding $n/(2^k \log n)$ jobs, instead of $n/2^k$. Again the exclusions in $E_{k,\tau}$ are split equally between jobs that are have the largest expected sizes, and those that are most likely be be big. Let the excluded set $E_k = \cup_i E_{k,2^i}$ be the union of the excluded sets for various possible values of τ_k. We could then construct our list by first picking in arbitrary order the jobs in $\mathcal{J} - E_1$, then picking in arbitrary order the jobs in $\mathcal{J} - E_1 - E_2$, and on the phase k, picking in arbitrary order the jobs in $\mathcal{J} - \cup_{i \le k} E_i$. Because E is the union of essentially all possible E_{k,τ_k}'s, we know that we are excluding the excluded jobs from the subproblem corresponding to the "right" τ_k. The redefinition of the $E_{k,\tau}$'s costs a log factor in our approximation ratio.

The remaining problem with this ordering is the possibility that the excluded sets may not be consistent. For example, a job j such that $j \notin E_1$ and $j \in E_2$ is an inconsistency as it is not possible to schedule j after τ_2 and before τ_1. To solve this we let the excluded set $E'_k = \cup_{i \ge k} E_i$ be the union of the previously defined excluded sets for later times. The algorithm LS then constructs its list by first picking in arbitrary order the jobs in $\mathcal{J} - E'_1$, then picking in arbitrary order the jobs in $\mathcal{J} - E'_1 - E'_2$, and on phase k, picking in arbitrary order the jobs in $\mathcal{J} - \cup_{i \le k} E'_i$. Because a job is excluded in E'_k if it is in any later excluded set E_i, we know that there will be no inconsistencies. This also guarantees the hereditary condition that $E'_{k+1} \subseteq E'_k$, which means that earlier scheduled jobs are not in later excluded sets. Because the size of the sets E_k are geometrically decreasing, these additional exclusions don't change the size of the excluded sets by more than a constant factor.

4 Algorithm LS

In this section we more formally describe the list scheduling algorithm LS, and introduce some notation that will be needed in the analysis. To aid in our later analysis, we describe the algorithm in terms of the complements of the excluded sets discussed in the last section. Recall $A_{k,\ell}$ is the complement of the excluded set $E_{k,\ell}$, and taking the complement of the union is equivalent to taking the intersection of the complements.

We assume that the number of machines $m \ge 2$. We show in Lemma 2 that we can assume without loss of generality that all possible job sizes are in the range $[1, n^{10}]$. Intuitively, if a job is sufficiently small, then it can change the total completion time objective by very little. The upper bound then follows by noting that at least two jobs have to become big to clog up the machines for a long period of time, and the probability that this happens is quite small. The proof of Lemma 2 can be found in Section 5.1. Let Δ be the smallest integer greater than $1000 \max\{m, \log n\}$ that is a power of two.

▶ **Lemma 2.** *Suppose we have a nonanticipatory fixed-priority algorithm that is α-approximate for the simplified instances of n jobs where every job j is instantiated to a size between 1 and n^{10}, i.e. $1 \leq P_j \leq n^{10}$ for all jobs j. Then one can get a $O(\alpha)$-approximation for an arbitrary instance consisting of n jobs.*

We now introduce our algorithm. For intuition guiding the development of the algorithm, we refer the reader to the overview of the algorithm given in Section 3.

```
Algorithm LS ($k \in [\log n]$, and $\ell \in [12 \log n]$)
1. For each pair of $k, \ell$, compute $A_{k,\ell}$ as follows. Let $\tau := 2^\ell$. Let $A_{k,\ell}$ be the intersection of
   the following two sets $A^h_{k,\ell}$ and $A^v_{k,\ell}$:
    - Let $A^h_{k,\ell}$ be the $n - \frac{n}{24 \cdot 2^k \cdot \log n}$ jobs with the smallest $h_{j,\ell}$ values
      where $h_{j,\ell} := \Pr[P_j \geq 2^\ell/\Delta]$.
    - Let $A^v_{k,\ell}$ be the $n - \frac{n}{24 \cdot 2^k \cdot \log n}$ jobs with the smallest $v_{j,\ell}$ values
      where $v_{j,\ell} := \sum_{s < 2^\ell/\Delta} s \cdot \Pr[P_j = s]$.
2. Define $A_k := \bigcap_{\ell \in [12 \log n]} A_{k,\ell}$.
3. Define $A'_k := \bigcap_{k \leq k' \leq \log n} A_{k'}$.
4. Consider $k$ in increasing order. For each $k$, schedule jobs in $A'_k \setminus A'_{k-1}$ in an arbitrary but
   fixed order assigning jobs to any available machine.
```

The following Lemmas are immediate from the algorithm's description, and will be useful for our analysis.

▶ **Lemma 3.** *It holds that*
- *For all k, ℓ, $n - \frac{n}{12 \cdot 2^k \log n} \leq |A_{k,\ell}| \leq n - \frac{n}{24 \cdot 2^k \log n}$.*
- *For all k, $n - \frac{n}{2^k} \leq |A_k| \leq n - \frac{n}{24 \cdot 2^k \log n}$.*
- *For all k, $n - \frac{n}{2^{k-1}} \leq |A'_k| \leq n - \frac{n}{24 \cdot 2^k \log n}$.*
- *For all k, $A'_k \subseteq A'_{k+1}$.*

5 Analysis

This section is devoted to proving Theorem 4.

▶ **Theorem 4.** *The algorithm LS is $O(\log^2 n + m \log n)$-approximate for scheduling n stochastic jobs non-preemptively on m identical machines with the objective of minimizing the total completion time in expectation.*

Our analysis is based on Lemma 8 that states how the solution to the subproblem parameterized by k, ℓ can be charged to the adversary's cost. That is, we will show if the algorithm cannot start all jobs in A_k, which is a subset of $A_{k,\ell}$, by a deadline $\tau = 2^\ell$, thereby delaying $n - |A_k|$ jobs after τ, then the adversary is more likely to delay a comparable number of jobs after time τ/Δ.

To formally state Lemma 8, we need to introduce some notation. Let $L(J')$ denote the *earliest* time when a machine becomes available after starting *all* jobs in J'; here the associated algorithm is implicitly given. Note that $L(J')$ depends on the realized processing time of jobs. Let A^*_k denote the $n - \frac{n}{24 \cdot 2^k \log n}$ jobs the adversary starts the earliest; recall that $|A^h_{k,\ell}| = |A^v_{k,\ell}| = n - \frac{n}{24 \cdot 2^k \log n}$, and $|A_{k,\ell}| \geq n - \frac{n}{12 \cdot 2^k \log n}$. Note that A^*_k could be stochastic while $A_{k,\ell}$ is deterministic.

The quantity $L^W(J')$ is defined similar to $L(J)$, but for the the worst *anticipatory* list scheduling algorithm W. So W knows the jobs sizes and it maximizes the earliest time when a machine becomes available after starting all the jobs in J'. Obviously LS performs better than the worst anticipatory algorithm. Lemma 5, Lemma 6, and Lemma 7 state straightforward properties of these times. The proof of Theorem 4 follows by application of Lemma 8, Lemma 5, Lemma 6 and basic algebra. The cornerstone of the analysis is the proof of Lemma 8, which we postpone until the end of the section.

▶ **Lemma 5.** *The function $L^W(\cdot)$ is monotone, i.e., for any realization of job sizes and any two sets of jobs, $J' \subseteq J$, we have $L^W(J') \leq L^W(J)$.*

▶ **Lemma 6.** *For all k and ℓ and for any realization of job sizes, $L(A'_k) \leq L^W(A'_k) \leq L^W(A_k) \leq L^W(A_{k,\ell})$.*

Proof. This follows from the the fact that $A'_k \subseteq A_k \subseteq A_{k,\ell}$, and that LS is a list scheduling algorithm. ◀

▶ **Lemma 7.** *For all k, $L(A^*_k) \leq L(A^*_{k+1})$.*

Proof. By definition of A^*_k, we know that $A^*_k \subseteq A^*_{k+1}$. Then, the lemma is immediate since the earlist time when a machine becomes available after the optimal scheduler starts all jobs in A^*_k can be only smaller than the analgously defined time after the same optimal scheduler starts all jobs in A^*_{k+1}. ◀

We now formally state our key lemma.

▶ **Lemma 8.** *For all k, we have $\Pr[L^W(A_k) \geq 2^\ell] \leq \Pr[L(A^*_k) \geq 2^\ell/\Delta] + O(\frac{1}{n^{12}})$.*

Before proving Lemma 8, we show how it implies Theorem 4.

Proof of Theorem 4. Since all jobs have sizes at most n^{10}, the maximum total completion time can be at most n^{12}. Hence we will proceed with our analysis ignoring the small additive term in the right-hand-side of Lemma 8 since it will add only 1 to the total completion time in expectation, and all jobs have sizes at least 1. Note that it suffices to bound $\mathbb{E}\sum_j S_j$ where S_j is j's starting time. This is because the algorithm's cost is $\sum_j S_j$ plus $\sum_j P_j$, and $\mathbb{E}\sum_j P_j$ is a clear lower bound to the adversary as we can observe in Lemma 15. We let $\mathbf{1}[\mathcal{E}]$ be an indicator variable that is 1 if the event $\mathcal{E}$ occurs and 0 otherwise, for some event $\mathcal{E}$.

$$\begin{aligned}
\sum_j S_j &\leq \sum_j \sum_{\ell \geq 0} 2^{\ell+1} \cdot \mathbf{1}[S_j \geq 2^\ell] \leq \sum_j \sum_{\ell \geq \log \Delta} 2^{\ell+1} \cdot \mathbf{1}[S_j \geq 2^\ell] + O(\Delta n) \\
&= \sum_{k \geq 1} \sum_{j \in A'_k \setminus A'_{k-1}} \sum_{\ell \geq \log \Delta} 2^{\ell+1} \cdot \mathbf{1}[S_j \geq 2^\ell] + O(\Delta n) \\
&\leq \sum_{k \geq 1} |A'_k \setminus A'_{k-1}| \cdot \sum_{\ell \geq \log \Delta} 2^{\ell+1} \cdot \mathbf{1}[L(A'_k) \geq 2^\ell] + O(\Delta n) \\
&\leq \sum_{k \geq 1} O(\frac{n}{2^k}) \cdot \sum_{\ell \geq \log \Delta} 2^\ell \cdot \mathbf{1}[L(A'_k) \geq 2^\ell] + O(\Delta n) \qquad \text{[Lemma 3]}
\end{aligned}$$

The first inequality follows since for some ℓ, $2^\ell \le S_j < 2^{\ell+1}$. By taking the expectation on both sides, we have

$$\begin{aligned}
\mathbb{E}\sum_j S_j &\le \sum_{k\ge1} O(\frac{n}{2^k}) \sum_{\ell\ge\log\Delta} 2^\ell \cdot \Pr[L(A'_k) \ge 2^\ell] + O(\Delta n)\\
&\le \sum_{k\ge1} O(\frac{n}{2^k}) \sum_{\ell\ge\log\Delta} 2^\ell \cdot \Pr[L^W(A_k) \ge 2^\ell] + O(\Delta n)\\
&\le \sum_{k\ge1} O(\frac{n}{2^k}) \sum_{\ell\ge\log\Delta} 2^\ell \cdot \Pr[L(A^*_k) \ge 2^\ell/\Delta] + O(\Delta n)\\
&\le \sum_{k\ge1} O(\frac{n}{2^k}) \sum_{\ell\ge1} \Delta\cdot 2^\ell \cdot \Pr[L(A^*_k) \ge 2^\ell] + O(\Delta n)\\
&\le 2\sum_{k\ge2} O(\frac{n}{2^k}) \sum_{\ell\ge1} \Delta\cdot 2^\ell \cdot \Pr[L(A^*_k) \ge 2^\ell] + O(\Delta n)
\end{aligned}$$

The second and third inequalities are due to Lemma 6 and Lemma 8, respectively. In the last inequality, we used the fact $L(A^*_2) \ge L(A^*_1)$, which follows from Lemma 7. We can charge $O(\Delta n)$ to the optimal cost since the nonanticipatory optimal solution must have total expected completion time at least $\sum_j \mathbb{E}P_j \ge n$, and our goal is to show a $O(\Delta \log n)$-approximation.

To upper bound the remaining terms, we lower bound OPT as follows.

$$\mathrm{OPT} \;\ge\; \sum_{k\ge2} |A^*_{k+1} \setminus A^*_k| \cdot L(A^*_k) \ge \Theta(1)\cdot \sum_{k\ge2} O(\frac{n}{2^k \log n}) \sum_\ell 2^\ell \cdot \mathbf{1}[L(A^*_k) \ge 2^\ell]$$

The first inequality follows since no job in $A^*_{k+1} \setminus A^*_k$ starts before time $L(A^*_k)$. By taking the expectation on this equation and combining it with the above equation, we conclude that our algorithm is $O(\Delta \log n)$-approximation, deriving Theorem 4. ◀

We are now ready to prove the key lemma.

Proof of Lemma 8. We will be concerned with the probability that the optimal adaptive algorithm cannot start $n' := n - \frac{n}{24\cdot 2^k \log n}$ jobs before time τ/Δ. We will say that a job j is big if its realized size $P_j \ge \tau/\Delta$, and say the job is small otherwise. We consider two cases depending on the volume of small jobs. Fix k and ℓ. For notational simplicity, let $\tau := 2^\ell$.

Case A: $\sum_{j\in A_k} v_j \ge \tau/2$: We first show in Lemma 9 and Lemma 10 that the aggregate size of the first n' small jobs that the optimal adaptive algorithms starts is likely at least $\tau/4$. Lemma 11 then shows that this is sufficient volume so that the optimal adaptive algorithm cannot start n' jobs before time τ/Δ.

To make this formal, we define a sequence of random variables $\{X_q\}$ where X_q refers to the "small" size of the q_{th} earliest job that is started by the optimal adaptive algorithm – X_q is set to P_j if $P_j < \tau/\Delta$, otherwise 0. Also let Y_q be the 0-1 random variable that becomes 1 if the q_{th} earliest job that the adversary starts becomes large, otherwise 0.

▶ **Lemma 9.** *$\sum_{q\in[n']} \mathbb{E}[X_q] \ge \tau/2$.*

Proof. This observation immediately follows from the fact that $A_k \subseteq A_{k,\ell} \subseteq A^v_{k,\ell}$, and $A^v_{k,\ell}$ consists of n' jobs with the smallest $v_{j,\ell} := \sum_{s<\tau/\Delta} s\cdot \Pr[P_j = s]$ values. ◀

▶ **Lemma 10.** *$\Pr[\sum_{q\in[n']} X_q \le \tau/4] \le \frac{1}{n^{12}}$.*

Proof. To apply Theorem 20, we scale down X_q by τ/Δ. Recall that $X_q \leq \tau/\Delta$ and $\Delta \geq 1000\max\{m, \log n\}$. By using Theorem 20 with $\mu \geq \frac{\tau}{2}/\frac{\tau}{\Delta} \geq \frac{\Delta}{2}$ and $\epsilon \geq 1/2$, we derive that the probability is at most $\exp(-\epsilon^2\mu/2) \leq \exp(-\Delta/16) \leq 1/n^{12}$. ◀

▶ **Lemma 11.** *If $\sum_{q\in[n']} X_q \geq \tau/4$, then $L(A^*_k) \geq \tau/\Delta$.*

Proof. For the sake of contradiction, suppose that $L(A^*_k) \leq \tau/\Delta$. Since each small job has size at most τ/Δ, a machine can be busy until time $2\tau/\Delta$ due to small jobs that are started by time τ/Δ. Hence it must be the case that $\sum_{q\in[n']} X_q \leq m \cdot 2\tau/\Delta < \tau/4$, which is a contradiction. ◀

Case B: $\sum_{j\in A_k} v_j \leq \tau/2$: We show in Lemma 12 that the algorithm W likely didn't start enough small jobs to even fill up one machine until time τ. Lemma 13 then shows that it must be the case that the algorithm W then must have started m big jobs before time τ. Lemma 14 then shows that the optimal adaptive algorithm must have started m big jobs before time τ/Δ and before starting n' jobs. Thus the optimal adaptive likely cannot finish n' jobs before time τ/Δ.

To make this formal, let X'_j denote job j's small size. That is, X'_j is set to P_j if $P_j \leq \tau/\Delta$, otherwise 0. Let Y'_j be the 0-1 random variable that becomes 1 if job j becomes large, otherwise 0. The differnece between X_q and X'_j (likewise between Y_q and Y'_j) is that X'_j is concerned with the size of a fixed job j while X_q is concerned with the size of the q_{th} earliest job the adversary starts – the q_{th} job cah change since the adversary is not necessarily a fixed-priority scheduler. By applying a concentration inequality, we can show,

▶ **Lemma 12.** $\Pr[\sum_{j\in A_k} X'_j \geq \tau] \leq 1/n^{12}$.

Proof. We scale down X_q by τ/Δ. By applying Theorem 20 with $\mu \leq \frac{\tau}{2}/\frac{\tau}{\Delta} \leq \frac{\Delta}{2}$ and $\epsilon = \frac{\tau}{\tau/\Delta}/\mu - 1 = \frac{\Delta}{\mu} - 1 \geq \frac{\Delta}{2\mu}$, we upper bound the probability by $\exp\left(-\frac{(\Delta/(2\mu))^2\mu}{2(1+(\frac{\Delta}{2\mu}-1)/3)}\right) \leq \exp\left(-\frac{(\Delta/(2\mu))^2\mu}{\Delta/\mu}\right) = \exp(-\Delta/4) \leq 1/n^{12}$. ◀

▶ **Lemma 13.** *If $\sum_{j\in A_k} X'_j < \tau$ and $L^W(A_k) \geq \tau$, then there must be at least m jobs in A_k with realized sizes at least τ/Δ.*

Proof. For the sake of contradiction, suppose there are less than m big jobs. Then there must exist a machine that is busy until time τ scheduling small jobs, which is a contradiction to the condition $\sum_{j\in A_k} X'_j < \tau$. ◀

▶ **Lemma 14.** $\Pr[\sum_{q\in[n']} Y_q \geq m] \geq \Pr[\sum_{j\in A_{k,\ell'}} Y'_j \geq m] \geq \Pr[\sum_{j\in A_k} Y'_j \geq m]$.

Proof. Notice that A_k is a subset of $A^h_{k,\ell'}$ with $\ell' = \log_2(\tau/\Delta)$. Since $A^h_{k,\ell'}$ consists of n' jobs with the smallest $h_{j,\ell'} := \Pr[P_j \geq 2^{\ell'} = \tau/\Delta]$ values, the probability that the adversary finds at least m big jobs while scheduling the first n' jobs it starts must be as large as the probability that our algorithm finds m big jobs while scheduling jobs in A_k. ◀

This concludes the proof of Lemma 8. ◀

5.1 Proof of the Simplifying Assumption (Lemma 2)

In this section we prove Lemma 2. Due to the space constraints, we defer the proof of Lemma 16, 17, and 18 to the full version of this paper.

We begin with the following simple lower bound on the adversary.

▶ **Lemma 15.** $\mathbb{E}\ \mathrm{OPT} \geq \mathbb{E}\sum_j P_j$.

Motivated by this lower bound, from now on we assume w.l.o.g. that $\mathbb{E}\sum_j P_j = 1$ by scaling jobs sizes uniformly.

We now show that one can assume that every job is instantiated to a size at least $1/n^2$. Let $\mathcal{I}^1$ be the instance obtained from the original instance $\mathcal{I}^0 := \mathcal{I}$ by replacing P_j with $P_j + 1/n^2$. We show that the optimal completion time can only double in the transition from $\mathcal{I}^0$ to $\mathcal{I}^1$. Let $\mathrm{OPT}(\mathcal{I})$ denote the adversary or its objective on instance $\mathcal{I}$.

▶ **Lemma 16.** $\mathbb{E}\ \mathrm{OPT}(\mathcal{I}^1) \leq 2 \cdot \mathbb{E}\ \mathrm{OPT}(\mathcal{I}^0)$.

▶ **Lemma 17.** *Given an algorithm A^1 for $\mathcal{I}^1$, one can derive an algorithm A^0 for $\mathcal{I}^0$ with the same expected total completion time or smaller.*

Hence assuming all jobs have sizes at least $1/n^2$ only loses factor 2 in the approximation ratio. Now we argue that if a job is instantiated to have a very large size, we can ignore such a bad case since it contributes to the algorithm's cost very little. To simplify our argument, we will assume that our algorithm is the *worst anticipatory fixed-priority* algorithm. That is, the worst algorithm does the following: it observes each job's realized size, and finds the worst ordering between jobs in J such that assigning each job to the earliest available machine according to the ordering maximizes the total completion time. If we can show that the event that there is a job that has a huge size can contribute to the expected total completion time by only a fraction of $\mathbb{E}\sum_j P_j = 1$, then we will be able to ignore such an event. Let BAD denote the worst algorithm we will consider, or its total completion time depending on the context. In the following, $\mathrm{BAD}(w)$ denote BAD's objective when an outcome (realization of job sizes) w occurs. Intuitively, BAD can have a huge total completion time only when at least two jobs are realized to have huge sizes, which can happen with a very small probability. This is where we use the fact $m \geq 2$.

▶ **Lemma 18.** *Let $\mathcal{E}$ be the event that* $\max_j P_j \geq n^8$. *Then* $\sum_{w \in \mathcal{E}} \mathrm{BAD}(w)\Pr[w] \leq O(1)\ \mathbb{E}\mathrm{OPT}$.

6 LP-based Algorithm with Machine Augmentation

▶ **Theorem 19.** *Suppose $m \geq \frac{36}{\epsilon^2}\log n$. Then there is a polynomial time $O(1)$-approximation that schedules n stochastic jobs non-preemptively on $(1+\epsilon)m$ identical machines, when compared against the adversary using m machines, with the goal of minimizing the total completion time in expectation.*

Proof. Let $x_{i,\tau}$ be the probability that the adversary schedules job i at time τ. Let $q_{i,d}$ denote the probability that job i has size no smaller than d. The following LP relaxation is due to [13].

$$\min \sum_{i,\tau} \tau \cdot x_{i,\tau} \quad s.t. \quad \sum_{\tau \geq 0} x_{i,\tau} \geq 1 \quad \forall i; \quad \sum_{i,\tau \leq t} q_{i,t-\tau} \cdot x_{i,\tau} \leq m \quad \forall t \geq 0; \quad x_{i,\tau} \geq 0 \quad \forall i, \tau \geq 0$$

The objective is the total expected starting time of all jobs. The first constraints say that each job must be scheduled. The second constraints ensure that at any time at most

m machines are used. These are valid constraints due to the nonanticipatory nature of the adversary: job i's size is realized independent of when it is started.

We now show a simple algorithm using $m' = (1+\epsilon)m$ machines. Since $\{x_{i,\tau}\}_\tau$ is a distribution over job i's starting times, we naturally set i's starting time S_i to τ with probability $x_{i,\tau}$. We order jobs in increasing order of S_i, and schedule job i on any available machine at time t – we will show that this is always possible with a high probability. If not possible, we switch to an arbitrary fixed-priority algorithm.

We first claim that we only need to consider times $1 \le t \le n^5$ in the LP. Unfortunately, we cannot use Lemma 2 here since this LP-based algrotihm is not a fixed-priority algorithm. However, we can use most of the simplifying argument in Section 5.1 with small tweak. We can show that one can assume without loss of generality that all jobs have sizes at least 1 and $n \le \mathbb{E}\sum_j P_j \le n^2$. Then, we know that any non-idle algorithm has total completion time at most $n^2 \mathbb{E} \max_j P_j \le n^4$ in expectation. This implies no optimal LP solution schedules a job by more than $1/n$ after time n^5, i.e. $\sum_{t' \ge n^5} x_{i,t'} \le 1/n$ for all i. This is because the second constriant of the LP is trivially satisfied for any optimal solution for all times after n^5. Hence we only need to consider times $1 \le t \le n^5$. This proves the LP has a size polynomial in n.

We will show that for each $1 \le t \le n^5$, the number of jobs whose intervals $(S_i, S_i + P_i)$ intersect time t is at least m' with a probability of at most $1/n^{23}$; let B_t refer to the bad event with respect to time t. To show $\Pr[B_t] \le 1/n^{23}$ fix a time $t \in [0, n^5]$. Observe that the probability that job i's interval intersects time t is $\sum_{i,\tau \le t} q_{i,t-\tau} \cdot x_{i,\tau}$, and let X_i is the 0-1 random variable that becomes 1 if such an event happens. By the second constraints of the LP, we have $\mathbb{E}\sum_i X_i \le m$. By the applying Bernstein inequalities (Theorem 21) with $\Delta = m' - m$, $b = 1$, and $V \le m$, we can upper bound the probability by $\exp(-\frac{\Delta^2}{2V + 2b\Delta/3}) \le \exp(-\frac{\epsilon^2 m}{2+2\epsilon/3}) \le \frac{1}{n^{12}}$. when $m \ge \frac{36}{\epsilon^2}\log n$ and $\epsilon \le 1$.

Now consider a fixed job i. The probability job i can be started at time S_i as suggsted by the LP is at least $1 - 1/n^{18}$ via a simple union bound over all times between 1 and n^5. If it is the case, we can charge i's starting time to the LP cost. Otherwise, we can still charge i's expected starting time when it starts before time n^{10} to $\mathbb{E}P_i \ge 1$. Now let $\mathcal{E}(q)$ denote the event that job i starts between time n^q and n^{q+1}. Note that for event $\mathcal{E}(q)$ to happen, there must be at least m jobs that have size at least n^{q-2} blocking all m machines. Hence $\Pr[\mathcal{E}(q)] \le \binom{n}{m} \cdot (\frac{1}{n^{q-4}})^m \le \frac{1}{n^{m(q-5)}}$; here Markov inequliay was used with $\mathbb{E}P_i \le n^2$. Hence the expected starting time of job i when it is at least n^{10} is at most $\sum_{q \ge 10} n^{q+1} \cdot \Pr[\mathcal{E}(q)] \le o(1)$ when $m \ge 3$. Again, we can charge this to $\mathbb{E}P_i \ge 1$. ◀

7 Concentration Inequalities

▶ **Theorem 20** ([6]). *Let the random variables $X_1, X_2, ..., X_n$ be independent, with $0 \le X_i \le 1$ for each i. Let $S_n = \sum X_i$, let $\mu = \mathbb{E}(S_n)$. Then, any $\delta > 0$, $\Pr[S_n \ge (1+\delta)\mu] \le \exp(-\frac{\delta^2 \mu}{2(1+\delta/3)})$ and $\Pr[S_n \le (1-\delta)\mu] \le \exp(-\frac{1}{2}\delta^2\mu)$.*

▶ **Theorem 21** ([6]). *Let $X_1, X_2, ..., X_n$ be n independent random variables such that for all $i \in [n]$, $X_i \le b$. Let $Y = \sum_{i=i}^n X_i$, $\mu := \mathbb{E}[Y]$, and $V := \mathrm{Var}[Y]$. Then it follow that*

$$\Pr[Y - \mu \ge \Delta] \le \exp(-\Delta^2/(2V(1 + (b\Delta/3V)))).$$

Acknowledgements We thank Marc Uetz for bringing this problem to our attention, for helpful discussions through the research process, and for his assistance during the writing process.

References

1 L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.

2 Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *FOCS*, pages 32–44, 1999.

3 David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010.

4 W. Horn. Minimizing average flowtime with parallel machines. *Operations Research*, 21:846– 847, 2006.

5 Blachander Krishnamurthy and Jennifer Wexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

6 Colin McDiarmid. Concentration. In Michel Habib, Colin McDiarmid, Jorge Ramirez-Alfonsin, and Bruce Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16 of *Algorithms and Combinatorics*, pages 195–248. Springer Berlin Heidelberg, 1998.

7 N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.

8 R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46:924–942, 1999.

9 Michael Pinedo. *Scheduling Theory, Algorithms, and Systems*. Springer, 2008.

10 M. H. Rothkopf. Scheduling with random service times. *Management Science*, 12:703–713, 1966.

11 A. S. Schulz. Stochastic online scheduling revisited. In B. Yang, D.-Z. Du, and C. Wang, editors, *Combinatorial Optimization and Applications*, volume 5165 of *Lecture Notes in Computer Science*, pages 448–457. Springer, 2008.

12 M. Skutella and M. Uetz. Stochastic machine scheduling with precedence constraints. *SIAM Journal on Computing*, 34:788–802, 2005.

13 Martin Skutella, Maxim Sviridenko, and Marc Uetz. Stochastic scheduling on unrelated machines. In *STACS*, pages 639–650, 2014.

14 Gideon Weiss. Approximation results in parallel machines stochastic scheduling. *Annals of Operations Research*, 26:195–242, 1990.

15 Gideon Weiss. Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. *Mathematics of Operations Research*, 17:255–270, 1992.

16 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.