An Efficient PTAS for Stochastic Load Balancing with Poisson Jobs

Anindya De
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
anindyad@cis.upenn.edu

Sanjeev Khanna
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
sanjeev@cis.upenn.edu

Huan Li
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
huanli@seas.upenn.edu

Hesam Nikpey
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
hesam@seas.upenn.edu

Abstract

We give the first polynomial-time approximation scheme (PTAS) for the stochastic load balancing problem when the job sizes follow Poisson distributions. This improves upon the 2-approximation algorithm due to Goel and Indyk (FOCS'99). Moreover, our approximation scheme is an efficient PTAS that has a running time double exponential in $1/\epsilon$ but nearly-linear in n, where n is the number of jobs and ϵ is the target error. Previously, a PTAS (not efficient) was only known for jobs that obey exponential distributions (Goel and Indyk, FOCS'99).

Our algorithm relies on several probabilistic ingredients including some (seemingly) new results on scaling and the so-called "focusing effect" of maximum of Poisson random variables which might be of independent interest.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Approximation algorithms analysis; Theory of computation → Scheduling algorithms

Keywords and phrases Efficient PTAS, Makespan Minimization, Scheduling, Stochastic Load Balancing, Poisson Distribution

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.37

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/2006.12670`

Acknowledgements We thank the anonymous reviewers for their valuable feedback. This work was supported in part by NSF awards CCF-1617851, CCF-1763514, CCF-1910534, CCF-1926872, and CCF-1934876.

1 Introduction

We consider the following fundamental problem in scheduling theory: given n jobs with job sizes $w_1, \ldots, w_n \geq 0$, assign jobs to m machines such that the maximum load of any machine (i.e., the total size of jobs assigned to the machine) is minimized. In other words, we want to partition $[n]$ into sets $S_1, \ldots, S_m$ so as to minimize $\max_{i\in[m]} \sum_{j\in S_i} w_j$. Often referred to as *load balancing* or *makespan minimization*, this is one of the classical NP-complete problems and along with its many variants, has been extensively studied both in theoretical computer science as well as operations research. While the exact problem is hard, this problem admits a polynomial time approximation scheme (PTAS) [12] and later work improved this to an efficient PTAS [16, 15, 14, 1]. Several other variants of this problem have also been studied – this includes (i) the *related machines case* where the machines can have different speeds [13]; (ii) the *unrelated machines case* where the job size itself depends on the machine on which it is scheduled [20]; and (iii) the *precedence constrained case* where there are precedence constraints on schedule of jobs [7, 5].

All the aforementioned variants of this problem have the common feature that the job sizes are known in advance to the algorithm designer. However, in many situations, there might be *uncertainty* in the job size. An obvious way to model this uncertainty is via the framework of stochastic optimization as follows – we have m machines and n jobs where the size of the i^{th} job is given by the random variable $\mathbf{W}_i$. If we now assign the jobs to m machines (given by $S_1, \ldots, S_m$), then the load of the j^{th} machine is given by the random variable $\sum_{i \in S_j} \mathbf{W}_i$. Similar to the case when the job sizes are *deterministic*, in *stochastic load balancing*, one would like to minimize the maximum load. However, since the maximum load (across machines) is itself a random variable – arguably, the most natural objective is to then minimize the expected maximum load. In other words, we seek to find a partition of $[n]$ into sets $S_1, \ldots, S_m$ so as to minimize $\mathbf{E}\left[\max_{j \in [m]} \sum_{i \in S_j} \mathbf{W}_i\right]$.

Throughout this paper, we assume that the random variables $\{\mathbf{W}_i\}_{i=1}^n$ are independent – that is the job sizes are independent of each other. Further, note that the algorithm designer is assumed to know the distribution of the random variables $\{\mathbf{W}_i\}_{i=1}^n$ (though, of course, not the actual realizations of the loads).

To our knowledge, Kleinberg, Rabani and Tardos [18] were the first to consider this problem in the algorithms community. They gave a $O(1)$-factor approximation algorithm for this problem. Soon thereafter, Goel and Indyk [9] considered the problem of obtaining better approximation for special classes of random variables – in particular, (i) if each $\{\mathbf{W}_i\}$ is an exponential random variable, they obtain a PTAS (though not an efficient one); (ii) if each $\{\mathbf{W}_i\}$ is a Poisson random variable, then they obtain a 2-approximation algorithm. In fact, this 2-approximation is obtained by considering the (deterministic) instance with loads $\{w_1, \ldots, w_n\}$ where $w_i = \mathbf{E}[\mathbf{W}_i]$ and then applying Graham's heuristic [10] on this instance.

Somewhat more complicated variants of this problem have also been considered – as an example, Gupta *et al.* [11] considered the problem of stochastic load balancing on unrelated machines. Here, the load of job i on machine j is given by a random variable $\mathbf{W}_{i,j}$. For this variant, [11] gave a $O(1)$ approximation algorithm (thus extending the guarantee of [18] to the case of unrelated machines). Similarly, Molinaro [22] considered the problem of minimizing the expected ℓ_p norm of the loads (the version we have can be seen as minimizing the expected ℓ_∞ norm of the loads). Despite all this impressive progress, the only case where we have a PTAS for stochastic load balancing is when all the loads $\{\mathbf{W}_i\}$ are exponential random variables. As the main result of this paper, we obtain an efficient PTAS for stochastic load balancing when all the loads are Poisson random variables.

▶ **Theorem 1.** *There is an algorithm* $\textsc{PoiScheduling}(n, m, \{\lambda_i\}_{i=1}^n, \epsilon)$ *that given an instance of the load balancing problem with n jobs and m machines where the size of the i^{th} job is $\mathbf{W}_i = \mathsf{Poi}(\lambda_i)$ (i.e. a Poisson random variable with mean λ_i), and a parameter $0 < \epsilon < 1$, outputs a job assignment whose expected maximum load satisfies $L \leq (1+\epsilon)L^*$, where L^* is the expected maximum load of an optimal assignment. The algorithm runs in time* $2^{2^{O(1/\epsilon^2)}} + O(n \log^2 n \log\log^2 n)$.

Theorem 1 is the first PTAS for stochastic load balancing with Poisson jobs. Prior to this result, the best known approximation algorithm for this setting was due to Goel and Indyk [9] (mentioned earlier) and had an approximation factor of 2. In fact, our PTAS is also an *efficient PTAS* – i.e., the running time remains polynomial in n even for some $\epsilon = o(1)$. In contrast, the PTAS from [9] for exponential random variables was *not* an efficient PTAS. Finally, we point out that our running time is doubly exponential in the error parameter ϵ. While this can be potentially improved to a singly exponential dependence in ϵ, it is unlikely to be improved further – in particular, [6] showed that under the ETH, any PTAS for even the deterministic load balancing problem must have a singly exponential dependence on ϵ[1].

1.1 Our techniques

At a high level, to design an algorithm for stochastic load balancing, we must come up with an *algorithmically tractable proxy* for the objective function $\mathbf{E}[\max_{j\in[m]} \sum_{i\in S_j} \mathbf{W}_i]$. However, the expected maxima of random variables (and more generally stochastic processes) can be notoriously difficult to reason about. Indeed, we point out that in the last fifty years, significant effort in probability theory has been devoted towards understanding the maximum of even simple families of random variables such as Gaussians [8, 24]. Despite this challenge, the hope is that by exploiting structural properties of Poisson random variables along with appropriate algorithmic primitives, we will be able to design an efficient PTAS for stochastic load balancing for Poisson jobs.

The starting points of our algorithm are two natural heuristics which have previously been analyzed in the context of stochastic load balancing.

1. The first heuristic is to construct an instance of (deterministic) load balancing where the size of the i^{th} job is $w_i = \mathbf{E}[\mathbf{W}_i]$. One can then apply the PTAS (say from [1]) to get an allocation of the n jobs into m machines. The obvious pitfall here is that the actual job size is a Poisson random variable which may typically be very far from its mean. In other words, this heuristic has a good guarantee provided

$$\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)] \approx \max_{j=1}^{m}[\mathbf{E}\ [\mathsf{Poi}(\mu_j)]],$$

where μ_j is the expected load size of the j^{th} machine[2]. Of course, the above relation may be far from true and indeed, we want to point out that while the left hand side $\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)]$ is just a function of $\mu_1, \ldots, \mu_j$, it is far from being a linear function of $\mu_1, \ldots, \mu_j$. It is easy to create an instance where the optimum obtained by replacing each Poisson load by its expectation is a constant factor away from the true optimum.
Despite this limitation, this heuristic is in fact of both theoretical and practical value. In particular, from a theoretical aspect, recall that $\mathsf{Poi}(\lambda)$ concentrates around λ (with standard deviation $\sqrt{\lambda}$). This can be leveraged to show that if the optimum allocation must necessarily have at least one machine with a (sufficiently) large load, then the allocation for the deterministic load balancing problem provides a near optimal allocation for the stochastic version as well.
2. The second heuristic is a *greedy algorithm* – namely, we first assign an arbitrary order to the jobs and *iteratively assign each job to the machine with the least current expected load.* This is the same as the Graham's rule [10], and is precisely how the authors of [9] obtained

[1] One can reduce an instance of deterministic load balancing to one of Poisson load balancing by scaling up all job sizes such that they all become at least $\omega(\epsilon^{-2}\log m)$. By Chernoff bounds and union bound this reduction preserves $(1 + O(\epsilon))$-approximation.

[2] Note that $\mathsf{Poi}(\lambda_1) + \mathsf{Poi}(\lambda_2) = \mathsf{Poi}(\lambda_1 + \lambda_2)$ if $\mathsf{Poi}(\lambda_1)$ and $\mathsf{Poi}(\lambda_2)$ are independent, so every machine's load is still a Poisson random variable.

a 2-approximation for load balancing Poisson jobs. The underlying rationale for this rule is the following cruical fact about Poisson random variables. Suppose $\mu_1 \geq \mu_2 \geq \mu_3 \geq \mu_4$ such that $\mu_1+\mu_4 = \mu_2+\mu_3$. Then, $\mathbf{E}[\max\{\mathsf{Poi}(\mu_1), \mathsf{Poi}\ (\mu_4)\}] \geq \mathbf{E}[\max\{\mathsf{Poi}(\mu_2), \mathsf{Poi}(\mu_3)\}]$. This fact can in fact be extended to prove that if there is an allocation such that the expected load is the same across all machines, then that is an optimum allocation. Of course, such an allocation might not exist – however, heuristically we might hope that if all the job sizes are small, then we can approximately equalize the expected load on the machines and that such an allocation might have a near-optimal expected maximum load.

It turns out that these heuristics (even when rigorously analyzed) are not sufficient to provide a PTAS for stochastic load balancing in all regimes of job sizes and (number of) machines. Therefore we need some other observations. The first crucial observation is that if there is a job size λ which is more than the average load (i.e. the total expected job size divided by m), then in the optimal allocation, such a job is assigned its own separate machine (Observation 17). This observation can be iteratively applied so that we are now left with job sizes $\{\lambda_i\}_{i=1}^{n'}$ and m' machines such that

$$\max_{i=1}^{n'}\{\lambda_i\} \leq \frac{\sum_{i=1}^{n'} \lambda_i}{m'} := \mu.$$

In other words, no job is larger than the average expected load across the m' machines, i.e., μ. With this simplification, we discuss another *familiar trick* in the context of allocation problems – namely we create a *rounded instance* such that each job size (now call it $\{\lambda_i'\}_{i=1}^{n'}$) is now in the interval $[\epsilon\mu, 2\mu]$. The rounding procedure we apply is identical to the rounding procedure used by [1] in the context of deterministic load balancing. A key property is that the number of different (expected) job sizes in this modified instance is a constant – i.e., only dependent on the target error parameter ϵ.

This rounding step highlights a key technical challenge our algorithm faces – namely, it is possible that by "multiplicatively dilating" the job sizes, the expected maximum load of the machines can change significantly. In other words, suppose $\mu_1, \ldots, \mu_m \geq 0$, then is it the case that for any $0 < \delta < 1$,

$$\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}((1+\delta)\mu_j)] \approx (1+O(\delta))\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)] \ ? \tag{1}$$

While intuitively this looks reasonable, it is not clear if this is true in full generality. Fortunately for us, we obtain the following dichotomy:

1. When μ is very large (this corresponds to the Case 1 in the analysis), we are able to show that the first heuristic above provides a PTAS – in other words, just substituting each stochastic job $\mathbf{W}_i$ with a deterministic job w_i such that $w_i = \mathbf{E}[\mathbf{W}_i]$ and then applying the PTAS for the deterministic case [1, 14, 15] gives a PTAS for the stochastic case. The underlying reason is that in this case, the expected maximum is essentially the same as the expected heaviest load across the m machines.
   The same algorithm also works if μ is in a "certain intermediate range" and m is sufficiently large (this corresponds to Case 3 in the analysis). In fact, in this case, even the greedy heuristic described earlier provides a PTAS. The underlying reason why the deterministic PTAS works is the following: in this regime, the expected maximum remains essentially the same even if all the loads were to go up by a factor of 2.
2. Outside of the above two cases, our heuristics (greedy or deterministic scheduling) fail to provably work. However, in these cases, we are able to prove (1). In other words, we are able to show that dilating or contracting each job size by a factor of $(1+\delta)$ affects

the expected maximum by only a factor of $1 \pm O(\delta)$. Thus, we can apply the rounding procedure from [1] to reduce to the case where the number of different job sizes is a constant. In fact, this is enough to obtain a PTAS for the stochastic load balancing problem though not an efficient PTAS.

Finally, to get an efficient PTAS, we leverage a third property of the "maximum of Poisson random variables" – namely, the so-called "focusing effect" [3, 2, 4]. Roughly speaking, it says that suppose we have m independent Poisson random variables (call them $\mathbf{X}_1, \ldots, \mathbf{X}_m$), each with mean μ, then there is an integer I such that $(\max_{i=1}^m \mathbf{X}_i) \in [I, I+1]$ with probability $1 - o(1)$ as $m \to \infty$. We extend this to (certain instances of) independent but not identically distributed Poisson random variables. Essentially such a "focusing effect", whenever it holds, allows us to express the expected maximum of the loads of m machines as a linear function of the allocation and then employ an integer linear program (ILP) to find the optimal allocation.

To explain how an ILP comes into the picture, first of all, we can assume that m (i.e., the number of machines) is sufficiently large in terms of the target error parameter δ. If this is not the case, then we can simply employ dynamic programming to find a good allocation (it is now an efficient PTAS because m is a constant). Once m is large, we show the following:

(a) Either there is a transition point $t = t(\mu_1, \ldots, \mu_m)$ such that $\max\{\mathsf{Poi}(\mu_1), \ldots, \mathsf{Poi}(\mu_m)\}$ sharply concentrates within $1 \pm O(\delta)$ of the transition point. In this case, we want to find the smallest t^*, for which there is an allocation with loads $\mu_1, \ldots, \mu_m$ such that $t^* = t(\mu_1, \ldots, \mu_m)$. We use ILP and binary search to find such t^*. Observe that in this case, the smallest such t^* will minimize the expected maximum load (up to $1 \pm O(\delta)$).

(b) Otherwise, there is a transition point $t = t(\mu, m)$ such that $\max\{\mathsf{Poi}(\mu_1), \ldots, \mathsf{Poi}(\mu_m)\}$ sharply concentrates in the set $[t-1, t]$. Observe that t only depends on μ and m, and hence can be easily computed. With the knowledge of t, we now use an ILP to find an assignment $\mu_1, \ldots, \mu_m$ which maximizes the probability that $\max\{\mathsf{Poi}(\mu_1), \ldots, \mathsf{Poi}(\mu_m)\} = t - 1$ and thus minimizes the expected maximum load.

1.2 Organization

In Section 2, we formally state the problem, establish some notations, and describe some properties of Poisson random variables that we will utilize. In Section 3 we give an overview of our concentration and scaling results for the maximum of Poisson random variables. In Section 4 we present our efficient polynomial-time approximation scheme and its analysis.

2 Preliminaries

We use $\mathsf{Poi}(\lambda)$ to denote a Poisson random variable with mean λ. Recall that $\mathbf{Pr}[\mathsf{Poi}(\lambda) = k] = e^{-\lambda} \cdot \frac{\lambda^k}{k!}$ for $k \in \mathbb{N}$. In the stochastic load balancing problem considered in this paper, we are given n jobs and m machines where the job sizes are independent Poisson random variables $\mathsf{Poi}(\lambda_1), \mathsf{Poi}(\lambda_2), \ldots, \mathsf{Poi}(\lambda_n)$. We will call λ_i the size of job i. Our goal is to assign the jobs to the machines so that the expected maximum load

$$L \overset{\text{def}}{=} \mathbf{E}\left[\max_{j=1}^{m} \sum_{i \in S_j} \mathsf{Poi}(\lambda_i)\right] \tag{2}$$

is minimized, where S_j is the set of jobs assigned to machine j.

It is well known that the sum of two independent Poisson random variables also follows a Poisson distribution, i.e. $\mathsf{Poi}(\lambda_1)+\mathsf{Poi}(\lambda_2) = \mathsf{Poi}(\lambda_1+\lambda_2)$. Therefore if we let $\mu_j = \sum_{i\in S_j}\lambda_i$, we can write (2) as $L = \mathbf{E}\left[\max_{j=1}^{m}\mathsf{Poi}(\mu_j)\right]$. We will call μ_j the load of machine j.

Henceforth, our analysis of Poisson random variables will mainly serve the purpose of characterizing the expected maximum load, and therefore we will use μ and $\mu_1, \mu_2, \ldots, \mu_m$ to denote the means when stating useful claims about Poisson distributions.

▶ **Definition 2.** *We write* $\mathsf{M}(m,\mu)$ *to denote the random variable whose value is the maximum of* m *i.i.d.* $\mathsf{Poi}(\mu)$.

In [9] the authors proved that Poisson distributions are log-concave:

▶ **Proposition 3** ([9]). *For any* $t \geq 0$, *the function*

$$f_t(\mu) = \log \mathbf{Pr}\left[\mathsf{Poi}(\mu) \leq t\right] \tag{3}$$

is decreasing and concave with respect to μ.

For any random variables $\mathbf{X}$ and $\mathbf{Y}$ taking values on $\mathbb{N}$, we say $\mathbf{X}$ *stochastically dominates* $\mathbf{Y}$, denoted by $\mathbf{X} \geq_{\text{sd}} \mathbf{Y}$, if $\mathbf{Pr}[\mathbf{X} \geq k] \geq \mathbf{Pr}[\mathbf{Y} \geq k]$ holds for every $k \in \mathbb{N}$. Note that for independent $\mathbf{X}, \mathbf{Y}$ we have $\mathbf{Pr}[\max\{\mathbf{X},\mathbf{Y}\} \geq k+1] = 1 - \mathbf{Pr}[\mathbf{X} \leq k]\,\mathbf{Pr}[\mathbf{Y} \leq k]$. Now by Proposition 3 we have the following:

▶ **Proposition 4** (Lemma 2.1 of [9]). *Given* $0 \leq \mu_1 \leq \mu_1' \leq \mu_2' \leq \mu_2$ *such that* $\mu_1+\mu_2 = \mu_1'+\mu_2'$, *it holds that* $\max\{\mathsf{Poi}(\mu_1), \mathsf{Poi}(\mu_2)\} \geq_{\text{sd}} \max\{\mathsf{Poi}(\mu_1'), \mathsf{Poi}(\mu_2')\}$.

Poisson random variables satisfy exponential tail bounds:

▶ **Proposition 5** (Theorem 4.4, Theorem 4.5 of [21]). *Let* $\mathbf{X}$ *be a Poisson random variable with mean* μ. *For* $0 < \delta < 1$ *we have*

$$\mathbf{Pr}[\mathbf{X} \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}, \tag{4}$$

$$\mathbf{Pr}[\mathbf{X} \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}. \tag{5}$$

In our analysis we will need to use Stirling's approximation to deal with factorials:

▶ **Proposition 6** (Stirling's approximation [23]). *For any integer* $n > 0$,

$$e\left(\frac{n}{e}\right)^n \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/(12n+1)} \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/12n} \leq en\left(\frac{n}{e}\right)^n. \tag{6}$$

3 Concentration and Scaling Results for Maximum of Poissons

In this section we present our concentration and scaling results for the maximum of independent Poisson random variables $\mathsf{Poi}(\mu_1), \mathsf{Poi}(\mu_2), \ldots, \mathsf{Poi}(\mu_m)$, which will be used to prove the correctness of our algorithm. Full proofs of these results are deferred to the full version of the paper.

Throughout we assume $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_m \geq 0$, and define $\mu = (\sum_{j=1}^{m}\mu_j)/m$. We use δ as an error parameter, which measures how well the maximum of Poissons is concentrated. We consider five different cases based on the relationship between μ, m, and μ_j's, and state our results for each of them. Note that while the ranges of μ in these cases are disjoint, we prove our lemmas below for slightly overlapping ranges of μ for ease of analyzing our algorithm in Section 4.

Fix $\delta \in (0, 1/10]$. We prove our results for the following cases respectively:

Case 1: $\frac{6}{\delta^2} \log m < \mu$.

Case 2: $\frac{1}{2^{1/\delta+1}} \log m < \mu \le \frac{6}{\delta^2} \log m$ and $m \ge 2^{2^{\frac{2}{\delta}}}$.

Case 3: $\frac{1}{m^\delta} < \mu \le \frac{1}{2^{1/\delta+1}} \log m$, $m \ge 2^{\frac{2}{\delta} \log \frac{2}{\delta}}$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$.

Case 4: $\frac{4 \log m}{m} < \mu \le \frac{1}{m^\delta}$, $m \ge 2^{100/\delta^2}$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$.

Case 5: $\mu \le \frac{4 \log m}{m}$, $m \ge 1000(1/\delta) \log^2(1/\delta)$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$.

For Case 1 we show that $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ is concentrated within $(1 \pm O(\delta))\mu_1$. For each of Case 2, Case 3, and Case 4 we define a certain transition point and show that $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ is concentrated around this point. For Case 5 we show that $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ takes value 0 or 1 with high probability. For all cases we show that the maximum value is robust to contraction or dilation of μ_j's. In particular $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ does not blow up by more than $1 + O(\delta)$ when we scale all μ_j's by $1 + \delta$. We call these *scaling results*.

▶ **Lemma 7** (Case 1). *Suppose $\delta \in (0, 1/10]$ and $\mu > \frac{6}{\delta^2} \log m$. Then*

$$\mu_1 \le \mathbf{E}\left[\max_{j=1}^m \mathsf{Poi}(\mu_j)\right] \le (1 + 5\delta)\mu_1. \tag{7}$$

▶ **Lemma 8** (Case 2). *Suppose $\delta \in (0, 1/10]$, $\frac{1}{2^{1/\delta+1}} \log m < \mu \le \frac{12}{\delta^2} \log m$, and $m \ge 2^{2^{\frac{2}{\delta}}}$. Define transition point $t_2 = t_2(\mu_1, \mu_1, \ldots, \mu_m)$ as the largest integer satisfying*[3]

$$\sum_{j=1}^m \mathbf{Pr}\left[\mathsf{Poi}(\mu_j) \ge t_2\right] \ge \frac{1}{3}. \tag{8}$$

Then for any random variable $\mathbf{X}$ taking values on $\mathbb{N}$, we have

$$(1 - 6\delta)\mathbf{E}\left[\max\{t_2, \mathbf{X}\}\right] \le \mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}(\mu_j), \mathbf{X}\right\}\right] \le (1 + 10\delta)\mathbf{E}\left[\max\{t_2, \mathbf{X}\}\right], \tag{9}$$

and

$$\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}((1+\delta)\mu_j), \mathbf{X}\right\}\right] \le (1 + 16\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}(\mu_j), \mathbf{X}\right\}\right]. \tag{10}$$

▶ **Lemma 9** (Case 3). *Suppose $\delta \in (0, 1/10]$, $\frac{1}{m^\delta} < \mu \le \frac{1}{2^{1/\delta+1}} \log m$, and $m \ge 2^{\frac{2}{\delta} \log \frac{2}{\delta}}$. Define transition point $t_3 = t_3(m, \mu) = \frac{\log m}{\log \frac{1}{\mu} + \log \log m}$. Then for any random variable $\mathbf{X}$ on $\mathbb{N}$ we have*

$$(1 - 4\delta) \max\{t_3, \mathbf{X}\} \le \max\{\mathsf{M}(m, \mu), \mathbf{X}\} \le (1 + 14\delta) \max\{t_3, \mathbf{X}\}, \tag{11}$$

and

$$\mathbf{E}\left[\max\{\mathsf{M}(m, 4\mu), \mathbf{X}\}\right] \le (1 + 20\delta)\, \mathbf{E}\left[\max\{\mathsf{M}(m, \mu), \mathbf{X}\}\right]. \tag{12}$$

[3] The choice of $\frac{1}{3}$ in the definition of t_2 is arbitrary. In principle any constant bounded away from both 1 and 0 suffices.

▶ **Lemma 10** (Case 4). *Suppose $\delta \in (0, 1/10]$, $\frac{4\log m}{m} < \mu \leq \frac{2}{m^\delta}$, $m \geq 2^{100/\delta^2}$, and $\mu_1, \dots, \mu_m \in [\mu/4, 4\mu]$. Define transition point $t_4 = t_4(m, \mu) = \lceil \gamma_4(m, \mu) \rceil$ where $\gamma_4(m, \mu) = \frac{\log m}{\log \frac{4}{\mu} + \log\log m}$. Let $\mathbf{W}$ be a Bernoulli random variable taking values on $t_4 - 1$ and t_4 where $\mathbf{Pr}\left[\mathbf{W} = t_4 - 1\right] = \prod_{j=1}^m \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right) \leq t_4 - 1\right]$. Then for any random variable $\mathbf{X}$ on $\mathbb{N}$ we have*

$$(1 - 5\delta)\mathbf{E}\left[\max\left\{\mathbf{W}, \mathbf{X}\right\}\right] \leq \mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \leq (1 + 16\delta)\mathbf{E}\left[\max\left\{\mathbf{W}, \mathbf{X}\right\}\right], \quad (13)$$

and

$$\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left((1+\delta)\mu_j\right), \mathbf{X}\right\}\right] \leq (1 + 16\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right]. \quad (14)$$

▶ **Lemma 11** (Case 5). *Suppose $\delta \in (0, 1/10]$, $\mu \leq \frac{8\log m}{m}$, $m \geq 1000(1/\delta)\log^2(1/\delta)$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$. Let $\mathbf{W}$ be a 0/1 Bernoulli random variable with $\mathbf{E}\left[\mathbf{W}\right] = 1 - \prod_{j=1}^m \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right) = 0\right] = 1 - e^{-m\mu}$. Then for any random variable $\mathbf{X}$ on $\mathbb{N}$ we have*

$$\max\left\{\mathbf{W}, \mathbf{X}\right\} \leq \mathbf{E}\left[\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right)\right] \leq (1 + 10\delta)\max\left\{\mathbf{W}, \mathbf{X}\right\}, \quad (15)$$

and

$$\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left((1+\delta)\mu_j\right), \mathbf{X}\right\}\right] \leq (1 + 10\delta)\,\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right]. \quad (16)$$

4 An Efficient Polynomial-time Approximation Scheme

Our PTAS for stochastic load balancing is heavily inspired by the approach of [1, 15] for the deterministic load balancing problem. Thus, we first begin with a recap of their approach.

4.1 Recap of the PTAS for deterministic load balancing

Consider any instance of *deterministic* load balancing where the job sizes are $\{\lambda_i\}_{i=1}^n$ and we have m machines – the goal is to find an assignment with smallest maximum load. The algorithms in [1, 15] proceed in two phases: In phase I, we assign "big" jobs to separate machines. Here big jobs are the maximal set of jobs whose size is greater than the remaining average load. In other words, it is the maximal set $B \subseteq [n]$ satisfying that $\forall i \in B$, $\lambda_i > (\sum_{i \notin B} \lambda_i)/(m - |B|) := \mu$. Exploiting the convexity of the objective function (i.e. the function $\max\{x_1, x_2, \dots, x_m\}$), [1, 15] show that an optimum assignment (i) assigns the jobs in B to their own separate machines; (ii) assigns the remaining jobs to the remaining machines in a way such that each of these machines have a load between $\mu/2$ and 2μ.

With this, we are only left with the problem of assigning the small jobs, i.e., the jobs not in B. A second key step here is to round the sizes of the remaining jobs, such that (i) the number of different job sizes is now[4] $\tilde{O}(1/\epsilon)$ and (ii) the potential number of different assignments to any single machine is $2^{\tilde{O}(1/\epsilon)}$. Crucially, both these numbers are just dependent on the target error parameter ϵ. With this rounding, [1, 14] formulate the problem of finding an optimal assignment on the remaining (rounded) jobs as an integer linear program with

[4] Recall that $\tilde{O}(f)$ denotes $O(f \log^c f)$ for some constant c.

$2^{\tilde{O}(1/\epsilon)}$ variables – referred to as a *configuration-IP*. The *configuration-IP* can be solved in time exponential in the number of variables and linear in the input length using algorithms in [19, 17].

▶ **Theorem 12** ([17]). *There is an algorithm ILP that solves an integer linear program with p variables in time $p^{O(p)}O(n)$ where n is the length of input.*

This leads to an overall running time of $2^{2^{\tilde{O}(1/\epsilon)}} + O(n \log n)$ [1], where the running time $O(n \log n)$ comes from the pre-processing step of sorting the job sizes (to find the big jobs).

▶ **Theorem 13** ([1]). *There is an algorithm* DetScheduling *that given an instance of the load balancing problem with n jobs and m machines where the jobs have deterministic sizes $\lambda_1, \lambda_2, \ldots, \lambda_n$, and a parameter $0 < \epsilon < 1$, outputs a job assignment whose maximum load is at most $(1+\epsilon)$ of the maximum load of an optimum assignment. The algorithm runs in time $2^{2^{\tilde{O}(1/\epsilon)}} + O(n \log n)$.*

The authors in [15] then use a *sparsification technique* to show that the *configuration-IP* has an optimum solution with a small support size, which leads to an improved running time of $2^{O(1/\epsilon)\log^4(1/\epsilon)} + O(n \log n)$. Later by improving the runtime of solving the ILP, a running time of $2^{O(1/\epsilon)\log^2(1/\epsilon)} + O(n \log n)$ was achieved [16].

4.2 Overview of our approach

We now give an overview of our approach for the stochastic load balancing problem. Recall that we have n jobs and m machines where the i^{th} job has size $\mathsf{Poi}(\lambda_i)$. Similar to the deterministic case [1, 15], we define "big" jobs as the maximal set of jobs whose (expected) size is greater than the remaining (expected) average load, i.e. the maximal set $B \subseteq [n]$ satisfying that $\forall i \in B$

$$\lambda_i > \frac{\sum_{i \notin B} \lambda_i}{m - |B|}. \tag{17}$$

By Proposition 3, the objective function is convex with respect to the machine loads (similar to [1, 15]). Thus, we assign the jobs in B to separate machines (see Lines 2–5 of Algorithm 1).

Assigning the small jobs (i.e., the jobs outside B) is however somewhat more complicated. As stated in Section 1.1, there are two principal difficulties in applying the approaches from [1, 15] to handling the remaining jobs. One is to discretize the job sizes, and the other is to formulate the problem of minimizing the expected maximum load as an integer linear program. The key to circumventing both these difficulties lies in the (technical) results on concentration and scaling of maximum of Poisson random variables proven in Section 3. To understand their role, let us begin with some notation. Let $m^{(1)} = m - |B|$ denote the number of the remaining machines. Consider an assignment of the remaining jobs and let $\mathsf{Poi}(\mu_1), \mathsf{Poi}(\mu_2), \ldots, \mathsf{Poi}(\mu_{m^{(1)}})$ be the corresponding distributions of the machine loads. Suppose $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_{m^{(1)}}$ and let $\mu = (\sum_{j=1}^{m^{(1)}} \mu_j)/m^{(1)}$. By an argument similar to the deterministic case [1, 15] (see Observation 18), we can restrict ourselves to the case when $\mu_j \in [\mu/2, 2\mu]$ for all $j \in [m^{(1)}]$. Let $\delta \in (0,1)$ be a target error parameter (roughly speaking, we set $\delta \approx \Theta(\epsilon)$).

Our results in Section 3 first imply that when μ is sufficiently large in terms of $1/\delta$ and $m^{(1)}$ (the condition of Lemma 7), or μ is in a certain intermediate range but $m^{(1)}$ is large enough in terms of $1/\delta$ (the condition of Lemma 9), it suffices to find an assignment by running the deterministic load balancing algorithm in Theorem 13. When μ does not satisfy the above conditions but $m^{(1)}$ is sufficiently large in terms of $1/\delta$, we have the following dichotomy:

1. $\max_{j=1}^{m^{(1)}} \mathsf{Poi}(\mu_j)$ is concentrated within $(1 \pm O(\delta))$ of a certain transition point $t = t(\mu_1, \ldots, \mu_{m^{(1)}})$ (Lemma 8), or
2. takes value $\lceil t \rceil - 1$ or $\lceil t \rceil$ with high probability, where the transition point $t = t(m^{(1)}, \mu)$ only depends on $m^{(1)}$ and μ (Lemmas 10 and 11).

Further, in all of the cases,

$$\mathbf{E}\left[\max_{j=1}^{m} \mathsf{Poi}((1+\delta)\mu_j)\right] \leq (1 + O(\delta))\mathbf{E}\left[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)\right] \tag{18}$$

(Lemmas 8, 10, and 11). Finally, when m is sufficiently small in terms of δ, we can use dynamic programming to get an efficient PTAS.

Let us now see how the above structural results are useful in algorithm design – first of all, (18) immediately allows us to discretize the job sizes by rounding up their means to the nearest integer power of $(1 + \delta)$, with proper handling of jobs with size below a certain threshold[5]. Once the job sizes are rounded, the existence of the transition points (rather, the precise definitions of these transition points in Section 3) can be used to construct integer linear programs which can find a near optimal solution. As an example, if we are in Case 4 as defined above, by Lemma 10 minimizing the expected maximum is the same as finding $\mu_1, \mu_2, \ldots, \mu_{m^{(1)}}$ such that the probability that the maximum load is $\lceil t \rceil$ is minimized. This finishes our overview of the PTAS.

4.3 Our PTAS

We give a full description of our efficient PTAS in Algorithm 1 as PoiScheduling, which calls Rounded (Algorithms 2), ILPScheduling (Algorithm 3), and DPScheduling as subroutines. The detailed description of DPScheduling is deferred to the full version as it uses mostly standard ideas.

PoiScheduling first handles the "big" jobs in the same way as deterministic case (Lines 2–5 of Algorithm 1). As before μ denotes the remaining average machine load (i.e. RHS of (17)) and $m^{(1)}$ denote the number of the remaining machines. PoiScheduling then does one of the following for the remaining jobs:

1. When μ is large enough to meet the condition of Case 1, or μ and $m^{(1)}$ meet the condition of Case 3, PoiScheduling directly uses the algorithm for deterministic case as a blackbox (Lines 7–9 of Algorithm 1).
2. When μ and $m^{(1)}$ meet the condition of Case 2, Case 4, or Case 5, PoiScheduling first rounds the sizes of the remaining jobs in the same way as [15] (Line 6 of Algorithm 1). Then for Case 2 it uses integer linear programming in conjunction with a binary search to find the smallest transition point t_2 achievable by an assignment of the remaining jobs, where the ILPs have linear objective functions and *configuration-IP* from [1, 15] as feasibility constraints (Lines 10–12 of Algorithm 1). Case 4 and Case 5 are also handled by integer linear programs (Lines 13-15 and Lines 16-17 of Algorithm 1 respectively).
3. When none of the conditions of Case 1 - Case 5 is met, namely $m^{(1)} \leq 2^{2^{O(1/\epsilon)}}$ and $\mu \leq O(\epsilon^{-2} \log m)$, PoiScheduling finds an assignment by dynamic programming (Line 19 of Algorithm 1).

PoiScheduling uses Rounded, ILPScheduling and DPScheduling as subroutines. Roughly speaking, Rounded takes a multi-set of job sizes and a parameter δ as input and outputs a multi-set of job sizes such that the number of different job sizes only depends

[5] The specific rounding scheme we use is identical to the one used in [15].

on δ. ILPSCHEDULING takes the jobs and the number of machines m, and a function f as input. The goal of ILPSCHEDULING is to find an assignment with loads $\mu_1, \dots, \mu_m$ such that $\sum_{j=1}^{m} f(\mu_j)$ is minimized. DPSCHEDULING takes the jobs and number of machine, and an error parameter ϵ as input and returns an assignment of jobs with at most $(1+\epsilon)$ error with respect to an optimum assignment by a dynamic programming.

By using the algorithm in [17] (Theorem 12) to solve the integer linear programs, POISCHEDULING achieves a running time double exponential in $1/\epsilon$ and nearly-linear in n. Note that although the algorithm in Theorem 12 needs integral coefficients, we can compute the coefficients with a high precision (inverse polynomial precision) which is sufficient to get our desired approximation and does not affect our running time, so we don't get into the details. We also note that while it is possible to use the sparsification technique in [15] to improve our running time of solving integer linear programs to single exponential in $1/\epsilon$, our dynamic program for the case when $1/\epsilon \geq \Omega(\log\log m)$ still takes time double exponential in $1/\epsilon$.

The performance of POISCHEDULING is characterized in Theorem 1. The performances of ROUNDED and DPSCHEDULING are characterized in Lemmas 14 and 15 respectively. We do not give a separate lemma for ILPSCHEDULING but analyze it in our proofs directly.

Note that Lemma 14 below only gives guarantees for how the job sizes and individual machine loads change after rounding; the lemma itself does not make assertions about the expected maximum load. Instead, guarantees for the latter will follow from our scaling results in Section 3.

▶ **Lemma 14** ([15]). *The algorithm $\{\lambda'_i\}_{i=1}^{n'} = \text{ROUNDED}(\{\lambda_i\}_{i=1}^{n}, \mu, \delta)$ runs in time $O(n)$. Suppose all $\lambda_i \leq \mu$, $\delta \in (0,1)$, and $\mu = (\sum_{i=1}^{n} \lambda_i)/m$ for an integer m. The number of different sizes in $\{\lambda'_i\}_{i=1}^{n'}$ is bounded by $O(\frac{1}{\delta}\log\frac{1}{\delta})$, and each $\lambda'_i = \delta\mu + k\delta^2\mu$ for some $k \in \mathbb{Z}_{\geq 0}, k \leq \frac{2}{\delta^2}$. n' is bounded by $O(m/\delta)$. For any assignment of $\{\lambda_i\}_{i=1}^{n}$ to m machines with loads $\mu_1, \mu_2, \dots, \mu_m$, there is an assignment of $\{\lambda'_i\}_{i=1}^{n'}$ to m machines with loads $\mu'_1, \mu'_2, \dots, \mu'_m$ such that all $\mu'_j \leq (1+5\delta)\mu_j$. Conversely, for any assignment $\{\lambda'_i\}_{i=1}^{n'}$ to m machines with loads $\mu'_1, \mu'_2, \dots, \mu'_m$, there is an assignment of $\{\lambda_i\}_{i=1}^{n}$ to m machines with loads $\mu_1, \mu_2, \dots, \mu_m$ such that all $\mu_j \leq (1+5\delta)\mu'_j$, and the latter assignment can be found in $O(n)$ time given the former assignment if both $\{\lambda_i\}_{i=1}^{n}$ and $\{\lambda'_i\}_{i=1}^{n'}$ are sorted.*

▶ **Lemma 15.** *Given jobs $\{\nu_i\}_{i=1}^{n^{(0)}}$, $\{\lambda_i\}_{i=1}^{n^{(1)}}$, number of machines m, and $\epsilon \in (0, \frac{1}{10}]$. Let $m^{(1)} = m - n^{(0)}$, $\mu = (\sum_{i=1}^{n^{(1)}} \lambda_i)/m^{(1)}$, and $\delta = \max\left\{\frac{\epsilon}{1000m^{(1)}}, 2^{-10^9/\epsilon^2}\right\}$. Suppose all $\lambda_i \leq \mu$, all $\nu_i > \mu$, and $\mu \leq 6000000\epsilon^{-2}\log m^{(1)}$. Then DPSCHEDULING$(\{\nu_i\}_{i=1}^{n^{(0)}}, \{\lambda_i\}_{i=1}^{n^{(1)}}, m, \epsilon)$ finds in $O(n^{(0)}\epsilon^{-4}\log^2 m^{(1)}) + (m^{(1)}/\delta)^{O(\frac{1}{\delta}\log\frac{1}{\delta})}$ time an assignment with expected maximum load $L \leq (1+\epsilon)L^*$, where L^* is the expected maximum load of an optimum assignment.*

A detailed proof of Lemma 15 is deferred to the full version of the paper. First we show how to prove Theorem 1 using the lemma.

4.4 Proof of Theorem 1

The following lemma shows that if Algorithm 1 does not use dynamic programming to find an assignment, then we have a good approximation.

▶ **Lemma 16.** *If Algorithm 1 returns an assignment without going into Line 19, then $L \leq (1+\epsilon)L^*$, where L, L^* is the expected maximum load of the assignment returned by Algorithm 1 and the optimum assignment respectively.*

Algorithm 1 $\textsc{PoiScheduling}(n, m, \{\lambda_i\}_{i=1}^{n}, \epsilon)$.

```
   Input  : Number of jobs $n$, number of machines $m$, job sizes $\{\lambda_i\}_{i=1}^{n}$, and $\epsilon \in (0,1)$
   Output : A job assignment $\phi : [n] \to [m]$
1  $\mu \leftarrow (\sum_{i=1}^{n} \lambda_i)/m$, and sort $\{\lambda_i\}_{i=1}^{n}$ such that $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$.
2  $n^{(1)} \leftarrow n$ and $m^{(1)} \leftarrow m$.
3  while $\lambda_{n^{(1)}} > \mu$ do
4      Assign the job with size $\lambda_{n^{(1)}}$ to the empty machine $m^{(1)}$: $\phi(n^{(1)}) \leftarrow m^{(1)}$.
5      $\mu \leftarrow \frac{m^{(1)}\mu - \lambda_{n^{(1)}}}{m^{(1)}-1}$, $n^{(1)} \leftarrow n^{(1)} - 1$, and $m^{(1)} \leftarrow m^{(1)} - 1$.
6  $\delta \leftarrow \frac{\epsilon}{1000}$ and $\{\lambda'_i\}_{i=1}^{n'} \leftarrow \textsc{Rounded}(\{\lambda_i\}_{i=1}^{n^{(1)}}, \mu, \delta)$.
7  if $\mu > \frac{6}{\delta^2}\log m^{(1)}$ or $\left(\frac{1}{(m^{(1)})^{\delta}} < \mu \le \frac{1}{2^{1/\delta+1}}\log m^{(1)}$ and $m^{(1)} \ge 2^{\frac{2}{\delta}\log\frac{2}{\delta}}\right)$ then
       // Case 1, Case 3
8      Create a new instance with $m^{(1)}$ machines and *deterministic* job sizes $\{\lambda_i\}_{i=1}^{n^{(1)}}$.
9      Run $\textsc{DetScheduling}$ in Theorem 13 to find a $(1+\frac{\epsilon}{5})$-optimum assignment
         $\phi' : [n^{(1)}] \to [m^{(1)}]$.
10 else if $\frac{1}{2^{\delta+1}}\log m^{(1)} < \mu \le \frac{6}{\delta^2}\log m^{(1)}$ and $m^{(1)} \ge 2^{2^{2/\delta}}$ then
       // Case 2
11     Use binary search to find the smallest $t_2 \in [\mu, 100\mu\log m^{(1)}]$ s.t. there is an
         assignment of $\{\lambda'_i\}_{i=1}^{n'}$ with loads $\{\mu_j\}_{j=1}^{m^{(1)}}$ s.t. $\sum_{j=1}^{m^{(1)}} \mathbf{Pr}[\mathsf{Poi}(\mu_j) > t_2] < \frac{1}{3}$; this
         is by $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to \mathbf{Pr}[\mathsf{Poi}(x) > t_2])$ for
         each guess of $t_2$ and checking if $\mathtt{opt} < \frac{1}{3}$.
12     $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to \mathbf{Pr}[\mathsf{Poi}(x) > t_2])$.
13 else if $\frac{4\log m^{(1)}}{m^{(1)}} < \mu \le \frac{1}{(m^{(1)})^{\delta}}$ and $m^{(1)} \ge 2^{100/\delta^2}$ then
       // Case 4
14     $t_4 \leftarrow \left\lceil \frac{\log m^{(1)}}{\log\frac{4}{\mu} + \log\log m^{(1)}} \right\rceil$.
15     Find an assignment of $\{\lambda'_i\}_{i=1}^{n'}$ with loads $\{\mu_j\}_{j=1}^{m^{(1)}}$ s.t. $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}[\mathsf{Poi}(\mu_j) < t_4]$ is
         maximized, by
         $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to -\ln(\mathbf{Pr}[\mathsf{Poi}(x) < t_4]))$.
16 else if $\mu \le \frac{4\log m^{(1)}}{m^{(1)}}$ and $m^{(1)} \ge 1000(1/\delta)\log^2(1/\delta)$ then
       // Case 5
17     Find an assignment of $\{\lambda'_i\}_{i=1}^{n'}$ with loads $\{\mu_j\}_{j=1}^{m^{(1)}}$ s.t. $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}[\mathsf{Poi}(\mu_j) = 0]$ is
         maximized, by
         $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to -\ln(\mathbf{Pr}[\mathsf{Poi}(x) = 0]))$.
18 else
19     $\phi' \leftarrow \textsc{DPScheduling}(\{\lambda_i\}_{i=n^{(1)}+1}^{n}, \{\lambda_i\}_{i=1}^{n^{(1)}}, m, \epsilon)$.
20 If $\phi'$ is an assignment of the rounded jobs $\{\lambda'_i\}_{i=1}^{n'}$, use Lemma 14 to covert it to an
     assignment of $\{\lambda_i\}_{i=1}^{n^{(1)}}$ such every machine's load overflows by no more than $(1+5\delta)$.
21 Return the assignment $\phi \cup \phi'$.
```

Algorithm 2 $\textsc{Rounded}(\{\lambda_i\}_{i=1}^n, \mu, \delta)$.

```
   Input  : Job sizes $\{\lambda_i\}_{i=1}^n$, $\mu$ s.t. all $\lambda_i \le \mu$, and $\delta \in (0,1)$
   Output : Rounded job sizes $\{\lambda'_i\}_{i=1}^{n'}$
1  $n' \leftarrow 0$ and $S \leftarrow 0$.
2  for $i \leftarrow 1$ to $n$ do
3      if $\lambda_i \ge \delta\mu$ then
4          $n' \leftarrow n' + 1$.
5          $\nu \leftarrow (1+\delta)^k \delta\mu$ where $k$ is the unique integer s.t.
             $(1+\delta)^{k-1}\delta\mu < \lambda_i \le (1+\delta)^k \delta\mu$.
6          $\lambda_{n'} \leftarrow l\delta^2\mu$ where $l$ is the unique integer s.t. $(l-1)\delta^2\mu < \nu \le l\delta^2\mu$.
7      else
8          $S \leftarrow S + \lambda_i$.
9  $S^{\#} \leftarrow k\delta\mu$ where $k$ is the unique integer s.t. $(k-1)\delta\mu < S \le k\delta\mu$.
10 $\lambda'_i \leftarrow \delta\mu$ for each $i = n'+1, \dots, n' + S^{\#}/(\delta\mu)$ and $n' \leftarrow n' + S^{\#}/(\delta\mu)$.
11 return $\{\lambda'_i\}_{i=1}^{n'}$
```

Algorithm 3 $\text{ILP}\textsc{Scheduling}(\{\lambda_i\}_{i=1}^n, m, f)$.

```
   Input  : Job sizes $\{\lambda_i\}_{i=1}^n$, number of machines $m$, and a function $f : \mathbb{R} \to \mathbb{R}$
   Output : An optimum value opt and a job assignment $\phi : [n] \to [m]$
1  $\mu \leftarrow (\sum_{i=1}^n \lambda_i)/m$.
2  Let $\pi_1 < \pi_2 < \dots < \pi_d$ be all different sizes in $\{\lambda_i\}_{i=1}^n$ and $\vec{\pi} \leftarrow (\pi_1, \pi_2, \dots, \pi_d)^T$.
3  Let $n_k$ be the number of jobs with size $\pi_k$ and $\vec{n} \leftarrow (n_1, n_2, \dots, n_d)^T$ (so
     $\sum_{k=1}^d n_k = n$).
4  $Q \leftarrow \left\{\vec{c} \in \mathbb{Z}_{\ge 0}^d : \vec{c}^T\vec{\pi} \le 4\mu\right\}$ (the set of possible assignments to a single machine).
5  Solve the following integer linear programming using Theorem 12:
```

$$\begin{aligned} \min & \sum_{\vec{c} \in Q} x_{\vec{c}}\, f(\vec{c}^T\vec{\pi}) \\ s.t. & \sum_{\vec{c} \in Q} x_{\vec{c}} = m \\ & \sum_{\vec{c} \in Q} x_{\vec{c}}\, \vec{c} = \vec{n} \\ & x_{\vec{c}} \in \mathbb{Z}_{\ge 0},\ \forall \vec{c} \in Q. \end{aligned} \tag{19}$$

```
6  Return the optimum value of the ILP and an assignment $\phi$ extracted from its
     solution.
```

We will provide a detailed proof of Lemma 16 in the next section. Roughly speaking, for Case 1 and Case 3, where we do not use the rounded jobs, using the guarantee of the deterministic load balancing algorithm we show that the maximum expected load of any machine is bounded. Then by using the concentration results from Section 3, we can bound the expected maximum load.

For three other cases, where the assignment obtained is for the rounded jobs, the proof consists of two main steps. In the first step, we compare the individual machine loads of the output assignment (of unrounded jobs) and the assignment of the rounded jobs given by ILPScheduling by Lemma 14. Then by the concentration and scaling results from Section 3, we bound the expected maximum load of the output assignment using the transition point of rounded jobs. In the second step, we bound the expected maximum of an optimum assignment of rounded jobs by that of an optimum assignment of unrounded jobs by scaling results from Section 3. Thus, we bound the expected maximum load of an output assignment by that of an optimum assignment.

Now we prove Theorem 1.

Proof of Theorem 1. First we analyze the running time when PoiScheduling does not use dynamic programming. If the algorithm invokes the efficient PTAS for deterministic case in Theorem 13, the running time is bounded by $2^{2^{\tilde{O}(1/\epsilon)}} + O(n \log n)$. Otherwise the algorithm will first do the rounding. By Lemma 14 after the rounding the number of different job sizes in $\{\lambda'_i\}_{i=1}^{n'}$ becomes $O(\frac{1}{\delta}\log\frac{1}{\delta}) = O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ and all job sizes are between $\delta\mu$ and 2μ. Now for each machine of $1, 2, \ldots, m^{(1)}$, we only need to consider the assignments to it with load at most 4μ. Therefore each machine can have at most $O(1/\delta) = O(1/\epsilon)$ jobs, and the number of different job profiles that can be assigned to one machine is at most $(1/\epsilon)^{O(1/\epsilon)} = 2^{O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$, which is the number of variables in the ILP. By Theorem 12 the ILP can be solved in $2^{2^{O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}} O(\log n)$ time. Since for Case 2 we need to do a binary search on interval $[\mu, 100\mu\log m^{(1)}]$, the total running time is bounded by $2^{2^{O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}} O(\log n \log\log n)$.

If PoiScheduling uses dynamic programming, we have $\delta \geq 2^{-10^9/\epsilon^2}$ and $m^{(1)} \leq 2^{2^{O(1/\epsilon)}}$. Therefore the total running time is bounded by $2^{2^{O(1/\epsilon^2)}} + O(n\epsilon^{-4}\log^2 n)$ by Lemma 15. Combining these two cases gives us the desired running time.

The approximation guarantee directly follows from Lemmas 16 and 15. ◀

5 Proof of Correctness using Concentration Results

In this section, we will prove the correctness of Algorithm 1 using the concentration results in Section 3. We refer to the case in Lemma 16 as the case with "large m" and the other case as the one with "small m". Here we only give the proof for large m but defer the proof for small m to the full version of the paper.

5.1 Analysis for large m

In this subsection, we prove that if Algorithm 1 does not call DPScheduling (Line 19), then $L < (1+\epsilon)L^*$ where L^* is the expected maximum load of an optimum assignment and L is the expected maximum load of the returned assignment by Algorithm 1. In our analysis, by large m we mean that the condition on Line 7, 10, 13, or 16 is satisfied. By small m we mean that none of those conditions is satisfied, i.e. the algorithm calls DPScheduling. In this subsection, we will also consider the case when the algorithm finishes assigning all jobs before Line 6.

▶ **Lemma 16.** *If Algorithm 1 returns an assignment without going into Line 19, then $L \leq (1+\epsilon)L^*$, where L, L^* is the expected maximum load of the assignment returned by Algorithm 1 and the optimum assignment respectively.*

Proof of Lemma 16. Let χ be the set of jobs each of which is assigned to a single machine in the first loop of Algorithm 1. Let Λ be the (multi-)set of the sizes of the jobs in χ. Let $\boldsymbol{\chi} = \{\mathsf{Poi}(\lambda) \,|\, \lambda \in \Lambda\}$ and $\mathbf{X} = \max_{\lambda\in\Lambda} \mathsf{Poi}(\lambda)$. Consider μ and $m^{(1)}$ as same as Algorithm 1 after Line 6. Suppose the algorithm returns an assignment where machine j has load μ_j. Let $\left\{\mu_j^{(1)}\right\}_{j=1}^{m^{(1)}}$ be the machine loads corresponding to the assignment ϕ' obtained *before* Line 20.

That is, if ϕ' is an assignment of the rounded jobs, $\mu_j^{(1)}$'s are the corresponding rounded loads. Without loss of generality, we assume $\mu_1^{(1)} \geq \mu_2^{(1)} \geq \ldots \geq \mu_{m^{(1)}}^{(1)}$. Let $\mu^{(1)} = \frac{\sum_{j=1}^{m^{(1)}} \mu_j^{(1)}}{m(1)}$. Note that if Algorithm 1 does not return an assignment using rounded jobs (Line 9), then $\mu = \mu^{(1)}$. There are six possible cases:

Case 0: The algorithm finishes assigning all jobs before Line 6. Then it must be the case that $n \leq m$. Therefore assigning every job to a single machine is the optimum solution.

Case 1: $\mu > \frac{6}{\delta^2} \log m^{(1)}$ and the algorithm goes to Lines 7–9. First we state an observation. Note that this is the same observation as **Observation 2.1** in [1] for deterministic case. Both our observation and the one in [1] follow from the convexity of the objective function with respect to the machine loads.

▶ **Observation 17.** *There is an optimum assignment such that each job in χ is solely assigned to a separate machine.*

Proof. Suppose in an optimum assignment there is a job in χ with size λ_i which is assigned to a machine with another job of size λ_j and suppose i is the biggest index such that a job with size λ_i shares its machine with another job. Let μ' be the value of μ in Algorithm 1 just before assigning λ_i to a machine, so we know $\lambda_i > \mu'$. By removing λ_j from its machine, all the other machines should have load at least μ', otherwise we can assign λ_j to a machine with load less than μ' and by Proposition 4, the expected maximum decreases which contradicts the optimality. But μ' is the average load of remaining machines (machines that do not contain any jobs from $\{\lambda_k | k > i\}$) and now each one has load more than μ', a contradiction. ◀

Let L^*_{DET} be the optimum answer of deterministic case for job sizes of $\{\lambda_i\}_{i=1}^{n^{(1)}}$ and $m^{(1)}$ machines, where $n^{(1)} = n - |\chi|$. Note that $L^* \geq L^*_{DET}$ and $L^* \geq \max_{\lambda \in \Lambda} \lambda$, by Observation 17. Since $\delta = \frac{\epsilon}{1000}$, by Lemma 7 we get:

$$\begin{aligned} L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \\ &\leq \mathbf{E}\left[\max\left\{\mathsf{Poi}\left((1+\epsilon/5)L^*_{DET}\right), \max_{j=2}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \quad \text{(by Theorem 13)} \\ &\leq (1+5\delta)\max\left\{(1+\epsilon/5)L^*_{DET}, \max_{\lambda\in\Lambda}\lambda\right\} \quad \text{(by (7) in Lemma 7)} \\ &\leq (1+\epsilon/200)(1+\epsilon/5)L^* < (1+\epsilon)L^*. \end{aligned} \tag{20}$$

Case 2: The algorithm goes to Lines 10–12. By calling ILPSCHEDULING multiple times, Algorithm 1 finds the smallest transition point t_2 for rounded jobs on $m^{(1)}$ machines as defined in Lemma 8. Note that if $\hat{L}$ is the optimum expected maximum for rounded version, then $\hat{L} \leq (1+80\delta)L^*$ (by (10) in Lemma 8 and Lemma 14). Since $\delta = \frac{\epsilon}{1000}$, $m^{(1)} > 2^{2^{100/\epsilon}}$ and $\frac{1}{2^{1/\delta+1}} \log m^{(1)} < \mu^{(1)} \log m^{(1)} \leq (1+5\delta)\frac{6}{\delta^2} \log m^{(1)} < \frac{12}{\delta^2} \log m^{(1)}$ (Lemma 14), then by Lemma 8 we get:

$$\begin{aligned} L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \\ &\leq \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left((1+5\delta)\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(Lemma 14)} \end{aligned}$$

$$\begin{aligned}
\ldots &\le (1+80\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(by (10) in Lemma 8)}\\
&\le (1+10\delta)(1+80\delta)\mathbf{E}\left[\max\left\{t_2, \mathbf{X}\right\}\right] \quad \text{(by (9) in Lemma 8)}\\
&\le \frac{(1+10\delta)(1+80\delta)}{1-6\delta}\hat{L} \quad \text{(by (9) in Lemma 8)}\\
&\le \frac{(1+10\delta)(1+80\delta)^2}{1-6\delta}L^*\\
&< (1+\epsilon)L^*
\end{aligned}$$

Case 3: $\frac{1}{(m^{(1)})^\delta} < \mu \le \frac{1}{2^{1/\delta+1}} \log m^{(1)}$ and $m^{(1)} \ge 2^{\frac{2}{\delta}\log\frac{2}{\delta}}$, and the algorithm goes to Lines 7–9. First we state an observation which is the same as **Observation 2.2** in [1] for deterministic case and again follows from the convexity of the objective function with respect to the machine loads

▶ **Observation 18.** *If for each i, $\lambda_i \le \mu$, then for each load μ_j in an optimum assignment we have $\mu/2 \le \mu_j \le 2\mu$.*

Proof. Suppose a machine has load $\mu_j > 2\mu$ and a job assigned to the machine is λ_i. $\lambda_i \le \mu$, so $\mu_j - \lambda_i > \mu$, so any other machines has load at least μ, a contradiction. So $\mu_j \le 2\mu$.

Suppose a machine has load $\mu_j < \mu/2$. So there is a machine with load $\mu_k > \mu$, so it has at least two jobs assigned. So there is a job λ_i in μ_k such that $\lambda_i \le \mu_k/2$, by taking it from μ_k its load would be $\mu_k - \lambda_i \ge \mu_k/2 > \mu/2 > \mu_j$, a contradiction as by reassigning λ_i to μ_j by Proposition 4 the expected maximum decreases. So $\mu/2 \le \mu_j$. ◀

Since $\delta = \frac{\epsilon}{1000}$ and $m^{(1)} > 2^{\frac{2}{\delta}\log\frac{2}{\delta}}$, the conditions of Lemma 9 holds and we get:

$$\begin{aligned}
L = \mathbf{E}&\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right]\\
&\le \mathbf{E}\left[\max\left\{\mathsf{M}(m^{(1)}, (1+\epsilon/5)2\mu), \mathbf{X}\right\}\right] \quad \text{(Observation 18 and Theorem 13)}\\
&\le \mathbf{E}\left[\max\left\{\mathsf{M}(m^{(1)}, 4\mu), \mathbf{X}\right\}\right] \quad \text{(Proposition 3)}\\
&\le (1+20\delta)\,\mathbf{E}\left[\max\left\{\mathsf{M}(m^{(1)}, \mu), \mathbf{X}\right\}\right] \quad \text{(by (12) in Lemma 9)}\\
&< (1+\epsilon)L^*.
\end{aligned}$$

Case 4: The algorithm goes to Lines 13–15. By putting $\delta = \frac{\epsilon}{1000}$ and Observation 18, an optimum assignment of rounded jobs satisfies the condition of Lemma 10. Let $\mathbf{W}$ be the Bernoulli random variable as described in Lemma 10 but with respect to the machine loads of an optimum assignment of the rounded jobs, then $(1-5\delta)\mathbf{E}\left[\max\left\{\mathbf{W}, \boldsymbol{X}\right\}\right] \le \hat{L}$ where $\hat{L}$ is the optimum expected maximum of rounded version. By finding an assignment maximimizing $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right) \le t_4 - 1\right]$, the value of $\mathbf{E}\left[\max\left\{\mathbf{W}, \boldsymbol{X}\right\}\right]$ would be minimized. Let the corresponding optimum $\mathbf{W}$ be $\mathbf{W}^*$. So we have $(1-5\delta)\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \le (1-5\delta)\mathbf{E}\left[\max\left\{\mathbf{W}, \mathbf{X}\right\}\right] \le \hat{L}$. Note that $\hat{L} \le (1+80\delta)L^*$ by (14) in Lemma 10 and Lemma 14.

As $\frac{4\log m}{m} < \mu^{(1)} \le (1+5\delta)\frac{1}{m^\delta} \le \frac{2}{m^\delta}$, by Lemma 10:

$$\begin{aligned}
L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \\
&\le \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left((1+5\delta)\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(Lemma 14)} \\
&\le (1+80\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(by (14) in Lemma 10)} \\
&\le (1+16\delta)(1+80\delta)\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \quad \text{(by (13) in Lemma 10)} \\
&\le \frac{(1+16\delta)(1+80\delta)}{1-5\delta}\hat{L} \\
&\le \frac{(1+16\delta)(1+80\delta)^2}{1-20\delta}L^* \\
&< (1+\epsilon)L^*. && (21)
\end{aligned}$$

Case 5: The algorithm returns an assignment on Line 17. With the same argument as the previous case, by maximizing $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right)=0\right]$, the value $\mathbf{E}\left[\max\left\{\mathbf{W}, \boldsymbol{X}\right\}\right]$ as defined in Lemma 11 would be minimized. Let the corresponding optimum $\mathbf{W}$ be $\mathbf{W}^*$. So we have $\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \le \hat{L}$ where $\hat{L}$ is the optimum expected maximum of rounded jobs with $m^{(1)}$ machines and $\hat{L} \le (1+50\delta)L^*$ by (16) in Lemma 11 and Lemma 14. So we get:

$$\begin{aligned}
L &\le \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left((1+5\delta)\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(Lemma 14)} \\
&\le (1+50\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(by (16) in Lemma 11)} \\
&\le (1+10\delta)(1+50\delta)\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \quad \text{(by (15)) in Lemma 11)} \\
&\le (1+10\delta)(1+50\delta)\hat{L} \\
&\le (1+10\delta)(1+50\delta)^2 L^* \\
&< (1+\epsilon)L^*.
\end{aligned}$$

So for all cases we have $L < (1+\epsilon)L^*$ and we are done. ◀

References

1 Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

2 Clive W Anderson, Stuart G Coles, and Jürg Hüsler. Maxima of poisson-like variables and related triangular arrays. *The Annals of Applied Probability*, pages 953–971, 1997.

3 CW Anderson. Extreme value theory for a class of discrete distributions with applications to some stochastic processes. *Journal of Applied Probability*, 7(1):99–113, 1970.

4 Keith M Briggs, Linlin Song, and Thomas Prellberg. A note on the distribution of the maximum of a set of poisson random variables. *arXiv preprint*, 2009. `arXiv:0903.4373`.

5 Chandra Chekuri and Michael Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41(2):212–224, 2001.

6 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 657–668. SIAM, 2014.

7 Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

8 D Darling. On the supremum of a certain gaussian process. *The Annals of Probability*, pages 803–806, 1983.

9 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 579–586, 1999. `doi:10.1109/SFFCS.1999.814632`.

10 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, pages 1563–1581, 1966.

11 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1274–1285. SIAM, 2018.

12 Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

13 Hochbaum, Dorit S and Shmoys, David B. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM journal on computing*, 17(3):539–551, 1988.

14 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 562–573, 2009. `doi:10.1007/978-3-642-02927-1_47`.

15 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 72:1–72:13, 2016. `doi:10.4230/LIPIcs.ICALP.2016.72`.

16 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 43:1–43:17, 2019. `doi:10.4230/LIPIcs.ITCS.2019.43`.

17 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. `doi:10.1287/moor.12.3.415`.

18 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.

19 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. `doi:10.1287/moor.8.4.538`.

20 Jan Karel Lenstra, David B Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

21 Michael Mitzenmacher and Eli Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

22 Marco Molinaro. Stochastic ℓ_p load balancing and moment problems via the l-function method. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 343–354. SIAM, 2019.

23 Herbert Robbins. A remark on stirling's formula. *The American mathematical monthly*, 62(1):26–29, 1955.

24 Michel Talagrand. Majorizing measures: the generic chaining. *The Annals of Probability*, 24(3):1049–1103, 1996.